International Journal of Research in Engineering, Science and Management
Volume 9, Issue 1, January 2026
www.ijresm.com | E-ISSN: 2581-5792 | RESAIM Publishing | www.resaim.com

Comparative Analysis of Machine Learning and Deep Learning Algorithms for Early Detection of Hardware Vulnerabilities in Hardware Security Systems

Milind Paraye[1], Abhishek Ratnu[1*], Shubham Sinha[1], Pratik Waghmode[1]
[1]*Department of Electronics and Telecommunication Engineering, Sardar Patel Institute of Technology, Mumbai, India*

***Abstract*: Early detection of hardware vulnerabilities is critical to ensuring the security and reliability of electronic systems. This study investigates and compares various machine learning and deep learning algorithms for their effectiveness in identifying hardware vulnerabilities during the early stages of the development lifecycle. The analysis focuses on algorithms commonly used in hardware security applications, including anomaly detection, classification, and pattern recognition, to detect issues such as hardware Trojans, side-channel attacks, and other security threats. Key performance metrics such as accuracy, detection speed, computational efficiency, and robustness against adversarial scenarios are evaluated. Furthermore, the study highlights the trade-offs between machine learning and deep learning approaches, considering their scalability and deployment feasibility in resource-constrained environments. The findings aim to provide insights into selecting the most suitable algorithm for early vulnerability detection, contributing to the advancement of secure hardware design and the mitigation of potential risks in critical systems.**

***Keywords*: Machine Learning, Deep Learning, Hardware Security, Vulnerability detection, Cybersecurity, Predictive Analysis.**

1. Introduction

The rapid advancement of modern electronic systems has brought about unprecedented functionality and performance but has also introduced significant security challenges. Hardware vulnerabilities, including hardware Trojans, side-channel attacks, and keylogging, pose substantial threats to systems operating in critical domains such as defence, healthcare, and finance. These vulnerabilities, if left undetected, can compromise system integrity, confidentiality, and availability, leading to devastating consequences. As such, the early detection of hardware vulnerabilities during the design and production stages is essential to safeguard electronic systems and ensure their reliability.

Machine learning (ML) and deep learning (DL) techniques have emerged as powerful tools for automating the detection of hardware vulnerabilities. ML models excel at identifying patterns and anomalies in large datasets, while DL approaches, such as Artificial Neural Networks (ANNs), leverage hierarchical representations to detect complex patterns. However, the adoption of these techniques in hardware security presents challenges, such as imbalanced datasets, computational efficiency, and the selection of optimal algorithms for specific vulnerabilities.

In this study, we propose a comprehensive framework for the early detection of hardware vulnerabilities by leveraging both ML and DL techniques. The process begins with data preprocessing, including cleaning and balancing the dataset using the Synthetic Minority Oversampling Technique (SMOTE) to address class imbalance. A wide range of ML models, including basic algorithms like Decision Trees and Support Vector Machines, as well as ensemble methods like Random Forests and Gradient Boosting, are implemented. To capture more intricate patterns and enhance detection accuracy, an ANN is employed, offering deep learning-based insights.

To make the findings accessible and actionable, a user-friendly web application has been developed using ReactJS and Flask. This platform allows users to interact with the results, visualize model performance, and understand the comparative effectiveness of different algorithms in detecting vulnerabilities. Key performance metrics, including accuracy, computational efficiency, and robustness against adversarial conditions, are evaluated to provide a holistic understanding of the strengths and limitations of each approach.

This work aims to bridge the gap between theoretical research and practical application in hardware security. By providing a detailed comparative analysis of ML and DL models, along with a visualization tool, the study offers a pathway for researchers and practitioners to adopt AI-driven methodologies in the secure design of hardware systems.

2. Related Work

Research on the application of machine learning and deep learning for hardware security has grown significantly in recent years. Early work focused primarily on traditional techniques such as rule-based methods and statistical analysis to detect

*Corresponding author: ratnu.abhishek200@gmail.com

hardware vulnerabilities. However, these approaches often struggled with the complexity and scale of modern hardware systems. As machine learning gained popularity, several studies began to apply classification algorithms like Support Vector Machines (SVMs) and Decision Trees for detecting hardware Trojans and side-channel attacks. These models showed promising results in identifying patterns in hardware behavior but were limited in their ability to handle complex, high-dimensional data.

Recent advancements have shifted towards ensemble models like Random Forests and Gradient Boosting, which improve detection accuracy by combining multiple weak classifiers into a stronger predictive model. Deep learning methods, particularly Artificial Neural Networks (ANNs), have also been explored for their capacity to learn intricate patterns and representations in large datasets. Notably, studies have demonstrated that deep learning can outperform traditional ML methods in identifying subtle vulnerabilities in hardware designs.

While existing works provide valuable insights, there remains a gap in comprehensively comparing various ML and DL algorithms for hardware vulnerability detection, particularly in terms of practical implementation and performance in real-world scenarios. This study addresses this gap by evaluating multiple algorithms and developing a platform to display the results.

3. Process

A. Dataset Preparation

The dataset used for this study consists of labelled instances representing hardware behaviors, including both vulnerable and non-vulnerable hardware configurations. It includes features such as electrical characteristics, signal patterns, and design parameters, which are crucial for identifying hardware vulnerabilities like Trojans and side-channel attacks. However, the dataset suffers from class imbalance, with a significantly lower number of vulnerable instances compared to nonvulnerable ones. To address this issue, the dataset undergoes a thorough cleaning process, which involves removing any irrelevant or missing data, handling outliers, and standardizing the features to ensure consistency.

To further mitigate the class imbalance, the Synthetic Minority Oversampling Technique (SMOTE) is applied. SMOTE generates synthetic instances of the minority class (vulnerable hardware) by interpolating between existing instances, thereby balancing the dataset. This preprocessing step enhances the performance of machine learning models by providing a more equitable representation of both classes, improving detection accuracy for hardware vulnerabilities.

B. Machine Learning models

This study utilizes a set of fundamental machine learning models for detecting hardware vulnerabilities. These models, each with distinct characteristics and algorithms, are implemented to classify hardware behavior as either vulnerable or non-vulnerable. Below is a detailed description of the models used and the formulas employed.

1) Linear Regression

Although typically used for regression tasks, Linear Regression is also employed here for binary classification through logistic transformation. The model attempts to find a linear relationship between the input features $\mathbf{x}$ and the output y. The prediction is given by:

$$y = \mathbf{w}^T\mathbf{x} + b$$

where $\mathbf{w}$ represents the weights assigned to the features, $\mathbf{x}$ is the feature vector, and b is the bias term. While Linear Regression isn't ideal for classification, it provides a baseline model by predicting continuous values that can be mapped to binary classes using a threshold.

2) Logistic Regression

Logistic Regression is a widely used model for binary classification tasks. It estimates the probability that a

given input $\mathbf{x}$ belongs to a particular class. The model uses a logistic function (sigmoid function) to transform the linear prediction into a probability between 0 and 1:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

where $P(y = 1 \mid \mathbf{x})$ is the probability that the output y is 1 (vulnerable), and $\mathbf{w}^T\mathbf{x}+b$ is the linear combination of input features. The decision rule is to classify an instance as vulnerable if the probability is greater than or equal to 0.5, otherwise as non-vulnerable.

3) Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful classification model that aims to find the optimal hyperplane to separate different classes in a high-dimensional feature space. The decision boundary is determined by maximizing the margin between the closest instances of each class (support vectors). The SVM optimization problem can be formulated as:

$$\text{Minimize} \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y_i(\mathbf{w}^T\mathbf{x}_i+b) \geq 1, \forall i$$

where $\mathbf{w}$ is the weight vector, b is the bias term, y_i is the class label of the i-th sample (+1 or −1), and $\mathbf{x}_i$ is the feature vector of the i-th sample. SVM works well for high-dimensional datasets, making it suitable for detecting complex hardware vulnerabilities.

4) K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) algorithm is a simple, instance-based learning algorithm used for classification. Given a new data point, the class is determined by the majority vote of its K-nearest neighbors in the feature space. The Euclidean distance between two data points $\mathbf{x}$ and $\mathbf{y}$ is computed as:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where n is the number of features. KNN is a nonparametric

method, meaning it makes no assumptions about the underlying data distribution, which can be advantageous when working with complex hardware behavior patterns.

5) Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees and combines their outputs to make predictions. Each tree is built using a random subset of the data and a random subset of features. The final classification is determined by majority voting across all the trees. Random Forest can be expressed as:

$$\hat{y} = \text{MajorityVote}(\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_M(\mathbf{x})\})$$

where $h_m(\mathbf{x})$ is the prediction from the m-th tree, and M is the total number of trees in the forest. Random Forest is particularly effective in handling overfitting and can capture complex, non-linear relationships in the data. It is a robust model well-suited for detecting hardware vulnerabilities.

C. Ensemble learning

Ensemble models are a powerful approach in machine learning where multiple individual models are combined to produce a final prediction. The primary idea behind ensembles is that combining several models can often result in improved accuracy, as it reduces the likelihood of errors that might occur in any single model. This is particularly useful when the individual models have different strengths and weaknesses, making their combination more robust.

One of the most common ensemble methods is bagging, where multiple models (typically decision trees) are trained on different subsets of the data and their predictions are averaged. Bagging works well when individual models are highly variable, as the variance of the ensemble prediction is reduced. The Random Forest algorithm is a well-known example of this technique, where a large number of decision trees are trained on random subsets of the data, and the final prediction is made by aggregating the individual predictions.

Another widely used ensemble method is boosting, where models are trained sequentially. Each subsequent model tries to correct the errors of the previous one, with a higher weight given to misclassified examples. This can significantly improve accuracy, especially in complex tasks. Algorithms like AdaBoost and Gradient Boosting use this technique to achieve higher performance.

Finally, stacking is another ensemble technique that involves training multiple different models and then combining their predictions using a meta-model. The meta-model learns how to best combine the predictions from the base models to maximize accuracy.

In terms of mathematical formulation, let y_i represent the prediction of the i-th model, and y_{ensemble} represent the final ensemble prediction. For bagging, the ensemble prediction is simply the average of all individual predictions:

$$y_{\text{ensemble}} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

For boosting, the ensemble prediction involves weighted sums of predictions from individual models, typically using weights determined by the model's performance on the training data. The ensemble prediction for boosting can be represented as:

$$= \text{X}\alpha i y i \qquad N\, y_{\text{ensemble}}$$
$$i=1$$

where α_i are the weights assigned to each model based on its performance. The goal of these methods is to reduce errors and increase accuracy by combining models that are complementary.

D. Deep Learning

Deep learning is a subset of machine learning that focuses on using neural networks with many layers to model complex patterns in data. Artificial Neural Networks (ANNs) are at the core of deep learning algorithms. ANNs are inspired by the human brain's structure, where layers of neurons are interconnected, each capable of learning specific features from the data. These networks have proven to be highly effective in solving complex tasks such as image recognition, natural language processing, and hardware security, where traditional machine learning models might struggle.

At a basic level, an ANN consists of three types of layers: the input layer, hidden layers, and the output layer. The input layer receives the features from the dataset, the hidden layers process the input data through weighted connections, and the output layer produces the final prediction or classification. The connections between neurons are characterized by weights, which are adjusted during training to minimize the prediction error.

The learning process in ANNs involves the backpropagation algorithm, which updates the weights of the connections in the network based on the error calculated in the output. Backpropagation uses the gradient descent optimization method to iteratively adjust the weights in the direction that minimizes the loss function. The loss function quantifies the difference between the predicted output and the true output, and its minimization is the goal of the learning process. The network's performance improves over time as the weights are adjusted to reduce this error.

A key advantage of deep learning is its ability to automatically learn features from raw data, eliminating the need for manual feature engineering. For example, in image recognition tasks, deeper layers of the network can automatically learn to recognize edges, shapes, textures, and objects, allowing the model to perform complex pattern recognition without human intervention. This ability to extract hierarchical features from the data makes deep learning particularly effective for tasks that require high-level abstraction.

In the context of hardware security, deep learning with ANNs has shown promise in detecting vulnerabilities such as hardware Trojans and side-channel attacks. In these applications, the network can learn to distinguish between

secure and compromised hardware based on features extracted from the data, such as signal patterns or power consumption. The training of the network requires large datasets of labelled examples, where the network learns to classify the data as secure or insecure based on the input features.

While deep learning offers powerful capabilities, it also comes with certain challenges. One of the main challenges is the need for large amounts of labelled data to train deep networks effectively. Additionally, deep networks are computationally expensive to train, requiring significant resources in terms of processing power and memory. Overfitting is another common issue, where the model learns the noise in the training data rather than the underlying patterns, leading to poor generalization to new, unseen data.

Despite these challenges, deep learning with ANNs remains one of the most powerful tools in machine learning, achieving state-of-the-art results in many fields. The continued development of more efficient algorithms, such as convolutional neural networks (CNNs) for image processing or recurrent neural networks (RNNs) for sequence data, has further enhanced the capabilities of deep learning models. As computational power increases and more data becomes available, deep learning is expected to continue transforming a wide range of industries, from healthcare to cybersecurity.

The mathematical formulation of an ANN involves the propagation of inputs through layers using activation functions. Let x represent the input vector, W the weight matrix, and b the bias term. The output of each neuron in a layer is computed as:

$$z = W \cdot x + b$$

The output a of the neuron is then passed through an activation function $f(z)$, which introduces non-linearity:

$$a = f(z)$$

The output from the final layer is compared with the true label, and the loss is computed using a loss function L. The weights are updated to minimize this loss through backpropagation and gradient descent.

E. Website and Flask Integration

Creating a dynamic and interactive website using ReactJS is a popular approach for developing modern web applications. ReactJS, a JavaScript library for building user interfaces, allows developers to create single-page applications (SPAs) with fast rendering and efficient state management. The core idea behind ReactJS is its component-based architecture, where each part of the user interface is built as a reusable and independent component. This modularity makes it easier to manage complex UIs and provides a seamless user experience.

For integrating machine learning models into a ReactJS website, Flask—a lightweight Python web framework—is commonly used as a backend to handle server-side logic. Flask allows easy integration with machine learning models by providing an HTTP interface through which the models can be exposed as RESTful APIs. Once the backend is set up, ReactJS can communicate with Flask using HTTP requests to send data and receive predictions from the machine learning model.

To integrate a machine learning model into a ReactJS website using Flask, the typical process involves several steps: 1. Model Training: First, the machine learning model is trained using a dataset, typically in Python using libraries like Scikitlearn, TensorFlow, or PyTorch. 2. Model Serialization: Once trained, the model is serialized into a format that can be easily loaded in the Flask application, such as a '.pkl' file for Scikitlearn models or a '.h5' file for Keras models. 3. Flask API: A Flask API is created to load the serialized model and define endpoints that accept input data (e.g., JSON) and return the model's predictions as a response. 4. ReactJS Frontend: The ReactJS frontend is designed to collect input from users (e.g., through forms or other interactive elements) and send this input to the Flask API using fetch or Axios. 5. Displaying Results: The frontend receives the response from Flask (which includes the model's predictions) and dynamically updates the UI with the results.

This architecture ensures seamless communication between the frontend and backend, providing an interactive platform for users to interact with machine learning models in real-time.

4. Results

A. Accuracy

Accuracy is a widely used metric for evaluating the performance of machine learning (ML) models. It is defined as the ratio of the number of correct predictions made by the model to the total number of predictions. Mathematically, accuracy can be expressed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

For classification problems, accuracy provides a straightforward measure of how well the model correctly identifies instances across all classes. While useful for balanced datasets, accuracy can be misleading in cases of imbalanced data. For instance, in a dataset where 95% of samples belong to one class, a model predicting only the majority class can achieve 95% accuracy without effectively identifying the minority class.

To address this limitation, additional metrics such as precision, recall, and the F1-score are commonly used. Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is calculated as:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

Recall, also known as sensitivity, measures the proportion of correctly predicted positive instances out of all actual positive instances. It is given by:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is particularly useful for datasets with class imbalances and is calculated as:

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

These metrics offer deeper insights into model performance. For example, high precision indicates fewer false positives, while high recall indicates fewer false negatives. The F1-score provides a single measure to evaluate the trade-off between precision and recall, making it especially effective when false positives and false negatives carry different costs.

MODEL	accuracy	Precision	recall	F1 score
Linear regression	0.57813	0.57425	0.68131	0.6231
KNN	0.9525	0.92783	0.98115	0.9490
Random forest	0.9831	0.97826	0.98901	0.9902
SVM	0.9438	0.9009	1	0.9478
ANN	0.7915	0.7923	0.7954	0.7934

Fig. 1. Visualization of Accuracy, Precision, Recall, and F1-Score

B. Confusion Matrix

A confusion matrix is a performance evaluation tool for classification models. It is a table that summarizes the model's predictions against actual outcomes, providing insights into its accuracy and errors. The matrix consists of four key components: True Positives (TP), correctly predicted positive instances; True Negatives (TN), correctly predicted negative instances; False Positives (FP), incorrect positive predictions (Type I error); and False Negatives (FN), incorrect negative predictions (Type II error). The confusion matrix helps compute critical metrics such as accuracy, precision, recall, and F1-score, enabling a comprehensive analysis of model performance, especially for imbalanced datasets.

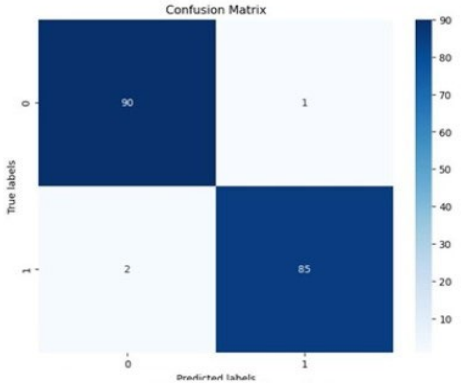

Fig. 2. Confusion matrix for Random Forest

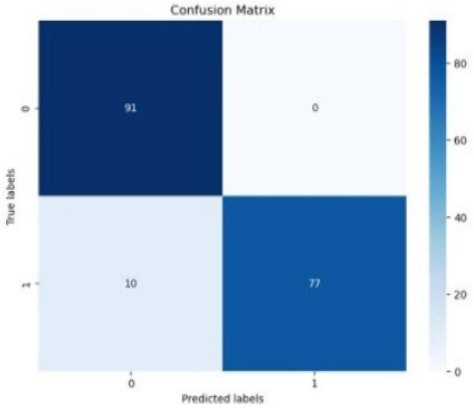

Fig. 3. Confusion matrix for SVM

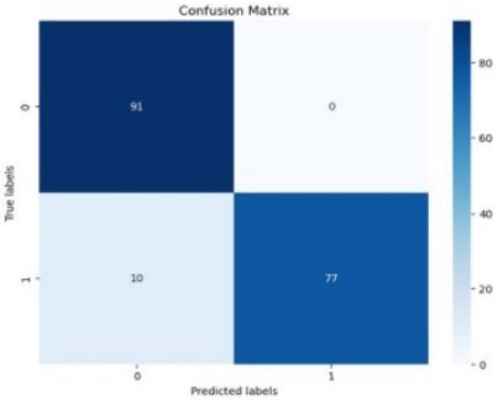

Fig. 4. Confusion matrix for KNN

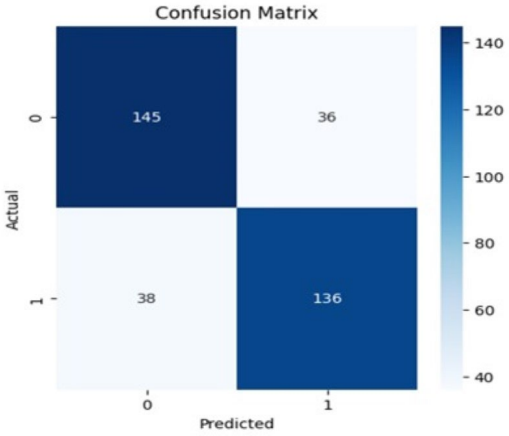

Fig. 5. Confusion matrix for artificial neural network

5. Conclusion

In this paper, we explored the application of various machine learning models, including logistic regression, random forest, support vector machines (SVM), and artificial neural networks (ANN), for detecting hardware vulnerabilities in electronic devices. Our paper highlighted the unique strengths and limitations of each model. Logistic regression served as a baseline with its simplicity and interpretability, while random forest demonstrated robustness and scalability for high-dimensional data. SVM effectively captured non-linear relationships using kernel functions, providing flexibility in modelling complex data patterns.

We observed varying degrees of generalization and adaptability across the models, emphasizing the importance of evaluating performance under diverse conditions, including adversarial scenarios. Practical considerations such as computational efficiency, interpretability, and deployment constraints were also addressed, underscoring the trade-offs between model complexity and real-world applicability.

Our findings contribute valuable insights into hardware vulnerability detection, particularly for Trojan and side-channel attack detection. Future research directions include refining model architectures, applying advanced feature engineering, and integrating anomaly detection techniques to further enhance hardware security. This work lays a foundation for developing resilient and secure electronic systems, helping to mitigate potential hardware threats in an evolving technological landscape.

References

[1] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "LLM for SoC security: A paradigm shift," *IEEE Access*, vol. 12, pp. 155498–155521, 2024.

[2] M. Akyash and H. Mardani Kamali, "Evolutionary large language models for hardware security: A comparative survey," *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, Apr. 2024.

[3] M. Nair, R. Sadhukhan, and D. Mukhopadhyay, "Generating secure hardware using ChatGPT resistant to CWEs," *Cryptology ePrint Archive*, Paper 2023/212, 2023. Available: https://eprint.iacr.org/2023/212.

[4] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "On hardware security bug code fixes by prompting large language models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4043–4057, 2024.

[5] X. Meng, A. Srivastava, A. Arunachalam, A. Ray, P. H. Silva, R. Psiakis, Y. Makris, and K. Basu, "Unlocking hardware security assurance: The potential of LLMs," *arXiv preprint* arXiv:2308.11042, Aug. 2023.

[6] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "(Security) assertions by large language models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4374–4389, 2024.

[7] G. Kokolakis, A. Moschos, and A. D. Keromytis, "Harnessing the power of general-purpose LLMs in hardware Trojan design," in *Applied Cryptography and Network Security Workshops (ACNS 2024)*, Lecture Notes in Computer Science, vol. 14586, Springer, Cham, 2024, pp. 176–194.

[8] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, Z. Xie, and H. Zhang, "AssertLLM: Generating and evaluating hardware verification assertions from design specifications via MultiLLMs," *arXiv preprint* arXiv:2402.00386, Feb. 2024.

[9] B. Mali et al., "ChIRAAG: ChatGPT informed rapid and automated assertion generation," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Knoxville, TN, USA, 2024, pp. 680–683.