EFFICIENT EXPLAINABLE AI AND ADVERSARIAL ROBUSTNESS USING FORMAL METHODS

Amira Jemaa

A thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Applied Science
(Electrical and Computer Engineering) at
Concordia University
Montréal, Québec, Canada

November 2024

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Amira Jemaa**

Entitled: **Efficient Explainable AI and Adversarial Robustness using Formal Methods**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. Nizar Bouguila

_____ External Examiner
Dr. Nizar Bouguila

_____ Internal Examiner
Dr. Moataz Chouchen

_____ Supervisor
Dr. Sofiène Tahar

Approved by _____
Dr. Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

_____ 2024

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Efficient Explainable AI and Adversarial Robustness using Formal Methods

Amira Jemaa
Concordia University 2024

Artificial Intelligence (AI) systems are increasingly used in critical applications, but their lack of transparency often hinders trust and reliability. Explainable AI (XAI) addresses this by making machine learning models more understandable and interpretable. Current XAI approaches, however, lack consistency or theoretical guarantees. Formal methods, which are rigorous mathematical reasoning tools, could play a significant role in overcoming these limitations by providing sound and consistent explanations for model decisions. This thesis builds upon an existing formal XAI tool, XReason, which uses logical reasoning to generate explanations for individual predictions. The contributions of this work are threefold. First, the tool is extended to support a powerful tree based model, Light Gradient Boosting Machine (LighGBM), which offers improved scalability and performance for large datasets. Second, it introduces explanations at the class level, enabling the analysis of general patterns in model behavior across different prediction categories. This provides insights into the factors shaping model decisions for each class and helps identify biases or inconsistencies in predictions. Third, adversarial robustness is explored by integrating methods to generate and detect adversarial examples. These adversarial samples expose vulnerabilities in the model by identifying subtle input changes that lead to incorrect predictions. Detection mechanisms are then developed to identify such inputs, enhancing the model's reliability. Experiments on a variety of datasets from different domains demonstrate that the extended framework produces consistent and robust explanations, both at the individual prediction level and across broader trends. By integrating formal methods, this work provides a practical formal XAI framework applicable to areas where trust in AI systems is essential.

To my beloved parents, Ali and Souad, my ever-supportive brother, Wassim, and my incredibly caring sisters, Takwa, Imen, and Ines, whose support and care have shaped who I am today.

Acknowledgments

First and foremost, I am profoundly thankful to Allah for His guidance and blessings to overcome challenges and see this work to fruition.

I would like to express my sincere gratitude to my supervisor, Dr. Sofiène Tahar, for giving me the opportunity to pursue my Master's degree under his guidance. His constant support, feedback and encouragement throughout this journey were invaluable in shaping both this thesis and my perspective on academic research. His expertise and availability helped me overcome many challenges and made this experience enriching and rewarding. I am also grateful to Dr. Adnan Rashid for his mentorship, technical support and feedback during my research. Special thanks to Dr. Maïssa Elleuch, who helped with the choice of the research topic at the start of my thesis.

I sincerely thank Dr. Nizar Bouguila and Dr. Moataz Chouchen for agreeing to serve on my thesis examining committee and taking the time to review my work.

I am very fortunate to have been part of the Hardware Verification Group (HVG), which provided a supportive and motivating environment. I would like to thank my colleagues Nour, Oumaima, Kubra, Elif and Alain who made this journey easier and more enjoyable.

I owe so much to my family, my parents and my siblings for their love, encouragement and unwavering belief in me. Their constant support has been the backbone of all my achievements. I am also incredibly grateful to my relatives in Montreal, especially my uncle Naceur and his family, for their generosity, which made a huge difference during this time. I am also deeply thankful to my friends Asma, Azza, and Omaima, whose support reached me across borders, to my roommate Imen for her kindness and and to my friends in Montreal who made my time there more enjoyable.

This work is the result of the collective support, guidance and encouragement I have received from these amazing people. Thank you for being a part of this journey.

Table of Content

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
ALE	Accumulated Local Effects
AUC	Area Under the Curve
CICIDS	Cybersecurity's Intrusion Detection System
CNF	Conjunctive Normal Form
CW	Carlini and Wagner Attack
DNN	Deep Neural Networks
DNF	Disjunctive Normal Form
FCE	Feature Contribution Explanation
FGSM	Fast Gradient Sign Method
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GAM	Generalized Additive Models
HSJ	HopSkipJump Attack
IDC	International Data Corporation
LightGBM	Light Gradient Boosting Machine
LIME	Local Interpretable Model-Agnostic Explanations
ML	Machine Learning
MUCs	Minimal Unsatisfiable Cores
PDP	Partial Dependence Plot
PGD	Projected Gradient Descent
RBO	Rank-Biased Overlap
ROC	Receiver Operating Characteristic
SHAP	SHapley Additive exPlanations
SMT	Satisfiability Modulo Theories

SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TPR	True Positive Rate
WSL	Windows Subsystem for Linux
XAI	Explainable Artificial Intelligence
XGB	eXtreme Gradient Boosting
ZOO	Zero-Order Optimization

Chapter 1

Introduction

Artificial Intelligence (AI) has been deeply embedded in our everyday lives, driving innovation across numerous industries and reshaping business models worldwide. According to the International Data Corporation (IDC), which is a global provider of market intelligence and analysis on the technology industry, global investment in AI is projected to reach nearly $630 billion by the year 2028, demonstrating a drastic increase from $12 billion in 2017 [1]. Meanwhile, Statista, which is a platform that consolidates and provides the statistical data and insights across various sectors, including technology, finance, and consumer behavior, anticipates that the broader AI market could exceed $1.8 trillion by 2030 [2]. As a result, the AI's widespread adoption in every sector is having a profound effect on society. AI is now omnipresent with systems making decisions for us every day from recommending products on Amazon to suggesting friends on Facebook and displaying targeted advertisements on Google search results. Moreover, the application of AI extends beyond consumer services to some critical sectors like cybersecurity and finance.

In cybersecurity, AI is used to detect and prevent security breaches, identify patterns of malicious activity and respond to threats in real time, helping to safeguard sensitive data and networks. Similarly, in finance, AI plays a key role in fraud detection, algorithmic trading and credit scoring, where AI-driven systems analyze a vast amount of data to make high-speed data-driven decisions. In such high-stakes environments, where the consequences of errors can be severe, understanding the reasoning behind AI's decisions is of utmost importance, where the opacity of many advanced models, often termed as *black boxes*, restricts users from comprehending how specific

decisions are reached. This highlights a need for explainability and transparency in AI, ensuring that its decisions, particularly in critical applications are well understood and can be trusted. Explainable AI (XAI) [3] can address this issue by opening up these black boxes, making it possible for humans to follow and validate the model's behavior.

1.1 Explainable AI

XAI refers to a set of methods and techniques designed to make the outcomes of machine learning models understandable to humans. Figure 1.1 illustrates the taxonomy of XAI, categorizing models into two types, such as interpretable models and the ones requiring post-hoc analysis methods to provide explanations after the training [3]. In the following, we elaborate on each of these methods.

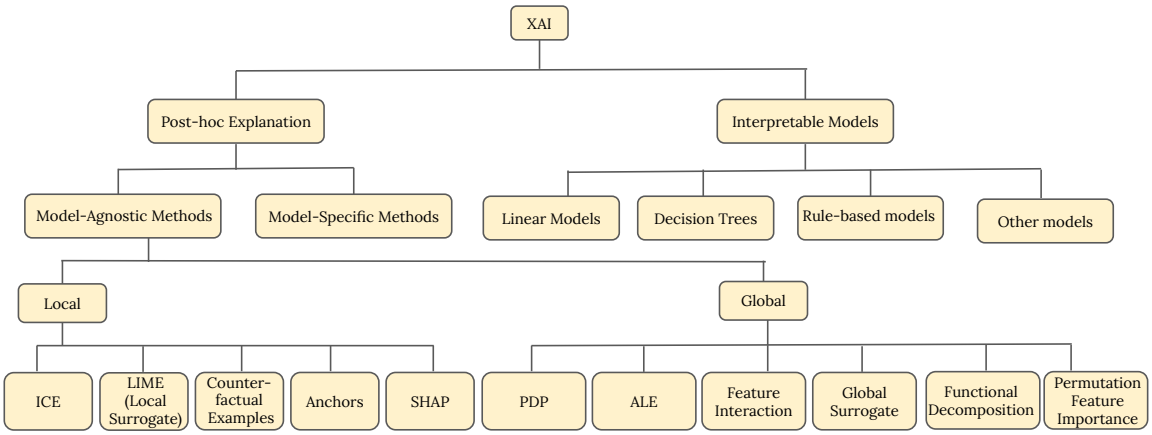

Figure 1.1: XAI Taxonomy

Interpretable Models

A machine learning model (ML) [4] is often deemed interpretable if its predictions can be easily understood [5]. Examples of interpretable models include linear/logistic regression [6, 7], decision trees [8, 9, 10], and rule-based learning models [11, 12]. In addition to these well-known interpretable models, there exist other models that exhibit varying degrees of interpretability. For instance, generalized additive models (GAMs) provide interpretability by combining linear terms with smooth functions, while prototype-based models such as k-nearest neighbors offer intuitive explanations

by comparing new instances to stored examples. These models often serve as a middle ground between inherently interpretable models and complex black-box models, depending on their design and application. While each of these models provides interpretability through its design, their real-world applicability can be limited by complexity and scalability. This includes issues like computational cost for training or inference and difficulties handling large datasets or high-dimensional features. For example, rule-based models may not perform well with complex data, and decision trees can become too large to manage effectively with large-scale datasets.

To be considered interpretable, a model should ideally meet one of the properties such as simulatability, decomposability, or algorithmic transparency. However, these properties are often narrowly defined and can vary significantly across different model types.

- **Simulatability:** It refers to the property that a person can follow the model's steps and comprehend each decision made. Although linear/logistic models, decision trees, rule-based models and Bayesian models are generally deemed interpretable, these models may become opaque if they are overly complex. For instance, a decision tree with many nodes may hinder interpretability, making it difficult to follow or understand each decision path.

- **Decomposability:** It tests interpretability through a model's components. For example, decision tree predictions are more comprehensible when each node represents a clear, known factor. However, decomposability often depends on whether individual components themselves are easily understood, which may not always be achievable in a complex real-world data.

- **Algorithmic Transparency:** It means the model's processes and decisions are clear and understandable to humans. Although transparent models like linear/logistic and Bayesian models utilize mathematical methods that theoretically improve interpretability, their transparency can diminish when these models rely extensively on abstract or high-dimensional representations.

Post-hoc Explanation

Post-hoc interpretation [13] refers to methods used to explain the predictions or behavior of a ML model after it has been trained, particularly for complex models

that are not inherently interpretable. These methods aim to provide insights into how models make decisions without modifying their internal structures. Simple models like linear/logistic regression and decision trees inherently offer good interpretability but often lack sufficient accuracy when applied to complex data. For more complex models, post-hoc interpretation techniques are frequently employed. There are two categories of post-hoc explanations, namely model-specific and model-agnostic methods.

Model-Specific Methods: Some explanation techniques are designed specifically for certain types of ML models and are referred to as model-specific methods. These methods exploit the internal structure and characteristics of the model to provide explanations tailored to its design. For example, rule-based approaches for support vector machines (SVM) [14] aim to extract decision rules, while layer-wise relevance propagation explains predictions in deep neural networks (DNN) [15] by tracing the contribution of individual inputs. Although these techniques can provide detailed insights into specific models, their applicability is often constrained to the model they are designed for and may not generalize across other types of models.

Model-Agnostic Methods: Model-agnostic methods are explanation techniques that can be applied to any ML model, as they do not rely on the internal structure of the model. Instead, they focus on understanding the relationship between the inputs and outputs. These methods are versatile and can be used with both simple and complex models, making them particularly useful in scenarios where model transparency is needed without modifying the model itself. These methods can further be classified based on the scope of their explanations into two categories: local and global approaches.

- **Local Model-Agnostic Methods**: These methods explain individual predictions. Examples include:
    - **LIME** (Local Interpretable Model-Agnostic Explanations) [16]: LIME interprets specific predictions by approximating a complex model with a simpler, interpretable model around the instance.
    - **SHAP** (SHapley Additive exPlanations) [17]: SHAP is a game-theoretic approach using Shapley values to allocate feature importance.

- **ICE** (Individual Conditional Expectation) [18]: ICE plots are visualization tools that illustrate how a feature affects the model's predictions for individual instances, highlighting unique response patterns.
- **Counterfactual Examples** [19]: Counterfactuals identify minimal changes needed to change a prediction.
- **Anchors** [20]: Anchors identify stable regions in the input space where predictions hold, providing interpretable if-then rules.

- **Global Model-Agnostic Methods**: These methods provide an overall understanding of the model's behavior across the dataset. Examples include:
  - **PDP** (Partial Dependence Plot) [21]: PDPs show the average effect of a feature on the model prediction, assuming independence from other features.
  - **Feature Interaction** [22]: It highlights interactions between features, giving insights into feature dependencies.
  - **ALE** (Accumulated Local Effects) [23]: ALE plots account for feature interactions and provide a more accurate alternative to PDPs in non-linear models.
  - **Global Surrogates** [24]: They approximate the black-box model with an interpretable one, allowing for an overall view of decision patterns.
  - **Functional Decomposition** [25]: It breaks down model predictions into interpretable components based on feature functions.
  - **Permutation Feature Importance** [26]: It assesses feature importance by permuting feature values and measuring the impact on model performance.

Despite the existence of a plethora of XAI methods that greatly help understanding and interpret AI outcomes, they are inherently heuristic, relying on simplifications that may not fully capture model behavior. These methods can introduce biases or inaccuracies, particularly in critical applications, and should be applied with caution. One emerging technology to address these issues is the use of formal methods [27], which are rigorous techniques and tools used to model complex systems as mathematical entities.

1.2 Formal XAI

Concise explanations are widely recognized as crucial for human decision-makers to understand and debug the systems. Decision trees have traditionally exemplified interpretable ML models due to their path-based explanations, which offer transparent reasoning for each prediction. This apparent clarity has driven significant interest in optimizing decision trees, despite the NP-hard challenge of finding the minimal tree structures [28]. However, recent research has revealed some fundamental issues with the assumption that decision trees are inherently interpretable. Studies, such as [29, 30, 31, 32], show that decision tree paths frequently incorporate non-essential features, even in supposedly optimal or sparse trees. This revelation challenges the traditional view of decision trees as naturally interpretable models and highlights a broader issue in ML: the need for genuinely concise and meaningful explanations.

As presented in the previous section, the current landscape of XAI is dominated by traditional heuristic methods. While these methods offer flexibility across different model architectures, they suffer from the following critical limitations.

- **Reliability Concerns**: XAI methods frequently produce inconsistent explanations for the same predictions. These methods, relying on local approximations, can generate significantly different explanations under minimal input changes, undermining their trustworthiness in critical applications.

- **Complexity and Redundancy**: Heuristic methods often generate overly detailed or redundant explanations, making it difficult for users to identify the truly relevant features. This excess information can lead to misinterpretation or decreased trust in the AI system's decisions.

- **Lack of Mathematical Foundations**: Without rigorous mathematical underpinnings, these approaches cannot guarantee the completeness or accuracy of their explanations. This absence of formal guarantees makes it impossible to ensure that explanations align with the model's actual decision-making process.

Formal XAI approaches [29] have emerged as a powerful solution to address the limitations of the traditional explainability methods. Formal XAI provides structured frameworks for addressing both *why* and *why not* questions about the model decisions, fostering accountability and enabling a thorough analysis of the model behavior.

Formal XAI refers to methods that provide precise, logically grounded explanations for model predictions.

They are mainly based on powerful mathematical logic and solvers, such as satisfiability (SAT)[33] and Satisfiability Modulo Theory (SMT) [34] solvers. Satisfiability is the problem of determining whether there exists an interpretation that satisfies the formula. It establishes whether the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to true. SAT solvers are used for Boolean formulas while SMT solvers can handle formulas that involve multiple theories, such as natural and real numbers.

Generally speaking, formal XAI offers the following key advantages in contrast to the traditional XAI approaches.

- **Mathematical Rigor**: Formal XAI methods are grounded in logic and optimization, producing explanations that accurately reflect the model's reasoning process. This mathematical foundation ensures consistency and reliability across similar instances.

- **Minimal Explanations**: A key strength of formal XAI is its ability to generate explanations using only the essential features needed for a decision, leaving out unnecessary details. This minimalism eliminates extraneous information, improving clarity for decision-makers and reducing the risk of misinterpretation.

- **Logical Consistency**: Techniques, such as abductive explanations [35] and minimal hitting sets [29] guarantee logical consistency across explanations, allowing stakeholders to verify reasoning patterns without any concern for arbitrary variations.

Formal XAI represents a significant advancement in making AI systems more trustworthy and deployable in high-stakes environments. By providing mathematically sound, minimal and consistent explanations, it addresses the key limitations of both traditional interpretable models and non-formal XAI methods. While the fundamental tension between accuracy and interpretability persists, formal XAI offers a promising framework for developing more transparent and reliable AI systems. This approach not only meets regulatory requirements but also provides users with a confidence needed to deploy AI systems in the critical applications. The continued development of formal XAI methods will be crucial in bridging the gap between the model

complexity and human understanding, ensuring that AI systems can be effectively and safely deployed across diverse high-stakes domains.

1.3 Adversarial Examples

As ML models become more integral to critical applications, their vulnerability to adversarial attacks [36] raises substantial concerns. Some approaches focus on manipulating input data through adversarial examples, which involve subtly altering the input to deceive models when making predictions. In contrast, other methods target the model itself, such as model poisoning [37], where malicious data is introduced during the training phase to disrupt the model's learning process. This approach fundamentally differs from input-level attacks. The concept of adversarial examples was introduced in the year 2013 which describes these carefully crafted inputs designed to exploit weaknesses in the behavior of the model [38]. Initially demonstrated in image data, adversarial examples showed that even small, targeted changes could cause models to make highly confident misclassifications.

Two main types of adversarial methods have emerged, namely white-box [39] and black-box attacks [40]. White-box attacks use the model's parameters and gradients, allowing the highly targeted manipulations. For example, the Fast Gradient Sign Method (FGSM) [41] generates adversarial examples by perturbing the input data in the direction of the model's gradient, specifically targeting the regions where the model is most sensitive. Similarly, Projected Gradient Descent (PGD) [42] enhances this approach with iterative adjustments to optimize the maximum misclassification probability. These white-box techniques have become standard for studying and testing model robustness, revealing where models are most vulnerable when the internal access is available.

In contrast, black-box attacks operate without any internal model information, relying solely on the input-output observations. A notable black-box method, Zeroth Order Optimization (ZOO) [43] approximates gradients by querying the model repeatedly with small input changes. This approach enables adversarial example generation for proprietary or inaccessible models by using only the input-output feedback. Black-box attacks have demonstrated that adversarial vulnerabilities can be exploited even in closed systems, proving that the detailed model knowledge is not always necessary.

As adversarial analysis extended beyond images, new methods, such as [44, 45, 46], emerged to handle text, audio and the tabular data, each requiring specific adaptations. In text, adversarial attacks typically involve small changes to words or tokens, while in audio, subtle waveform distortions can mislead the speech recognition systems, although the perturbations remain unnoticed by human listeners. Tabular data, often used in applications like fraud detection and healthcare, pose unique challenges when researchers attempted to apply image-based adversarial analysis methods. Tabular data is composed of structured fields, such as categorical, Boolean, and continuous variables that are often domain-specific and require adherence to realistic constraints. For instance, certain fields in tabular data, like credit history or age, may be fixed by the application and not modifiable. To create realistic adversarial examples, researchers implemented constraints on which fields can be changed, ensuring that only editable, plausible fields are modified.

As adversarial examples became more diverse and their implications more complex, the need for methods to understand and mitigate these vulnerabilities grew. This led to the parallel development of XAI techniques and adversarial defenses, with XAI playing a crucial role in addressing adversarial challenges. Methods, such as LIME and SHAP explaining which features influence model decisions, have since been instrumental in the adversarial research. For example, in the structured data, XAI-driven insights guide perturbations towards the significant yet subtle features, making adversarial examples harder to detect while maintaining their effectiveness [47]. To further refine this process, custom norms have been introduced to prioritize perturbations on features that are less likely to be reviewed by human observers, enhancing the stealth of attacks in the real-world scenarios.

The integration of XAI into the adversarial workflows highlights the need for robust, transparent and reliable explainability methods that support both the generation and defense against adversarial attacks. By clarifying the model behavior and revealing decision boundaries, XAI helps researchers and practitioners build more resilient ML systems, prepared to withstand increasingly sophisticated adversarial threats.

1.4 Formal XAI Tools

In the recent past, only a few formal XAI tools have been introduced to provide concise formal explanations. In the sequel, we will elaborate on three prominent formal XAI tools, namely, XReason [48], Silas [49], PyXAI [50].

1.4.1 XReason

XReason [48] is an advanced analytical tool developed in 2019 to provide comprehensive abductive explanations of eXtreme Gradient Boosting model (XGBoost) classifer [51] model decisions. The tool initially employed SMT-based solvers to generate precise, minimal explanations that ensure logical consistency in the model interpretation. The tool's capabilities were substantially enhanced through the introduction of MaxSAT-based encoding for extracting explanations [52, 53, 54]. MaxSAT solvers extend classical SAT solvers by finding solutions that satisfy the maximum number of clauses [52]. These solvers are used to compute explanations by identifying minimal sets of feature values necessary for a model's prediction. This improvement proved particularly effective in analyzing large-scale tree ensembles, offering increased computational efficiency while maintaining the ability to generate minimal explanations. XReason is also compatible with established explanation tools, such as Anchor, LIME and SHAP. This integration allows researchers to validate and enhance the quality of the heuristic explanations. The framework demonstrates considerable versatility, effectively processing both the continuous and categorical data types. To ensure a scientific reliability, XReason includes detailed experimental configurations in its repository [48]. These setups enable researchers to reproduce essential performance measurements, including explanation size, processing speed, and memory requirements. Figure 1.2 depicts the flow of XReason. XReason supports both binary and multiclass classification tasks. Through these features, XReason has established itself as a valuable resource in the research area of formal model interpretation.

XReason has made notable contributions in the domain of formal XAI, but also has key limitations:

- XReason supports only XGBoost models, which limits its use in tasks requiring efficient handling of large datasets.

- The XReason tool provides only local explanations, lacking support for global interpretability.
- The current XReason framework does not handle adversarial example generation or detection.

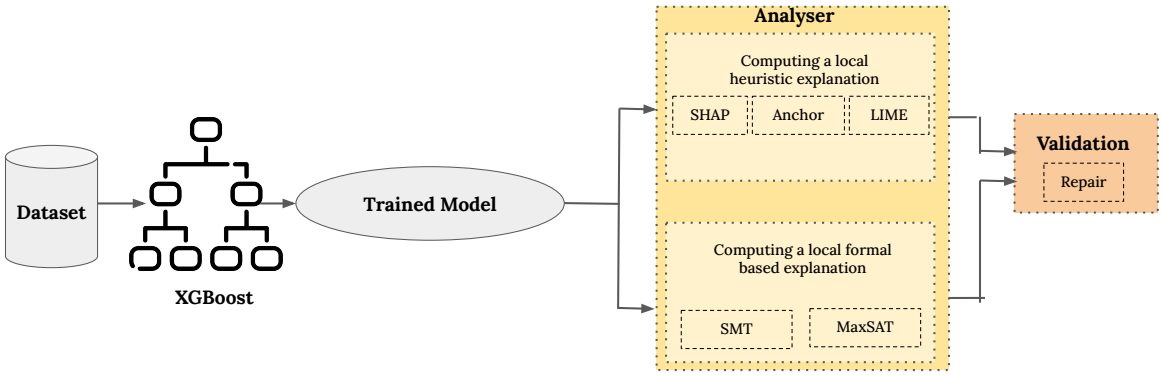

Figure 1.2: Overview of the XReason Tool

1.4.2 Silas

Introduced in 2021, Silas [49] is a data mining and predictive analytics software built using a customized Random Forest model [55], capable of handling the structured data. It is designed for binary classifiers built from tabular data and is available in both a free version and a commercial version. As shown in Figure 1.3, Silas has two unique features [56]:

- **Model Insight:** This tool adopts an SMT (Satisfiability Modulo Theories) [34] automated reasoner, which extends traditional satisfiability (SAT) solvers [33] by incorporating theories such as arithmetic, bit-vectors, and arrays. SMT reasoners enable efficient handling of logical formulas with constraints from these theories. In this context, the SMT reasoner is used to identify the key features involved in the model's decision-making by simplifying the forest and deriving only a small set of decision rules. One rule is extracted for each class.
- **Model Audit:** This feature uses SMT to mathematically prove that the prediction model conforms to the user's specifications. Users can also re-train models with predefined constraints to ensure they are correct.

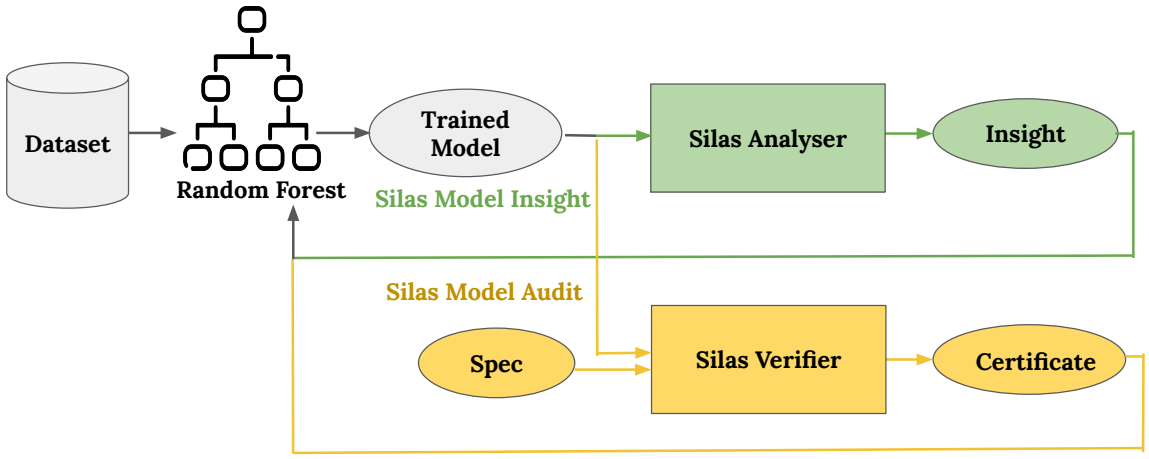

Figure 1.3: Overview of the Silas Tool

Recently, the developers of Silas extended the tool by introducing adversarial samples using insights from Minimal Unsatisfiable Cores (MUCs) [57], which are the smallest sets of features that, when altered, change the model's decision [58]. This approach turns the model's decision-making process into logical statements, helping to identify features that, if modified, can alter the model's prediction. The process involves defining a search area around the original input, where each feature is assigned a threshold specifying the extent of the allowable change.
While Silas has notable strengths, it also comes with some limitations:

- Silas only works only with a single model type (Random Forest), which limits its flexibility for other machine learning architectures.
- It supports only binary classifiers, making it unsuitable for multi-class tasks.
- Silas is computationally intensive for large datasets or high-dimensional models due to the use of SMT solvers.
- It is not open-source, which restricts customization and broader use by researchers.

1.4.3 PyXAI

PyXAI (Python eXplainable AI) [50] is a Python library recently introduced in 2024 that provides formal explanations using SMT for tree-based ML models [59]

including DecisionTreeClassifier (DT) [60], RandomForestClassifier (RF) [55], and gradient boosting models (BT) such as XGBClassifier [51], XGBRegressor [51] and LGBMRegressor [61]. The explanations generated by PyXAI are local and post-hoc. As shown in Figure 1.4 [50], the types of explanations PyXAI supports include: (1) Abductive explanations: These provide insights into why a particular instance was classified in a specific way by the model, addressing the *Why?* question. In regression, they explain why the model's output for the given instance lies within a specified range; (2) Contrastive explanations: These answer the *Why not?* question by clarifying why the instance was not classified in an expected way according to the user's perspective. PyXAI also includes functionality for correcting tree-based models when their predictions conflict with the specific user knowledge.

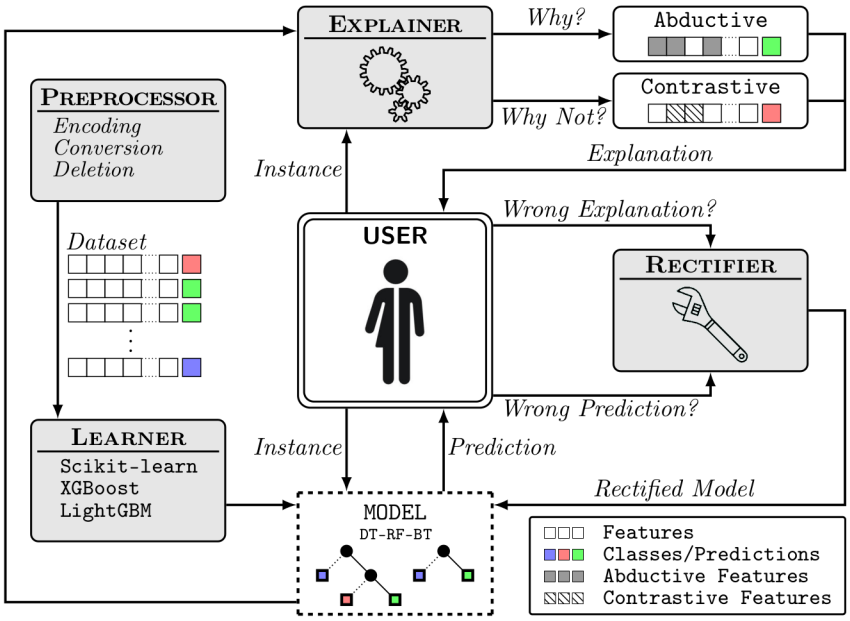

Figure 1.4: Overview of the PyXAI Tool

PyXAI is designed for general applications involving classification and regression tasks [59]. Although PyXAI offers valuable functionality, it has some limitations:

- PyXAI generates only local, post-hoc explanations, which may not be sufficient for users seeking global insights.
- It relies on SMT solvers, which can lead to scalability issues when dealing with large models or datasets.

- PyXAI does not handle adversarial example generation or detection.
- It is relatively new, with lack of extensive documentation or community support.

1.5 Problem Statement

Artificial Intelligence (AI) has become integral to various industries, including critical domains like cybersecurity, transportation and medicine. However, the reliance on complex "black-box" models raises significant concerns about trust. To address this, Explainable AI (XAI) methods provide transparency, making AI decisions more interpretable. Despite significant progresses, existing XAI approaches are heuristic in nature, often lacking consistency and guarantees, limiting their reliability in critical scenarios.

Formal XAI methods overcome these issues by offering mathematically rigorous, minimal, and logically consistent explanations. Among the few existing tools, XReason stands out by addressing classification problems. XReason provides abductive formal explanations [62] for XGBoost models using a SAT solver. It explains *why* an XGBoost model makes a particular prediction for a given sample by identifying a minimal subset of features responsible for the decision.

This thesis addresses these gaps by extending XReason with support for the scalable and high-performance Light Gradient Boosting Machine model (LightGBM) [61], class-level explanations, adversarial examples generation and adversarial examples detection. These enhancements create a comprehensive formal XAI framework that ensures transparency, reliability, and robustness in diverse applications. We call the extended version of XReason: XReason+.

1.6 Proposed Methodology

In this Master's thesis, we propose to enhance XReason's capabilities in several key areas:

- **Model Support:** We expand XReason to support LightGBM [61] in addition to XGBoost, enabling formal reasoning and explanations across multiple ML models.

- **Class-wise Explanations:** We introduce class-wise 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.

Figure 1.5 presents our proposed methodology in extending the XReason tool. The XReason+ process begins by training either XGBoost or LightGBM models on the provided training data. Once the model is trained, the test data is processed to compute both instance-level and class-wise formal explanations using the MaxSAT 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.

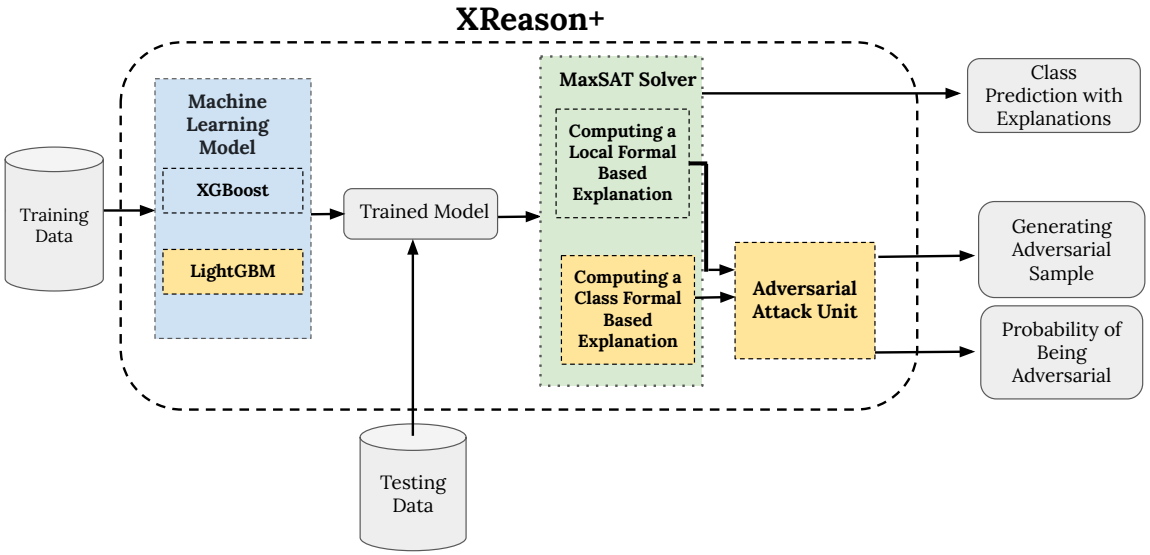

Figure 1.5: Proposed Methodology

The XReason+ tool is available as an open-source project on GitHub [63]. To

validate the effectiveness of the proposed methodologies, a series of experiments were conducted using various datasets, including the CICIDS-2017 dataset [64] for network security applications, the Breast Cancer Wisconsin dataset [65] for healthcare classification tasks, the Banknote Authentication dataset [66] for evaluating adversarial detection methods, as well as the Image Segmentation [67], Ecoli [68], and Iris [69] datasets to assess class-wise explanations and their effectiveness in capturing class-specific feature patterns. These datasets provide a diverse range of applications, ensuring a comprehensive evaluation of the proposed methodologies in different contexts.

All experiments were conducted on a machine running Windows Subsystem for Linux (WSL) with Python 3.10 [70], utilizing Windows 10 with WSL (Ubuntu 20.04), an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz (1.80GHz, 4 cores, 8 logical processors), and 8 GB of physical RAM.

1.7 Thesis Contributions

The contributions of this thesis can be summarized as follows where the publications for these contributions can be found in the Biography section at the end of the thesis document.

- **Encoding LightGBM Models:** Implemented an encoding for LightGBM models, enabling efficient reasoning and explanation generation. The encoding approach allows the formalization of LightGBM models into a logical format suitable for MaxSAT solvers. Experimental results, validated on the Breast Cancer dataset, demonstrated 100% correctness of the encoding by matching predictions from the encoded and original models [Bio-Cf1,Bio-Cf2].

- **Class-wise Explanations:** Introduced class-level formal explanations in XReason+, enabling a comprehensive understanding of model behavior for specific prediction classes. Experiments on datasets such as Image Segmentation, Ecoli, and Iris demonstrated the effectiveness of these explanations in capturing class-specific feature patterns and outperformed heuristic methods in explanation consistency [Bio-Cf1, Bio-Cf2].

- **Generation and Detection of Adversarial Examples:** Developed methods for generating adversarial samples and detecting them using formal explanations. Experiments on tabular datasets showed that the adversarial generation approach effectively exposed model vulnerabilities, while the detection mechanism identified adversarial samples with high accuracy, enhancing model security [Bio-Cf1].

- **Tool Development:** Implemented the new XReason+ tool in Python, extending the original XReason tool with new functionalities, including support for LightGBM models, class-wise explanations, and adversarial example handling. The tool is available as an open-source project, available online on github [Bio-T1].

1.8 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 describes the propositional encoding of LightGBM models, starting with gradient boosting concepts, logical foundations, and a formal algorithm for encoding decision trees. Correctness and performance of the encoded model are validated experimentally.

Chapter 3 focuses on class-wise explanations, presenting methods for generating both local and class-wise explanations using MaxSAT solvers, with experimental evaluations demonstrating the effectiveness of these explanations in capturing class-specific feature patterns and comparing them to heuristic approaches like SHAP.

Chapter 4 discusses adversarial robustness, detailing techniques for generating and detecting adversarial samples using formal explanations, with experimental evaluations that assess the robustness of the proposed methods and their effectiveness in identifying adversarial vulnerabilities.

Chapter 5 presents a case study on network security using the CICIDS-2017 dataset, evaluating the robustness and correctness of formal explanations and comparing adversarial generation and detection results. Chapter 6 concludes the thesis with a summary of contributions while proposing future research directions.

Chapter 2

LightGBM Model Encoding

This chapter covers the encoding of LightGBM models into formal logical representations. It includes an overview of Gradient Boosting Tree Models, the logical foundations for encoding, a detailed algorithm for the process and experimental results validating the correctness and performance of the encoded models on the Breast Cancer dataset [65].

2.1 Gradient Boosting Tree Models

Gradient boosting [71] is a powerful ensemble learning technique that builds a model in a stage-wise fashion by sequentially adding decision trees to minimize a specified loss function. LightGBM [61] is a widely used gradient boosting framework designed for efficiency and scalability. Unlike traditional tree-based models that use a level-wise growth strategy, LightGBM employs a leaf-wise growth strategy. This approach prioritizes growing the leaf with the highest potential to reduce loss, resulting in deeper and more complex trees. In [72], the authors have shwon that LightGBM is much faster in training compared to XGBoost. The study also highlights that LightGBM handles large datasets effectively and is well-suited for tasks requiring fast computational performance. While this design enhances training speed and predictive accuracy, it introduces challenges in model interpretability due to the unbalanced and intricate tree structures.

2.1.1 Function Representation

To formally represent the structure of LightGBM models, we begin by defining functions that map feature inputs to outputs. In LightGBM, the ensemble model consists of multiple decision trees, each acting as a function that contributes to the final prediction. Each tree can be viewed as a function $f : D \to C$, where D is the domain of feature values and C is the codomain of predictions or classes. As shown in Figure 2.1, each tree function $f(d) = c$, with $d \in D$ and $c \in C$, encapsulates the mapping from feature inputs to outputs based on the decision rules specified at its nodes [29].

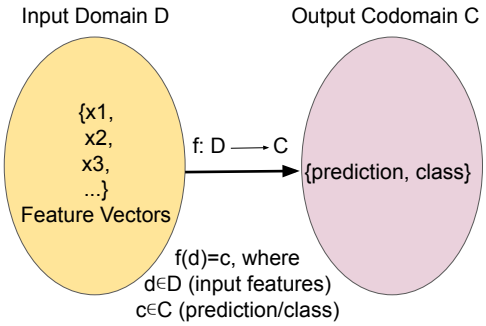

Figure 2.1: Function Representation

Parameterization of Functions

Each decision tree in LightGBM is defined by a set of parameters representing the thresholds or conditions for feature splits. By parameterizing these functions, we can represent different paths within a tree, where each path corresponds to a unique combination of conditions. The LightGBM model can thus be represented as a family of parameterized functions, with each function's parameters specifying the thresholds and feature values at each node [29].

In this context, a parameterized function can be represented as:

$$f(d; \pi_1, \pi_2, \ldots, \pi_n)$$

where $\pi_1, \pi_2, \ldots, \pi_n$ denote the split thresholds or conditions specific to each tree. This parameterization enables LightGBM to represent complex mappings by adjusting conditions across multiple trees in the ensemble.

Piecewise Function Representation

Building on the parameterized functions, each decision tree in LightGBM can be considered as a *piecewise function* over the feature space. Each path of the tree corresponds to a unique region defined by the feature splits along that path. Within each path (region), the function is constant and leads to a specific prediction. As an ensemble of such piecewise functions, LightGBM captures complex decision boundaries by aggregating outputs across multiple trees. This piecewise approach allows LightGBM to model non-linear relationships between features and predictions by iteratively refining each path based on the prediction error. Thus, the ensemble effectively produces a refined, accurate mapping from D to C.

2.1.2 Logical Foundations and Notations

To systematically encode the structure of LightGBM's decision trees, we establish logical foundations and notations that frame each decision tree as a set of logical expressions. By representing decision paths using logical formulas, we create a consistent framework that supports interpretability tasks.

Encoding Split Conditions as Propositional Variables

In the context of LightGBM, each split condition is an inequality involving a feature, such as $x_i > t$ or $x_i \leq t$, where x_i is a feature and t is a threshold. To represent these conditions in propositional logic, we introduce propositional variables corresponding to these conditions. For example, let:

$$p_{i,t} = \begin{cases} \text{True}, & \text{if } x_i > t \\ \text{False}, & \text{otherwise} \end{cases}$$

Each propositional variable $p_{i,t}$ captures the truth value of the split condition. The logical expressions along a path are then constructed using these variables [29].

Literals and Propositional Formulas

A *literal* is a propositional variable or its negation. In our encoding, literals represent the truth or falsity of split conditions. For example, $p_{i,t}$ is a positive literal

representing $x_i > t$, while $\neg p_{i,t}$ represents its negation, i.e., $x_i \leq t$. By combining literals using logical connectives, we form *propositional formulas* that express the conditions along a path within a decision tree.

Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF)

Propositional formulas can be represented in either *Conjunctive Normal Form (CNF)* or *Disjunctive Normal Form (DNF)*:

- **Conjunctive Normal Form (CNF):** A formula is in CNF if it is a conjunction of one or more clauses, where each clause is a disjunction of literals. For example:

$$(p_{1,5} \vee \neg p_{2,3}) \wedge (\neg p_{3,1} \vee p_{4,2})$$

  CNF is useful for representing global constraints in a model.

- **Disjunctive Normal Form (DNF):** A formula is in DNF if it is a disjunction of one or more terms, where each term is a conjunction of literals. For instance:

$$(p_{1,5} \wedge \neg p_{2,3}) \vee (p_{3,1} \wedge p_{4,2})$$

  DNF is particularly useful for representing the entire set of paths leading to a prediction, as each term corresponds to a unique path.

Path Representation using Propositional Logic

Each path from the root to a leaf in an LightGBM tree can be represented as a conjunction of literals, forming a logical expression that captures the sequence of conditions defining the path as shown in Figure 2.2. For instance, a path with conditions $x_1 > 5$, $x_2 \leq 3$ and $x_3 = 1$ can be encoded as:

$$p_{1,5} \wedge \neg p_{2,3} \wedge p_{3,1}$$

Alternatively, these conditions can also be expressed explicitly as:

$$(x_1 > 5) \wedge (x_2 \leq 3) \wedge (x_3 = 1)$$

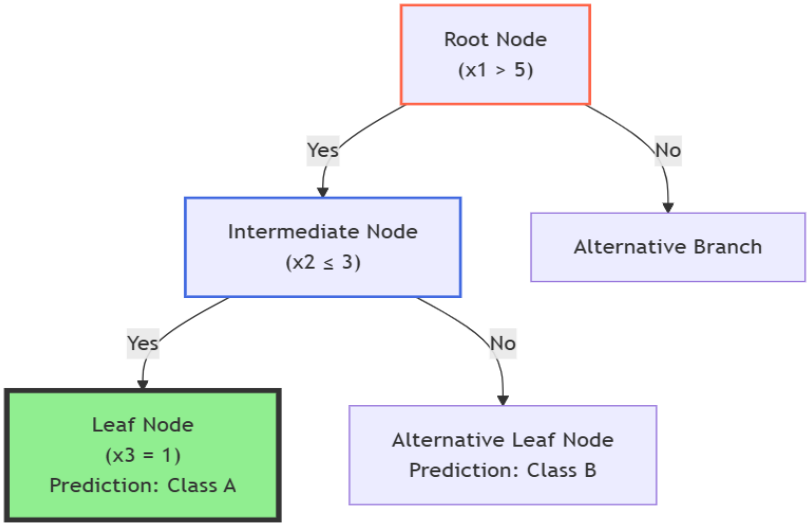

Figure 2.2: Decision Path

Here, each condition corresponds to a decision rule and their conjunction defines a specific region in the feature space that leads to a particular prediction. By representing all paths leading to a particular prediction and combining them using disjunctions, we can express the tree's overall decision logic in Disjunctive Normal Form (DNF). This structured encoding approach provides a basis for analyzing and interpreting individual paths and their contribution to the model's predictions.

By establishing these logical foundations and notations, we provide the framework needed to represent LightGBM's decision trees in a formalized logical structure that supports interpretability tasks.

2.1.3 Classification Problem Setup

To encode the LightGBM model effectively, we formalize the elements of the classification problem, including features, domains, classes and the classification function. This structured representation provides a consistent framework for the encoding process.

Problem Definition

A classification problem is defined as a 4-tuple:

$$M = (F, D, K, \kappa)$$

where

- $F = \{f_1, f_2, \ldots, f_n\}$ is the set of features.
- $D = D_1 \times D_2 \times \ldots \times D_n$ is the domain, where D_i is the domain of feature f_i.
- K is the set of possible classes or outcomes.
- $\kappa : D \to K$ is the classification function that maps feature vectors to class labels.

Feature Domains

Each feature f_i has an associated domain $D_i \subseteq \mathbb{R}$ (for numerical features) or a finite set of categories (for categorical features). The domain defines the range of possible values that the feature can take. When encoding the model, we consider these domains to ensure that the logical expressions accurately represent the feasible conditions.

Output Classes

The set K represents all possible class labels for the classification task. In binary classification, $K = \{0, 1\}$, while in multiclass classification, K includes multiple class labels. Each leaf node in an LightGBM tree is associated with a prediction that corresponds to a class label or a probability score.

Classification Function

The classification function κ maps feature vectors to class labels based on the aggregated decisions of all trees in the LightGBM model. For an instance $d \in D$, the prediction $\kappa(d)$ is determined by summing the contributions from each tree and applying a decision rule, such as selecting the class with the highest score.

Instance Representation

An instance $d = (d_1, d_2, \ldots, d_n)$ is a vector of feature values, where $d_i \in D_i$. In the encoding process, we represent the instance's traversal through the decision trees using logical expressions based on the truth values of the split conditions at each node.

2.1.4 Decision Tree Structure for Encoding

To encode LightGBM's decision trees formally, we define the tree structure in terms of nodes and edges that represent feature-based splits leading to predictions at the leaf nodes. Each path from the root to a leaf encapsulates a series of conditions that must be satisfied for an instance to reach a particular prediction.

Tree Representation

A decision tree T is represented as a pair $T = (V, E)$, where:

- $V = V_{\text{internal}} \cup V_{\text{leaf}}$ is the set of nodes, including internal (split) nodes and leaf nodes.
- E is the set of directed edges connecting nodes, representing the flow of decisions based on split conditions.

Each internal node $v \in V_{\text{internal}}$ is associated with a split condition on a feature and each leaf node $v \in V_{\text{leaf}}$ contains a prediction.

Split Conditions and Nodes

At each internal node, a split condition $x_i > t$ divides the data into two branches:

- Left child node: corresponds to $\neg p_{i,t}$ (i.e., $x_i \leq t$).
- Right child node: corresponds to $p_{i,t}$ (i.e., $x_i > t$).

These conditions are used to construct logical expressions representing the paths an instance can take through the tree.

Paths and Leaf Nodes

A path from the root to a leaf node is a sequence of split conditions represented by a conjunction of literals. Each path defines a unique combination of feature conditions leading to a specific prediction at the leaf node. By enumerating all such paths, we can represent the entire decision logic of the tree.

Encoding the Tree Structure

To encode the tree structure, we represent each path π_j leading to a leaf node v_{leaf} as a logical formula:

$$\pi_j = l_1 \wedge l_2 \wedge \cdots \wedge l_m$$

where l_k are literals corresponding to the split conditions along the path. The entire tree can then be expressed as a disjunction of these path formulas:

$$T = \pi_1 \vee \pi_2 \vee \cdots \vee \pi_p$$

This DNF formula represents all possible paths that result in a prediction within the tree.

2.2 Algorithm for LightGBM Encoding

2.2.1 Overview of LightGBM

LightGBM follows the gradient boosting framework, where an ensemble of decision trees is built sequentially to minimize a loss function. Each tree corrects the residual errors of its predecessors by optimizing the gradient of the loss function. The steps of the LightGBM algorithm are outlined below [61]:

(1) **Input Data and Preprocessing**:

- **Input:** A dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i is a feature vector and y_i is the corresponding label.
- **Preprocessing:** Features are binned into discrete values using a histogram-based technique, reducing memory usage and computation.

(2) **Initialization**:

- Initialize the model $F_0(x)$ with a constant value, typically the mean of the target values for regression or the log-odds for classification.
- Define the loss function $L(y, F(x))$ based on the task (e.g., mean squared error for regression, cross-entropy for classification).

(3) **Iterative Tree Construction**: For $t = 1$ to T (number of boosting iterations):

(a) **Compute Gradients and and Second-Order Derivatives**:

$$g_i = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}, \quad h_i = \frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2}.$$

(b) **Feature Histogram Construction**: For each feature, calculate histograms of gradients and Hessians using the binned feature values. This step aggregates statistics to determine the best split points.

(c) **Split Point Selection**: For each feature, identify the split point that maximizes the gain in the objective function:

$$\text{Gain} = \frac{\left(\sum_{i \in \text{left}} g_i\right)^2}{\sum_{i \in \text{left}} h_i + \lambda} + \frac{\left(\sum_{i \in \text{right}} g_i\right)^2}{\sum_{i \in \text{right}} h_i + \lambda} - \frac{\left(\sum_{i \in \text{all}} g_i\right)^2}{\sum_{i \in \text{all}} h_i + \lambda},$$

where λ is a regularization parameter.

(d) **Leaf-wise Tree Growth**: Grow the tree by selecting the leaf with the highest potential gain (leaf-wise growth strategy). This strategy produces deeper trees compared to level-wise growth, enhancing accuracy for complex patterns.

(e) **Update Tree Structure**: Add the split to the tree and assign predictions to the new leaves based on the aggregated gradients and Hessians.

(4) **Model Update**: Update the model by adding the predictions of the newly built tree, weighted by the learning rate η:

$$F_t(x) = F_{t-1}(x) + \eta \cdot f_t(x),$$

where $f_t(x)$ is the function represented by the t-th tree.

(5) **Termination**: Stop if the number of iterations T is reached, or if the improvement in the loss function falls below a predefined threshold.

2.2.2 Key Features of LightGBM

- **Histogram-based Splitting**: Features are discretized into bins, reducing the computational cost of finding split points.
- **Leaf-wise Growth**: By focusing on the most promising leaf, LightGBM achieves deeper trees and better accuracy compared to traditional level-wise growth.
- **Support for Categorical Features**: Categorical features are directly handled by optimizing the split points for their unique values.
- **Regularization Techniques**: Includes Lasso (L1) and Ridge (L2) regularization [73] and feature importance pruning to avoid overfitting.

2.2.3 Formal Encoding Algorithm

Building on the principles outlined in Section 2.1, we now translate the decision-making process of LightGBM into a formal representation suitable for explanation generation. We propose a formal encoding of LightGBM to produce precise explanations capturing the model's decision-making process. 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 like $f_1 > 0.5$ 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. Formally, encoding a tree is a logical formula representing the conjunction of feature conditions leading to a particular leaf.

Algorithm 2.1 outlines the steps to encode a decision tree into a logical representation. 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.

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.

Algorithm 2.1 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 $Thresholds \leftarrow \emptyset$, $Lvars \leftarrow \emptyset$, $Paths \leftarrow \emptyset$
for each feature f_i **do**
    Extract thresholds t_i and add to $Thresholds(f_i)$
    Assign logical variables L_i for splits and add to $Lvars(f_i)$
end for
for each branch b_i **do**
    Traverse and extract constraints, form path p_i from $Lvars$ and add to $Paths$
end for
for each path p_i in $Paths$ **do**
    Expand p_i into logical constraints $[L_1, L_2, ..., L_n, 0]$ and add to final set
end for
for each new path from further splits **do**
    Repeat the previous steps for new paths
end for
Ensure paths respect tree order and encode feature domains with logical variables
Return: Encoded paths P

2.3 Experimental Evaluation

We evaluated the encoded LightGBM on various datasets, whereas in this chapter, we only present the results obtained using the Breast Cancer dataset [65], a well-known dataset for binary classification. This dataset contains 569 samples with 30 numerical features, representing cell nuclei attributes from digitized images. The target variable distinguishes between two classes: *malignant* and *benign*. The dataset was split into 70% for training and 30% for testing, maintaining the class distribution.

2.3.1 Correctness of Encoded Models

Ensuring the correctness of the encoded model is an important step to verify that the logical representation matches the decision-making process of the original LightGBM model. Predictions from the encoded model were compared to those of the original model across both training and test datasets. This validation would demonstrate that the encoding process accurately represents the model's behavior for the evaluated data. The encoding approach directly translates the structure and logic of the LightGBM model into a formal representation. By systematically encoding decision paths and split conditions, it captures the key elements of the model.

The predictions of the encoded model were compared with those of the original LightGBM model on both training and test datasets. The encoded model achieved 100% agreement with the original model for all training and test samples, confirming the correctness of the encoding process. This consistency demonstrates that the encoded model accurately replicates the original model's behavior across both seen and unseen data.

2.3.2 Performance Metrics

Both the trained ML model (original) and its representation in propositional logic (encoded) were evaluated using standard classification metrics. The results were identical, demonstrating the reliability of the encoding process. The consistent results across both seen and unseen data provide confidence in the encoding's ability to replicate the original decision logic. Below, we define each metric [74]:

- **Precision:** Precision quantifies the proportion of correctly identified positive samples, i.e., true positives (TP), among all samples predicted as positive. False positives (FP) are incorrectly predicted positive samples. It is given by:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity):** Recall measures the proportion of actual positive samples correctly identified by the model. It considers true positives (TP) and false negatives (FN), where (FN) are positive samples incorrectly classified as

negative. It is defined as:

$$Recall = \frac{TP}{TP + FN}$$

- **F1-Score:** The F1-score balances precision and recall, providing a harmonic mean of the two. It is calculated as:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **Accuracy:** Accuracy represents the proportion of correctly classified samples to the total number of samples. Mathematically:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP and TN are the true positives and true negatives, while FP and FN are the false positives and false negatives, respectively.

2.3.3 Experimental Results

The previous sections thoroughly address the experimental process by detailing the evaluation of the encoded LightGBM model on the Breast Cancer dataset, using XReason+ to ensure the encoding accurately replicates the original model. We have collected experimental results by comparing predictions on testing datasets, with performance metrics calculated to validate the process. The performance metrics of the encoded model, including Precision, Recall, F1-Score, and Accuracy, are summarized in Table 2.1.

Table 2.1: Performance Metrics of the Encoded Model

Metric	**Value**
Precision	0.9243
Recall (Sensitivity)	0.9240
F1-Score	0.9241
Accuracy	0.9240

These metrics confirm that the predictive capabilities of the encoded model are

preserved, with no loss in performance compared to the original model. This consistency validates the accuracy and reliability of the encoding process.

2.4 Summary

This chapter details the encoding process for LightGBM models, starting with an overview of Gradient Boosting Tree Models and their functional representations. Logical foundations were introduced to frame decision trees as formal structures using propositional logic, CNF and DNF representations. An algorithm for encoding these structures was presented, showcasing how decision paths are systematically translated into logical constraints.

We have implemented LightGBM model encoding as part of the XReason+ tool (Figure 1.5). Thereafter, we have conducted experiments using the Breast Cancer dataset. The experimental results demonstrated the correctness of the encoded model, achieving perfect agreement with the original LightGBM model across both training and test datasets. Standard performance metrics confirmed that the encoding preserves the model's predictive capabilities.

The next steps of our proposed XReason+ tool include developing class-wise explanations and integrating adversarial example generation and detection, which are addressed in the following chapters. The next chapter specifically focuses on class-wise explanations.

Chapter 3

Class-wise Explanation

This chapter discusses the generation of formal local explanations for LightGBM models using SAT and MaxSAT solvers. It covers local explanations for individual predictions and class-wise explanations that summarize intervals of key features for each class. Experimental results validate the approach and compare formal local explanations with heuristic methods like SHAP, using several datasets, including the Image Segmentation dataset [67], the Ecoli dataset [68], and the Iris dataset [69].

3.1 Use of SAT and MaxSAT Solvers in Explanations

We propose to use the MaxSAT solver in generating formal local explanations for predictions made by LightGBM models, as it identifies the smallest set of features responsible for a prediction while maintaining computational efficiency. Building on the propositional encoding methodology detailed in Chapter 2, this section focuses on the computational processes and optimization strategies enabled by these solvers, ensuring the extraction of minimal, sufficient explanations with formal guarantees.

3.1.1 Propositional Encoding as a Foundation

The encoding of LightGBM models into CNF, as explained in Chapter 2, serves as the foundation for using SAT and MaxSAT solvers. This encoding represents the decision logic of tree ensembles in terms of propositional variables and clauses,

capturing the paths and thresholds that lead to predictions. Each decision tree is expressed as a set of hard clauses, ensuring that the logical constraints governing the model's behavior are satisfied.

3.1.2 SAT Solvers for Satisfiability Queries

SAT solvers are employed to verify whether a given subset of feature values is sufficient to guarantee a specific prediction. This involves checking the satisfiability of the CNF formula representing the LightGBM model, augmented with constraints enforcing the candidate feature subset. A satisfiable result indicates that the specified subset is enough to ensure the model's output, while an unsatisfiable result implies that additional features are necessary.

3.1.3 Optimization with MaxSAT Solvers

To identify *minimal* explanations, we use the MaxSAT solver which extends the capabilities of SAT solvers by allowing the inclusion of *soft clauses* with associated weights. Soft clauses represent conditions that are preferred but not mandatory to satisfy, enabling optimization in scenarios where not all constraints can be met [75]. For formal local explanations, the objective is to minimize the number of features included in the explanation while preserving the model's prediction.

The optimization process involves the following steps:

(1) **Input Preparation:** The CNF encoding of the LightGBM model is combined with a candidate feature subset, represented as soft clauses.

(2) **Objective Function Construction:** The MaxSAT solver is tasked with maximizing the satisfaction of soft clauses while ensuring all *hard clauses*, which are mandatory constraints that must always be satisfied, are upheld.

(3) **Iterative Refinement:** The solver iteratively evaluates and adjusts the candidate subset until a minimal set of features sufficient for the prediction is identified.

(4) **Output:** The result is a subset-minimal explanation that formally guarantees the prediction.

3.1.4 Formal Guarantees of Explanations

By employing MaxSAT solver, the approach ensures that the explanations are both *sound* and *minimal*. Soundness implies that the identified feature subset is always sufficient to guarantee the model's prediction, while it minimally ensures that no smaller subset can achieve the same result. This distinguishes MaxSAT-based methods from heuristic explainability techniques, which lack such formal guarantees.

3.1.5 Benefits and Challenges

The use of SAT and MaxSAT solvers offers several advantages:

- **Scalability:** The propositional encoding avoids computationally expensive linear constraints, making the approach efficient for large ensembles.
- **Precision:** Formal methods provide exact explanations, avoiding the inaccuracies often associated with heuristic methods.
- **Flexibility:** The optimization framework can adapt to a variety of LightGBM configurations and datasets.

3.2 Local Explanations

Local explanations aim to provide a detailed understanding of why a specific prediction was made for an individual instance. These explanations are generated by identifying the minimal subset of feature-value pairs sufficient to guarantee the model's output, leveraging the propositional encoding of the LightGBM model and the optimization capabilities of SAT and MaxSAT solvers.

3.2.1 Workflow for Generating Local Explanations

The process of generating local explanations involves three key steps: determining the predicted class for the instance, obtaining a consistent model for the instance and extracting the minimal explanation.

Step 1: Determining the Predicted Class

The first step of the process starts by determining the class predicted by the LightGBM model for the given instance. The respective steps are as follows [52] :

(1) The inputs to the process are the instance's feature values and the thresholds of all decision trees in the LightGBM model.

(2) The paths taken by the instance through the tree ensemble are analyzed to identify the relevant splitting features and their associated values.

(3) A *hypothesis* is generated, consisting of a minimal set of feature-value constraints that ensure the instance follows the correct paths through the trees.

(4) The hypothesis, along with the propositional formulas encoding the LightGBM model, is input into a SAT/MaxSAT solver.

(5) The output of the process is a *model* (truth value assignments for the propositional variables) representing the logical consistency of the hypothesis and the encoded model.

The solver's output is then used to identify the specific leaf nodes reached by the instance and the sum of the leaf outputs determines the predicted class.

Step 2: Obtaining a Consistent Model

The second step involves verifying the consistency of the hypothesis with the encoded model:

(1) The input to the process is the hypothesis and the LightGBM model encoding in CNF.

(2) A MaxSAT solver processes the input to find a *model*, which is an assignment of truth values that satisfy the hypothesis and all constraints of the encoding.

(3) The output of the process is a model that ensures the hypothesis is consistent with the decision paths and the predicted class.

Step 3: Extracting the Explanation

The final step is to compute the minimal explanation:

(1) The input to the process is the propositional encoding of the LightGBM model for the predicted class and the tree thresholds.

(2) The MaxSAT solver evaluates subsets of features to determine the most critical features required for the prediction. This is done iteratively, removing features and verifying whether the remaining subset still guarantees the prediction.

(3) The output of the process is the minimal subset of features sufficient to explain the prediction.

The solver ensures that the explanation is both *sound* (it guarantees the prediction) and *minimal* (it includes the smallest number of necessary features).

3.2.2 Advantages of the Formal Local Explanations

The use of formal methods for local explanations provides several benefits:

- **Soundness:** The generated explanations are guaranteed to be sufficient for the prediction, avoiding inaccuracies often present in heuristic methods.
- **Minimality:** The explanations include the smallest set of features required for the prediction, ensuring interpretability.
- **Scalability:** The use of efficient SAT and MaxSAT solvers enables the generation of explanations for complex models and large datasets.

3.2.3 Example of a Local Explanation

Consider an instance with the following feature values:

$$\text{Feature 1: } x_1 = 5.3,$$
$$\text{Feature 2: } x_2 = 2.8,$$
$$\text{Feature 3: } x_3 = 1.2.$$

Suppose the LightGBM model predicts Class A for this instance. By applying the proposed method, the following explanation is generated:

- The minimal subset of features, $\{x_1, x_3\}$, is sufficient to justify the prediction.
- Feature 2 (x_2) is deemed irrelevant for this specific instance, as removing it does not change the model's decision.

3.3 Class-wise Explanations

The notion of *class* in ML refers to the distinct categories or labels that a classification model predicts. Each class represents a specific group of data points with shared characteristics as defined by the target variable in the dataset. For instance, in a binary classification task, the classes could represent "positive" and "negative" outcomes, while in multiclass problems, the classes may correspond to multiple categories like different species of flowers or types of network intrusions.

The Class-wise 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 3.1 details the procedure for building the class-wise explanations. 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., $M(x_i) = c$).

For each instance, a formal local 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-wise 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.

Algorithm 3.1 Class-wise Explanation Building

Require: Trained model M, training dataset D_{train}

Ensure: Class-wise 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 local 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

3.4 Experimental Results

This section evaluates the reliability and robustness of our explanation methods using three datasets: Image Segmentation [67], Ecoli [68], and Iris [69]. By comparing feature frequencies, explanation lengths, and class-level explanations, we aim to understand how well the methods work across different models and domains.

3.4.1 Analysis of Local Explanations

In our experiments, we utilized the Image Segmentation dataset from the UCI ML Repository [67] and the Ecoli dataset [68] from the same repository. The Image Segmentation dataset consists of instances drawn from a database of outdoor images, where each instance represents a 3x3 region characterized by 19 high-level features

such as contrast, intensity and various color metrics, with the goal of classifying these regions into different terrain types (e.g., grass, sky, foliage). The Ecoli dataset, on the other hand, consists of instances representing the cellular localization sites of proteins in Ecoli, characterized by 7 features such as the amino acid composition, with the goal of classifying the protein's localization site. We trained XGBoost and LightGBM models using the same hyperparameter settings to ensure a fair comparison of their explanations. Specifically, both models were trained with a maximum tree depth of 3 and 20 trees. For all generated explanations corresponding to correct predictions, we carefully checked for counterexamples, instances where the feature values identified as important would lead to a different class prediction and did not find any. This observation reinforces the reliability of the explanations.

Comparison of XGBoost and LightGBM for the Segmentation Dataset

We conduct a comparison of feature frequencies in explanations generated by XGBoost and LightGBM for the segmentation dataset. Figure 3.1 presents a comparative analysis of feature frequency in the generated explanations across both models. The results reveal interesting patterns in how these models prioritize different features in their explanations. While both models show similar overall trends for certain features (e.g., Feature 10, Feature 16 and Feature 19 maintaining high importance across both models), there are notable divergences in others. The intersection analysis (shown in green) provides particularly valuable insights, as it highlights features that consistently appear in explanations regardless of the underlying model. This consistency across different models strengthens the reliability of our formal XAI approach, while the differences underscore the importance of model-agnostic explanation methods. For instance, Feature 10 shows the highest frequency in both models with significant overlap, suggesting its fundamental importance to the underlying problem rather than being an artifact of any particular model's architecture. The rightmost bars represent the average number of times each feature appears in the explanations, showing that both XGBoost and LightGBM have similar averages (70-72 appearances per feature), with an intersection of about 40 appearances. This similar average frequency of feature appearances suggests that both models exhibit comparable patterns in their explanations, despite their architectural differences, further supporting the robustness of our formal XAI approach across different model architectures.

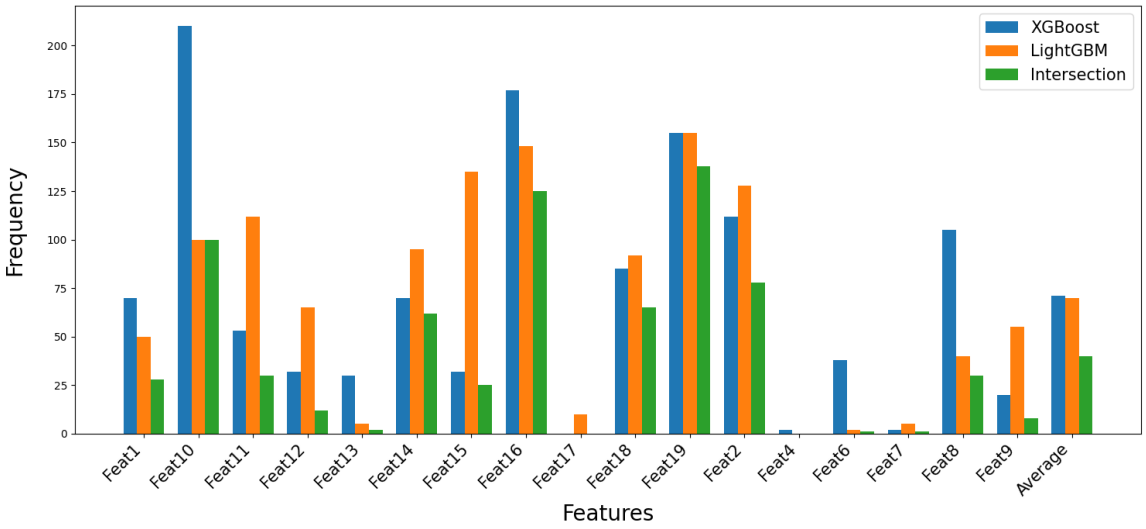

Figure 3.1: Features Frequency for Different Cases for the Segmentation Dataset

To further analyze the explanation characteristics, we examined the distribution of explanation lengths generated by both models, as shown in Figure 3.2. The graph reveals that both XGBoost and LightGBM predominantly generate explanations using 3 to 11 features, with a clear peak at 4 features for both models (approximately 50 occurrences). This preference for relatively concise explanations aligns with the goal of interpretability in XAI, as shorter explanations are generally more comprehensible to human users. Interestingly, while both models show similar patterns for shorter explanations (3-4 features), they exhibit some divergence in their behavior for medium-length explanations (5-8 features). LightGBM maintains a more gradual decline in frequency across these lengths, suggesting a more uniform distribution of explanation complexities. In contrast, XGBoost shows a steeper decrease in frequency for longer explanations, indicating a stronger preference for more concise explanations. The relatively low frequency of explanations with 9-11 features for both models suggests that highly complex explanations are rare, regardless of the underlying model architecture. This consistency in explanation length patterns further supports the robustness of our formal XAI approach across different model implementations.

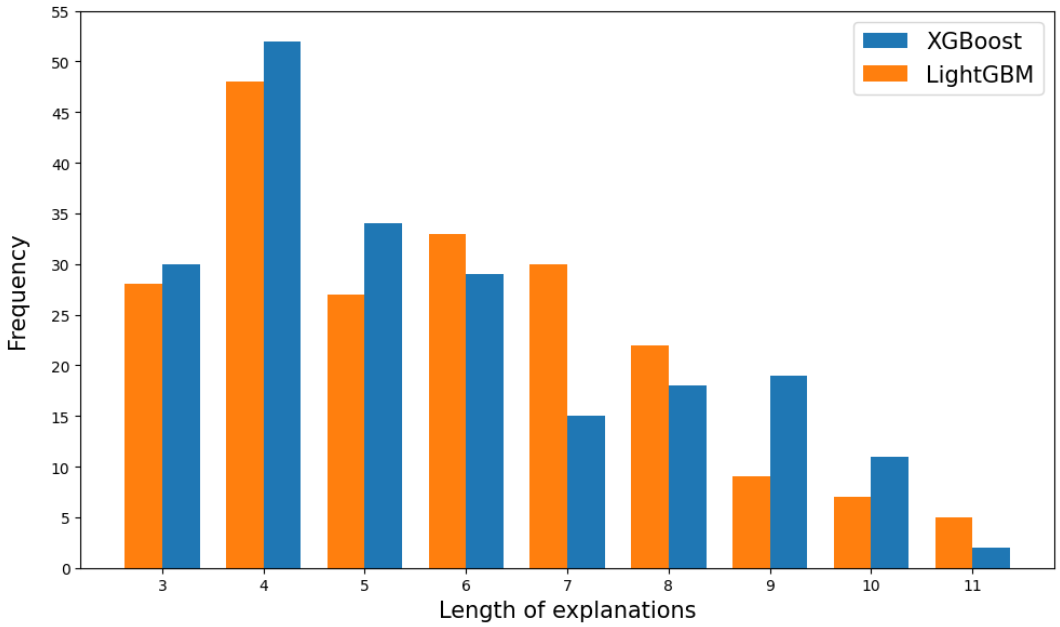

Figure 3.2: Explanation Length Frequency for the Segmentation Dataset

Comparison of Formal Explanation and SHAP for the Ecoli Dataset

Figure 3.3 shows the frequency of features identified as important in explanations for the Ecoli dataset. For each instance, we compared the features identified by the Local Explanation method with the top features selected using SHAP's feature importance rankings. This comparison helps highlight the differences in how the two methods prioritize features.

When two features are marked as important by the Local Explanation method, we selected the top two features ranked by SHAP for the same instance. The goal here is to demonstrate that while SHAP sometimes identifies highly ranked features, it can lead to counterexamples where altering these features changes the prediction class. In contrast, the Local Explanation method ensures that no such counterexamples occur, providing correctness in the explanations.

For example, the Feature *mcg* was identified 305 times by the Local Explanation method, compared to 259 times by SHAP. This shows that SHAP may overlook important features in some cases. On the other hand, the Feature *gvh* appeared 325 times in SHAP explanations but only 109 times in Local Explanation, with an intersection of 108 occurrences. This indicates areas where the two methods agree and differ.

These results demonstrate that SHAP lacks robustness, as it can select features that lead to counterexamples, making its explanations less reliable compared to the Formal Local Explanation method.

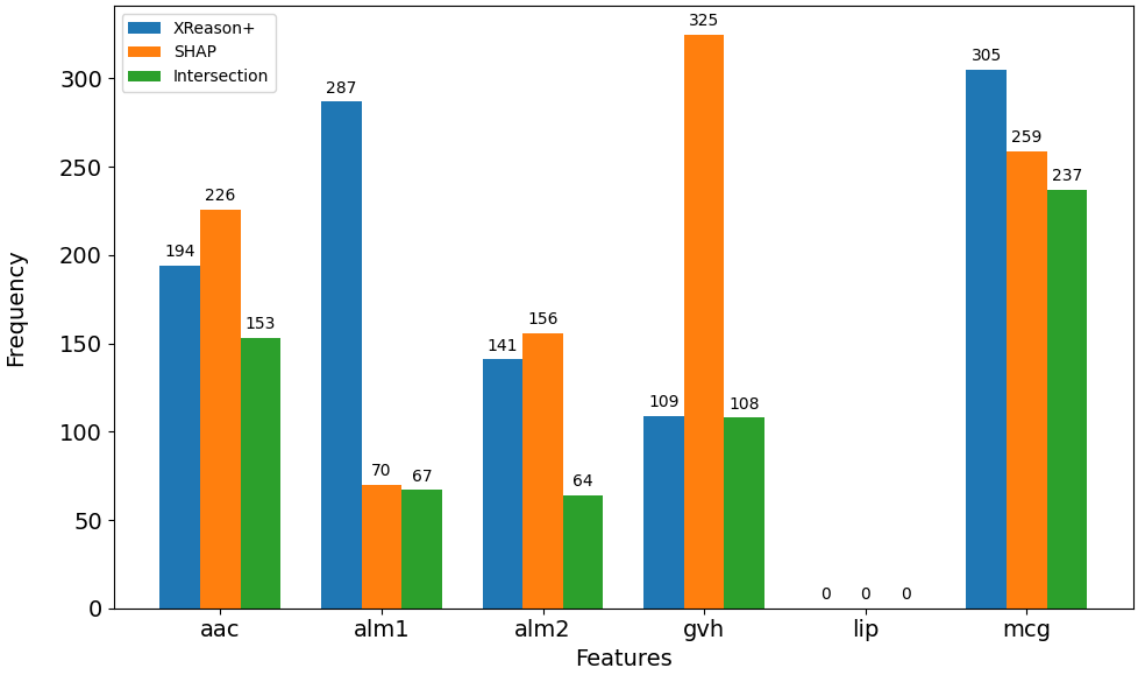

Figure 3.3: Features Frequency for Different Cases for the Ecoli Dataset

Out of the 210 explanations generated, we found that there were 0 identical explanations between the Formal Local Explanation method and SHAP. This finding highlights that SHAP often identifies different features as important, some of which, when altered, can lead to counterexamples where the prediction class changes. In contrast, the Formal Local Explanation method guarantees correctness by ensuring that the features identified are consistent with maintaining the prediction's validity.

By using the top n features from SHAP for comparison, we demonstrated that Formal Local Explanation offers a more robust and reliable approach. The differences observed further emphasize the value of Formal Local Explanation in applications requiring explanations that are not only insightful but also free from errors caused by potential counterexamples.

3.4.2 Analysis of Class-wise Explanations

The class-wise explanations were analyzed for their ability to capture the most representative features for each class. Using the Iris dataset [69], a well-known benchmark in classification tasks, we examined how features like sepal length, sepal width, petal length, and petal width contributed to distinguishing between the three flower species. Figure 3.4 illustrates the frequency of features appearing in explanations across different classes. For each class, the most frequent features typically align with the domain knowledge, indicating that the formal local explanation method successfully identifies the core drivers of predictions. Certain features appear exclusively in the explanations for specific classes, reinforcing their role as distinguishing factors for those classes. This analysis demonstrates how the class-wise explanations provide a high-level understanding of the model's decision boundaries, while also ensuring that the identified features are interpretable and actionable.

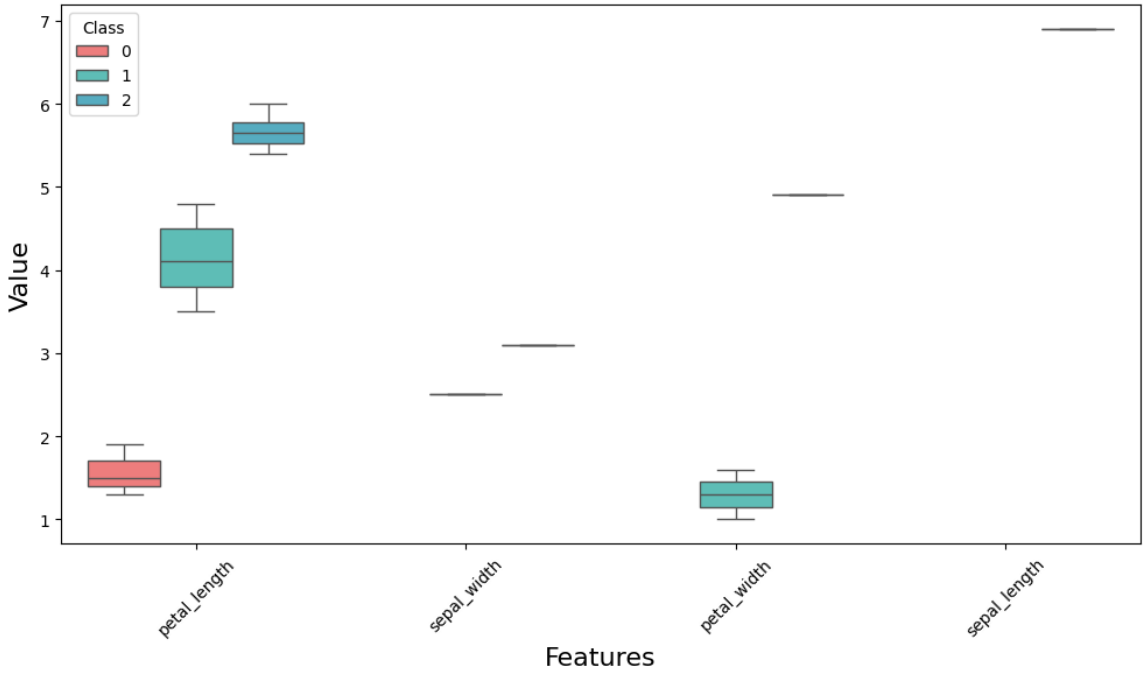

Figure 3.4: Feature Value Distributions Across Classes for the Iris Dataset

To assess the stability of the generated explanations, we compared the intervals of feature values across classes. For features shared between multiple classes, there was no overlap in their intervals, suggesting that the explanation method accurately captures class-specific patterns. For example, *petal_length* has distinct, non-overlapping

ranges for each class: Class 0 with values clustered between 1.0 and 2.0, Class 1 between 3.5 and 4.8 and Class 2 above 5.4. Similarly, *petal_width* differentiates Class 1 (1.0–1.6) and Class 2 (around 4.9). These results validate the precision of the formal local explanations in differentiating classes based on their feature distributions.

The analysis of class-wise explanations uncovered several key insights:

- **Feature Importance Across Classes:** Features such as *petal_length* and *petal_width* are consistently important across classes, with distinct value intervals reflecting their critical role in classification.
- **Model Robustness:** The lack of overlap in feature intervals between classes indicates that the model maintains clear boundaries, enhancing robustness.
- **Interpretability:** Aggregating feature intervals offers a concise, interpretable summary of what defines each class, aiding domain experts in understanding model decisions.

We implemented the functionality of computing class-wise explanations in the XReason+ tool, enabling users to generate and analyze these explanations efficiently for various datasets.

3.5 Summary

This chapter describes methods for generating local and class-wise explanations. Local explanations use SAT and MaxSAT solvers to identify minimal feature subsets needed for individual predictions, ensuring they are sound and interpretable. We proposed an algorithm that outlines the process for constructing class-wise explanations by aggregating important features from training instances per class and computing feature value intervals. We have implemented a functionality for computing class formal based explanations, that is part of the XReason+ tool (Figure 1.5). Experimental results confirmed the reliability of these explanations, showing consistency with model predictions and highlighting the benefits of formal methods over heuristic ones like SHAP. The chapter demonstrates that formal local explanations are robust and offer reliable insights into LightGBM model behavior. The next chapter will explore how these explanations are leveraged for generating and detecting adversarial examples.

Chapter 4

Adversarial Examples

This chapter focuses on adversarial examples, a critical challenge in ML where small input perturbations can lead to incorrect predictions. In the context of XAI, such attacks not only alter predictions but also compromise the integrity of model explanations. We introduce a framework for generating adversarial examples using formal explanations and detecting them based on discrepancies in class-wise explanations. Experimental results validate the approach using the Banknote Authentication dataset [66]. The potential effects of the adversarial attacks can be described in two main contexts: (1) attacks, which affect both predictions and explanations; (2) attacks, which manipulate explanations without changing the output predictions. Our proposed framework focuses on the first context, where we use the formal explanations to create adversarial perturbations that impact both the model's predictions and its explanations.

4.1 Adversarial Example Generation using Formal Methods

Adversarial example generation is a critical process in evaluating the robustness of ML models. This process involves crafting examples that are minimally perturbed but are sufficient to alter the model's prediction. In this section, we describe in detail the steps involved in generating adversarial examples using formal methods. The approach is outlined in Algorithm 4.1, which consists of two primary phases: (1) analyzing the training data to extract key features and their value ranges for each class

and (2) generating adversarial examples by perturbing these features under specific constraints.

Algorithm 4.1 Adversarial Example Generation

Require: Trained model M, test input x_{test}, class-wise explanations $\mathcal{E}_c$, training dataset D_{train}

Ensure: Adversarial example x_{adv}

procedure GENERATEADVERSARIALEXAMPLE(x_{test})
  $y_{\text{test}} \leftarrow M(x_{\text{test}})$ ▷ Predict label of test input
  Extract $\mathcal{E}_{\text{test}}$ for x_{test} ▷ Extract most important features and values
  $x_{\text{adv}} \leftarrow x_{\text{test}}$ ▷ Initialize adversarial example
  for each (f, v_f) in $\mathcal{E}_{\text{test}}$ **do**
    Retrieve $[a_f^{y_{\text{test}}}, b_f^{y_{\text{test}}}]$ from $\mathcal{E}_{y_{\text{test}}}$
    Retrieve $[f_{\min}, f_{\max}]$ from D_{train}
    Find minimal $\epsilon \neq 0$ such that:
    $v'_f = v_f + \epsilon$
    $v'_f \in [f_{\min}, f_{\max}]$
    $v'_f \notin [a_f^{y_{\text{test}}}, b_f^{y_{\text{test}}}]$
    Update f in x_{adv} to v'_f
    if $M(x_{\text{adv}}) \neq y_{\text{test}}$ **then** ▷ Check if prediction changed
      break ▷ Stop perturbation if adversarial
    end if
  end for
  return x_{adv}
end procedure

The following subsections elaborate on each step of the process outlined in Algorithm 4.1.

Feature Extraction and Interval Definition

The first phase of the adversarial example generation process focuses on understanding the characteristics of the training data, specifically the most important features and their values that contribute to model predictions. This is achieved by utilizing formal explanation techniques, which identify the features and their values

necessary to preserve the model's decision for a given instance. Unlike traditional feature importance methods, this approach treats all extracted features equally, as any change to these features can potentially alter the prediction.

In our work, we focus exclusively on continuous data. This means that the features extracted through formal explanations are numerical and can take on a range of values, which is essential for defining meaningful intervals for perturbation.

The formal explanation mechanism is applied to instances from the training dataset D_{train} that belong to each class c in the set of all classes $\mathcal{C}$. The explanation of each instance provides a minimal set of feature-value pairs:

$$\mathcal{E}_x = \{(f_1, v_1), (f_2, v_2), \ldots, (f_k, v_k)\},$$

where f_i represents a feature and v_i is its value for the instance x. The union of these explanations over all instances of class c forms the set $\mathcal{F}_c$, which contains all most important features for the class. These features and their values are key to capturing the model's behavior for class c.

To aggregate these feature values across multiple instances, we use K-means clustering [76] to group similar values. K-means is a widely used clustering algorithm that partitions data points into K clusters by minimizing the sum of squared distances between data points and their respective cluster centroids. Specifically, for a feature f and its values $\mathcal{V}_f^c$ observed in D_{train} for class c, the K-means algorithm operates as follows:

- **Initialization:** Randomly initialize K cluster centroids within the range of feature values.
- **Assignment Step:** Assign each value $v \in \mathcal{V}_f^c$ to the nearest centroid based on the Euclidean distance.
- **Update Step:** Recalculate the centroids as the mean of all values assigned to each cluster.
- **Convergence:** Repeat steps 2 and 3 until the cluster assignments no longer change or a predefined number of iterations is reached.

The resulting clusters are used to define intervals $[a_f^c, b_f^c]$ for each feature f in class c. These intervals represent the typical value ranges for the feature within the class.

Formally:

$$a_f^c = \min(\text{cluster bounds of } f), \quad b_f^c = \max(\text{cluster bounds of } f).$$

The use of K-means ensures that the intervals capture natural groupings of feature values, which helps improve robustness to noise or outliers in the data. These intervals serve as the foundation for generating adversarial examples.

Generating Adversarial Examples

The second phase involves crafting adversarial examples for a given test input x_{test}. The process begins by extracting the features and their values from the formal explanation of x_{test}. This explanation, denoted as $\mathcal{E}_{\text{test}}$, contains the most important feature-value pairs for the prediction of x_{test}. These pairs are used to determine which features need to be perturbed to generate an adversarial example.

Each feature f in $\mathcal{E}_{\text{test}}$ is perturbed according to a set of constraints. The goal is to modify the value v_f of feature f to a new value v'_f such that the prediction of the model changes. However, the perturbation must satisfy the following conditions:

- **Data Range Constraint:** The perturbed value v'_f must lie within the overall range of feature f observed in D_{train}, i.e., $v'_f \in [f_{\min}, f_{\max}]$.
- **Class Range Constraint**: The perturbed value v'_f must fall outside the typical interval $[a_f^{y_{\text{test}}}, b_f^{y_{\text{test}}}]$ associated with the predicted class y_{test}. This ensures that the perturbation moves v_f into a region of the feature space that does not typically align with the predicted class.

The algorithm iteratively applies perturbations to the features in $\mathcal{E}_{\text{test}}$ until the model's prediction changes. Once the prediction changes (i.e., $M(x_{\text{adv}}) \neq M(x_{\text{test}})$), the process stops to ensure minimal perturbation.

Evaluation of Adversarial Example Generation

To assess the effectiveness of the adversarial example generation approach, we define these evaluation metrics:

- **Success Rate:** The percentage of adversarial examples that successfully change the model's prediction. This metric quantifies the ability of the method to fool

the model:

$$Success\ Rate = \frac{Number\ of\ successful\ adversarial\ examples}{Total\ number\ of\ test\ inputs}$$

- **Perturbation Magnitude:** The average L_2 distance, also known as the Euclidean distance, between x_{test} (the original test sample) and x_{adv} (the corresponding adversarial example) measures the subtlety of the perturbations. Smaller distances indicate more realistic adversarial examples:

$$Perturbation\ Magnitude = \frac{1}{N}\sum_{i=1}^{N} \|x_{adv,i} - x_{test,i}\|_2$$

where N is the total number of adversarial examples, $x_{test,i}$ represents the i-th test sample, and $x_{adv,i}$ is the corresponding adversarially perturbed version of that sample.

4.2 Detection of Adversarial Examples using Formal Explanations

The process of detecting adversarial examples involves leveraging formal explanations to compare the features of an input sample with the class-wise explanations derived from the training data. The detection process evaluates whether the input sample's features align with the characteristics expected for the predicted class. This is done by performing two checks: (1) verifying that the features identified in the input explanation are relevant to the predicted class and (2) ensuring that their values fall within the typical intervals defined in the class-wise explanations.

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

Algorithm 4.2 outlines the process of detecting adversarial examples. For each feature in the explanation of x_{input}. Algorithm 4.2 performs the following checks:

(1) **Feature Relevance Check:** The algorithm starts by checking if the feature f is part of the class-wise explanations $\mathcal{E}_{y_{\text{input}}}$. If it turns out that $f \notin \mathcal{E}_{y_{\text{input}}}$, then the feature is considered irrelevant for the predicted class y_{input}. In this case, the discrepancy count d is increased by 1, and the algorithm moves on to check the next feature.

(2) **Value Range Check (if relevant):** If $f \in \mathcal{E}_{y_{\text{input}}}$, the algorithm retrieves the interval $[a_f^{y_{\text{input}}}, b_f^{y_{\text{input}}}]$, which defines the expected range for f in the predicted class. It checks whether the input feature value $V_{\text{input}}(f)$ lies within this interval; if not, then the discrepancy count d is incremented by 1.

If either of the above checks fails for a feature, it is considered a discrepancy. The total number of discrepancies is accumulated, and the adversarial likelihood score s_{adv} is calculated as the ratio of discrepancies to the total number of features in the input explanation $\mathcal{F}_{input}$, where $\mathcal{F}_{input}$ represents the set of features used to generate the explanation:

$$s_{adv} = \frac{\textit{Number of discrepancies}}{\textit{Total number of features in } \mathcal{F}_{input}}.$$

To improve efficiency, the algorithm halts further checks if the adversarial likelihood score exceeds a predefined threshold τ, indicating that the sample is highly likely to be adversarial.

Algorithm 4.2 Adversarial Sample Detection Using Class-Wise Explanations

Require: Trained model M, class-wise explanations $\{\mathcal{E}_c\}$, input sample x_{input}, threshold τ

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-wise 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
    Compute interim adversarial likelihood score:

$$s_{\text{adv}} \leftarrow \frac{d}{|\mathcal{F}_{\text{input}}|}$$

    if $s_{\text{adv}} \geq \tau$ **then** ▷ Stop if likelihood exceeds threshold
        break
    end if
end for
return s_{adv}

4.3 Experimental Results

This section presents experiments on detecting adversarial examples in Banknote Authentication Dataset to assess the reliability and robustness of the proposed methods. It examines the success rate and perturbation magnitude of adversarial generation and evaluates the detection method to ensure accurate identification of manipulated instances.

4.3.1 Dataset and Experimental Setup

The Banknote Authentication Dataset, sourced from the UCI ML Repository [66], consists of features extracted from digital images of banknotes. These features include variance, skewness, kurtosis of the wavelet-transformed image and entropy. The dataset is commonly used for binary classification tasks, with labels indicating whether a banknote is authentic (class 1) or forged (class 0).

The dataset contains 1,372 samples, with each sample described by four numerical features and a binary target variable. For the experiments, 138 samples were used, with 70% allocated for training and 30% for testing. The training dataset was used to train the LightGBM model, while the test dataset was used for adversarial generation and detection experiments.

4.3.2 Generation of Adversarial Examples

The main goal of this process was to slightly modify the input features from the test dataset, keeping the altered instances as close as possible to the original ones while causing the model to misclassify them. This method emphasizes generating adversarial examples that remain realistic and within the bounds of the dataset's feature distribution.

The generation process was repeated ten times using the same perturbation threshold for all runs. This setup made it possible to assess variations caused by the inherent randomness of the adversarial generation algorithm. Each run resulted in a unique set of adversarial examples, varying in the number of samples that fooled the model, the transitions between classes (e.g., from class 0 to class 1 and vice versa) and the extent of the applied perturbations, quantified by the average Euclidean distance between the original and adversarial samples.

Table 4.1 provides a comprehensive summary of the results across the ten runs. For each run, the table reports the percentage of samples that fooled the model. It also provides a breakdown of class transitions, showing the percentage of samples initially classified as 0 (authentic banknotes) that were misclassified as 1 (forged banknotes) and vice versa. On average, 34.52% of the samples successfully fooled the model, with an average Euclidean distance of 13.70 between original and adversarial examples. The average runtime for generating adversarial examples across all runs was 3.67 seconds.

Table 4.1: Adversarial Examples Generation

Run	Fooled (%)	0 to 1 (%)	1 to 0 (%)	Avg. Euclidean Distance
1	40.48	34.48	53.85	16.155
2	30.95	24.14	46.15	11.100
3	33.33	10.34	84.62	16.901
4	26.19	13.79	53.85	11.970
5	33.33	27.59	46.15	9.201
6	40.48	24.14	76.92	15.482
7	30.95	20.69	53.85	13.898
8	38.10	20.69	76.92	16.554
9	28.57	10.34	69.23	13.804
10	42.86	27.59	76.92	11.892

In addition to the numerical details provided in the above table, Figure 4.1 offers a visual representation of key metrics across the ten runs. The graph plots both the percentage of samples that fooled the model and the average Euclidean distance for each run. This visualization highlights the variability in the adversarial generation process, illustrating how random seeds may affect the outcomes. It is observed that the largest average Euclidean distances do not always result in the input fooling the model. For example, the highest percentage of samples that fooled the model (42.86%) had an average Euclidean distance of 11.89 between the generated adversarial examples and the original test samples, which is below the overall average.

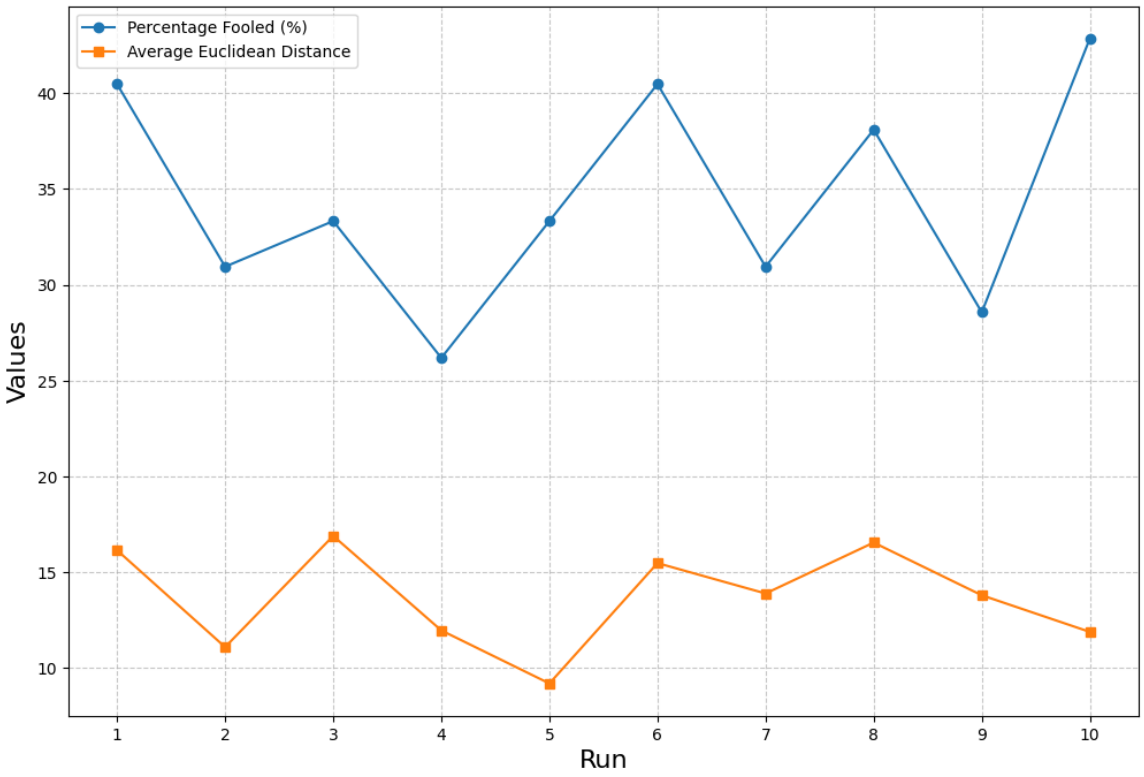

Figure 4.1: Percentage of Fooled Samples and Average Euclidean Distance Across 10 Runs

4.3.3 Detection of Adversarial Examples

The detection framework was tested on adversarial samples generated in the Banknote Authentication dataset. Out of the test dataset, the adversarial generation process resulted in:

- **Samples that changed from 0 to 1:** 8 out of 29 original class 0 samples (27.59%).
- **Samples that changed from 1 to 0:** 10 out of 13 original class 1 samples (76.92%).

All 18 adversarial samples were used as input for the detection framework, achieving a detection accuracy of 100%, with all 18 samples correctly identified as adversarial. The detection mechanism successfully flagged all adversarial samples by identifying feature value discrepancies against the expected class-wise explanation intervals. For the 8 samples changing from class 0 to class 1, feature values were outside the typical

intervals for class 0 and aligned with class 1's feature space. Similarly, the 10 samples changing from class 1 to class 0 showed perturbations that moved them out of class 1's expected intervals. This approach proved particularly effective for the Banknote Authentication dataset, where feature distributions and their impact on predictions could be explicitly captured through formal explanations.

4.4 Summary

This chapter addresses the challenge of adversarial examples, focusing on their dual impact on model predictions and explanations. We introduced approaches for the generation of adversarial examples using formal explanations and detect them by analyzing discrepancies in class-wise explanations. The generation process ensured minimal yet effective perturbations, altering both predictions and associated explanations. The detection mechanism leverages class-wise explanations to identify inconsistencies between the expected and observed feature values, enabling precise identification of adversarial samples.

As part of XReason+ (Figure 1.5), we implemented the adversarial example generation and detection components. Experimental evaluations using the Banknote Authentication dataset demonstrated the methods' effectiveness in generating subtle adversarial examples and achieving high detection accuracy, highlighting its potential for enhancing model robustness against adversarial attacks. The next chapter presents a case study based on network security demonstrating the application of all components of XReason+ .

Chapter 5

Case Study: Network Security Application

This chapter presents a case study showcasing the application of the proposed XReason+ tool in the field of network security. Using a customized version of the CICIDS-2017 dataset [64], the study evaluates the tool's capabilities in explaining and enhancing the robustness of LightGBM models. The chapter covers the dataset's characteristics, model performance metrics, robustness of explanations compared to SHAP and LIME and the use of formal explanations for adversarial example generation and detection.

5.1 Dataset Overview

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 [64], 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. [77], which reduces the feature set to 19 essential attributes, selected for their

relevance to network traffic patterns. Table 5.1 summarizes the features utilized from the customized CICIDS-2017 dataset for analysis [77].

Table 5.1: Description of the Customized CICIDS-2017 Dataset

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.

5.2 Model Performance

We trained an LightGBM model on the training set and evaluated its performance on the testing set, following the recommendations of Li et al. [77]. The evaluation was conducted using key metrics: Accuracy, Precision, Recall, F1 Score, and Area Under the Receiver Operating Characteristic (ROC) Curve (AUC). The ROC curve illustrates the trade-off between the True Positive Rate (TPR) and False Positive Rate

(FPR) at various thresholds, while the AUC quantifies the model's overall ability to distinguish between attack and non-attack samples. Table 5.2 summarizes the metrics and the corresponding values obtained, which range from 90% to 93%.

Table 5.2: 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

5.3 Robustness and Correctness of Formal Explanations

Many previous studies have used SHAP and LIME to explain models trained on CICIDS-2017 data (e.g., [78, 79]). 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 in XReason+ 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 sizes 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 XReason+, we used Spearman's Rank Correlation [80], Kendall's Tau [81] and Rank-Biased Overlap (RBO) [82] 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.

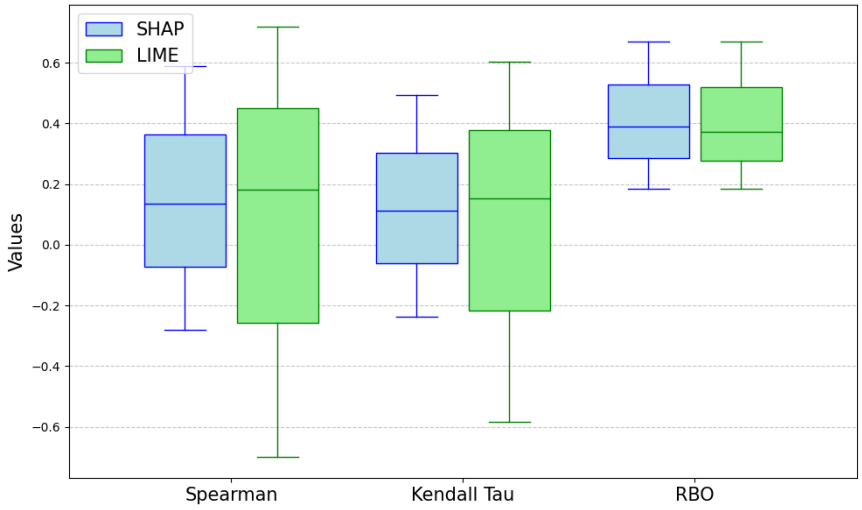

Figure 5.1: Comparison of SHAP and LIME Rankings with XReason+

The results for each metric are summarized as follows:

  - **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.

Figure 5.1 shows the variability and averages of Spearman's Rank Correlation, Kendall's Tau, and RBO metrics for SHAP and LIME, comparing their alignment with the formal method.

In summary, the formal explanation method for LightGBM predictions in XReason+ 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 examples that exploit the model's vulnerabilities.

5.4 Adversarial Examples Generation and Detection

5.4.1 Adversarial Examples Generation

To evaluate the robustness of our approach, we applied adversarial example generation techniques across the entire test set, consisting of 8,853 samples. Our formal explanation-based method demonstrated notable effectiveness, successfully fooling the model in 2,821 cases (31.86%). To provide context for these results, we compared our approach against two widely recognized adversarial generation methods, HopSkipJumpAttack (HSJ) [83] and Carlini Wagner (CW) attacks [84], both of which have been applied to network traffic in recent studies [85].

HopSkipJumpAttack (HSJ)

The HopSkipJumpAttack (HSJ) is a decision-based adversarial attack designed to craft adversarial examples for black-box ML models [83]. It focuses on generating perturbations without requiring access to the model's internal parameters or gradients.

Key characteristics of the HSJ attack include:

- **Black-Box Methodology:** Operates exclusively on model decision outputs, making it suitable for models where gradients or internal architectures are inaccessible.
- **Iterative Boundary Search:** Utilizes a systematic search strategy involving "hop," "skip," and "jump" steps to efficiently navigate the input space and identify the decision boundary.
- **Minimal Perturbation Principle:** Produces adversarial examples with minimal modifications, ensuring semantic similarity to the original input.
- **Computational Efficiency:** Designed to be less computationally intensive than other black-box attack techniques, facilitating practical adversarial testing.

The attack can be formulated as the following optimization problem [83]:

$$\min_{\delta} \|\delta\|_p \quad \text{subject to } f(x+\delta) \neq f(x), \tag{5.1}$$

where x is the input, δ the perturbation, f the model's decision function and $\|\delta\|_p$ the p-norm of the perturbation.

The HSJ algorithm employs a unique strategy involving:

(1) **Hop:** Random sampling of initial candidate perturbations.

(2) **Skip:** Directional estimation of the decision boundary.

(3) **Jump:** Refined search towards the decision boundary.

This approach makes HSJ effective against models with complex decision boundaries.

Carlini & Wagner (CW)

The Carlini & Wagner (CW) attack is a white-box adversarial attack renowned for its ability to produce high-quality adversarial examples [84].

Key features of the CW attack include:

- **Custom Optimization Formulation:** Uses a tailored optimization objective to minimize perturbation while achieving misclassification.
- **Flexible Distance Metrics:** Supports L_0, L_2, and L_∞ distance metrics, which are commonly used to measure perturbations in adversarial example generation. L_0 counts the number of features modified, L_2 measures the Euclidean distance between original and perturbed inputs, and L_∞ captures the maximum change to any feature, enabling tailored generation strategies based on specific requirements.
- **Provable Effectiveness:** Demonstrates superior performance across various neural network architectures.
- **Theoretical Robustness:** Provides a strong foundation for understanding and exploiting model vulnerabilities.

The attack's optimization problem is expressed as [84]:

$$\min_{\delta} \|W(\delta)\|_p + c \cdot \max(0, \text{loss}(f(x+\delta), t)), \tag{5.2}$$

where $W(\delta)$ constrains the perturbation, c controls the balance between perturbation size and adversarial strength, f is the model, x the input and t the target class.

The attack involves:

(1) Reformulating the constrained optimization problem into an unconstrained form.

(2) Using optimization techniques such as Adam [86] or L-BFGS [87].

(3) Iteratively refining the adversarial example.

CW attacks have been instrumental in identifying vulnerabilities across various domains, including image classification and malware detection.

5.4.2 Results and Comparison

We run experiments to compare the adversarial example generation methods: XReason+, HSJ, and CW. The experiments were conducted using a trained LightGBM model and evaluated on the same CICIDS-2017 dataset. For each method, we calculated the percentage of samples that successfully fooled the model, the misclassification rates for each class (Class 0 to 1 and Class 1 to 0), and the average Euclidean distance of adversarial perturbations. The results are shown in Table 5.3.

Table 5.3: Comparison of Adversarial Generation Methods

Method	**Fooled Samples (%)**	**Class 0 to 1 (%)**	**Class 1 to 0 (%)**	**Avg. Distance**
XReason+	31.86	13.41	66.84	1.670
HSJ	8.27	2.54	19.13	0.275
CW	1.14	1.09	1.24	0.11

From the above table, it becomes clear that XReason+ achieves the highest success rate, with 31.86% of samples successfully fooling the model. This performance significantly surpasses that of HSJ (8.27%) and CW (1.14%), underscoring its effectiveness in exposing vulnerabilities within the model. The results clearly position XReason+ as the most effective method among the three.

Building on this, XReason+ also reveals a striking disparity in class-specific misclassifications. It misclassifies 66.84% of samples from Class 1 to Class 0, a much higher rate than the 13.41% observed for misclassifications from Class 0 to Class 1. This asymmetry may be attributed to intrinsic dataset properties. By comparison, HSJ and CW exhibit lower misclassification rates. Specifically, HSJ shows 19.13% misclassifications from Class 1 to Class 0 and 2.54% in the opposite direction, while CW maintains balanced but minimal rates of 1.24% and 1.09%, respectively.

The effectiveness of XReason+ is further characterized by the magnitude of adversarial perturbations. On average, XReason+ generates perturbations with a Euclidean distance of 1.670, indicating substantial modifications to input data. This likely contributes to its high success rate. In contrast, HSJ achieves smaller perturbations, averaging 0.275, while CW produces the most imperceptible perturbations at 0.11. These differences reflect distinct priorities in balancing effectiveness against imperceptibility.

Taken together, these findings illustrate the unique strengths and trade-offs of each

method. XReason+ emerges as the most effective for generating adversarial examples, making it well-suited for robustness evaluation. However, its larger perturbations highlight a trade-off between success rate and subtlety.

5.4.3 Detection of Adversarial Examples

Using our formal explanation detection mechanism, as described in Algorithm 4.2, we identified 1,731 out of the 2,821 adversarial examples generated by our approach as likely adversarial, resulting in a detection rate of 61.36%. While methods like HSJ and CW are commonly used for adversarial generation, they are not directly applicable to adversarial detection, particularly in the context of tabular data.

5.5 Summary

This chapter demonstrated the application of XReason+ in network security using the customized CICIDS-2017 dataset. The LightGBM model achieved strong performance, with metrics like Accuracy and F1 Score exceeding 90%. Formal explanations in Xreason+ proved more robust and reliable than SHAP and LIME, showing 100% consistency and correctness in identifying critical features for predictions.

The adversarial examples were generated with minimal perturbations, achieving a misclassification rate of 31.86%. The detection mechanism based on formal explanations identified 61.36% of adversarial examples, underscoring its effectiveness in addressing model vulnerabilities. This case study highlights the utility of XReason+ in real-world applications, ensuring both interpretability and robustness.

Chapter 6

Conclusion

6.1 Summary of Contributions

Machine Learning models are widely used in critical domains like cybersecurity, where transparency and trust are essential. Despite their effectiveness, the reliance on complex black-box models raises significant concerns, as their opaque nature limits interpretability. Explainable AI (XAI) methods aim to address these issues, but heuristic approaches often lack consistency and guarantees, making them unreliable in critical scenarios.

Formal XAI methods provide a solution by offering mathematically rigorous, logically consistent, and minimal explanations. Among the existing tools, XReason stands out for its formal approach but is limited to supporting XGBoost models, providing only local explanations, and lacking mechanisms for adversarial example generation and detection. This work extends XReason to address these gaps, introducing support for LightGBM models, class-level explanations, and methods for generating and detecting adversarial examples. These contributions create a comprehensive framework that ensures transparency, reliability, and robustness.

A propositional encoding was developed for LightGBM models, achieving perfect agreement with the original models across training and test datasets. This encoding preserves predictive performance while enabling formal explanations. Local explanations use SAT/MaxSAT solvers to identify minimal feature subsets needed for individual predictions, ensuring they are interpretable and sound. Class-level explanations aggregate important features across instances, capturing class-specific patterns and

enabling global interpretability. Experimental results confirmed that these formal explanations are consistent with model predictions and outperform heuristic methods like SHAP.

Adversarial robustness was addressed by generating adversarial examples with minimal perturbations, which altered both predictions and associated explanations. Detection mechanisms were implemented to analyze discrepancies in class-wise explanations, successfully identifying adversarial samples. Experiments using the Banknote Authentication dataset demonstrated the generation of subtle adversarial examples and high detection accuracy, highlighting the robustness of the approach.

We have implemented the above development in an extended version of XReason, named XReason+, which is freely available on-line as a GitHub repository. We validated the combined functionalities of our XReason+ tool using the CICIDS-2017 dataset, demonstrating strong performance in a network security context. The LightGBM model achieved over 90% accuracy and F1 scores, and formal explanations showed 100% consistency with model predictions. Adversarial examples were generated with a misclassification rate of 31.86%, and 61.36% of these were successfully detected, confirming the effectiveness of the proposed methods in addressing model vulnerabilities.

This work presents a comprehensive extension of XReason, combining formal explanations with adversarial robustness techniques. The resulting framework provides a reliable solution for high-stakes applications, ensuring consistent interpretability and enhanced security in AI-driven decision-making.

6.2 Future Directions

In following we discuss several research directions aiming to build upon the foundations established in this work in order to advance even more the field of *Formal XAI*:

- **Support for other ML and Deep Learning Models:** Adding support for other ML models, such as SVM [88] and Deep Learning models, would make the approach more versatile. Deep learning is used in many areas like image processing and language models, where explanations are still challenging. Expanding to these models could help apply the method in more real-world

situations.

- **Exploring Other Solvers:** Investigating the use of different solvers, beyond MaxSAT, could improve the scalability and efficiency of the approach. This would open up new possibilities for tackling more complex problems in decision-making and optimization tasks.

- **Robustness to Concept Drift:** Investigating how formal explanations handle concept drift [89] would be useful. Concept drift occurs when data distributions change over time, which can affect the local formal explanations and class-wise explanation intervals. Exploring the stability of these explanations under evolving data and developing adaptive methods is an important next step.

- **Experimenting with High-Dimensional Datasets:** Exploring the use of datasets with higher dimensions, such as those with 100 or more features, could provide valuable insights into the scalability of the approach. This would allow for an assessment of the model's performance and the potential need for optimization in the encoding process to handle such data efficiently.

- **Contrastive Explanations:** The current explanations focus on identifying the most important features for a prediction, but they do not address *why* a specific outcome occurred instead of another. Contrastive explanations can provide insights into what needs to change for an alternative decision, which is particularly helpful in domains like healthcare or finance, where understanding decision boundaries is critical for actionable insights.

- **Better Adversarial Detection:** Current detection methods could be enhanced by exploring alternative strategies tailored to adversarial scenarios. These improvements could make the detection process more robust and accurate, ensuring better identification of adversarial examples across different datasets and applications.

Bibliography

[1] Mariana Fang. A deep dive into IDC's global AI and generative AI spending . https://blogs.idc.com/2024/08/16/a-deep-dive-into-idcs-global-ai-and-generative-ai-spending/. Accessed: [2024].

[2] Bergur Thormundsson. Artificial intelligence (AI) worldwide - statistics facts. https://www.statista.com/topics/3104/artificial-intelligence-ai-worldwide/#topicOverview. Accessed: [2024].

[3] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[4] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[5] Riccardo Guidotti and Salvatore Ruggieri. On the stability of interpretable models. In *Neural Networks*, pages 1–8. IEEE, 2019.

[6] Zoran Bursac, C Gauss, D Williams, and David Hosmer. A purposeful selection of variables macro for logistic regression. *Source Code Biology and Medicine*, 3:173, 2007.

[7] Joanne Peng, Kuk Lee, and Gary Ingersoll. An introduction to logistic regression analysis and reporting. *Journal of Educational Research*, 96:3–14, 2002.

[8] John Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[9] Steven M. Rovnyak, Stein Kretsinger, James Thorp, and Donald Brown. Decision trees for real-time transient stability prediction. *IEEE Transactions on Power Systems*, 9(3):1417–1426, 1994.

[10] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[11] Ulf Johansson, Rikard König, and Lars Niklasson. The truth is in there - Rule extraction from opaque models using genetic programming. In *International Florida Artificial Intelligence Research Society Conference*, volume 2, pages 658––663. AAAI press, 2004.

[12] John Ross Quinlan. Generating production rules from decision trees. In *International Joint Conference on Artificial Intelligence*, pages 304–307. Morgan Kaufmann Publishers Inc., 1987.

[13] Zachary C Lipton. The mythos of model interpretability: In Machine Learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[14] Nahla Barakat and Andrew Bradley. Rule extraction from support vector machines: A sequential covering approach. *IEEE Transactions on Knowledge and Data Engineering*, 19:729–741, 2007.

[15] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Machine Learning*, volume 70, pages 3319 – 3328, 2017.

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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.

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

[18] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24:44–65, 2013.

[19] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law Technology*, 31:841–887, 2017.

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Conference on Artificial Intelligence*, volume 32, pages 1527–1535. AAAI press, 2018.

[21] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189 – 1232, 2001.

[22] Andreas Henelius, Kai Puolamäki, Henrik Boström, Lars Asker, and Panagiotis Papapetrou. A peek into the black box: exploring classifiers by randomization. *Data Mining and Knowledge Discovery*, 28(5–6):1503–1529, 2014.

[23] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 82(4):1059 – 1086, 2020.

[24] Vida Panitch. Global surrogacy: Exploitation to empowerment. In *Gender Justice and Development: Vulnerability and Empowerment*, pages 97–112, 2017.

[25] Dingmar van Eck, Daniel A McAdams, and Pieter E Vermaas. Functional decomposition in engineering: a survey. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 48043, pages 227–236, 2007.

[26] Philip Adler, Casey Falk, Sorelle A. Friedler, Tionney Nix, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. Auditing black-box models for indirect influence. *Knowledge and Information Systems*, 54(1):95 – 122, 2018.

[27] Gerard O'Regan. *Concise guide to formal methods*. Springer, 2017.

[28] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete . *Information Processing Letters*, 5(1):15–17, 1976.

[29] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On tackling explanation redundancy in decision trees (extended abstract). In *International Joint Conference on Artificial Intelligence*, pages 6900 – 6904. ACM, 2023.

[30] Joao Marques-Silva and Xuanxiang Huang. Explainability Is Not a Game. *Communications of the ACM*, 67(7):66–75, 2024.

[31] Joao Marques-Silva. Logic-Based Explainability in Machine Learning. In *Reasoning Web. Causality, Explanations and Declarative Knowledge*, volume 13759 of *LNCS*, pages 24–104. Springer, 2023.

[32] Joao Marques-Silva and Alexey Ignatiev. No silver bullet: interpretable ML models must be explained. *Frontiers in Artificial Intelligence*, 6:1–15, 2023.

[33] Armin Biere, Marjin Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2021.

[34] Leonardo De Moura and Nikolaj 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.

[35] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for Machine Learning models. In *AAAI Symposium on Educational Advances in Artificial Intelligence*, volume 33, pages 1511–1519. AAAI Press, 2019.

[36] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[37] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Conference on Security Symposium*, pages 1623 – 1640. USENIX Association, 2020.

[38] Christian Szegedy. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[39] V Porkodi, Murugan Sivaram, Amin Salih Mohammed, and V Manikandan. Survey on white-box attacks and solutions. *Asian Journal of Computer Science and Technology*, 7(3):28–32, 2018.

[40] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In *International conference on machine learning*, pages 2484–2493. PMLR, 2019.

[41] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.

[42] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv*, abs/1706.06083, 2017.

[43] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: Zeroth Order Optimization based black-box attacks to deep neural networks without training substitute models. In *Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[44] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17:151–178, 2020.

[45] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. TextBugger: Generating Adversarial Text Against Real-world Applications. *CoRR*, abs/1812.05271, 2018.

[46] Tianyu Du, Shouling Ji, Jinfeng Li, Qinchen Gu, Ting Wang, and Raheem Beyah. Sirenattack: Generating adversarial audio for end-to-end acoustic systems. In *Asia Conference on Computer and Communications Security*, pages 357–369. ACM, 2020.

[47] Aditya Kuppa and Nhien-An Le-Khac. Adversarial XAI methods in cybersecurity. *IEEE transactions on information forensics and security*, 16:4924–4938, 2021.

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

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

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

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

[52] Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and Joao Marques-Silva. Using MaxSAT for Efficient Explanations of Tree Ensembles. In *Conference on Artificial Intelligence*, volume 36, pages 3776–3785. AAAI, 2022.

[53] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On validating, repairing and refining heuristic ML explanations. *CoRR*, abs/1907.02509, 2019.

[54] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On formal reasoning about explanations. In *Knowledge Representation and Automated Reasoning*, 2020.

[55] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[56] Gelin Zhang, Zhe Hou, Yanhong Huang, Jianqi Shi, Hadrien Bride, Jin Dong, and Yongsheng Gao. Extracting optimal explanations for ensemble trees via automated reasoning. *Applied Intelligence*, 53:14371–14382, 2022.

[57] Inês Lynce and Joao Marques-Silva. On Computing Minimum Unsatisfiable Cores. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 305–310, 2004.

[58] Shucen Ma, Jianqi Shi, Yanhong Huang, Shengchao Qin, and Zhe Hou. MUC-driven Feature Importance Measurement and Adversarial Analysis for Random Forest. *arXiv preprint arXiv:2202.12512*, 2022.

[59] Gilles Audemard, Jean-Marie Lagniez, Pierre Marquis, and Nicolas Szczepanski. PyXAI: An XAI Library for Tree-Based Models. In *International Joint Conference on Artificial Intelligence*, pages 8601–8605, 2024.

[60] Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[61] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[62] Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and Joao Marques-Silva. From contrastive to abductive explanations and back again. In *Advances in Artificial Intelligence*, volume 12414 of *LNCS*, pages 335–355. Springer, 2020.

[63] XReason+. `https://github.com/hvg-concordia/XReasonP`. [Online; accessed 2024].

[64] Iman Sharafaldin, Arash Habibi Lashkari, and Ali 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.

[65] Matjaz Zwitter and Milan Soklic. Breast Cancer. `https://doi.org/10.24432/C51P4M`, 1988.

[66] Volker Lohweg. Banknote Authentication. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C55P57.

[67] Image Segmentation. UCI Machine Learning Repository, 1990. DOI: https://doi.org/10.24432/C5GP4N.

[68] Kenta Nakai. Ecoli. UCI Machine Learning Repository, 1996. DOI: https://archive.ics.uci.edu/dataset/39/ecoli.

[69] Ronald Aylmer Fisher. Iris. UCI Machine Learning Repository, 1936. DOI: https://doi.org/10.24432/C56C76.

[70] WSL. `https://documentation.ubuntu.com/wsl/en/latest/`. [Online; accessed 2024].

[71] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neuro Robotics*, 7:1–12, 12 2013.

[72] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937–1967, 2021.

[73] Tim Hesterberg, Nam Choi, Lukas Meier, and Chris Fraley. Least Angle and L1 Regression: A Review. *Statistics Surveys*, 2:61–93, 2008.

[74] David MW Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[75] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019.

[76] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global K-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

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

[78] Thi-Thu-Huong Le, Haeyoung Kim, Hyoeun Kang, and Howon Kim. Classification and Explanation for Intrusion Detection System Based on Ensemble Trees and SHAP Method. *Sensors*, 22:1154–1182, 2022.

[79] Shruti Patil, Vijayakumar Varadarajan, Siddiqui Mohd Mazhar, Abdulwodood Sahibzada, Nihal Ahmed, Onkar Sinha, Satish Kumar, Kailash Shaw, and Ketan Kotecha. Explainable Artificial Intelligence for intrusion etection system. *Electronics*, 11(19):3079–3102, 2022.

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

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

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

[83] Jianbo Chen, Michael Jordan, and Martin Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. In *Symposium on Security and Privacy*, pages 1277–1294. IEEE, 2020.

[84] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *Symposium on Security and Privacy*, pages 39–57. IEEE, 2017.

[85] Lisa-Marie Geiginger and Tanja Zseby. Evading Botnet Detection. In *Symposium on Applied Computing*, pages 1331 – 1340. ACM, 2024.

[86] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503 – 528, 1989.

[88] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[89] Gerhard Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–10, 1994.

Biography

Education

- **Concordia University**: Montreal, Quebec, Canada.
  M.A.Sc., Electrical & Computer Engineering (January 2023 - December 2024)
- **École Nationale des Sciences de l'Informatique**: Manouba, Tunisia.
  Engineering Diploma, Computer Science (September 2019 - October 2022)
- **Institut Préparatoire aux Etudes d'Ingénieurs de Tunis**: Tunis, Tunisia.
  (September 2017 - August 2019)

Awards

- Mitacs Globalink Graduate Fellowship, Canada (January 2023-August 2023)
- Special Entrance Award, Concordia University, Canada (January 2023-August 2023)
- Split Merit Scholarship, Concordia University, Canada (January 2023-December 2023)

Work History

- **Research Assistant**, Hardware Verification Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada (2023-2024).

- **Data Scientist Intern**, École de Technologie Supérieure, Montreal, Quebec, Canada (2022).

Publications

Conference Papers

- [**Bio-Cf1**] Amira Jemaa, Adnan Rashid, and Sofiène Tahar. Extending XReason: Formal Explanations for Adversarial Detection. In International Congress on Information and Communication Technology (ICICT), Lecture Notes in Networks and Systems (LNNS). Springer, 2025. To appear.

- [**Bio-Cf2**] Amira Jemaa, Adnan Rashid, and Sofiène Tahar. Leveraging Formal Methods for Efficient Explainable AI. In Women in Formal Methods Workshop, Conference on Intelligent Computer Mathematics, Montreal, Canada, August 2024.

Tools

- [**Bio-T1**] Amira Jemaa, XReason+, GitHub Repository: https://github.com/hvg-concordia/XReasonP