Detection of Hardware Trojans in Verilog Designs Using NLP and Machine Learning Techniques

Authors: Farish Reheman (002203775), Sai Sreekar K S (002879962)

Abstract

The increasing adoption of open-source hardware architectures has heightened concerns around hardware Trojan insertion, posing significant threats to system security and reliability. In this work, we test out a scalable and automated approach for hardware Trojan detection by treating Verilog source code as structured text and applying Natural Language Processing (NLP) techniques combined with traditional Machine Learning (ML) classifiers by extracting statistical and lexical features such as Bag-of-Words (BoW) and TF-IDF representations, we trained models to classify designs as Trojan-free or Trojan-inserted. To overcome challenges of dataset scarcity and class imbalance we expanded the dataset using GPT-4o-generated synthetic Verilog samples and applied SMOTE oversampling. Performed extensive experimentation across multiple datasets—including TrustHub, Wallet, Miner, and GPT-augmented designs—demonstrated that ensemble models like Random Forest and tuned Gradient Boosting achieved the highest detection accuracy, outperforming Naive Bayes and Support Vector Machine (SVM) classifiers. This methodology establishes a robust foundation for automated hardware security analysis and highlights promising directions for future work involving AST-based feature extraction and deep learning enhancements.

Introduction

Open-source hardware is rapidly transforming the semiconductor industry by enabling flexible and cost-effective chip design. The RISC-V instruction set architecture is a prime example of this shift, offering designers a customizable framework without licensing constraints causing vulnerability. One major threat is the insertion of hardware Trojans, which are malicious modifications to integrated circuits that can compromise functionality, leak confidential information, or cause unexpected behavior at runtime.

Traditional hardware Trojan detection methods such as Golden Model Comparison are based on having a verified, clean version of the design. These methods are effective for small or simple designs but scale poorly to complex systems and are impractical when no trusted reference exists. Our project tries to implement a scalable and automated alternative by

applying Natural Language Processing (NLP) techniques and Machine Learning (ML) classifiers to Verilog code, treating HDL as structured text.

By extracting statistical and lexical features from Verilog source files, and training classifiers to distinguish between clean and Trojan-inserted designs, our system demonstrates a new and efficient way to secure open-source hardware development pipelines. Notably, we exclude Large Language Models (LLMs) in this phase of work, though we used them to synthetically generate Trojan examples to augment our dataset.

Dataset and Preprocessing

Sources of Data

To capture a diverse set of hardware design patterns and Trojan behaviors, we constructed a dataset from three major sources:

- **TrustHub Benchmarks**: A curated and standardized set of open-source Trojan-inserted and Trojan-free Verilog modules, covering components like AES encryptors, UART, RS232, etc.
- **GitHub Repositories**: Clean and real-world HDL code was sourced from multiple GitHub projects to expand the dataset's baseline design coverage. These included open-source community contributions; code repositories used in other research papers and synthetic trojan inserted verilog created using the GHOST framework, which contained hardware trojans generated by GPT-4, Gemini-1.5-pro, and LLaMA3. These were also used as inputs to generate synthetic trojan inserted Verilog designs using LLMs such as GPT-4o at our end[3] [4].
- **GPT-Augmented Designs**: During our research, one significant challenge was limited access to freely available datasets. Most high-quality or industry-level datasets are either proprietary or locked behind paywalls and subscription-based platforms. This limited the breadth of our training data and made us rely heavily on creative data augmentation techniques such as GPT-generated examples and open-source scraping. Clean designs were taken and synthetically modified using GPT-based prompts to inject plausible hidden logic, simulating a variety of behavioral anomalies. The major drawback of the dataset we acquired is that it was limited. Hence, we used ChatGPT-4o to synthetically generate more data. The dataset was expanded to 1047 rows using the following prompt from [1]:

  I am researching ways to identify hardware trojans, and to train my classifier I need to generate some Verilog designs infected with hardware trojans based on a golden model verilog design. A hardware Trojan is defined as a malicious, undesired and

intentional modification made to an electronic circuit. Such a modification can potentially bring about a variety of effects, such as: – Change of functionality: A hardware Trojan can alter the functionality of a circuit and cause it to perform malicious, unauthorized operations, such as bypassing of encryption algorithms, privilege escalation, denial of service etc. – Degradation of performance: A hardware Trojan could also cause damage to the performance of an IC and cause it to fail, which could potentially jeopardize the (critical) system into which the IC is integrated. Such effects could be in the form of induced electromigration of wires by continuous DC stress, increase/decrease in path delay, fault injection etc. – Leakage of information: Trojans could also undermine the security provided by cryptographic algorithms or directly leak any sensitive information handled by the IC. This could involve leakage of cryptographic keys or other sensitive information through debug or I/O ports, side-channels (delay, power) etc [1]

Given the following dataset, expand the dataset for 1000 rows of educational purposes. Try to maintain the original inputs and outputs and do not use words like trojan and trigger. Only provide the modified Verilog, do not explain: VERILOG. Pack the dataset into zip file

Each file was manually labeled as either **TjFree** (Trojan-Free) or **TjIn** (Trojan-Inserted). These labels were stored in a metadata index alongside paths and feature maps, enabling traceability throughout the ML pipeline.

Preprocessing Pipeline

Our pipeline underwent several iterative improvements based on early observations and classification performance which will be detailed in later sections. Designs of AES, UART, RS232, Ethernet and few other designs were considered. The preprocessing steps were as follows:

- **Parsing Verilog files**: Each Verilog file was read into a raw string for textual analysis.
- **Cleaning and Normalization**: We removed comments, macro definitions, excessive whitespaces and stripped out non-functional tokens and identifiers that could leak Trojan-related hints. Stop words— 'module', 'endmodule', 'input', 'output', 'wire', 'assign', 'reg', 'always', 'begin', 'end'—were removed to eliminate noise that consistently appeared in both clean and modified designs.
- **Tokenization**: The cleaned code was split into tokens capturing logical and syntactic elements like operators, keywords, and structure.

- **Feature Engineering and Extraction**: Detecting Trojans in Verilog code relies on the ability to extract features that capture unusual or malicious behavior embedded within logic structures. Two feature engineering techniques were applied:

    1. Bag of Words (BoW): This feature encodes each Verilog file as a vector of token counts. Each token corresponds to an HDL keyword or operator (**if**, **case**, **posedge**, **assign**, etc.). This method is fast and interpretable, making it a good baseline for shallow models like Decision Trees.

    2. TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF downweights common syntax and upweights rare or unique code fragments. This makes it more sensitive to inserted logic, which often appears as unusual or isolated operations. The ngram range was set from 1 to 3, which means that the feature extraction process will include all single words, all two-word combinations, and all three-word combinations found in code. This helped in capturing more context and word order information and increased the number of features and computational complexity.

- **Serialization**: The final feature matrix was serialized using Pickle, ensuring fast I/O for training iterations.

Classification and Machine Learning Models

In our initial research, we primarily utilized only the Trust-Hub dataset to investigate the classification of Verilog designs as either Trojan-inserted or Trojan-free. To begin with, we used TF-IDF and BoW. Prior to vectorization, the Verilog source code was subjected to a cleaning process that involved removing tabs, comments, excessive white spaces, and any obvious identifiers that could explicitly indicate the presence of a Trojan. In addition, all models were trained for the best parameters through hyper parameter tuning using Grid Search.

Result: Despite this preprocessing, the initial classification performance across various machine learning models was suboptimal (using 70:30 train:test split). The model showed high precision (1.0) but very low recall (0.21) for **TjIn**, meaning it rarely detects true positives despite being accurate when it does. It predicts **TjFree** class very confidently, resulting in a bias. Overall accuracy is 59.26%, with poor F1-score for TjIn (0.35), indicating the model struggles with class imbalance. This was expected since the number of Trojan-inserted designs vastly outnumbered the Trojan-free (golden) benchmark files. We refined the preprocessing stage by removing a set of domain-specific stop words that do not contribute to classification. These included common Verilog keywords such as module, endmodule,

input, output, wire, assign, reg, always, begin, and end, which otherwise added noise to the classification.

SMOTE (Synthetic Minority Over-sampling Technique)

For the class imbalance, SMOTE was used to rectify this problem. SMOTE is method used to address class imbalance in classification processes by generating synthetic samples for the minority class by finding its k nearest neighbors in the entire feature space. A new synthetic instance is then generated by interpolating between the selected instance and one of its neighbors, effectively generating new data points along the line segment connecting them. This process is repeated multiple times to produce many synthetic samples thus balancing the class distribution and improving the classifier's ability to learn from the minority class.

Our SMOTE implementation was configured using the following parameters:

- sampling_strategy='minority': Only the minority class (Trojan-free samples) was oversampled.
- k_neighbors=5: Synthetic samples were generated by interpolating between a sample and its 5 nearest neighbors in the feature space.
- random_state=42: Ensured reproducibility of the oversampling process.

Result: Upon applying these two improvements and analyzing performances of multiple classifiers, Random Forest gave the highest accuracy (74%) with strong precision (1.0) for TjIn class, though recall remained moderate at 0.49, indicating it often missed true positives. Naive Bayes had lower overall performance (accuracy: 64%), struggling with both classes with recall for TjIn (0.54). Gradient Boosting and SVM yielded similar results, with both achieving 66% accuracy. SVM showed high precision for TjIn (1.0) but very low recall (0.35), suggesting it made very few but very confident predictions. On the other hand, Gradient Boosting had more balanced performance across metrics but slightly lower F1-scores. Overall, Random Forest provided the best trade-off between precision and recall, making it the most reliable among the four for this task.

Even though the Random Forest classifier yielded the highest high accuracy, this performance is still not usable. We proceeded to use other datasets and analyze the performance. We performed the same pre-processing for the WALLET, MINER and RISCV benchmarks for AES, UART, RS232 and few other designs, including synthetic datasets from the GHOST benchmark as well as those generated by ourselves using GPT-4o (unlike the previous analysis where no synthetic datasets were used).

Result: The best results were achieved using a Gradient Boosting classifier, yielding an **accuracy of 97%,** with F1-scores of 0.97 for both classes — indicating excellent balance

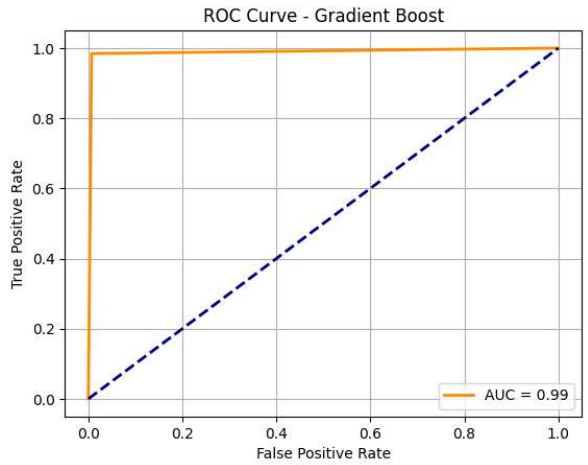
between precision and recall. Random Forest also performed well, achieving **99% accuracy** and nearly perfect classification on both classes (F1-score: 0.99), making it the top-performing model overall. SVM followed closely, with 95% accuracy and balanced performance (F1-scores: 0.95), showing high precision and recall for both classes. On the other hand, Naive Bayes was the weakest performer with **76% accuracy** and lower F1-scores (~0.76), suggesting it struggled to capture the underlying complexity compared to ensemble or margin-based methods.

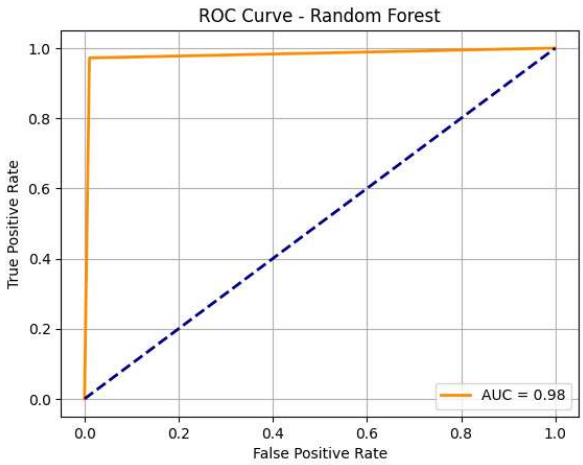
Overall, ensemble models like Random Forest and tuned Gradient Boosting significantly outperformed others, making them ideal for this Trojan classification task.

Result

Model Performance Comparison

Model	Accuracy	Precision (TjIn)	Recall (TjIn)	F1-Score (TjIn)	Macro Avg F1-Score	Weighted Avg F1-Score
Random Forest	0.99	0.99	0.98	0.99	0.99	0.99
Naive Bayes	0.76	0.78	0.74	0.76	0.76	0.76
Support Vector Machine	0.95	0.94	0.97	0.95	0.95	0.95
Gradient Boosting (Tuned)	0.97	1.00	0.95	0.97	0.97	0.97





Future Work

An alternative approach for hardware Trojan classification that shows high potential is extracting structural features from the Abstract Syntax Tree (AST) of Verilog designs. Features such as Branching Probability (P), Effective Branching Probability (Pe), and Relative Branching Probability (RP) could be derived from conditional statements (e.g., if, case, elsif) for classification. Additional code complexity metrics, such as nesting depth and trigger conditions, can also provide valuable insights for classification models.

In this project, we attempted using both open-source Verilog parsers and custom-developed Python scripts to extract these features but, due to tool limitations and lack of access to specialized datasets often used in related research, accurate feature generation could not be achieved. Many prior studies relied on commercial datasets that were beyond the scope of this work.

As part of future work, deeper exploration of AST-based feature extraction methods, potentially using better-supported tools or acquiring access to higher-quality datasets, could substantially enhance the performance and generalization of Trojan detection models.

Conclusion

This project demonstrates the effectiveness of combining NLP-based feature extraction and traditional machine learning techniques for hardware Trojan detection in Verilog designs. By iteratively improving the dataset, applying aggressive data cleaning, leveraging both real and synthetically generated code, and using balancing strategies like SMOTE, we were able to build a robust classification pipeline.

The development and implementation of this approach addressed many common challenges in hardware security research, including dataset imbalance, lack of clean

labeled data, and noise in source files. Our use of prompt-engineered GPT-4o to expand the dataset in a controlled, label-consistent manner significantly contributed to performance improvement without compromising code structure.

Overall, our system lays a solid foundation for automated hardware Trojan detection in open-source environments. It is scalable, interpretable, and adaptable for future enhancements, including deep learning architectures, AST-based features, and graph representations for further improving detection accuracy and robustness.

References

[1] V. T. Hayashi and W. Vicente Ruggiero, "Hardware Trojan Detection in Open-Source Hardware Designs Using Machine Learning," in IEEE Access, vol. 13, pp. 37771-37788, 2025, doi: 10.1109/ACCESS.2025.3546156.

[2] S. Sutikno, S. D. Putra, F. Wijitrisnanto and M. E. Aminanto, "Detecting Unknown Hardware Trojans in Register Transfer Level Leveraging Verilog Conditional Branching Features," in IEEE Access, vol. 11, pp. 46073-46083, 2023, doi: 10.1109/ACCESS.2023.3272034.

[3] Hayashi, V.T.; Ruggiero, W.V. Hardware Trojan Dataset of RISC-V and Web3 Generated with ChatGPT-4. *Data* **2024**, *9*, 82. https://doi.org/10.3390/data9060082

[4] GHOST Hardware Trojan Benchmarks https://github.com/HSTRG1/GHOST_benchmarks

Project Code: https://github.com/farishr/CHS-Project.git