Extending XReason: Formal Explanations for Adversarial Detection

Amira Jemaa, Adnan Rashid, and Sofiène Tahar

Department of Electrical and Computer Engineering
Concordia University, Montreal, QC, Canada,
{a_jem, rashid, tahar}@ece.concordia.ca

Abstract. Explainable Artificial Intelligence (XAI) plays an important role in improving the transparency and reliability of complex machine learning models, especially in critical domains such as cybersecurity. Despite the prevalence of heuristic interpretation methods such as SHAP and LIME, these techniques often lack formal guarantees and may produce inconsistent local explanations. To fulfill this need, few tools have emerged that use formal methods to provide formal explanations. Among these, XReason uses a SAT solver to generate formal instance-level explanation for XGBoost models. In this paper, we extend the XReason tool to support LightGBM models as well as class-level explanations. Additionally, we implement a mechanism to generate and detect adversarial examples in XReason. We evaluate the efficiency and accuracy of our approach on the CICIDS-2017 dataset, a widely used benchmark for detecting network attacks.

Keywords: Explainable AI, LightGBM, Formal Explanations, Adversarial Robustness, XReason.

1 Introduction

Artificial Intelligence (AI) has revolutionized various industries by automating complex tasks and enabling data-driven decision-making. However, as AI systems become more sophisticated, concerns about their transparency and trust have emerged, leading to the rise of Explainable Artificial Intelligence (XAI) [1]. XAI allows the decision-making processes of AI models to be comprehended by human users which is a crucial requirement for raising trust in high-stakes domains such as healthcare, finance, and network security. Moreover, AI models remain vulnerable to adversarial attacks [2], where malicious inputs can manipulate outcomes. XAI plays a pivotal role in mitigating these risks by providing insights into model behavior, aiding in the detection of vulnerabilities, and strengthening the robustness and security of AI applications.

Well-known XAI tools, such as SHAP [3], LIME [4], are model-agnostic, meaning that they can be applied to any machine learning model without requiring access to the internal structure of the model. These tools work by treating the model as a black box and focusing on explaining the relationship between

inputs and outputs. While model-agnostic methods are useful for a variety of models, they have significant limitations. In fact, these methods are often heuristic, meaning they can generate different explanations for the same prediction. Furthermore, they are not always correct, which can result in counterexamples, where the explanations do not hold.

On the other hand, formal methods in XAI have emerged as a robust alternative for generating precise and reliable explanations. Formal methods translate complex machine learning models into mathematical logic representations [5]. Subsequently, formal methods ensure the soundness and completeness of explanations by identifying the minimal set of features responsible for the model's decision. This approach strengthens the model's defenses against adversarial attacks by accurately identifying exploitable features in its decision-making process. Moreover, the generation of adversarial examples, i.e., small perturbations in input data designed to mislead a model, highlights the importance of understanding the feature-level behavior. In this regard, the identification of the features influencing decisions of the model allows the generation of more targeted and effective adversarial samples. XAI is essential in this context as it reveals the feature importance, enabling precise manipulation of only those features that drive the model's prediction. However, this strategy is only reliable when the truly relevant features are accurately identified. By rigorously analyzing the decision-making process, formal methods ensure that perturbations are applied to the correct features, improving both the generation and detection of adversarial examples.

A few formal XAI tools, such as XReason [6], Silas [7] and PyXAI [8], have been recently introduced to provide formal explanations. For instance, XReason uses XGBoost models [9] and SAT solvers [10] to provide detailed, instance-level explanations, while Silas focuses on Random Forest models [11] and employs SMT solvers [12]. PyXAI, on the other hand, focuses on models such as RandomForest Classifier, XGBoost Classifier, XGBoost Regressor, and LGBM Regressor with the use of SAT solvers. After considering these options, we selected XReason due to its open-source nature and flexibility, allowing for customization and extensibility. While XReason is effective for XGBoost models, it lacks support for other more efficient machine learning models, such as LightGBM [13], which is a gradient boosting framework that is widely recognized for its efficiency and scalability, especially when dealing with large datasets and requiring faster training times. Additionally, XReason only provides instance-level explanations, which focus on single predictions without offering a broader understanding of model behavior across classes. To overcome these limitations, we propose in this paper to extend XReason by integrating support for LightGBM and introduce class-level explanations. These explanations identify the most important features for each class, providing a more comprehensive view of the model's decision-making process. Moreover, since XReason does not address adversarial robustness, we also propose to incorporate the ability to both generate and detect adversarial examples.

The rest of the paper is organized as follows: In Section 2, we explain the methods used in our framework for extending XReason. Section 3 describes our approach for the propositional encoding of LightGBM. In Section 4, we discuss class-level explanations. In Section 5, we present our methods for generating and detecting adversarial examples. Finally, in Section 6, we present our experiments using the CICIDS-2017 dataset [14], where we evaluate the robustness and correctness of our formal explanation method in generating and detecting adversarial examples.

2 Proposed Framework

The proposed methodology builds on the existing tool XReason[6], which addresses classification problems by providing abductive formal explanations (AXps) [15] for XGBoost models using a SAT solver. XReason explains *why* an XGBoost model makes a particular prediction for a given sample by identifying a minimal subset of features responsible for the decision. Figure 1 presents our proposed extensions to XReason, called XReason+. Our framework enhances XReason's capabilities in several key areas:

- **Model Support:** We expand XReason to support LightGBM in addition to XGBoost, enabling formal reasoning and explanations across multiple machine learning models.
- **Class-Level Explanations:** We introduce class-level explanations, which create intervals for the most important features defining each class. This provides a broader view of model behavior compared to instance-based explanations, offering insights into how features contribute to predictions for each class.
- **Adversarial Sample Handling:** Using formal explanations, we implement an adversarial attack mechanism that can both generate and detect adversarial samples. Detection is based on calculating the probability of a sample being adversarial by analyzing changes in explanations in response to small input perturbations.

As depicted in Figure 1, the XReason+ process begins by training either XGBoost or LightGBM (LGBM) models on the provided training data. Once the model is trained, test data is processed to compute both instance-level and class-level formal explanations using a MaxSAT [16] solver. These explanations are then used to produce class predictions for the test data, accompanied by formal explanations. Moreover these explanations are used as input for the adversarial attack unit, which can either generate adversarial samples from the test data or detect the probability of the test data being adversarial by analyzing explanation changes in response to small perturbations.

In the next section, we describe the formal encoding of LightGBM models in XReason. This encoding captures the exact paths leading to predictions, enabling formal analysis.

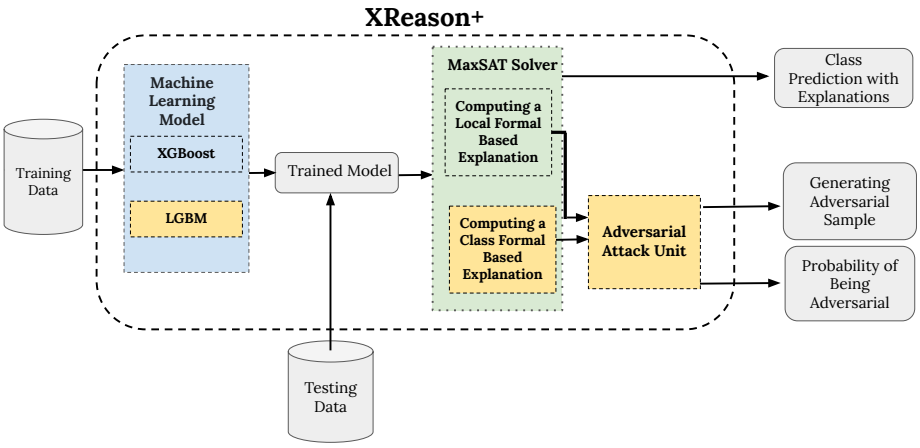

Fig. 1: XReason+ Tool.

3 Propositional Encoding of LightGBM

3.1 Overview of LightGBM

LightGBM is a widely used gradient boosting framework designed for efficiency and scalability. Its leaf-wise growth strategy differs from the level-wise approach in traditional tree-based models. LightGBM prioritizes leaves that offer the highest reduction in loss, resulting in deeper and more complex trees. While this design improves training time and accuracy, it complicates interpretability due to the unbalanced tree structures.

In this section, we propose a formal encoding of LightGBM model to produce precise explanations that capture the model's decision-making process.

3.2 Propositional Representation of LightGBM

Each decision tree in LightGBM can be represented as a series of logical constraints that encode the conditions (splits) at each node. For instance, a split condition such as $f_1 > 0.5$ is encoded as a Boolean variable, where $f_1 = 1$ if the condition holds and $f_1 = 0$ otherwise. Each path through the tree represents a conjunction of these Boolean variables, and each leaf node contains the prediction (or score) that the tree generates. Formally, the encoding of a tree is a logical formula representing the conjunction of feature conditions that lead to a particular leaf. By encoding the trees in this manner, we can capture the exact decision path taken by an instance and represent it as a formal logical expression.

3.3 Algorithm for Encoding Decision Trees

Algorithm 1 outlines the steps to encode a decision tree into a logical representation in XReason. The algorithm starts by receiving a decision tree as input, which includes its features, thresholds (the values used to split data), and branches. The first step is to collect all the thresholds used in the tree for each feature. Then, for each of these thresholds, the algorithm assigns logical variables that represent the decision points in the tree where the data is split. Once the thresholds and logical variables are assigned, the algorithm moves to the branches of the tree. For each branch, it creates a logical path, which describes how the features and thresholds lead to a specific decision.

Algorithm 1 Propositional Encoding of Decision Tree

Input: Decision tree with nodes, features f_i, thresholds t_i, and branches b_i.
Output: Encoded paths P with logical constraints.
Initialize $Thresholds \leftarrow \emptyset$, $Lvars \leftarrow \emptyset$, $Paths \leftarrow \emptyset$
Step 1: Threshold Encoding
for each feature f_i **do**
    Extract the thresholds t_i used for splitting
    Add t_i to $Thresholds(f_i)$
end for
Step 2: Logical Variable Encoding
for each feature f_i **do**
    Assign logical variables L_i for each split point
    Add L_i to $Lvars(f_i)$
end for
Step 3: Path Representation
for each branch b_i in the tree **do**
    Traverse the branch and extract the feature and logical variable constraints
    Formulate the path p_i as a set of logical conditions from $Lvars$
    Add p_i to $Paths$
end for
Step 4: Path Expansion
for each path p_i in $Paths$ **do**
    Expand p_i into a set of logical constraints: $[L_1, L_2, ..., L_n, 0]$
    Add the expanded constraints to the final encoded set
end for
Step 5: Additional Path Encoding
for each new path generated by further splits **do**
    Repeat Steps 3 and 4 for the new paths
end for
Step 6: Order and Domain Encoding
Ensure that each path respects the hierarchical order of the tree
Encode the domain of each feature and its corresponding logical variables
Finalize the encoded representation for all paths
Return: Encoded paths P

These paths are then expanded into a set of logical rules or constraints that represent how the decision-making process works in the tree. If the tree splits further and creates new paths, the algorithm applies the same process to these additional paths. In the final steps, the algorithm ensures that the order of decisions in each path matches the structure of the tree and that the feature values are encoded correctly.

4 Class-Level Explanations

The Class-Level Explanation constructs explanations for each class in the model by aggregating important features from training instances that belong to a specific class. This process helps identify the common characteristics that define each class and can be used to understand the behavior of the model at the class level.

Algorithm 2 details the procedure for building the class-level explanations in XReason. The algorithm begins by iterating over all classes in the trained model M. For each class c, it creates an empty set $\mathcal{E}_c$ to store the important features. It then analyzes each instance x_i in the training dataset D_{train}, where the model predicts the class c (i.e., $M(x_i) = c$).

Algorithm 2 Class-Level Explanation Building

Require: Trained model M, training dataset D_{train}
Ensure: Class-level explanations $\mathcal{E}_c$ for each class c
procedure BUILDCLASSLEVELEXPLANATIONS
  for each class c in model M **do**
    Initialize $\mathcal{E}_c$ as an empty set
    for each instance x_i in D_{train} where $M(x_i) = c$ **do**
      Extract important features $\mathcal{F}_i$ from formal explanation of x_i
      $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup \mathcal{F}_i$
    end for
    for each important feature f in $\mathcal{E}_c$ **do**
      Collect all values V_f^c of feature f in $\mathcal{E}_c$
      Determine interval $[a_f^c, b_f^c]$ using clustering on V_f^c
    end for
  end for
end procedure

For each instance, a formal explanation method is used to extract the important features $\mathcal{F}_i$. These latter highlight the key input features that contributed to the model's decision for that particular instance. The important features of each instance x_i are added to the class-level explanation set $\mathcal{E}_c$, which accumulates the significant features for the entire class.

Once the important features for all instances belonging to the class c have been collected, the algorithm identifies the key ranges of values for each feature. For each important feature f in $\mathcal{E}_c$, it collects all values V_f^c observed across the

instances. Using a clustering technique, the algorithm determines an interval $[a_f^c, b_f^c]$, which represents the typical range of feature values for the class c.

These intervals provide a condensed representation of the features that are most relevant for each class, offering insights into how the model differentiates between classes based on specific feature ranges. This information can be valuable for both interpreting the model and for downstream tasks, such as adversarial sample generation, where these feature intervals can be used to manipulate input data to fool the model. In next section, we propose an approach to generate and detect such adversarial attacks.

5 Adversarial Example Detection Method

Algorithm 3 outlines the process of detecting adversarial examples by comparing the explanations of an input sample with the class-level explanations derived from the training data. The detection relies on two checks: verifying the importance of the features and ensuring that their values fall within expected intervals for the predicted class.

The process begins by predicting the label of the input sample, x_{input}, using the trained model M. Once the label is predicted, the important features and their values are extracted from the formal explanation of x_{input}. For the predicted class, class-level explanations $\mathcal{E}_{y_{\text{input}}}$ are retrieved, which contain the important features and their typical intervals.

Algorithm 3 Adversarial Sample Detection Using Class-Level Explanations

Require: Trained model M, class-level explanations $\{\mathcal{E}_c\}$, input sample x_{input}
Ensure: Adversarial likelihood score s_{adv}

$y_{\text{input}} \leftarrow M(x_{\text{input}})$ ▷ Predict label of input sample
Extract important features $\mathcal{F}_{\text{input}}$ and their values V_{input} from explanation of x_{input}
Retrieve class-level explanations $\mathcal{E}_{y_{\text{input}}}$
Initialize discrepancy count $d \leftarrow 0$
for each feature f in $\mathcal{F}_{\text{input}}$ **do**
  if f is not in $\mathcal{E}_{y_{\text{input}}}$ **then**
    $d \leftarrow d + 1$ ▷ Feature not expected to be important
  else
    Retrieve interval $[a_f^{y_{\text{input}}}, b_f^{y_{\text{input}}}]$
    if $V_{\text{input}}(f) \notin [a_f^{y_{\text{input}}}, b_f^{y_{\text{input}}}]$ **then**
      $d \leftarrow d + 1$ ▷ Value outside expected interval
    end if
  end if
end for
Compute adversarial likelihood score $s_{\text{adv}} \leftarrow \dfrac{d}{|\mathcal{F}_{\text{input}}|}$
return s_{adv}

The algorithm then compares the features from the input sample with the class-level explanations. For each important feature, two checks are performed:

- **Feature Importance Check:** The algorithm verifies if the feature is considered important for the predicted class by checking its presence in the class-level explanation.
- **Feature Value Interval Check:** If the feature is important, its value is checked against the interval defined in the class-level explanation for the predicted class.

If a feature is either not present in the class-level explanation or its value falls outside the defined interval, it is considered a discrepancy. The number of such discrepancies is accumulated, and the adversarial likelihood score s_{adv} is computed as the ratio of discrepancies to the total number of important features in the input sample.

6 Case Study: CICIDS-2017 Dataset

As a case study for our proposed XReason+ tool, we utilize a customized version of the Canadian Institute for Cybersecurity's Intrusion Detection System 2017 (CICIDS-2017) dataset [14], which is widely used for network security research. The original dataset simulates both normal network traffic and various types of network attacks, making it highly representative of real-world conditions. It contains over 3 million records, 80 network features and 14 attack types, with an imbalanced class distribution. This dataset is particularly suitable for classification tasks, as it includes a labeled target variable indicating whether each instance represents normal traffic or an attack. We adopt the modified version of CICIDS-2017 proposed by Li et al. [17], which reduces the feature set to 19 essential attributes, selected for their relevance to network traffic patterns. Table 1 summarizes the features utilized from the customized CICIDS-2017 dataset for analysis.

6.1 Preprocessing and Model Performance

To ensure the quality and relevance of the data, we applied additional preprocessing steps:

- **Duplicate Removal**: We eliminate any duplicate records in the dataset.
- **Feature Selection**: Using the Autospearman method [18], which automatically removes features with high Spearman correlation to each other, we reduce the set to 18 critical features.
- **Data Splitting**: The dataset is split into 70% for training (20,655 samples) and 30% for testing (8,853 samples).

We trained a LightGBM model, following the recommendation of Li et al.[17], and evaluated the model's performance on the testing set using key metrics: Accuracy, Precision, Recall, F1 Score, and Area Under the ROC Curve (AUC) [19]. Table 2 explains the metrics and provides the respective values obtained, which range from 90% to 93%.

Table 1: Description of the Customized CICIDS-2017 Dataset [17].

Feature	**Description**
Flow Duration	Duration of the flow in microseconds.
Total Length of Fwd Packets	Total length of packets sent in the forward direction.
Fwd Packet Length Max	Maximum size of packets in the forward direction.
Fwd Packet Length Mean	Average size of packets in the forward direction.
Bwd Packet Length Max	Maximum size of packets in the backward direction.
Bwd Packet Length Min	Minimum size of packets in the backward direction.
Flow IAT Mean	Average inter-arrival time between packets within the flow.
Flow IAT Min	Minimum inter-arrival time within the flow.
Fwd IAT Min	Minimum inter-arrival time of forward packets.
Fwd Header Length	Total length of headers in the forward packets.
Bwd Header Length	Total length of headers in the backward packets.
Fwd Packets/s	Number of packets sent per second in the forward direction.
Bwd Packets/s	Number of packets sent per second in the backward direction.
Min Packet Length	Minimum size of packets within the flow.
URG Flag Count	Count of packets with the URG (urgent) flag set.
Down/Up Ratio	Ratio of bytes sent in the forward direction to bytes received.
Init_Win_bytes_forward	Initial window size in bytes for the forward packets.
Init_Win_bytes_backward	Initial window size in bytes for the backward packets.
min_seg_size_forward	Minimum segment size observed in the forward packets.
Label	Binary label indicating normal (0) or attack (1) traffic.

Table 2: Model Performance Metrics

Metric	**Definition**	**Value**
Accuracy	The overall proportion of correct predictions for both attack and non-attack samples.	0.920
Precision	Measures the proportion of correctly identified attacks out of all samples classified as attacks.	0.930
Recall	Measures the proportion of actual attacks that were correctly identified by the model.	0.924
F1 Score	The balance between Precision and Recall, showing how well the model detects attacks without missing or misclassifying them.	0.923
AUC	Indicates how well the model can distinguish between attacks and non-attacks.	0.909

6.2 Robustness and Correctness of Formal Explanations

Many previous studies have used SHAP and LIME to explain models trained on CICIDS-2017 data (e.g., [20, 21]). While these methods provide valuable insights into feature importance, they are prone to instability and lack formal guarantees. In contrast, our formal explanation method provides consistent and provably correct explanations.

Robustness of Explanations To assess the robustness of these methods, we applied SHAP, LIME, and our formal approach on the same instances twice. This comparison helps to evaluate the consistency of the feature values and rankings across both runs. We obtained following results:

- **SHAP** demonstrated **100% consistency** when run twice on the same instance. Both the feature importance scores and the resulting features rankings were identical in each run.
- **LIME** showed 0% consistency between runs. The feature rankings and importance scores changed each time, indicating that LIME's explanations are highly variable and not reliable.
- **XReason+**: Our formal explanation approach is fully deterministic, producing identical explanations and rankings in both runs. Given the same instance, the method consistently identifies the same features and assigns the same ranks, demonstrating robustness across repeated runs.

Correctness of Explanations The formal explanation method provides 100% correct explanations by identifying minimal sets of features that are guaranteed to be responsible for the prediction. We aim to evaluate how closely the rankings of the most important features for a given instance, as determined by SHAP and LIME, align with the rankings produced by the formal explanation method.

- **Ranking in Formal Explanations**: Features included in our formal explanation are assigned rank 1, while all other features are assigned the next following rank.
  Example: For this instance with values:

  ```
  [0.05230066, -0.71498734, -0.59981065, -0.0993616, -0.24724035,
  -0.32174034, 0.903657, -0.20532016, 0.18419513, 0.61847005,
  0.45396074, 0.25984816, -0.2581599, -0.17959084, 0.67872539,
  -0.11816232, 0.39183229, 1.5487759, -0.05288477]
  ```

  the features *Bwd Packet Length Max* (0.05230066) and *Fwd Packet Length Max* (0.61847005) represent the maximum size of packets in the backward and forward directions, respectively. Our formal explanation was:

  ```
  "IF Bwd Packet Length Max == 0.05230066
  AND Fwd Packet Length Max == 0.61847005 THEN 0"
  ```

In this case, we assigned rank 1 to the features included in the explanation: *Bwd Packet Length Max* and *Fwd Packet Length Max*. Since these two features are included, all other features were ranked 3.

– **Comparison with SHAP and LIME:** SHAP and LIME generate feature importance scores, and we generated the rankings based on the absolute scores, with the most important feature receiving rank 1, the next receiving rank 2, and so on. To compare these ranks with the formal method, we used Spearman's Rank Correlation [22], Kendall's Tau [23], and Rank-Biased Overlap (RBO) [24] to evaluate the results. Spearman and Kendall range from -1 to 1, where higher values indicate stronger agreement between ranks. RBO ranges from 0 to 1, with higher values representing greater overlap between ranked lists.

  - **Spearman's Rank Correlation**:
    * For SHAP, the Spearman values range from -0.2818 to 0.5892, with an average of 0.1352.
    * LIME exhibited greater variability, with Spearman values ranging from -0.6983 to 0.7201 and an average of 0.1811.

  - **Kendall's Tau**:
    * SHAP's Kendall Tau values range from -0.2361 to 0.4936, with an average of 0.1133.
    * LIME's Kendall Tau values ranged from -0.5850 to 0.6032, with an average of 0.1517.

  - **Rank-Biased Overlap (RBO)**:
    * SHAP and LIME exhibited moderate overlap in the top-ranked features compared to the formal method, with RBO values ranging from 0.1849 to 0.6684. SHAP had an average RBO of 0.3885, while LIME had an average of 0.3715.

In summary, the formal explanation method for LightGBM predictions proves to be more robust and correct compared to SHAP and LIME. Given this strong foundation of reliability and correctness, we leveraged our formal explanations to explore their utility in generating and detecting adversarial examples. By using the key features identified through our method, we can craft and detect adversarial samples that exploit the model's vulnerabilities.

6.3 Adversarial sample generation and detection

To evaluate the robustness of our approach, we applied the adversarial sample generation method across the entire test set, consisting of 8,853 samples. Out of these, 2,821 samples (31.86%) successfully fooled the model, highlighting its vulnerability to adversarial attacks. The average Euclidean distance between the original and adversarial samples was 1.67, indicating that only small perturbations were needed to manipulate the model's predictions.

Further analysis showed a significant difference in the impact of adversarial attacks based on the original class. Out of the 3,058 samples that were originally predicted as attacks (class 1), 2,044 (66.84%) were misclassified as normal traffic (class 0) after the perturbation. Moreover, 777 out of the 5,795 samples originally predicted as normal (class 0) were flipped to attacks (class 1), which represents 13.41%.

Using our formal explanation-based detection method, we were able to identify 1,731 out of the 2,821 adversarial examples (those that changed the model's prediction) as likely adversarial, achieving a detection rate of 61.36%. This demonstrates the effectiveness of the detection mechanism in identifying samples that exploit model vulnerabilities, particularly in the case of misclassified attack samples.

7 Conclusion

In this paper, we proposed a framework extending the XReason tool by adding formal explanations for LightGBM models, class-level explanations, and a method to generate and detect adversarial samples. Our approach provides deterministic and consistent explanations, addressing the limitations of heuristic methods like SHAP and LIME. We applied our XReason+ tool to the CICIDS-2017 dataset, demonstrating its effectiveness in generating adversarial samples and improving robustness. In the future, we plan to extend the developed framework to support additional machine learning models and apply it to other large-scale datasets. We also aim to enhance the adversarial detection mechanism.

References

[1] A. Darwiche and A. Hirth. On the reasons behind decisions. In *European Conference on Artificial Intelligence*, pages 712–720, 2020.

[2] H. Baniecki and P. Biecek. Adversarial attacks and defenses in explainable artificial intelligence: A survey. *Information Fusion*, page 102303, 2024.

[3] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[4] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you? explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, ACM 2016.

[5] J. Marques-Silva and A. Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 12342–12350, 2022.

[6] XReason. https://github.com/alexeyignatiev/xreason. [Online; accessed 2024].

[7] Silas. https://www.depintel.com/silas_download.html. [Online; accessed 2024].

[8] PyXAI. https://www.cril.univ-artois.fr/pyxai/. [Online; accessed 2024].

[9] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining*, pages 785–794, ACM 2016.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2021.

[11] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[12] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, and Q. Ye. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2017.

[14] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*, pages 108–116, 2018.

[15] A. Ignatiev, N. Narodytska, N. Asher, and J. Marques-Silva. From contrastive to abductive explanations and back again. In *Advances in Artificial Intelligence*, pages 335–355, LNCS vol, 2020. Springer.

[16] A. Ignatiev, A. Morgado, and J. Marques-Silva. RC2: An efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019.

[17] Y. Li and S. Abdallah. IoT data analytics in dynamic environments: From an automated machine learning perspective. *Engineering Applications of Artificial Intelligence*, 116:105366, 2022.

[18] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude. AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models. In *International Conference on Software Maintenance and Evolution*, pages 92–103, IEEE 2018.

[19] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[20] T. T. Le, H. Kim, H. Kang, and H. Kim. Classification and explanation for intrusion detection system based on ensemble trees and shap method. *Sensors*, 22(3):1154, 2022.

[21] S. Patil, V. Varadarajan, S. M. Mazhar, A. Sahibzada, N. Ahmed, O. Sinha, S. Kumar, K. Shaw, and K. Kotecha. Explainable artificial intelligence for intrusion detection system. *Electronics*, 11(19):3079, 2022.

[22] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[23] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[24] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems*, 28(4):1–38, 2010.