UNIVERSITY OF CALIFORNIA,
IRVINE

Graph Neural Network for Integrated Circuits and Cyber-Physical Systems Security

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Rozhin Yasaei

Dissertation Committee:
Professor Mohammad Abdullah Al Faruque, Chair
Professor Fadi Kurdahi
Assistant Professor Zhou Li

2023

DEDICATION

I dedicate this to my incomparable parents, Mino and Sasan. Their endless outpouring of love, unwavering support, and constant words of encouragement have fostered in me such resilience and determination that have been invaluable during my journey. Their lessons of tenacity continue to reverberate, shaping my perspective.

In addition, my heartfelt gratitude extends to my brother, Farbod. His steadfast presence, undying support, and unwavering faith in me have fortified my resolve, instilling in me the strength and confidence needed to overcome the most daunting of obstacles.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor and committee chair, Professor Mohammad Abdullah Al Faruque, for all of his help and valuable insights through my research. I would also like to express my deepest appreciation to my committee members, Professor Fadi Kurdahi and Professor Zhou Li, for dedicating their valuable time to review my work and provide insightful comments.

I would like to extend my sincere thanks to Dr. Sina Faezi for inspiring me to find my path in research. Without his irreplaceable guidance, ingenious suggestions, and profound belief in my abilities, I would not have been able to complete this dissertation today.

I would like to thank my friends, lab mates, and colleagues in the Autonomous and Intelligent Cyber-Physical Systems (AICPS) laboratory for the cherished time spent together in the lab and in social settings. I would like to extend my gratitude to my collaborators, Prof. Sitao Huang, Dr. Sina Faezi, Shih-Yuan Yu, Yasamin Moghaddas, Luke Chen, Anomadarshi Barua, Yicheng Zhang, Felix Hernandez, Qingrong Zhou, Tommy Nguyen, and Emad Kasaeyan Naeini for all the research discussions and contributions we had throughout my research endeavor.

I would like to thank UCI's Department of Electrical Engineering and Computer Science for their support that allowed me to thrive on this beautiful path. I would like to thank the Office of Naval Research (ONR) for the award N00014-17-1-2499 for supporting my research on hardware Trojan detection. Any opinions, findings, conclusions, or recommendations expressed in this Dissertation are those of the author and do not necessarily reflect the funding agencies' views. I thank IEEE and ACM for the permission to include the content of my dissertation, which portion of it was originally published in transactions and conference papers.

VITA

Rozhin Yasaei

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2023**
University of California Irvine (UCI) *Irvine, CA, USA*

Master of Science in Computer Engineering **2021**
University of California Irvine (UCI) *Irvine, CA, USA*

Bachelor of Science in Electrical Engineering **2018**
Sharif University of Technology *Tehran, Iran*

RESEARCH EXPERIENCE

Graduate Researcher **2018–2023**
University of California Irvine (UCI) *Irvine, CA, USA*

Researcher **2022**
NASA Frontier Development Lab (FDL), SETI *Mountain View, CA, USA*

TEACHING EXPERIENCE

Pedagogical Fellow **2021-2023**
University of California Irvine *Irvine, CA, USA*

Teaching Assistant and Lecturer **2019-2023**
University of California Irvine *Irvine, CA, USA*

Teaching Assistant **2015–2018**
Sharif University of Technology *Tehran, Iran*

Teacher **2015-2017**
Farzanegan High School *Tehran, Iran*

REFEREED JOURNAL PUBLICATIONS

MuFAD: Multi-Modal Data Fusion for Anomaly Detection in the Internet of Things through Graph Learning **2023**
IEEE Internet of Things Journal

Hardware Trojan Detection using Graph Neural Networks **2022**
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Golden Reference-Free Hardware Trojan Localization using Graph Convolutional Network **2022**
IEEE Transactions on Very Large Scale Integration (VLSI) Systems

Brain-Inspired Golden Chip Free Hardware Trojan Detection **2021**
IEEE Transactions on Information Forensics and Security

Stealing Neural Network Structure through Remote FPGA Side-channel Analysis **2021**
IEEE Transactions on Information Forensics and Security

REFEREED CONFERENCE PUBLICATIONS

HW2VEC: A Graph Learning Tool for Automating Hardware Security **2021**
IEEE International Symposium on Hardware Oriented Security and Trust (HOST'21)

GNN4IP: Graph Neural Network for Hardware Intellectual Property Piracy Detection **2021**
Design automation Conference (DAC'21)

GNN4TJ: Graph Neural Networks for Hardware Trojan Detection at Register Transfer Level **2021**
IEEE/ACM Design Automation and Test in Europe Conference (DATE'21)

HTnet: Transfer Learning for Golden Chip-Free Hardware Trojan Detection **2021**
IEEE/ACM Design Automation and Test in Europe Conference (DATE'21)

IoT-CAD: Context-Aware Adaptive Anomaly Detection in IoT Systems Through Sensor Association **2020**
IEEE/ACM International Conference On Computer Aided Design (ICCAD)

ABSTRACT OF THE DISSERTATION

Graph Neural Network for Integrated Circuits and Cyber-Physical Systems Security

by

Rozhin Yasaei

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2023

Professor Mohammad Abdullah Al Faruque, Chair

This Ph.D. dissertation presents a comprehensive investigation into addressing security and reliability challenges in embedded and Cyber-Physical Systems (CPS). Our research leverages advanced machine learning techniques such as Graph Neural Networks (GNN) to develop novel methodologies for cross-layer security analysis.

This dissertation addresses the growing risk posed by the globalization of the Integrated Circuit (IC) supply chain, whereby the majority of the design, fabrication, and testing processes have been outsourced to untrusted third-party entities across the globe. This development has significantly increased the threat of malicious modifications, known as Hardware Trojans (HTs), being inserted into Third-Party Intellectual Property (3PIP). HTs pose a substantial risk to IC integrity, functionality, and performance. Despite numerous HT detection methods proposed in existing literature, most limitations include reliance on a golden reference circuit, lack of generalizability, limited detection scope, low localization resolution, and manual feature extraction and property definition. Furthermore, the equally important task of HT localization has been neglected. This research proposes an innovative, golden reference-free method for HT detection and localization at the pre-silicon stage of IC development, employing models based on GNN. The circuit design is converted into a graph that is an intrinsic data structure for hardware design and captures the computational structure and

data dependencies. We develop a graph classification model to distinguish between HT-free and circuits infected with known or even unknown HTs. To push the boundaries further, we extract node attributes from the HDL code and devise a Graph Convolutional Network (GCN) that facilitates automatic feature extraction, enabling the classification of nodes as either Trojan or benign. This methodology offers an automated approach to HT detection and localization, relieving designers of the need for time-consuming manual code review. The developed method achieves exceptional performance in detecting HT-infected circuits and locating the HT. The approach outlined in this dissertation sets a new benchmark for HT detection and localization, offering a scalable, efficient, and highly accurate tool for securing the pre-silicon IC supply chain.

This dissertation expands to encompass the challenges facing IP piracy. The productivity gap, coupled with time-to-market pressure, has led to increased interest in hardware Intellectual Property (IP) core design within the semiconductor industry, dramatically reducing design and verification costs. Recognizing these challenges, this dissertation proposes a novel IP piracy detection methodology, modeling circuits and assessing similarity between IP designs. Contrary to traditional methods that embed a signature within the circuit design, our method does not introduce additional hardware overhead, nor is it vulnerable to removal, masking, or forging attacks. This approach effectively exposes IP infringements, even when the original IP is complicated by the adversary to deceive the IP owner. To represent the circuit accurately for modeling, we translate the hardware design into a data-flow graph due to similar data types and properties and subsequently model it using state-of-the-art graph learning methods. This approach effectively complements the GNN-based techniques proposed earlier in this dissertation, presenting a robust and comprehensive suite of solutions for security and reliability challenges in the semiconductor industry.

Moving to the CPS domain, the dissertation addresses security challenges in IoT systems through the development of adaptive anomaly detection methods. The first proposed ap-

proach utilizes IoT sensor data and fog computing to ensure data integrity and detect anomalous incidents. The proposed methodology incorporates our sensor association algorithm, LSTM neural networks, and Gaussian estimation for real-time anomaly detection. The dissertation further extends the research to multi-modal data fusion, where the integration of sensor and communication data using GNN enables improved anomaly detection, source identification, and recovery in IoT systems.

Overall, this dissertation showcases the application of advanced techniques such as GNN and machine learning in enhancing security and reliability in hardware design and IoT systems. The proposed methodologies for anomaly detection, hardware Trojan detection, IP piracy detection, and cross-layer security analysis contribute to advancing the state-of-the-art in ensuring the integrity and security of critical systems in the digital era.

Chapter 1

Introduction

The 21st century is an era of elegantly engineered components, devices, and systems that can communicate across different platforms to provide information on demand and make necessary decisions. The ability to connect real-world applications with modeling techniques is crucial to successful research and a critical need in today's world. This includes abstracting a problem or system to its essence, pinpointing pieces of collectible data that capture the system's essence, and devising effective data-driven models to learn and interpret the data. Reliability and security are currently being explored from a single-layer perspective in the literature. However, a holistic model of a system to analyze cross-layer security is still a challenging necessity, which this dissertation intends to address.

Cyber-physical systems are conventionally studied to be mathematically defined in the control systems domain. Such approaches construct definitive or state-based models that overlook the information hidden in various data generated by the system. For example, human interaction with the system is an essential factor that is hard to define mathematically. Therefore, we intend to leverage the abundance of data in the modern world to convert theoretical models to statistical models that embed our prior knowledge of the system as well

as the information derived from data analysis.

Constructing a digital model of a real-world system is challenging because modern systems are quite complex; They comprise hardware, software, and communication layers that often interact with the physical world and generate multi-modal data such as design and architecture, sensor recording, communication network data, side-channel emissions, human-computer interaction, etc. In addition to generated data, the mutual information and correlation patterns shared among various components of the system, known as context, are shown to be very valuable in advanced security and reliability analysis [174, 109, 110, 182, 118]. Integrating multi-modal data and extracting the context are keys to a holistic model, leading to new knowledge discovery levels, and can be leveraged to analyze and enhance security, reliability, and even performance. We pursue this vision in this dissertation and develop a strategy to apply machine learning to hardware and system for cross-layer security and reliability.

To follow this strategy, various challenges emerge, but advances in data fusion and learning models support the vision of developing such multi-modal models. We utilize Graph Neural Networks (GNN) to learn and interpret knowledge graphs, a data structure that can embody multi-modal Euclidean and non-Euclidean data. For example, GNN is shown superior performance when the interconnection between system components is dense. When the data generated in each component is complex, Convolutional Neural Networks (CNN) is a suitable choice for automated feature extraction and learning. Understanding textual data, especially human-computer interaction, would require different techniques known as natural language processing. Adopting mathematical knowledge of the system in the statistical model is another challenge that requires fundamental adjustment in the machine learning model.

Moreover, many apparently different problems, in fact, share some common characteristics. Understanding these inherent characteristics enables us to tackle deeper problems and de-

velop better solutions. Building upon this idea, machine learning introduces transfer learning that studies the ability to apply a model trained for a system to a different but related target system. Transfer learning can be another influential strategy in modeling complex systems. Figure1.1 illustrates the integration of various data sources to create a model of a system such as hardware, automotive vehicle, Internet of Things (IoT), etc.

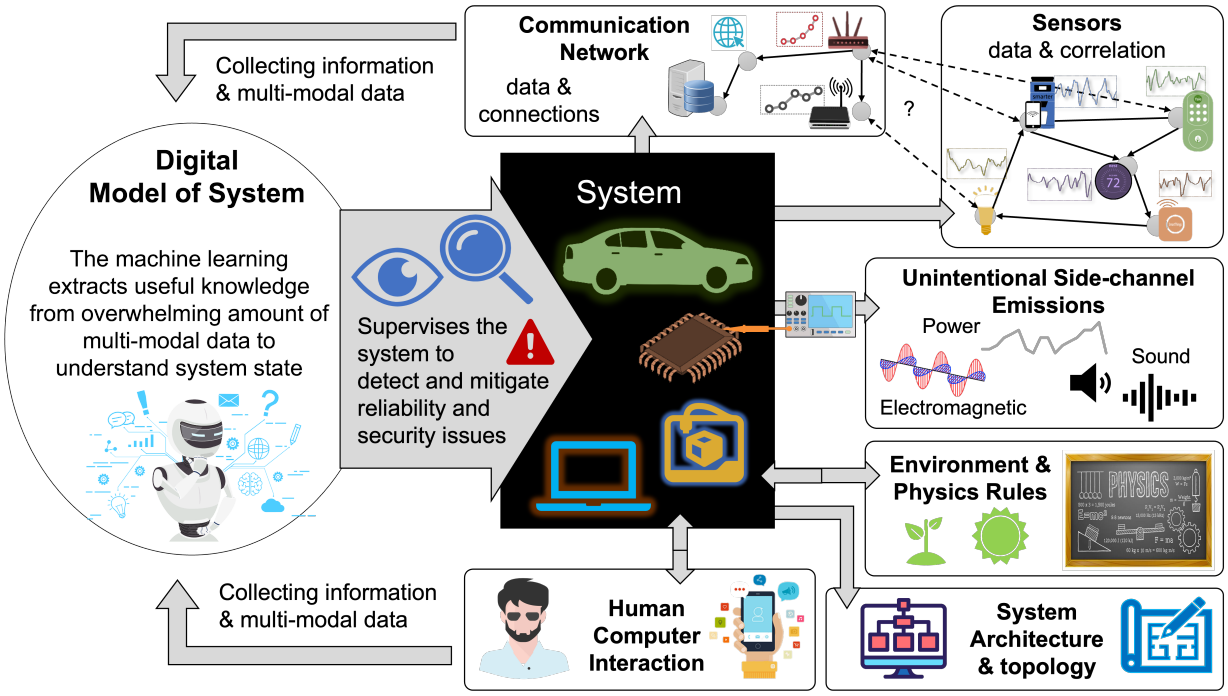


Figure 1.1: The research vision overview; integrating various sources of information about the system and extracting practical knowledge from them to understand and secure the system.

A salient attribute of digital systems is their hierarchical architecture, wherein high-level systems are constructed upon foundational layers. In the context of computing systems security, this hierarchical structure is vital as the security and reliability of the high-level design are contingent upon the trustworthiness of the underlying layers. Figure 1.2 exemplifies the fact that cybersecurity cannot be effectively achieved without ensuring security at the hardware platform upon which it operates.

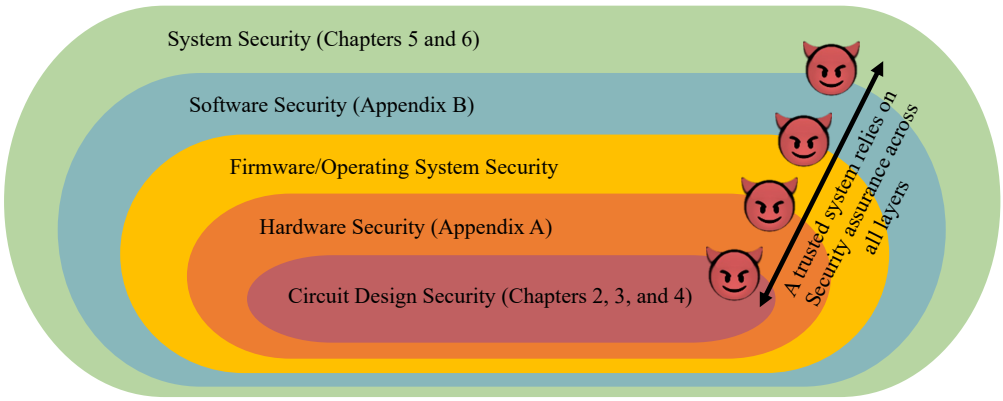


Figure 1.2: The hierarchical structure of a system from the hardware design, fabricated IC, firmware/operating system, up to the software.

1.1 Related Works and Research Challenges

Numerous approaches have been proposed to mitigate the HT and IP piracy threats in the literature. However, they suffer from several shortcomings, as explained further. Currently, there is no universal method and design automation tool for detecting all types of HTs. Thus, a new scalable method and the associated tool are required for unknown HT detection that is expandable as new HTs are introduced. The existing pre-silicon HT detection solutions fall into five main categories [172, 173]; (i) test pattern generation, (ii) formal verification, (iii) code analysis, (iv) machine learning, and (v) graph similarity techniques. These methods propose new ideas but have several shortcomings; reliance on golden-reference, unable to identify unknown HTs, burdening the designer with a manual review of code, unable to guarantee HT detection, limited detection scope to some specific type of HTs, not being scalable, or too complex.

Conventionally, the IP protection techniques fall into preventive (i.e., logic encryption, camouflaging, metering, and split manufacturing) and detective (i.e., digital signature) methods. All these methods add excessive implementation overhead to the hardware design that limits their applications in practice. Moreover, they mostly focus on security at the IC level, while many commercial IPs comprise the soft IPs due to flexibility, independence of plat-

form technology, portability, and easy integration with other components. The high level of abstraction makes IP protection more challenging since it is easier for an adversary to slightly change the source code and redistribute it illegally at the lower levels of abstraction. Although the existing preventive countermeasures deter IP theft, they cannot guarantee IP security as the adversaries keep developing more sophisticated attacks to bypass them. Therefore, an effective IP piracy detection tool is crucial for IP providers to disclose the theft. To this end, the state-of-the-art piracy detection method embeds signatures of IP owners known as watermark and legal IP user known as a fingerprint, in the circuit design to assure authorship and trace legal/illegal IP usage. IP watermarking and fingerprinting are prone to removal, masking, or forging attacks that attempt to omit the watermark, distort its extraction process, or embed another watermark in IP.

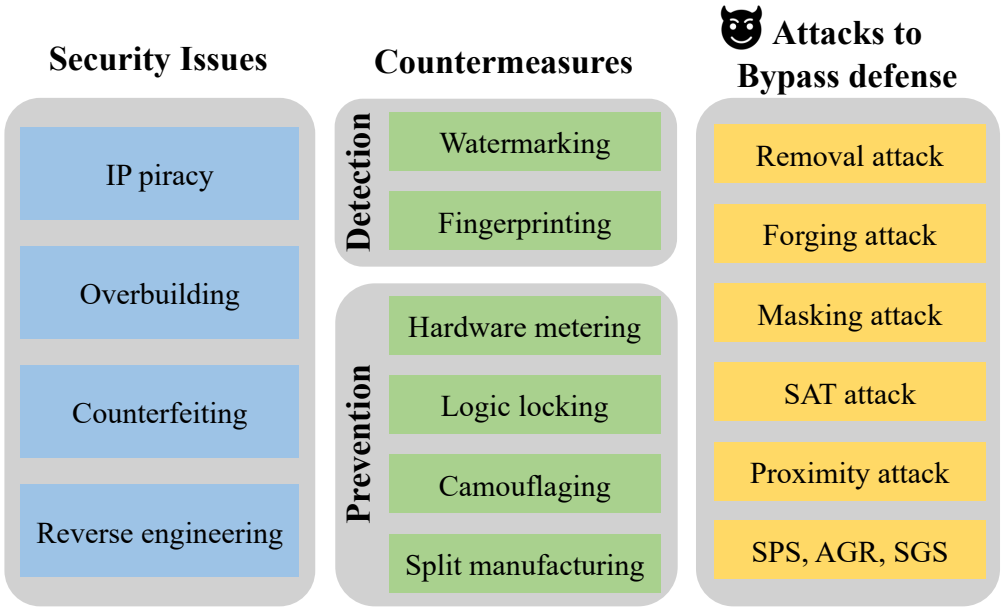


Figure 1.3: IC security issues and countermeasure besides Hardware Trojan.

1.2 Dissertation Contributions

1.2.1 Hardware Security

The time-to-market pressure and continuous growing complexity of hardware designs have promoted the globalization of the integrated circuit supply chain and pushed most of the design, fabrication, and testing process from a single trusted entity to various untrusted third-party entities worldwide. Outsourcing has made the semiconductor industry vulnerable to security threats such as hardware Intellectual Property (IP) piracy or hardware Trojans (HT). Hardware Trojan Detection. HT is a minimal malicious circuit inserted by an adversary in an IC to leak sensitive information, change functionality, degrade performance, or deny the service of the chip. It remains hidden and inactive to avoid detection during testing and only gets triggered under rare circumstances in run-time. HT is known to be the underlying reason for several severe system failures. Thus, HT is one of the major hardware security concerns, and we have worked on approaches to tackle this security threat in the design stage (pre-silicon) and after chip fabrication (post-silicon). To ensure the trustworthiness of IC design, it is essential to ascertain the authenticity of in-house and third-party IPs. Detection of HT in the early stages of the design flow is beneficial because removing it would be very expensive later. The existing HT detection solutions suffer from limitations; reliance on trusted golden reference (unavailable in practice), unable to identify unknown HTs, burdening the designer with a manual review of hardware, unable to guarantee HT detection, and limited detection scope to some specific types of HTs, not scalable, or too complex. To overcome these challenges, we proposed the first methodology that encodes IC design as a graph and models it with GNN [176] (Chapter 2). A circuit is non-Euclidean data that shares similar characteristics with the graph data structure. Thus, based on data flow, we generate a heterogenous graph representation of hardware design register transfer level code or netlist [172] (Chapter 2). We utilized graph data to develop a GNN model that

learns circuits and Trojans' behavior and key features and distinguishes them through graph classification. This research resulted in a golden reference-free end-to-end fully automated security defense that demonstrated high performance in detecting known and unknown HTs in the pre-silicon stage. In the subsequent research [173] (Chapter 3), we pushed the limits even further to locate the Trojan in the victim circuit, which is currently a challenge, and the limited number of proposed methods have low accuracy and require an exhaustive manual search. We developed a node classification GNN model that focuses on the graph nodes and connection to perform node classification and determine the Trojan nodes. Later, an automated algorithm maps the malicious nodes to their counterpart in the circuit. The evaluation demonstrated it could locate all the Trojans with very high accuracy, fully automated. We further investigated the HT detection in a manufactured IC as a second layer of defense against overseas untrusted rouge foundries. In the post-silicon stage, the internal design of the IC is hardly accessible without the destruction of the chip. Therefore, we studied hardware side-channel power and Electromagnetic emissions as a non-invasive method to detect the presence of HTs. Common machine learning classification methods require a trusted golden chip which is not available in practice. Therefore, we collected electromagnetic and power side-channel signals for a library of known HTs and developed a convolutional neural network model on a library of Trojans to learn the best discriminative features for HT detection. Then, we devised a neural network architecture to perform transfer learning and apply the learned knowledge from the library to the circuit under test. Eventually, the learning outcome is passed to an anomaly detection mechanism that monitors the chip and reports malicious activities upon HT activation in run-time [50] (Appendix A). Later, we proposed a brain-inspired HT detection architecture using Hierarchical Temporal Memory. Similar to the human brain, this solution is resilient against natural changes that might happen in the side-channel measurements while accurately detecting the chip's abnormal behavior when the HT gets activated. We used a self-referencing method for HT detection, eliminating the golden chip's need [51] (Appendix A). IP Piracy Detection. The superiority of GNN

led to leveraging the GNN-based approach for IP piracy, a serious security issue, especially in IP-intensive economies such as the US. To increase productivity under time-to-market pressure, IP core design has grabbed substantial attention from the semiconductor industry and has dramatically reduced the design and verification cost. However, the globalization of the IC supply chain poses a high risk of theft for design companies that share their most valuable assets, IPs, with other entities. We proposed a novel methodology for IP piracy detection that models the circuits and assesses the similarity between IP designs instead of the conventional techniques of inserting and extracting a signature to prove the ownership. This method does not add any hardware overhead as a signature and is not vulnerable to removal, masking, or forging attacks. It also effectively exposes the infringement between two IPs even if the adversary complicates the original IP through obfuscation techniques to deceive the IP owner. We convert each circuit to a data-flow graph with attribute vectors assigned to each node. Then, a GNN-based model is constructed on labeled pair of similar and dissimilar circuit graphs to learn the critical properties that define circuits, evaluate similarity on a scale of -1 to 1, and determines IP piracy. The results indicated that this method could detect even partial similarity with high performance [175] (Chapter 4). Another expensive IP of tech companies is the machine learning models, often developed with expensive computation and private datasets. Many companies use public server services such as AWS, which makes them vulnerable to IP piracy. We explored machine learning security on FPGA-based servers, and due to our cross-later security perspective, we leveraged software/hardware co-design to construct an attack model. It resulted in a remote side-channel attack that can steal the machine model configuration running on multi-tenant FPGA servers and revealed the vulnerability of these systems [190] (Appendix B). The experiences with the GNN model have shown its high potential for modeling hardware and system, and the methodology to leverage it led to a patent. To facilitate this multi-disciplinary research, we created an open-source library of GNN models called HW2VEC [183] and a few datasets for the public. We later employed GNN for hardware design automation as well. Figure 1.4 rep-

resents the big picture of the methodologies we developed based on GNN for understanding and securing complex hardware designs.

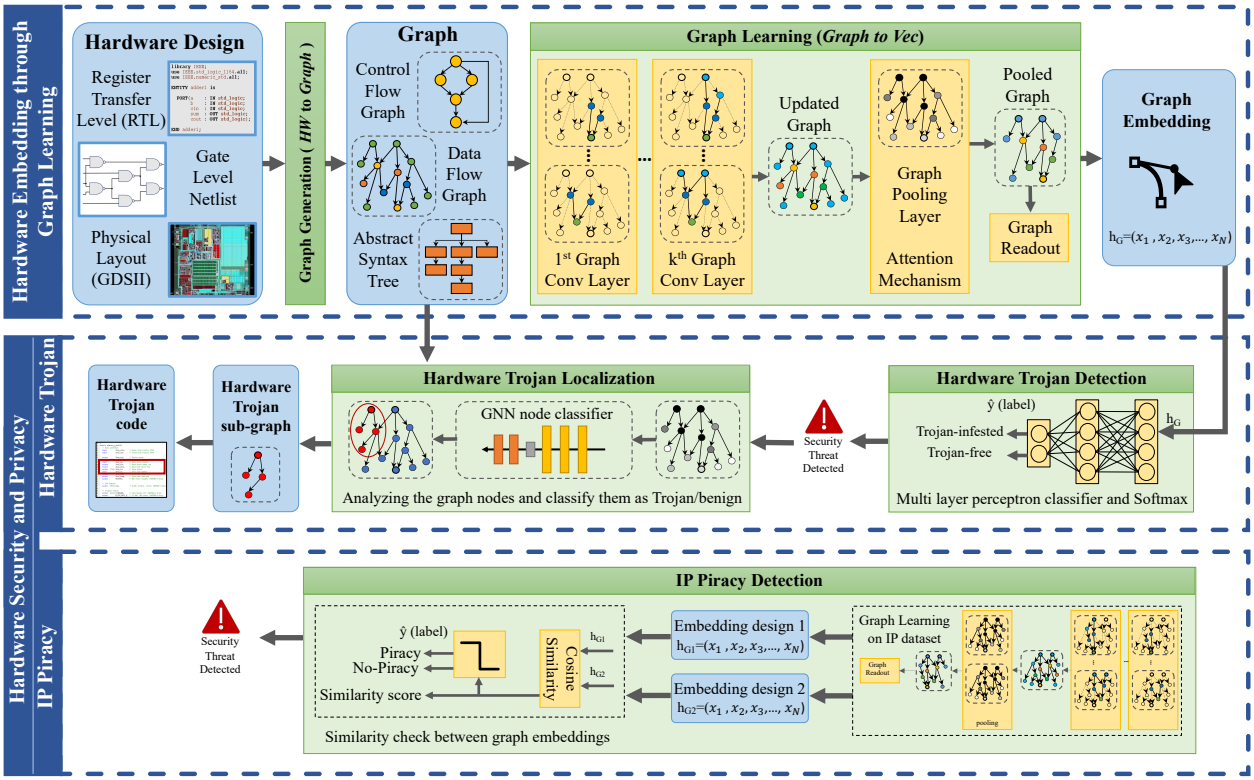


Figure 1.4: The overview of hardware security methodologies developed in this dissertation.

1.2.2 Embedded and Cyber-Physical Systems Security

In today's world, we are surrounded by interconnected networks of intelligent devices called the Internet of Things (IoT) that monitor and control our home, health [133, 135, 134], vital infrastructure [34, 157, 158, 17], factory planes [108, 181, 32, 31, 180, 47], etc. The widespread presence of IoT systems and their critical applications emphasize the importance of their security and integrity. IoT devices are vulnerable to attacks and failures due to low computational resources, cost constraints, and tight time-to-market. It is challenging to mitigate these attacks and failures because of the multi-disciplinary nature of IoT, which brings together the physical domain through sensor interaction and the cyber domain through

the communication network and the cloud.

We started investigating IoT systems security from a bottom-up perspective. Our approach is motivated by the observation that strongly correlated patterns might be found in sensor data, and the discovery of such coherent clusters of sensors is essential in revealing abnormality and attack toward the system from different points of view. We proposed an adaptive context-aware anomaly detection method [174] that captures the system's physical properties to ensure the integrity of IoT sensor data and identify anomalous incidents. In this approach, we devised a novel sensor association algorithm that generates fingerprints of sensors, clusters them, and extracts the context of the system. Based on the contextual information, a predictor model, which comprises a long short-term memory neural network and Gaussian estimator, detects anomalies. Then, a consensus algorithm identifies if the anomaly is an environmental incident or a security and reliability issue. This model wholly or partially updates itself to adapt to the variation in the environment and changes in IoT system architecture. The experiments showed that context awareness and adaptability are crucial features in modeling cyber-physical systems, which enhance the performance and robustness of the anomaly detection model (Chapter 5).

Although stand-alone approaches are proposed in the literature for network intrusion detection or sensor anomaly detection, a holistic model needs to be included to integrate the information from both domains and extract the valuable context shared among different system components. We proposed a multi-modal data fusion methodology that fuses sensor and communication data. We integrate IoT physical and cyber elements into a knowledge graph representation that signifies the correlation between elements as a connection and provides embedding for data generated by each component. We constructed a GNN model to learn the context and normal state of the system and detect abnormal activities. We further studied the signatures of networks and sensor attacks to determine the source of the anomaly to facilitate fast and informed recovery after an incident. Experiments on the

greenhouse monitoring IoT systems demonstrated that this holistic multi-modal anomaly detection model, on average, achieves a 22% F1-score improvement over the single-modal approaches (Chapter 6).

1.3 Dissertation Organization

This dissertation embarks on a comprehensive exploration of cross-layer security, starting from hardware security and methodically progressing toward an understanding of higher-level embedded and cyber-physical system security. In Chapters 2 and 3, we delve into one of the primary security threats plaguing the semiconductor industry: hardware Trojans (HTs).

Chapter 2 unveils a hardware Trojan detection model that leverages a Graph Neural Network (GNN) classifier, detailing the process of translating hardware design into a graph format that accurately encapsulates its intrinsic structure. This chapter culminates in the evaluation of the proposed model, demonstrating its superior performance in both precision and speed compared to existing methods.

Building on the HT detection discourse, Chapter 3 tackles the subsequent phase of HT localization and elimination from the compromised circuit. We discuss a node classifier model rooted in GNN and delineate algorithms designed to convert the circuit into a graph with attached node attributes, identify Trojan nodes within the graph, and map the nodes back to their HDL code representation.

Having demonstrated the potential of graph learning in the hardware domain, Chapter 4 confronts another IC security threat: IP piracy. Borrowing the graph generation process from Chapter 2, we propose a new approach toward IP piracy detection. Instead of relying on traditional signature-based defense mechanisms, an automated GNN-based tool is introduced

to assess the similarity between different hardware designs.

The lessons learned in Chapters 2, 3, and 4, such as the importance of context and understanding the relationships between various components for security assurance, serve as cornerstones for the following chapters. Shifting from the low-level circuit to the system level, Chapter 5 presents a context-aware model for Cyber-Physical System (CPS) security, demonstrated within an Internet of Things (IoT) system. We discuss the pressing need for IoT security and introduce a novel methodology. This methodology extracts sensor fingerprints to identify clusters of related sensors, leveraging this sensor association information for enhanced fault and attack detection.

Finally, Chapter 6 integrates the GNN development insights from Chapters 2, 3, and 4 and the context-aware anomaly detection perspective from Chapter 5. It presents a novel approach for cross-layer anomaly detection in the physical and communication layers of IoT systems. Here, we elaborate on a GNN model that performs data fusion between multimodality communication network data and sensor measurements.

The appendices A and B provide a summary of three collaborative research projects that explore post-silicon IC security and software adversary attacks, effectively bridging the gaps in the cross-layer security exploration proposed in the main body of the dissertation.

Chapter 2

Hardware Trojan Detection

2.1 Introduction

The scale and complexity of modern System-on-Chip (SoC) designs have made it increasingly challenging and expensive for chip manufacturers to design, fabricate, and test every component in-house. The time-to-market pressure and resource constraints have pushed SoC designers to outsource hardware designs and use Third-Party Electronic Design Automation (3P-EDA) tools and Intellectual Property (IP) cores from various vendors worldwide. Using Third-Party IPs (3PIP) can be cost-effective due to the re-usability of IP cores so that chip manufacturers can reallocate their resources to meet market demands. However, the security and trustworthiness of 3PIPs are not always guaranteed, and reliance on untrusted IPs and EDA tools greatly raises the risks of HT insertion by rogue entities in the Integrated Circuit (IC) supply chain.

HT refers to an intentional and malicious modification of an IC that is usually designed to leak information, change functionality, degrade performance, or deny the service of the chip. Due to the wide applications of ICs in military systems, critical infrastructures, medical

devices, etc., the consequences of an undetected HT in a chip can be life-threatening. For example, an actual demonstration of the HT threat occurred in 2007, when a suspected nuclear installation in Syria was bombed by Israeli jets because Syrian radar was disabled by a remote kill switch backdoor in its commercial off-the-shelf microprocessor [6]. In 2012, an undocumented hardware backdoor was found in the Actel/Microsemi ProASIC3 chips used in military-grade FPGAs [147] that allowed the extraction of secret keys, enabling an adversary to modify the chip's configurations and gain full control of the chip. Furthermore, it is projected that the global semiconductor IP market will reach 7.3 Billion by 2025, with a compounded annual growth rate of 5.5% from 2020-2025 [1], and the security concerns about untrusted IPs can significantly damage the market.

Figure 2.1 shows the typical life cycle of an SoC design, starting from system-level specification to fabrication, in which several stages of the IC supply chain are marked as potential points of HT injection. A rogue in-house designer/team can manually modify or add malicious functions in the hardware design at any design abstraction level (system level, behavioral level, and logic level). Moreover, the 3P-EDA tools used for behavioral, logic, and physical synthesis may insert HT into the synthesized RTL and gate-level netlist code. Using untrusted 3PIP in various design stages introduces a means for adversaries to tamper and infect a design with HTs. To ensure the trustworthiness of an SoC design, it is crucial to ascertain the authenticity of 3PIPs. The 3PIPs are typically classified into three categories based on their format; Soft IP (i.e., synthesizable Verilog or VHDL in the RTL format), Firm IP (i.e., gate-level netlist), and Hard IP (i.e., GDSII files and custom physical layout format).

The flexibility of IP cores in higher levels of abstraction makes it easier for the attacker to design and implement various malicious functions. It is crucial to identify HTs early on in the design stages because it becomes increasingly expensive to remove them later. Detecting a few lines of HTs in an industrial-strength IP with thousands and hundreds of thousands of

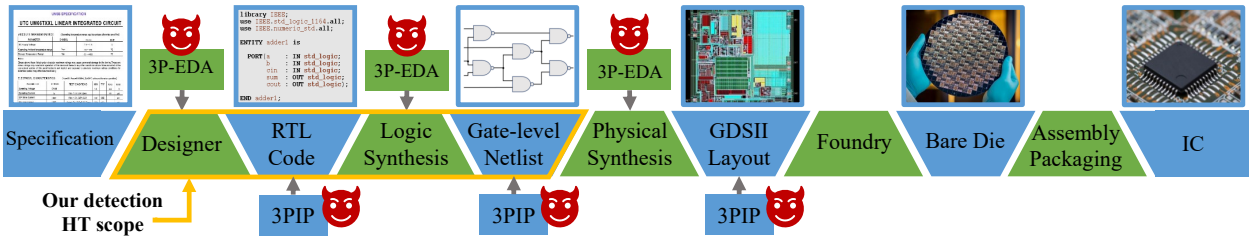


Figure 2.1: Semiconductor supply chain, pre-silicon HT injection points.

code lines is extremely challenging. Any work requiring manual code review is error-prone, time-consuming, and not scalable. The need for a scalable pre-silicon security verification method is highlighted in many technical documents, such as a recent white paper from Cadence and Tortuga Logic [53] because existing solutions fail to safeguard the hardware design from the constantly evolving HTs designed by adversaries. We further elaborate on this necessity through a motivational example.

2.1.1 Motivational Example

The HT detection problem has always been a back-and-forth tug-of-war. HTs are stealthy by design and are composed of a payload and trigger. They are usually very small and inactive, with minimal effects on the chip until the trigger circuit is activated under very rare circumstances and triggers the payload to perform its malicious activities. The HT detection problem has always been a back-and-forth tug of war. Whenever new HT detection methods are proposed in the literature capable of detecting currently known HTs, new HTs are designed to bypass state-of-the-art detection methods. This behavior can be observed by looking at the trend in HT detection over the past decade. One of the earliest defense mechanisms, [69] proposed a novel Unused Circuit Identification (UCI) technique that identifies suspicious circuitry not being used or activated during design verification. However, the authors in [148] later designed a new type of HT called Stealthy Malicious Circuits (SMC), which could bypass UCI by hiding HT in nearly-unused logic. Further, FANCI [160] was

successful in detecting SMC by identifying the low control value exhibited by the nearly-unused logic, but it was later defeated by DeTrust [186], which designed a new class of HTs with stealthy implicit triggers. Mero[23] generates test vectors that are capable of activating HTs with low trigger probabilities for detection. Still, it fails to generate test vectors that can activate "hard-to-trigger" HTs with trigger probabilities less than 10^{-6} [138].

Among the more recent works that use Trusthub benchmarks, [130] is supposed to detect HTs that leak sensitive data, such as secret keys in cryptographic cores, as in the AES-T600 benchmark. However, it fails to detect another similar HT benchmark, AES-T700. Later, [131] identifies different data leaking HTs (e.g., AES-T600 and AES-T700) by adding data leaking as an additional security property for model checking. However, it fails for HTs that execute other malicious functions, such as chip degradation (e.g., the AES-500 benchmark), rather than data leakage. A recent paper [56] showed that modeling the hardware design as a graph can be beneficial in the hardware security domain. However, the HT detection scope of [123, 56] based on graph similarity algorithms is limited to known HTs in the method's library. There have also been methods that convert the hardware design into a graph representation of the circuit and extract HT features to build a database of HT features for detection [27], but the detection scope is also limited to the HTs in its database.

Despite the various HT detection methods proposed in the literature, the HT problem remains significant as there exists no single method that can detect all different types of HTs. While most detection algorithms can only detect known HTs, new HTs are designed to circumvent existing detection methods. Thus, a new flexible technique for detecting unknown HTs is needed, one that can be expanded as new HTs emerge.

2.1.2 Research Challenges

The existing pre-silicon HT detection methods have several shortcomings:

1) **Reliance on golden-reference**: Most HT detection methods rely on a golden HT-free reference circuit to compare the design under test with the golden reference and flag it as HT-infected in the case of deviation. Nevertheless, a golden reference is hard to obtain in practice, especially infeasible at the IP level.

2) **Limited detection scope**: Some HT detection algorithms are constructed based on a library of known HTs. Consequently, they fall short in detecting unknown HTs. Another set of methods assumes particular properties in the Trojan design of the trigger or payload, limiting the detection scope to specific HTs.

3) **Manual code review**: Many HT countermeasures mark the parts of the design under test which are prone to HT insertion. However, this does not guarantee that an HT is present/detected, and it would burden the circuit designer with a manual review of the suspicious parts, which is tedious and error-prone for large designs.

4) **Scalability and complexity issue**: Due to the growing complexity of modern ICs, scalability is an essential feature for any hardware design tool but many current techniques and algorithms are so complicated that they face time or memory issues for large designs.

2.1.3 Chapter Contributions

To address these challenges, **we propose a golden reference-free pre-silicon HT detection approach that takes advantage of state-of-the-art machine learning techniques to learn the circuit behavior and detect the anomalous and malicious presence of Trojan inside the design.** Since hardware is a non-euclidean, structural type of data in nature, we use the graph data structure to represent the hardware design and generate the Data Flow Graph (DFG) for both RTL codes and gate-level netlists. We leverage GNN to model the behavior of the circuit. The scalability of our method stems from

the fully automated process of graph generation, feature extraction, and detection without any manual workload. Automatic feature extraction is crucial when new HTs are discovered, as our model will be readily updated without the need to define new properties or introduce additional feature engineering as in previous works. Our contributions are outlined below:

- We propose a novel approach for modeling the hardware design to ensure its security. Our methodology models the circuit as its intrinsic representation, a graph, and we leverage GNN to extract the critical features of hardware design and learn its behavior.
- We construct two fully automated models for HT detection in the pre-silicon stage. Each model includes a DFG generation pipeline followed by a GNN-based graph learning flow, and it is developed and customized for its target hardware design, either RTL code or gate-level netlist. The models discover even unknown HTs without relying on golden HT-free references or manual code reviews from the circuit designer. Our models are faster than existing methods and scalable for large designs.
- We create a Trojan DFG dataset consisting of RTL codes and gate-level netlists. We expand the HT-infested RTL benchmarks from Trusthub and perform logic synthesis and optimization to acquire the corresponding gate-level netlist.
- We survey the pre-silicon HT detection techniques in the literature, analyze state-of-the-art and make a comprehensive comparison with our approach.

2.1.4 Threat Model

This work aims to determine whether an RTL code or gate-level netlist is infected with a malicious Trojan circuit or not. There is no assumption on the type of HT and the design of the trigger or payload. Therefore, our method can detect trigger-based or always active Trojans with a payload circuit to modify functionality, degrade performance, leak confidential

data, or deny the service. The only premise is that the HT is inserted in the design stage through the following attack scenarios: 1) A rogue in-house designer intentionally manipulates the RTL/netlist design manually. 2) The 3P-EDA tool used for design/logic synthesis and analysis inserts an HT into the synthesized RTL/netlist code automatically. 3) 3PIP vendor is not trustworthy, and a malicious circuit is hidden in the IP.

2.2 Related Works and Background

The majority of the pre-silicon HT detection techniques in the literature fall into four main categories, described below:

2.2.1 Hardware Trojan

An HT consists of two fundamental parts: payload and trigger. The payload is the implementation of the malicious behavior of the HT. This malicious behavior could lead to information leaks, such as the secret key of a cryptographic core which would enable unrestricted access to sensitive data. It could change the functionality of the circuit to sabotage or cause harm. It can deny the service of certain functionalities of the chip or degrade the performance by increasing power consumption. The trigger is an optional circuit that monitors various signals or events in the base circuit and activates the payload when a specific signal or event is observed. HTs without triggers are usually always-on Trojans. HTs with triggers in the context of digital circuits can either be combinational or sequential. Combinational triggers activate upon a specific set of signal inputs. Sequential triggers rely on a specific sequence of signals or events. A sequential trigger could be based on a counter reaching a specific value or when a signal pattern repeats over a certain duration. A sequential trigger is more difficult to activate compared to a combinational trigger due to the vast number of states

that are required to be checked. We classify the triggers of HTs into three main categories; i) time bomb, ii) cheat code and iv) always on. The time bomb trigger is activated after numerous clock cycles, and the cheat code trigger depends on one or a sequence of specific inputs in the base circuit.

HTs are designed with malicious intent and to stay undetected. HT designers can reduce the likelihood of their HTs from being detected by using a common technique that involves placing them in rare internal nodes with exceedingly low activation probabilities. Rare internal nodes are difficult to detect because they are often outside the functional context of the intended design, and they often go undetected by traditional test methods. Therefore, many HT detection methods take into account the rare internal node characteristic of HTs as part of their detection schemes.

2.2.2 Hardware Trojan Detection Methods

Test Pattern Generation: Traditional test pattern generation has been widely used in both pre-silicon and post-silicon manufacturing stages. The idea is to create an extensive set of test vectors with varying input logic combinations to validate a circuit's output against the intended design output. The intended design output must come from a trusted golden reference chip of the original circuit design. These test patterns are typically generated based on the system specifications of the design, which describes the functionalities and behaviors of the circuit. However, due to their stealthy nature, HTs remain inactive and well-hidden during simulation and testing. Many HTs can bypass detection from these test vectors by simply not modifying the functionalities of the original circuit. More importantly, most HTs only activate under particular rare events. This makes it difficult for traditional test generation methods to trigger HTs that use internal node triggers with extremely low activation probability.

To tackle these issues, the authors in [23, 138] use automatic test pattern generation (ATPG) to create effective test vectors by identifying rare input logic at internal nodes to increase the probability of triggering the HT. However, even with this strategy, the scale of modern designs makes it possible for HT designers to create extremely low activation triggers that could evade detection by combining multiple rare signals and having the trigger conditions occur over multiple steps. [103] proposed a new test generation method by mapping the trigger activation problem to a clique cover problem which can detect extremely rare triggers but is shown to have unstable coverage performance. A recent study employs reinforcement learning techniques in conjunction with rare node excitation, as well as controllability and observability analysis to generate test vectors with improved trigger coverage and test generation time. Most methods above assume a full-scan chain design which simplifies ATPG by converting sequential elements into combinational elements with scan flip flops. However, not all designs have full-scan chains due to design constraints such as power, area, and additional hardware components required [18]. Therefore, a partial-scan chain design has been adopted. [39] tackles the challenges of partial-scan designs by combining ATPG with model checking for more efficient test vector generation and improved HT coverage. Still, there is no guarantee of success in the test pattern generation approach, and most test generation techniques are time-consuming due to their iterative nature. More importantly, in order to achieve full coverage, test pattern generation would need to generate and test all the possible cases, but given the scale of modern ICs, it is simply infeasible since the state space for test vectors grows exponentially to the number of rare input signals.

Formal Verification (FV): Formal verification is a mathematical proof-checking technique that relies on the security and trust policies defined in the system-level specification. It verifies the integrity of the design using common verification methods such as property checking, equivalence checking, and model checking. In order to apply formal verification, the 3PIP design must first be converted from an HDL language like Verilog to an equivalent

model in the proof-checking format using a formal language like Gallina, a tactic language developed as part of the well-known theorem prover called Coq. The 3PIP is delivered with a separate code representing the 3PIP but written in formal language and used for proof-checking; this code is known as proof-carrying code (PCC). PCC can either be provided by the 3PIP vendor directly or they can provide a Soft IP that can be converted into a proof-checking format.

Applying FV methods, [150] formulate taint-propagation properties that verify the data flow between signals in a design to identify unintentional design bugs. HTs, however, are intentional by nature, so the criteria for bug detection do not directly apply to HT detection. A similar approach is able to detect information leakage [131] and malicious modification to registers [130] by applying information leakage and register modification as criteria for the security properties. While using formal verification on 3PIP proves the predefined security properties, its detection scope is limited to the properties stated in the system-level specification. Only specific types of HTs can be detected because the properties are insufficient to cover all the various types of malicious behaviors that HTs can exhibit. [130], for example, defines "no-critical-data-corruption" which can only detect data-corrupting HTs. [131] modifies the security criteria from data corruption to data leakage. Both approaches employ model checking, which does not scale to large designs because model checking is NP complex and suffers from state explosion. [59] combines theorem proving with model checking to overcome the state explosion issues of model checking; however still suffers from the limited scope issue. Information flow tracking has been used to model security properties that can provide wider coverage of HTs [119]. FV does not depend on HT trigger conditions for detection, so it does not suffer from the issue of not being able to detect HTs due to low trigger probability. However, it is still possible for 3PIP vendors to intelligently manipulate the proofs and security properties to evade FV.

Code Analysis (CA): Code analysis or code coverage is a technique that analyzes the execution of the RTL or gate-level netlist code. To verify the 3PIP for trust, code analysis uses metrics such as line, statement, toggle, and finite state machine (FSM) coverage and compares against the design specification to ascertain the suspicious signals that imitate the HT. These coverage metrics respectively check which lines and statements are executed, what signals are being switched in the gate-level netlist, and which states are reached during execution. If anything less than 100% coverage is reported, then the 3PIP design is considered to be HT infested.

Using CA, [120] extracts state transition graphs from gate-level netlist and reports state transitions that are vulnerable to HT injection. This approach burdens the designer with manual analysis of the suspicious regions to identify the possible HT. It is also limited to the design's combinational logic. FANCI [160] proposed a control value metric that measures the degree of influence a given input has on the operation and output of a circuit. It looks for nets in the HDL code with very low control values and marks them as suspicious nets. VeriTrust [185] returns the nets that are not driven by functional inputs as potential triggers for an HT. DeTrust [186], on the other hand, proposes an attack that exploits the vulnerabilities of FANCI and VeriTrust by modifying the HTs with stealthy implicit triggers. Trigger logic for these HTs is distributed over multiple stages with a combination of sequential and combinational logic alongside logic that is part of the intended design, making them much more challenging to detect. In order to execute/cover all the lines of the code, code analysis requires an effective test bench to cover all the execution scenarios of the design. This causes the same problems as test pattern generation in that the number of test patterns needed for higher coverage scales poorly with larger designs, and verification time increases to an infeasible amount. It has also been shown that even 100% coverage does not guarantee that a 3PIP is HT-free [76].

Machine Learning (ML): Machine learning is a powerful technique that has received a lot of attention lately and has revolutionized different fields of study [174, 14]. The majority of ML-based works for HT detection depend on side-channel signal analysis [50, 51, 15], which delays the detection until the post-silicon stages. More recent approaches have focused on extracting various features during the design phase of the circuit to classify the HT-free and HT-infested 3PIPs. For example, the gradient boosting algorithm uses the Abstract Syntax Tree (AST) of RTL code [64], the multilayer neural network uses Trojan-net features of gate-level netlist [65], the artificial immune system uses DFG and Control Flow Graph (CFG) of RTL code [184], and the probabilistic neural network uses CFG of RTL code [40]. Most of these models rely on an HT-free golden reference. However, a trusted reference is not guaranteed because the untrusted 3PIP vendors are the ones that provide the source code and specifications that may include hidden HTs as part of the golden reference. Moreover, reference-based methods may be inconclusive or too complex for exhaustive verification, especially for large designs. With classical ML models, the models' performance heavily depends on the quality of the selected features. This typically requires a lot of upfront resource investment in feature engineering and can be quite time-consuming. Additional feature engineering would also be required to account for newly developed and unknown HTs. We propose a scalable, golden reference-free model that leverages a potent ML technique, GNN, which performs automatic feature extraction and learns the circuit's behavior, both intended and malicious behaviors, to detect known and unknown HTs.

2.2.3 Hardware as a Graph

A graph is an intuitive representation of a hardware design. The vast network of gates in a circuit can be naturally represented by the interconnections of nodes as gates and edges as wires in a graph. Graph-related problems have been around for decades, with firm roots in discrete mathematics and computer science. There is a rich pool of knowledge in graph

theory and many well-established algorithms were developed to solve these graph problems. By transforming a hardware design into its graphical representation, we can apply the same concepts and algorithms in solving graph problems to solve hardware design problems.

Various graph representations of traditional EDA problems are shown in [105]. For example, logical verification can be modeled using a rooted directed graph to represent the boolean function [21]. Many EDA problems can be converted into graph problems. However, many graph problems suffer from NP complexity and need to overcome scalability issues due to large modern IC designs. In more recent works, [123] propose analyzing the data/control flow graph of the circuit to locate the HTs. The authors create a library of HTs and use sub-graph matching algorithms to find the graphs of such known HTs in the graph of the 3PIP designs. Graph matching is an NP-complex problem that does not scale to large designs. To improve the accuracy and computation time, [56] introduces a new graph similarity heuristic tailored for hardware security. These methods can detect only the HTs that have the same graph representation as known HTs in their library, while attackers will design a diverse set of HTs. Recently, GNN has been deployed for reverse engineering to assist with malicious logic detection. In this direction, ReIGNN [36] has combined GNN with structural analysis to classify registers in a netlist. GNN-RE [10] leverages GNN to identify the boundaries of modules in a flattened netlist and classify the functionality of sub-circuits.

Recently GNN models have grabbed attention in EDA and hardware security [183, 175, 176]. GNN is deep learning that operates on the graph and has the advantages of machine learning methods but can be applied to non-Euclidean data. Our intuition behind using GNN is that graphs are an intrinsic representation of a hardware design. Representing the hardware design in the form of DFGs allows us to capture the behavior between signals, sequential elements, and combinational elements in the circuit. Through the DFGs, the GNN will learn to distinguish normal circuits from circuits that contain malicious functions.

2.3 Methodology

Our methodology is built on the premise of the existence of a feed-forward function f that determines whether the hardware design p is infected with a malicious Trojan circuit or not. To approximate the function f, our methodology comprises three main steps:

i) Firstly, we extract the DFG G from hardware design p through an automated DFG generation pipeline.

ii) Secondly, the DFGs are passed to a GNN framework that includes graph convolution layers and an attention-based graph pooling layer for graph learning and feature extraction. Further, the graph readout operation generates a vectorized graph embedding, denoted as $\mathbf{h_G}$ based on the discerning features learned by the GNN model.

iii) Lastly, Multi-Layer Perceptron (MLP) is used to perform classification on the graph embedding and output the HT label y.The HT label y is the output of function f, as given in Equation 2.1.

$$y = f(p) = \begin{cases} (1,0), & \text{if the design is HT-infected} \\ (0,1), & \text{if the design is HT-free,} \end{cases} \tag{2.1}$$

We describe some background information about GNN and provide more details regarding our GNN model and MLP classifier in section 2.3.1. In our approach, we target the hardware design in the pre-silicon stage, including the RTL and gate-level netlist. Although RTL codes and netlists are both represented using a Hardware Description Language (HDL) such as Verilog, they are very different in terms of structure, level of abstraction, and code size (number of signals and operations). Thus, we construct two distinct HT detection models with different graph generation pipelines, as elaborated in the following sections.

2.3.1 Graph Convolutional Networks

Many data structures such as social network data or hardware designs can be naturally formulated as graphs. Graph learning encompasses fundamental challenges since the graph size and topology vary among data samples. To address this issue, GNN is introduced. It is a graph-based deep learning model that extracts features from non-Euclidean data structured as a graph. The architecture we use is inspired by the Spatial Graph Convolution Neural Networks (SGCN). As depicted in Figures 2.3 and 2.4, the architecture of our GNN models for both RTL and netlist HT detection mainly includes convolutional layers, an attention-based pooling layer, a readout unit, and an MLP classifier.

The input to GNN is a graph $G = (V, E)$ where E is the set of directed edges and V is the set of vertices. The edges are represented as the adjacency matrix A and each node embodies a feature vector a_v that specifies the node attributes. In general, the convolution operation in an SGCN is defined by a node's spatial relations. The spatial convolution has a *message propagation phase* and a *read-out phase. The intuition behind message passing is that nodes pass feature vectors to their immediate graph neighbors and through an iterative process, the information is accumulated as node embeddings. Each iteration is basically one layer of graph convolution and by increasing SGCN layers, nodes can reach further nodes and gather information deeper in the graph. The message propagation phase involves two sub-functions:* ***AGGREGATE*** *and* ***COMBINE*** *functions, given by:*

$$a_v^{(l)} = \boldsymbol{AGGREGATE}^{(l)}(\{h_u^{(l-1)} : u \in N(v)\}), \tag{2.2}$$

$$h_v^{(l)} = \boldsymbol{COMBINE}^{(l)}(h_v^{(l-1)}, a_v^{(l)}), \tag{2.3}$$

where $N(v)$ is the set of nodes connected to node v, ***AGGREGATE*** *collects the feature vectors of neighboring nodes to produce an aggregated feature vector $a_v^{(l)}$ for layer l.* ***COMBINE*** *will combine the previous node feature $h_v^{(l-1)}$ with $a_v^{(l)}$ to produce the next feature vector $h_v^{(l)}$. The message propagation is carried out for a pre-determined number of l iterations. In our model, the* ***AGGREGATE*** *and* ***COMBINE*** *functions of massage passing are performed by the Graph Convolution Network (GCN) [80]. Concatenation of aggregated message vectors $a_v^{(l)}$ for all $v \in V$ forms the matrix $\mathbf{X^{(l)}}$, which we call the node embedding matrix.* The GCN layers update the node embeddings for each iteration l of *message propagation* as follows:

$$\mathbf{X^{(l+1)}} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}\mathbf{X^{(l)}}W^{(l)}) \tag{2.4}$$

where W^l is a trainable weight used in the GCN layer. $\widehat{A} = A+I$ is the adjacency matrix of G used for aggregating the feature vectors of the neighboring nodes and I is an identity matrix that adds the self-loop connection to make sure the previously calculated features will also be considered in the current iteration. This self-loop acts similarly to the COMBINE function where the accumulated messages are combined with the previous node feature vector. $\widehat{D}$ is the diagonal degree matrix used for normalizing $\widehat{A}$. $\sigma(.)$ is the activation function such as the Rectified Linear Unit (ReLU). We initialize the embedding $\mathbf{X}_i^{(0)}$ for each node $i \in V$ based on our initial intuition about the graph data in a specific application. We denote the final propagation node embedding $\mathbf{X}^{(l)}$ as $\mathbf{X}^{prop}$. Regarding the complexity of our model, GCN complexity grows almost linearly with the number of nodes for sparse graphs, and the circuit connections are sparse, producing a sparse DFG [94]. The other factor that affects the computation and memory usage is the length of node feature vectors which is a relatively small number (e.i. compared to raw image) in our case.

2.3.2 Attention-based Pooling

An attention-based graph pooling layer is then applied to the node embedding $\mathbf{X}^{prop}$. According to [81], such an attention-based pooling layer allows the model to concentrate on a local part of the graph and is regarded as part of a unified computational block of a GNN pipeline. We perform *top-k filtering* on nodes based on the scores predicted from a separate trainable GNN layer [85], as follows:

$$\alpha = \mathbf{SCORE}(\mathbf{X}^{prop}, \mathbf{A}),\ \mathbf{P} = \text{top}_k(\alpha) \tag{2.5}$$

where α denotes the coefficients predicted for nodes by the graph pooling layer. $\mathbf{P}$ represents the indices of the pooled nodes, which are chosen from the top k nodes ranked by α. The number k used in *top-k filtering* is computed as $k = pr \times |V|$, where pr is a pre-defined constant called pooling ratio. We consider only a constant fraction pr of the embeddings of the nodes in the DFG to be relevant (i.e., 0.6). The pooling ratio like other hyper-parameters of the GNN model is tuned based on the application through design exploration by the model developer. The node embeddings and edge adjacency information after pooling are denoted by $\mathbf{X}^{pool}$ and $\mathbf{A}^{pool}$ which are calculated as follows:

$$\mathbf{X}^{pool} = (\mathbf{X}^{prop} \odot \tanh(\alpha))_{\mathbf{P}},\ \mathbf{A}^{pool} = \mathbf{A}^{prop}{}_{(\mathbf{P},\mathbf{P})} \tag{2.6}$$

where $\odot$ represents an element-wise multiplication, $()_{\mathbf{P}}$ refers to the operation that extracts a subset of nodes based on P and $()_{(\mathbf{P},\mathbf{P})}$ refers to the adjacency matrix between the nodes in this subset.

2.3.3 Graph Embedding Generation

In the read-out phase, our model aggregates the node embeddings acquired from the graph pooling layer, $\mathbf{X}^{pool}$ and extracts the graph-level features which is a vector called graph embedding $\mathbf{h}_G$ for DFG G.

$$\mathbf{h}_G = \mathbf{READOUT}(\mathbf{X}^{pool}) \tag{2.7}$$

Where the **READOUT** operation may be summation, average, or the maximum of each feature dimension, overall node embeddings, denoted as *sum-pooling*, *mean-pooling*, or *max-pooling* respectively.

2.3.4 Multi-Layer Perceptron Classifier

Our GNN model generates a graph embedding for each DFG that represents the essential features of the hardware design. We further process the embedding vector $\mathbf{h}_G$ with an MLP layer and a Softmax activation function to produce the final prediction as follows,

$$\hat{Y} = \mathbf{Softmax}(\mathbf{MLP}(\mathbf{h}_G)) \tag{2.8}$$

This layer reduces the number of hidden units used in $\mathbf{h}_G$ and produces a two-dimensional output representing the probabilities of both classes (HT-infested or HT-free). Finally, the predicted values in $\hat{Y}$ are normalized using the Softmax function, and the class with the higher predicted value is chosen as the detection result.

To train the model, we compute the cross-entropy loss function, denoted as H, between the

ground-truth label Y and the predicted label $\hat{Y}$, described as follows:

$$\arg\min H(Y,\hat{Y}) = \arg\min \sum_{y_i \in Y, \hat{y_i} \in \hat{Y}} y_i log_e(\hat{y_i}) \tag{2.9}$$

The model is trained through an iterative process using the gradient descent method to minimize our cross-entropy loss function.

2.3.5 GNN for HT Detection in RTL

In this section, we explain our first model that processes the hardware design in RTL and determines the presence of HT in the code. The overview of the model is illustrated in Figure 2.3. The first part of the figure indicates the pipeline to extract data flows of RTL code and structure them as a directional graph, DFG. The next part presents the architecture of the machine learning model that we developed to classify the RTL graphs.

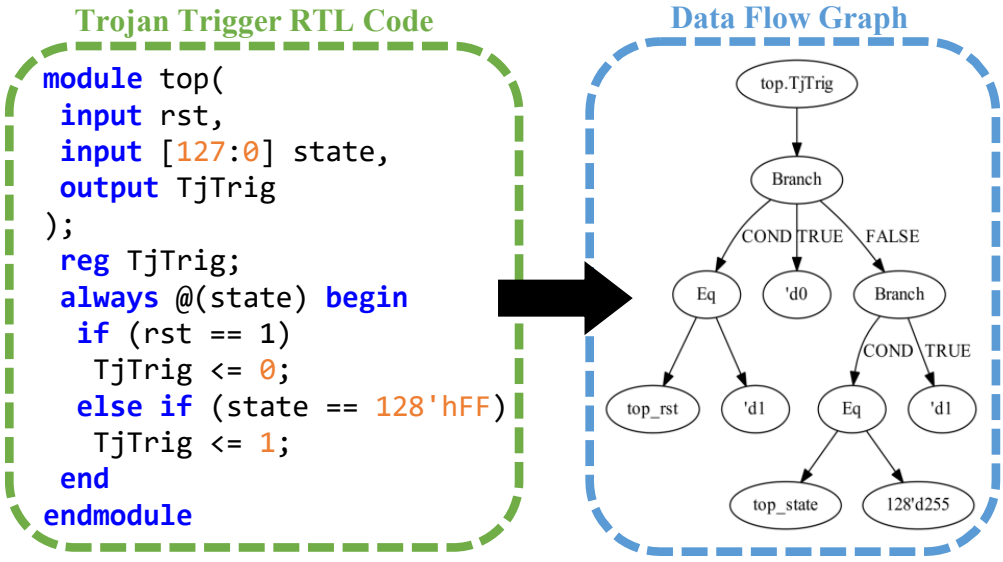


Figure 2.2: The RTL code of a Trojan trigger and its DFG.

2.3.6 RTL Graph Generation

A graph is a non-euclidean data structure that retains the topological information as well as signals. Similarly, the hardware design describes the circuits in terms of circuit elements and their connections. Given that these are naturally aligned, we convey the hardware design by employing DFG. DFG captures the data dependencies between signals and operations in the circuit and provides a fundamental expression of the computational structure. We develop an automated conversion process of the circuit to DFG through our DFG generation pipeline, as illustrated in Figure 2.3(a). The automated pipeline comprises several phases: preprocess, parser, data flow analyzer, and merge.

Due to the complexity and size of circuits, a digital circuit is often designed in a hierarchy, with multiple modules in different files. Consequently, the graph generation procedure starts with combining the files and flattening the design to a single RTL code. The preprocessing phase also resolves any syntax incompatibilities (i.e., invalid signal names). Next, we use a hardware design toolkit called Pyverilog [153] to parse and analyze the Verilog code. The Pyverilog parser extracts the abstract syntax tree from the code and passes it to the data flow analyzer to generate a DFG for each signal in the circuit such that the signal is the root node. To have a single graph representation for the whole circuit, we fuse all the signal DFGs and trim the disconnected sub-graphs and redundant nodes in the merge phase. Figure 2.2 exemplifies an RTL code and its corresponding DFG.

In addition to graph topology, node types also contain valuable information for graph learning and are used to create the node feature vectors. We initialize the node feature vector to be the one-hot encoding of the node type. The merged DFG generally contains three categories of node types; operation, constant, and signal. Twenty-eight different types of operation nodes are recognized in RTL graphs, and nodes are tagged by their operation name accordingly (e.i. AND, XOR, etc.). The constant tag indicates that the node represents a number and

is tagged as constant regardless of its value. Notably, more variation is observed in signal nodes because their tags are extracted from signal names in the RTL code designated by the designer. We assign signal nodes distinctive tags based on their name in the RTL code because semantic data is concealed in the signal names. After scanning various RTL codes, we create a list of signal names that provide semantic information and are prevalent in different designs. For example, a node name that spells "clk" or "clock" generally refers to the circuit clock. We tag the signals node based on the list, and the nodes with a new signal name not in the list are tagged the same as the "general" signal node. Therefore, the feature vector length is independent of the circuit, and eventually, the length of node feature vectors for RTL graphs is 300 nodes. In contrast, such high-level information does not exist in a netlist, and signal names are arbitrary, only to show wires. Thus, we ignore the signal names in the netlist and tag the signals either as input, output, or intermediate in the netlist.

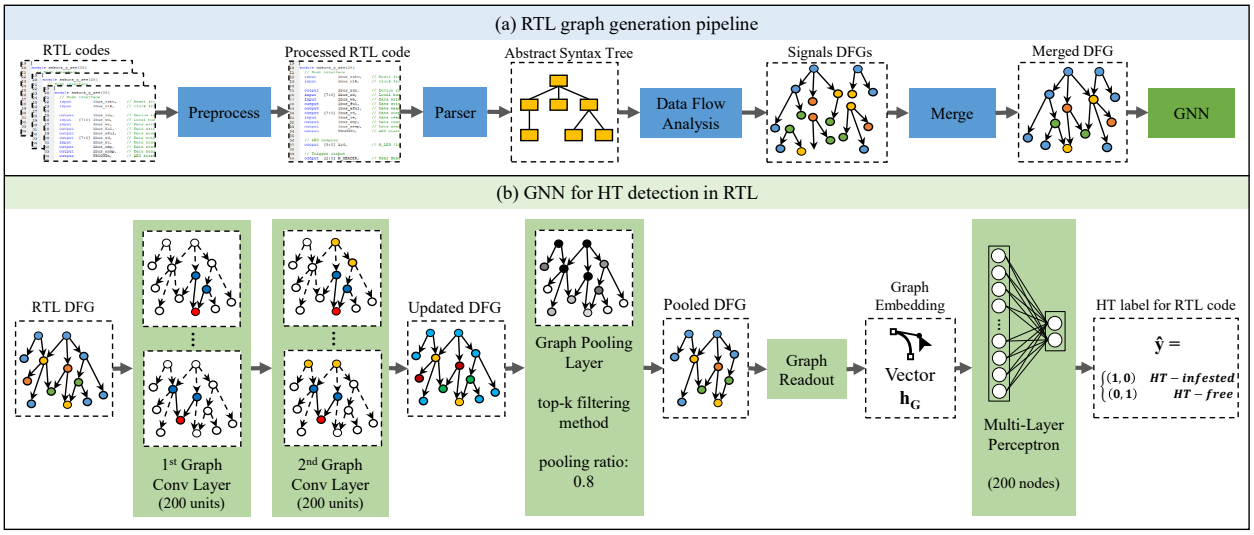


Figure 2.3: The RTL DFG generation pipeline and GNN model for Trojan detection in RTL code.

2.3.7 HT Detection Model for RTL

Figure 2.3(b) demonstrates the architecture of our RTL Trojan detection model based on graph convolutional networks. The mathematical background of GNN and details of each

layer are elaborated in section 2.3.1, and this section presents the architecture of our customized GNN model for HT detection in RTL. When the DFG of RTL design is generated by the graph generation pipeline, it is passed to the GNN model for graph learning. To be precise, the model inputs are the adjacency matrix of DFG, which indicates the connections among nodes and a feature vector for each node. Our model architecture includes two convolutional layers with 200 hidden units that perform message passing and modify the node feature vectors. Then, we use a graph pooling layer as our attention mechanism, which pushes the model to focus on the critical nodes. Our pooling method is k-filtering with a ratio of 0.8, which scores the nodes and keeps the nodes with the top 80% scores. Further, the readout unit extracts the graph embedding vector from the node feature vectors and passes it to the MLP with 200 nodes to classify the RTL code as HT-free or HT-infected. This architecture is achieved after some design space exploration which is discussed in section 2.4.4.

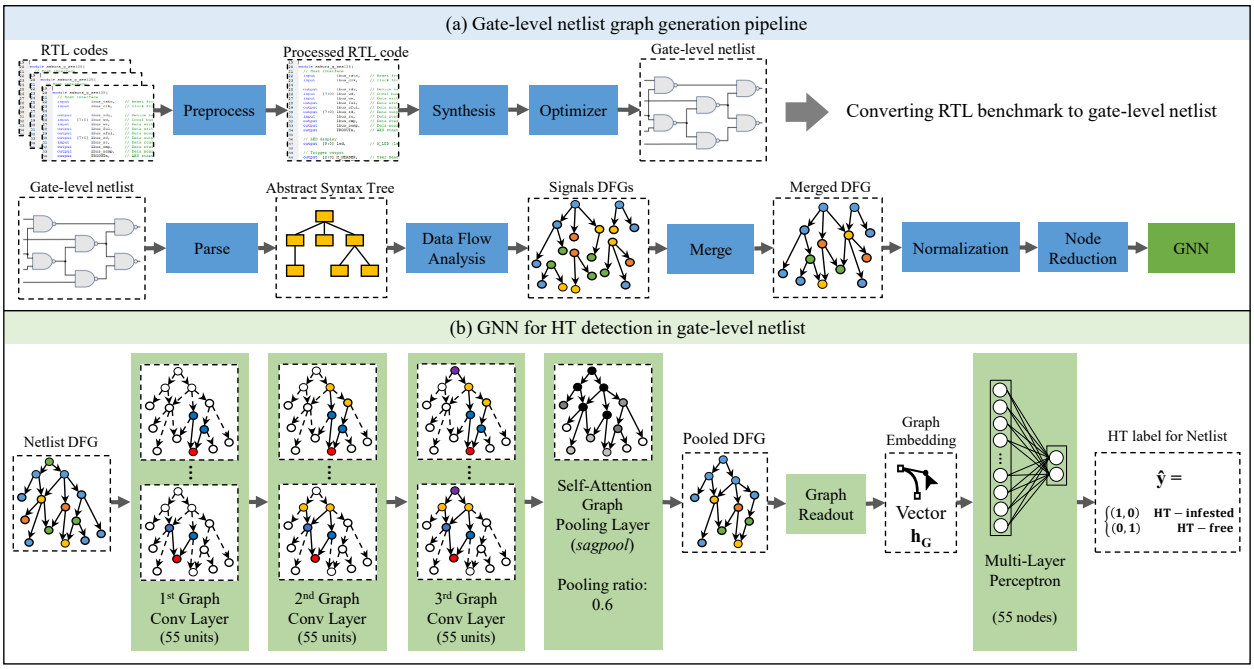


Figure 2.4: The netlist DFG generation pipeline and GNN model for Trojan detection in the gate-level netlist.

2.3.8 GNN for HT Detection in Netlist

Gate-level netlist is different from the RTL design as it is closer to the actual hardware implementation and very detailed. The high-level behavior of RTL code vanishes in the netlist, and it is substituted by numerous gates and signals defined to imitate the circuits components and their connection. As a result, the graph representation of the netlist is different from RTL with a considerably larger size. Consequently, our RTL HT detection model fails to perform with the expected accuracy for netlist, and we develop a customized graph generation pipeline and GNN model for finding HT in a netlist, as shown in Figure 2.4.

2.3.9 HT Benchmarks Synthesis to Netlist

Following the netlist graph generation procedure in Figure 2.4(a), we employ the open-source RTL synthesis tool, called Yosys [164] to synthesize the RTL HT benchmarks and generate the gate-level netlists. A custom script is used to automate the RTL to gate-level netlist synthesis and perform some additional processing steps to fix the syntax incompatibilities for Pyverilog. The Yosys logic synthesis steps are as follows: 1) The Yosys RTL front end converts the RTL code into the RTL intermediate Language (RTLIL) internal cell representation. 2) The built-in optimizer is called to remove unused signals and cells, optimize finite state machines, and translate memories to primary memory cells. 3) The built-in technology mapper and ABC tool are used to map the internal cell representation to a custom standard cell library made up of generic gates (AND, OR, XOR, NOT, etc.) and output as a generic gate-level netlist code in Verilog. One issue during logic synthesis is that the optimization step removes unused signals and cells. According to Yosys, unused signals and cells are defined as those that do not modify an output signal. This becomes a problem as there are HTs that do not alter any output of the circuit. For example, a Trojan is designed

to degrade the performance of the chip by causing additional power consumption through excessive shifting. To resolve this issue, we specify in Yosys to not remove any user-defined signals and cells so that it will retain the HT code.

2.3.10 Netlist Graph Generation

After RTL synthesis, the netlist in Verilog format is generated. The synthesized netlist code also introduces syntax incompatibilities with Pyverilog, so we perform some processing to remove these issues. Finally, the cleaned-up gate-level netlist is passed to the Pyverilog parser. The remaining steps for parsing the AST, generating a DFG for each signal, and merging and trimming are similar to the RTL code analysis to produce a single DFG that represents the whole circuit. The final DFG for gate-level netlist is a rooted directed graph that shows data dependency from the output signals (the root nodes) to the input signals (the leaf nodes). It is defined as graph $G = (V, E)$ where E is the set of directed edges and V is the set of vertices. We define $V = \{v_1, v_2, ..., v_n\}$ where v_i represents signals, constant values, and operations such as xor, and, branch, or branch condition. We define $E = e_{ij}$ for all i, j such that $e_{ij} \in E$ if the value of v_i depends on the value of v_j, or if the operation v_j is applied on v_i.

2.3.11 Netlist Graph Optimization

Behavioral characterization of RTL is lost in the netlist due to a lower level of abstraction, and the circuit is described as a detailed, complex sea of gates. Consequently, the netlist graph is significantly larger than its RTL counterpart, thwarting HT detection. For instance, the RTL DFG of AES benchmark, the most complicated benchmark in our dataset, holds on average 13K nodes per graph. Conversely, the AES benchmark for gate-level netlist graph

contains 300K nodes per graph on average. The increased graph size drastically expands the memory footprint for both the dataset and the model, leading to memory shortage. Therefore, we reconstruct our dataset and model to reduce the memory footprint.

After reviewing various netlists and their DFGs, we identified unnecessary details in the signals and operations that are ineffective in HT detection. To reduce the graph size, we conduct an automated graph optimization process to eliminate the excessive details of the netlist. Our graph optimizer trims the graph generated by Pyverilog and removes redundant nodes, whose removal does not affect the overall hardware design data-flow representation. For example, the nodes tagged as *concatenation* and *part selection* by Pyverilog provide detailed information that in HT detection applications are not required. The *concatenation* node appears for certain node types, such as the logic gates used in the netlist. For example, an output node Y depends on the result of an OR gate operation node between two input signal nodes, A and B. The *concatenation* node exists between the OR gate and the two input signal nodes to signify that the OR gate depends on two signals. When the redundant *concatenation* node is removed, the signal nodes are directly connected to the OR gate without the intermediate *concatenation* node, indicating data dependency without any information loss.

The *part selection* node appears when there is a bit-wise assignment. For instance, assigning bits five to ten for a wire node A to bits three to eight for a wire node B. Node A will be connected to node B with a *part selection* node making up of 3 components; the least significant bit of A, the most significant bit of A, and the data signal A itself. We are only concerned about the data dependency in the HT detection application and not the bit-wise detail of the data. Therefore, we exclude the *part selection* nodes and only keep the data signal in the dependency relation. With these changes, we reduced the netlist graph of the AES benchmarks from approximately 300K nodes to 100K nodes. Overall, the average number of nodes for the entire dataset was reduced by 50%. Figure 2.5 demonstrates

examples of netlist DFG optimization through the elimination of *concatenation* and *part selection* nodes.

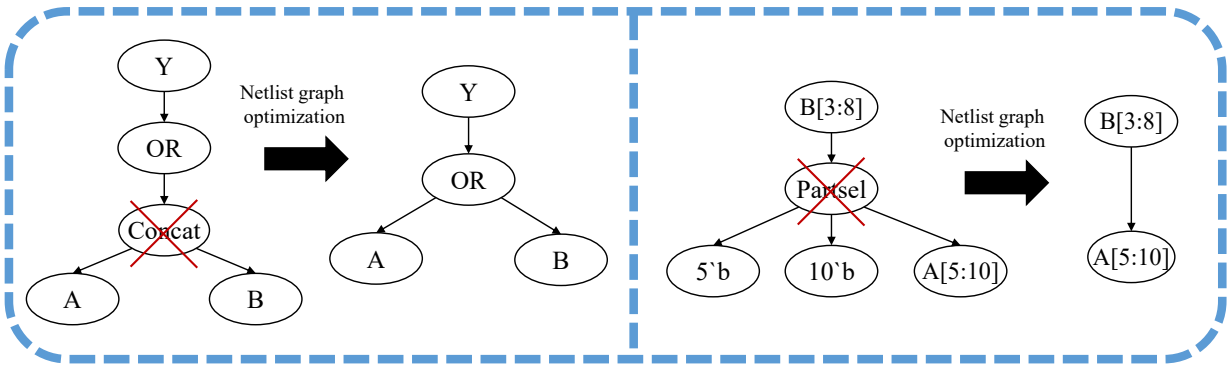


Figure 2.5: Netlist graph optimization by removing *concatenation* and *part selection* nodes.

2.3.12 Netlist Graph Normalization

Although we have significantly reduced the overall size of our netlist dataset, it is still comparably larger than its RTL counterpart. To reduce the memory overhead further, we modify an integral part of the GNN model itself, the node feature vector dimension. The feature vector is used as part of the graph convolution process for node embedding. It is initialized based on intuition about the application and data concealed in the graph nodes. The node feature matrix is an N by D matrix where N is the number of nodes and D is the dimension of the feature vector. We have reduced N in the graph optimization step through node reduction. Therefore, we apply a node normalization step to reduce the feature vector dimension to resolve this issue.

In our application, nodes in a netlist DFG have different types such as signals, constant values, and logic operations. Each signal node in the DFG has a unique name, and tagging the nodes based on their name leads to large feature vectors while the signal names hardly convey any semantic data or behavioral description (i.e. Wire1, Wire2). We normalize the DFGs by generalizing the node tags. Instead of having each unique signal name, constant

value, and type of operation as a one-hot encoding, we categorize them into 17 classes.

For example, all nodes with no in-degrees will be assigned as input signals, and all nodes with no out-degree will be designated as output signals. This also applies to the generic gate nodes where each gate is assigned a specific value. Normalization drastically decreases the number of node tags from 300 to only 17, reducing the feature vector dimension and subsequently, the required memory for graph learning. Moreover, the accurate initialization of the feature vector can improve the performance of GNN to converge fast to high accuracy. Our intuition is that the feature vectors correlate to their node type, and by normalization, we extract this information that is carried over to GNN.

2.3.13 HT Detection Model for Netlist

After the netlist DFG is generated and optimized, it is passed to the GNN model along with its normalized node feature vectors. The architecture of our netlist-customized HT detection model is demonstrated in Figure 2.4(b). We reach this model based on our intuition about the distinct characteristics of netlist after conducting various experiments, some of which are presented in section 2.4.4. The most challenging characteristic of the netlist is the graph size which is relatively larger than the RTL graph for the same circuit, even after optimization. Additionally, it is too detailed and missing the high-level behavioral information of circuits, making the feature extraction challenging. Due to these traits, our netlist model has one more graph convolutional layer than the RTL model (3 layers in total) to perform more in-depth massage passing and feature extraction. Moreover, the number of hidden units is decreased to 55 to minimize the resource utilization of the model. Next, we utilize self-attention graph pooling to revise the nodes and focus on those with strong influence. We got the best performance for the pooling ratio of 0.6, which is lower than the RTL ratio. It matches the initial intuition that there are many unnecessary nodes in the netlist graphs

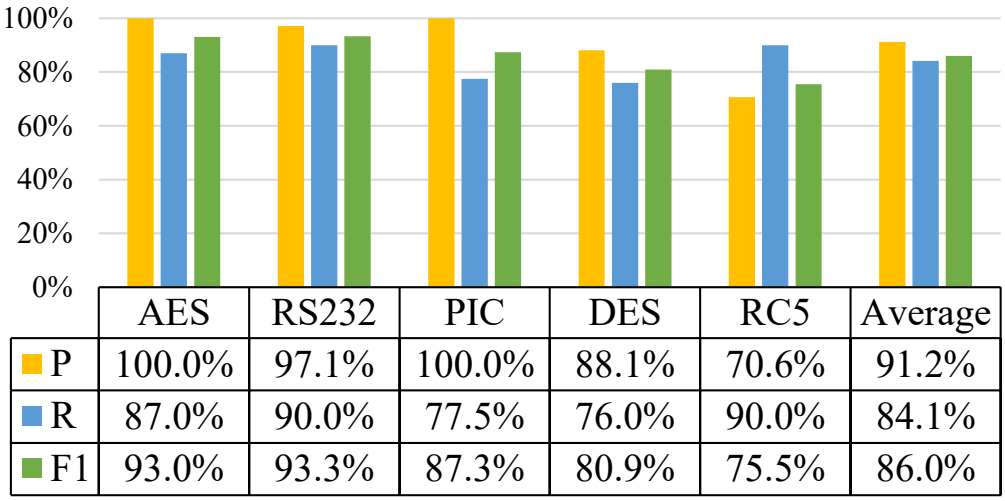
that can be omitted through pooling. The rest of the model is very similar to the RTL-customized model, and it creates graph embedding with a readout unit followed by an MLP for classification.

2.4 Evaluation

In this section, we explain our dataset and evaluation, report the results, and compare our proposed method with state of the art.

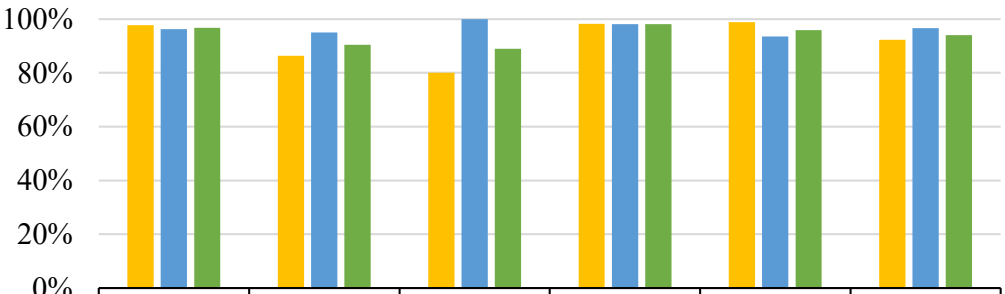
2.4.1 Dataset

We create a dataset of RTL codes and gate-level netlists based on the benchmarks in Trusthub [143] which contain 34 varied types of HTs inserted in 3 base circuits: AES, PIC, and RS232. To expand our dataset, we extract the HTs design and use them as additional HT data instances. Moreover, we integrate some of the extracted HTs into new HT-free circuits, DES and RC5, which increases the number of HT-infested samples. To balance the dataset and increase the number of HT-free samples, we add different variations of open-source HT-free circuits in addition to new ones, including DET, RC6, SPI, SYN-SRAM, VGA, and XTEA to the dataset resulting in a dataset of 132 combined instances of 47 HT-Free and 85 HT-infested RTL codes. We also synthesized a corresponding generic gate-level netlist for each RTL code, which doubles the number of data instances. In our dataset, the number of nodes in netlist DFG of the base circuits on average is as follows: 109130 nodes in AES, 41649 nodes in DES, 107675 nodes in RC5, 10628 nodes in PIC, and 1307 nodes in RS232.



	AES	RS232	PIC	DES	RC5	Average
P	100.0%	97.1%	100.0%	88.1%	70.6%	91.2%
R	87.0%	90.0%	77.5%	76.0%	90.0%	84.1%
F1	93.0%	93.3%	87.3%	80.9%	75.5%	86.0%

Figure 2.6: The performance of our method in gate-level netlist HT detection.



	AES	RS232	PIC	DES	RC5	Average
P	97.8%	86.4%	80.0%	98.2%	98.9%	92.3%
R	96.3%	95.0%	100.0%	98.1%	93.5%	96.6%
F1	96.8%	90.5%	88.9%	98.1%	95.9%	94.0%

Figure 2.7: The performance of our method in RTL HT detection.

2.4.2 HT Detection Results

We evaluate our method performance through an unknown HT detection scenario. To create this scenario, we apply the leave-one-out method, where we leave out one of the base circuits' HT-free and HT-infected benchmarks for testing and train the model on the rest of the benchmarks. This setup satisfies the claims of golden chip-free and unknown HT detection, as both golden HT-free versions of the circuit under test and the injected HT circuit are unknown to the model. We repeat this process 20 times for each base circuit and count the True Positive (TP), False Negative (FN), and False Positive (FP) to measure the evaluation

metrics, Precision (P), Recall (R), or True Positive Rate, and F_β *score* as follows:

$$P = \frac{TP}{TP+FP}, R = \frac{TP}{TP+FN}, F_\beta score = \frac{(1+\beta^2)*P*R}{\beta^2*P+R}$$

Recall measures how well our model can detect HT-infested samples. This metric is intuitive, but it is not sufficient for evaluation, especially in an unbalanced dataset. It is also essential to have a low false-positive value which is the number of HT-free samples incorrectly classified as HT. Thus, precision is necessary as it measures the number of correctly classified HT samples over all samples classified as HT-infested, including the false positives. The F_β *score* is the weighted average of precision and recall that better presents the overall performance.

Figures 2.7 and 2.6 demonstrate the performance of our models for HT detection in RTL and netlist. Higher performance in RTL HT detection is justified by the intuition that RTL code is the behavioral description of the circuit, and the high level of abstraction assists with the learning process and facilitates the detection process. Moreover, the DFG generated from RTL is considerably smaller, and it is easier for the model to capture the key feature of the design. DFGs of netlists are more complicated and detailed, which impedes the learning process and requires a more complicated model. The complexity of the model can be adjusted by changing the number of hidden units and layers.

On the other hand, our models perform better for AES, RC5, and DES compared to RS232 and PIC because the former circuits are all encryption cores and the model has more data instances to learn their behavior while the latter circuits are a communication protocol and a processor. The performance for RS232 and PIC can be enhanced by including more circuits similar to PIC and RS232 in the dataset.

There exist circuits in RTL and netlist that do not follow the same pattern. For example, HT detection in RC5 has higher accuracy in RTL compared to RS232, whereas the reverse relation is recorded in the netlist. To analyze this case, multiple influential factors should be

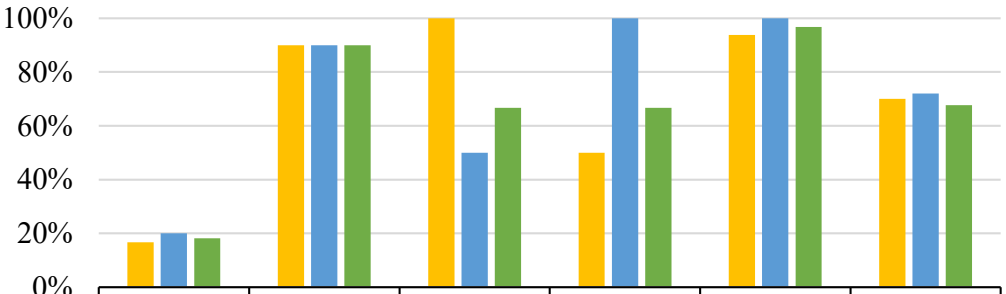
considered, such as the complexity of the circuit, the size of circuit DFG after optimization, the number of similar data samples in the dataset, etc. The combination of these factors has created such performance patterns, and by concentrating on a single aspect, the results may seem sporadic. In the case of RS232 and RC5 circuits, RS232 is a relatively simple circuit, and its netlist DFG is small and intuitive, which can be why high HT detection performance in RS232 is maintained, even in the netlist.

2.4.3 Effect of Attention Mechanism

In this section, we investigate the effects of our attention mechanism, the pooling layer, on the performance of the model. In this case study, we keep the hyperparameters and architecture of the GNN model for HT detection in netlist the same and only change the pooling ratio to 1, which means the operation does not occur. The results of this experiment are depicted in Figure 2.8. As the results indicate, in the absence of the pooling layer, learning does not happen correctly for most circuits. For example, for DES circuit recall equals 1 and precision is 0.5, which means the model has not learned the distinction between HT-free and HT-infected circuits and has labeled all the test samples as HT-infected. This experiment highlights that the attention mechanism plays a vital role in the learning process since it pushes the model to concentrate on the critical parts of the graph.

2.4.4 GNN Hyperparameters

We develop two GNN models for HT detection in RTL and netlist. We determine hyperparameters of each model based on our intuition about the unique characteristics of our application, previous knowledge developed by common potent GNN architectures in the literature, and GNN design space exploration. The intuition toward the application usually helps with better configuration initialization and faster convergence toward the optimal ar-



	AES	RS232	PIC	DES	RC5	Average
P	16.7%	90.0%	100.0%	50.0%	93.8%	70.1%
R	20.0%	90.0%	50.0%	100.0%	100.0%	72.0%
F1	18.2%	90.0%	66.7%	66.7%	96.8%	67.7%

Figure 2.8: The performance of our model without pooling layer.

chitecture. After testing a setting, we analyze the performance and decide what parameters to modify. For example, if we notice under-fitting, we increase the model complexity and computational power.

In the experiments on RTL, we use 2 GCN layers with 200 hidden units for each layer. For the graph pooling layer, we use the pooling ratio of 0.8 to perform *top-k filtering*. For **READOUT**, we use *max-pooling* for aggregating node embeddings of each graph. Our model uses 1 MLP layer to reduce the number of hidden units from 200 to 2 used in $\mathbf{h}_G$ for predicting the result of HT detection. In training, we append a dropout layer with a rate of 0.5 after each GCN layer. We train the model for 200 epochs using the batch gradient descent algorithm with batch size 4 and a learning rate of 0.001.

The gate-level netlists tend to produce considerably larger DFGs compared to RTL codes, even after optimization and node reduction steps. Thus, the netlist HT detection model requires a different configuration than the RTL model with an optimized structure to be easily trained using GPUs. In our constrained computation platform of a GPU with 14GB RAM, we limit the hidden units to 75 hidden units for 2-Layer networks, 58 for 3-layer, and 40 for 4-layer. We performed random search and grid search over a range of hidden layers, hidden units, pool ratio, and batch size. We found the average F1-score across all

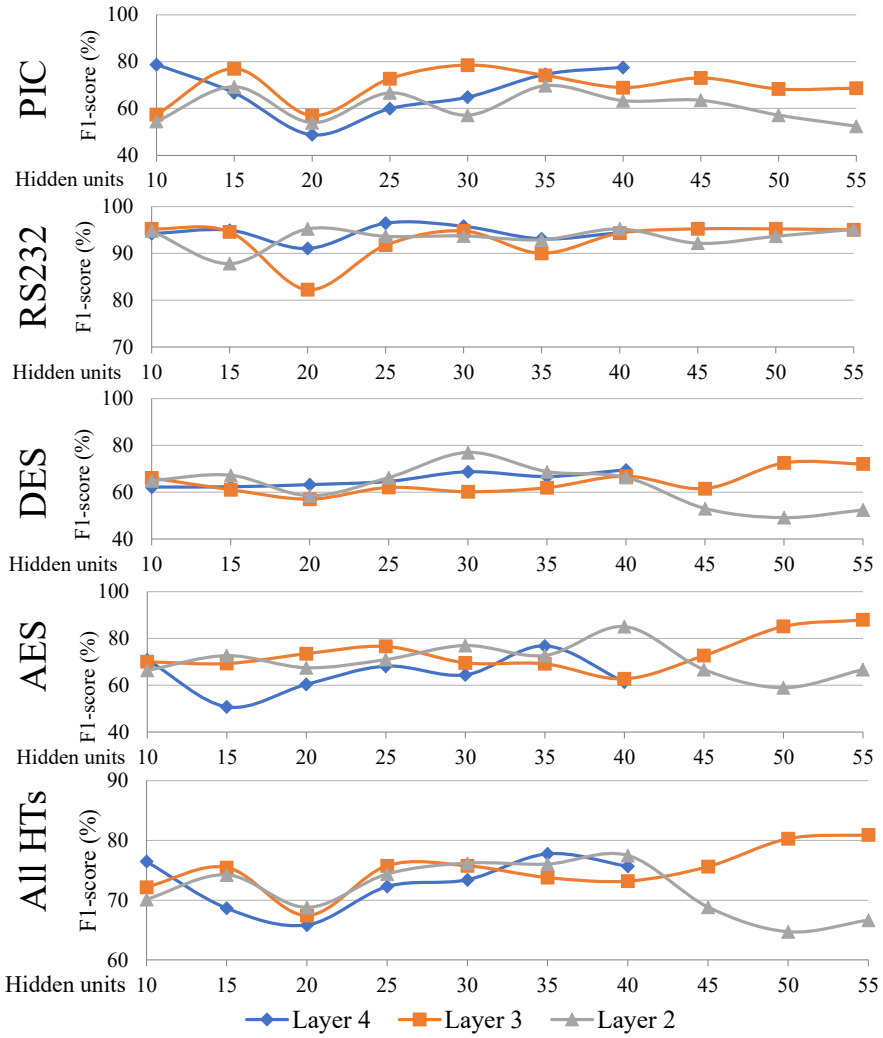


Figure 2.9: The model performance under different settings for the number of convolutional layers and hidden units in each layer.

configurations for each layer is shown in Figure 2.9. According to the figure, the 3-layer network has the best average F1 score compared to the 2-layer and 4-layer networks. Fixing our model to a 3-layer network, we then find the best-hidden unit, pool ratio, and batch size by comparing the F1-score for all pairs of combinations across all 5 golden reference-free benchmarks. We found the best configuration for the gate-level netlist model to be a 3-layer network with 55 hidden units and a pool ratio of 0.6 for performing *sagpool*, or self-attention graph pooling. The **READOUT** phase and the MLP are the same as the RTL model, and we train this model for 200 epochs using the batch gradient descent algorithm with batch size 2 and a learning rate of 0.001.

Table 2.1: Timing of HT detection per sample and training models.

HT detection in RTL						
Benchmark	AES	RS232	PIC	DES	RC5	**Ave.**
Detection time(s)	0.0268	0.0169	0.0182	0.0222	0.0215	**0.0211**
Training time (s)	220.37	251.18	274.18	261.71	284.53	**258.39**

Table 2.2: Timing of HT detection per sample and training models.

HT detection in netlist						
Benchmark	AES	RS232	PIC	DES	RC5	**Ave.**
Detection time(s)	14.07	13.70	13.40	14.07	13.37	**13.72**
Training time (s)	1796	5449	5346	5164	4215	**4394**

2.4.5 Timing

We train and test the RTL model on an NVIDIA GeForce GTX 1080 graphic card with 8GB memory and the gate-level netlist model on a V100 GPU with 16GB memory. The timing results for the base circuits are summarized in Table 2.1 and Table 2.2. Although training can be time-consuming, it occurs once, and the trained model finds the HT very fast in 21.1ms on average for RTL and 13.72 seconds for gate-level netlist which are both comparable to other works.

2.4.6 Comparison with State of the Art

In this section, we compare leading pre-silicon HT detection techniques, in terms of quantitative and qualitative metrics as well as in a case study.

Qualitative comparison:

We compare our models against other works using 3 essential qualitative metrics for HT detection; i) golden reference-free, ii) unknown HT detection, and iii) automated process. Designing an effective HT detection method without white-box knowledge or access to a

trusted reference design is very challenging as both are not readily available. Traditional algorithmic approaches that utilize heuristics can satisfy the golden reference-free condition, whereas ML techniques naturally rely on datasets containing both HT-free and HT-infested designs to train and classify test circuits. The next important metric is the ability to detect unknown HTs. In this regard, ML-based approaches have more success as the features of existing circuits are learned and used to classify similar circuits or unseen circuits. Techniques such as [123, 56] and [131, 119] limit their detection scope to HTs that exist in the method's library or only to security properties that are explicitly defined. To the best of our knowledge, all the existing approaches require feature engineering or manual property definition. However, our approach is capable of HT detection with an automated feature extraction process, which removes the need for a manual feature or property extraction. Thus, our models can easily expand to various types of circuits and can scale to industrial-level IP designs through retraining if a new type of HT is discovered.

Quantitative comparison:

For quantitative comparison, we analyze the precision, recall, and timing metrics. For timing, we compare the average detection time for the AES benchmarks due to being the largest benchmarks in our dataset in terms of graph size.

It should be noted that the exact comparison between the reported timings is not possible as the computing platforms differ on a paper-to-paper basis. However, the relative difference between the timings shows that our models are faster than if not comparable to others. The timing of algorithmic methods (FV, CA, and GM) highly depends on the hardware design complexity. Their memory usage and detection time drastically grow for large designs which can cause timeout or memory shortage problems. On the other hand, the circuit's complexity does not have a notable effect on the detection time of the proposed RTL method according to Table 2.4, which makes it scalable for large designs. Our gate-level netlist detection time is consistent across benchmarks of varying complexity which shows that it is also not greatly

affected by the circuit complexity.

For recall and precision, both our RTL and gate-level netlist have comparable results. Although it is challenging to compare against algorithm methods since most papers only report a list of known HT benchmarks that their models can successfully detect while we test our model with the unknown HTs scenario, which is mostly not detectable by those algorithms. Furthermore, FP and TP for computing precision and the performance of the methods on HT-free samples are not available in some papers to compare.

Table 2.3: Comparing HT detection methods in a case study.

Benchmark	**Victim Circuit**	**Trigger**	**Payload**	**Ours (ML)**	**[56] (GM)**	**[123] (GM)**	**[131] (FV)**	**[119] (FV)**
AES-T900	AES encryption core	Time bomb	Leak data	✓✓	✓	✓✓	✓✓	✓✓
RS232-T500	UART serial communication	Time bomb	Deny service	✓✓	✓	✓✓	✗	✓✓
AES-T1900	AES encryption core	Time bomb	Degrade chip	✓✓	✓	✓✓	✗	✗
AES-T2000	AES encryption core	Cheat code	Leak data	✓✓	✗	✗	✓✓	✓✓
RS232-T700	UART serial communication	Cheat Code	Deny service	✓✓	✗	✗	✗	✓
AES-T1800	AES encryption core	Cheat code	Degrade chip	✓✓	✗	✗	✗	✗

2.4.7 Case Study

We analyze the ML, GM, and FV techniques in a case study and investigate if the models can detect 6 different types of HTs. In Table 2.3, the first top 3 HT benchmarks are known to the Method Under Test (MUT) and exist in its library of HTs whereas the other 3 HTs are unknown. ✓✓indicates that the MUT can detect the HT and it is explicitly reported in its paper. ✓\✗shows that this case is not tested in the paper but it is supposed to detect\not to detect the HT according to authors' claims and assumptions. The results indicate that

GM methods rely on the HT library and memorize them and consequently, cannot recognize the HTs out of the library. FV methods depend on the predefined properties for HTs and cannot identify new types of HTs. On the other hand, our method is an ML-based method that learns the HT behaviors and can pinpoint different types of HT, even the unknown ones.

2.5 Discussion

The size of the graph dataset plays an important role in determining how well our models can learn to generalize with unknown circuit designs. The more nodes in the graph, the more difficult it is for the learning process. The difference in nodes between the gate-level netlist representation versus the RTL representation is several orders of magnitudes larger. Due to the memory limitations, the complexity of our model was also limited to a specific range of layers and hidden units. Future works should look at how to further reduce the graph sizes for the gate-level netlist or to use a different type of graph representation for the hardware circuit.

2.6 Chapter Concluding Remarks

We propose a novel golden reference-free approach to finding unknown HT in both RTL codes and gate-level netlists. We generate DFGs for both RTL and netlist codes and employ the GNN to construct two models that infer the presence of HT from the generated graphs in RTL and gate-level netlist. Our method automatically extracts the features of graphs and learns the behavior of the hardware design. Our model is trained and tested on a DFG dataset created by expanding the Trustub benchmarks. The RTL results indicate that the proposed method discovers HT with 97% recall and 94% F1-score very fast in 21.1ms. The

gate-level netlist results indicate an 84% recall and 86% F1 score with an average detection time of 13.72 seconds.

Table 2.4: Comparing the performance of our method with the state-of-art methods for HT detection in 3PIP.

Paper (year)	**Technique category - Method**	**Prec** [1]	**Rec**[2]	**Time (s)**	**Automated feature extraction**	**Golden reference free**	**Unknown Trojan detection**
Ours (2021)	ML - Graph neural network on RTL graph	92%	97%	0.026	✓	✓	✓
Ours (2021)	ML - Graph neural network on netlist graph	91%	84%	13.72	✓	✓	✓
[184] (2018)	ML - Artificial immune system	87%	85%	NA	✗	✓	✓
[64] (2019)	ML - Gradient boosting algorithm	NA	100%	1.36	✗	✗	✓
[65] (2017)	ML - Multi-layer neural networks	NA	90%	NA	✗	✗	✓
[40] (2017)	ML,GM - Subgraph isomorphism	NA	100%	1.15	✗	✓	✗
[56] (2020)	GM - Graph similarity	NA	NA	NA	✗	✓	✗
[123] (2017)	GM - Subgraph matching	NA	100%	5.02	✗	✓	✗
[74] (2019)	CA - Socio-network analysis	98%	98%	NA	✗	✓	✗
[119] (2017)	FV - Information flow analysis	NA	100%	292.85	✗	✓	✗
[131] (2016)	FV - Model checking	NA	100%	96.13	✗	✓	✗

[1]Precision [2]Recall

Chapter 3

Hardware Trojan Localization

3.1 Introduction

The growing complexity of Integrated Circuits (IC), time-to-market pressure, and expensive design and manufacturing processes have promoted the globalization of the semiconductor industry. Outsourcing the fabrication and depending on third-party hardware IP blocks and EDA tools raise the risk of intentional and malicious manipulation of the circuit, known as a Hardware Trojan (HT). Figure 3.1 demonstrates the IC supply chain and the involved parties which are vulnerable points of HT insertion. Currently, HT is a significant hardware security concern with devastating consequences such as denial of service, malfunctioning, data leakage, and performance degradation in the chip. The attackers usually design HT to be a tiny circuit hidden inside the main design, normally inactive with minimal effect on the chip's functionality and specification. The HT often gets triggered under rare circumstances, and consequently, it can escape detection by routine simulation and functional testing.

Trojan detection is crucial to ascertain the authenticity of 3PIPs and prevent the negative consequences of HTs, as elaborated in Chapter2. However, HT detection does not suffice to

ensure the fabrication of a trustworthy chip, and HT localization is the next essential step. The hardware IPs fall into three classes based on their format and level of abstraction; Soft IP (i.e., synthesizable Verilog or VHDL source code), Firm IP (i.e., placed RTL block and netlist), and Hard IP (i.e., physical layout and GDSII). Soft IP is the most popular IP core, and IP trust revolves around it. However, it has the most vulnerability against HT insertion because the flexibility and high level of abstraction in Register Transfer Level (RTL) codes facilitate the HT design and implementation for the attacker [156].

Manual review of hardware design to pinpoint HT is very time-consuming and error-prone, especially for an industrial-level large design. Due to the paramount importance of the HT threat, numerous defense mechanisms are proposed in the literature to determine if the design is infested with HT. Still, they fail to locate it in the IC design. Some works analyze parameters and side-channel data of circuits such as polynomials of gate-level implementation [52], thermal map [155], or path delays [137] to pinpoint the disturbance introduced by HT. They have the premise that a trusted Trojan-free reference design called golden reference exists to compare against the parameters of the circuit under test, which is an unrealistic assumption. In order to obtain the golden reference, the whole process of IC design, test, and fabrication should be performed by in-house trusted teams, EDA tools, and manufacturing facilities that would be very expensive and infeasible in practice.

HT defense methods based on formal verification and code analysis define some properties for HT, analyze the hardware design, and mark the areas satisfying the predefined properties.

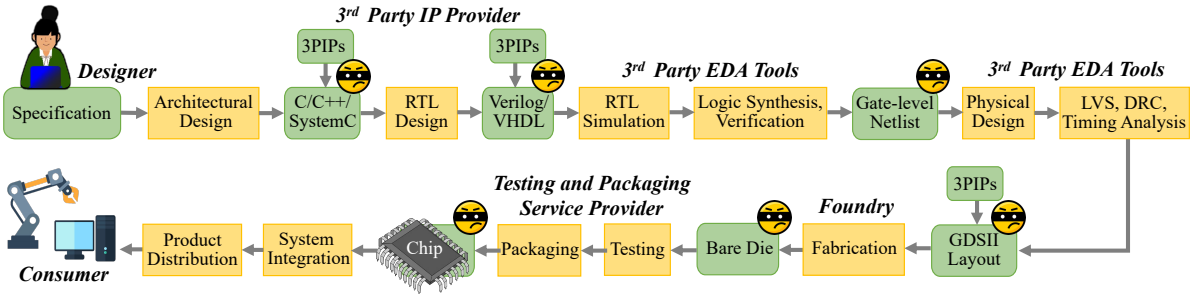


Figure 3.1: IC supply chain, vulnerable to HT insertion in different stages.

For example, [75] examines RTL codes using word-level statistics of the inputs and tags the arithmetic blocks with rare nets as vulnerable to HT. [149] flags the unused portion of the circuit as malicious. [160] measures the degree of control that an input signal has on the operation and outputs of the circuit and marks weakly-affecting inputs as possible HT triggers. These works narrow down the search space for HT. However, they still burden the designer to manually review the suspicious areas, which can be a large circuit due to low localization resolution. Moreover, most of the existing solutions require manual property definition or feature extraction, and they fail to outline a comprehensive set of properties or features representing all kinds of HTs. Consequently, they are effective only for particular HTs.

There is an increasing trend to explore the graph representation of hardware for security purposes [55] because hardware design is a non-Eulicidian structural data that shares similar properties with a graph. The graph is a mathematical structure that represents the relation between pairs of objects. It preserves the topological information that makes it the best match for modeling the fundamental objects in the hardware design process [106]. For instance, the graph is leveraged to represent the hardware design in [123, 55] for HT mitigation. Still, Graph matching algorithms fail to recognize unknown HTs and are not scalable to large designs due to high complexity.

Deep learning has introduced potent techniques that revolutionized many fields of study [174], but it operates on Euclidean data and cannot be directly used for hardware design. The current deep learning models for HT mitigation examine the side-channel emissions of the fabricated chip, which are time-series data [50]. To fill the gap and apply machine learning to hardware design, we convert the design from the textual format of HDL code to a graph and leverage GCN, which is like deep learning operating on graphs. A recent work [176] proposes a graph classification model based on a graph neural network to find whether the circuit is infested with HT or not, but it fails to locate the Trojan.

In order to overcome the shortcomings of current approaches, we propose a novel, golden reference-free HT localization method in the pre-silicon stage. We generate a graph representation for the hardware design, assign attributes to nodes in the graph, classify nodes, and locate the Trojan in HDL code based on the graph node's class. Our node classification model is based on GCN that automatically aggregates the features in graph nodes through the graph convolution operation. We create a dataset of hardware designs by inserting HT benchmarks from TrustHub [2] to different circuits. Our methodology is trained on this dataset to learn the behavior and features of Trojan nodes. Then, the trained model can locate the Trojan nodes based on their malicious abnormal behavior in even new and unknown Trojans in a fully automated process without any need for manual review.

3.2 Research Challenges

HT detection and localization is a difficult problem, and the current solutions suffer from the following shortcomings:

- **Reliance on golden reference**: A Trojan-free circuit called golden reference for comparison with the circuit under test is not available in the real world, and the golden reference-dependent methods are not practical.
- **Unable to generalize**: Various types of HTs have been discovered so far, and new HT designs are continuously introduced. Due to the variety in HT design and specification, defining a template or some properties that describe all HTs is challenging. Consequently, many countermeasures fail to generalize and are limited to known HTs or only HTs with a specific trigger or payload.
- **Low localization resolution**: Some works output the areas of the circuit that are vulnerable to HT insertion and due to low localization resolution, they burden the

designer with an exhaustive review of suspicious regions.

- **Manual feature extraction**: Algorithmic and classic machine learning approaches rely on an expert to define properties and extract features from the circuit, which is error-prone and exhausting.

- **Scalability**: With the increased complexity of ICs, scalability has become an essential characteristic of any circuit analysis tool, but complex algorithms fail to scale for large designs.

3.3 Chapter Contributions

We surmount the aforementioned research challenges and propose a novel, golden reference-free approach for HT localization that is fully automated with no need for manual revision by experts. To the best of our knowledge, this is the first work to apply GCN for HT localization. Our contributions can be summarized as:

- The hardware design HDL code is converted to a data-flow graph using the hardware design toolkit [154]. We develop an algorithm to extract the attributes of nodes and assign an attribute vector to each one.

- We construct a node classification model based on GCN that automatically aggregates the features for each node in the graph representation of hardware design, learns their behavior and marks the malicious nodes.

- We develop a Trojan labeling algorithm that provides a mapping from HDL code to its graph and labels the nodes in the graph as Trojan or benign. This algorithm determines the HT label vector of the training dataset, which is deployed by the GCN model for training and calculating classification loss.

- We survey the existing pre-silicon HT detection and localization methods and their shortcomings to picture the current state and challenges of this research area as well as the potential of graph learning for advancement in hardware security.

3.4 Related Works and Background

3.4.1 Hardware Trojan Localization

The majority of defense mechanisms against HT focus on its detection, and there are inadequate works with the capability to locate the Trojan circuit. For example, [137, 155] perform HT localization with the assumption that the design pipeline is trusted and the attacker resides in the foundry. Both works have the unrealistic premise that a golden reference is available. [137] proposes a satisfiability-based test pattern generation scheme that detects and locates the Trojan inserted by the foundry by comparing the timing and path delays of the suspicious IC with a golden IC. [155] extracts the Trojan activity factor from the redundant thermal map and performs HT localization by comparing the thermal side-channel of the target chip with the golden reference.

Code analysis is one of the conventional pre-silicon HT defense mechanisms that inspects the HDL code to ascertain suspicious signals in the circuit, and it is mainly restricted to combinational logic. In this technique, the code is scanned based on coverage metrics (toggle, line, state, etc.) to find the potential areas of HT presence. Different methods propose various definitions for a suspicious area, such as unused circuit identification [68], weakly affecting inputs [160], and low dependence on functional inputs [185]. Due to the exclusive definition of HT, later [149] defeats [68] and [185] and [160] get bypassed by the new HT attack [186]. [75] proposes a framework to analyze RTL codes using word-level statistics of the inputs. It locates the arithmetic blocks with rare nets to be reviewed as candidates vulnerable to HT

and can only identify HTs that are always on or triggered by current inputs. Code analysis suggests the circuit areas are susceptible to HT insertion and cannot actually locate the HT or even guarantee its detection.

[52] introduces a formal method based on symbolic algebra by extracting polynomials from the gate-level implementation of the untrustworthy IP and comparing them with the golden reference polynomials. [74] leverages principal features of social network analysis to outline the relation between design properties and locate HT. This approach applies only to combinational Trojans.

3.4.2 Graph in Hardware Applications

Hardware design is non-Eulicidian structural data that shares similar properties with a graph. The graph is a mathematical structure that represents the relation between pairs of objects. It preserves the topological information that makes it the best match for modeling the fundamental objects in the hardware design process. Thus, the graph is leveraged to represent the hardware in numerous Electronic Design Automation (EDA) problems which shift the problem to choosing the appropriate algorithm from the many well-known graph algorithms and applying it directly or with a slight change to solve the problem. However, developing an effective approach for each problem is still challenging. Furthermore, many problems are NP-hard with large sizes, which makes efficiency a major concern and leads to scalability issues. To tackle the complexity issue, data-driven learning techniques have grabbed much attention. The classical machine learning models include an initial step of manual feature extraction, which is followed by model training based on a large set of data instances [106].

The next generation of machine learning models leverages convolutional layers in deep learning models, making the feature extraction process automated through learning. Recently, deep learning models have been developed with high resiliency against adversarial attacks

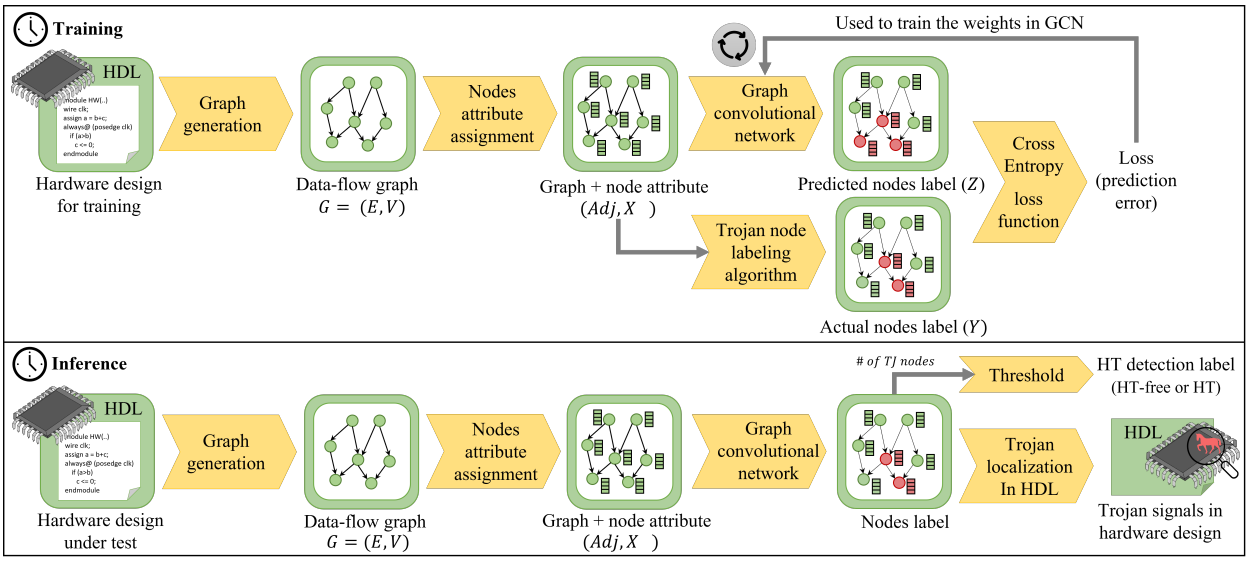


Figure 3.2: Overview of our HT localization methodology in training and inference phases.

[14]. Although that deep learning has improved the performance in various applications, it cannot be directly applied to graphs because it was originally developed for Euclidean data, and notable extra endeavors are needed to extract features from graphs and encode the structural information. In response, the graph learning method is introduced, which defines the convolutional operation on graphs and automates the feature extraction from graphs. There have been a few works investigating the advantages of graph learning for hardware security [176, 175, 183] and hardware design automation such as test point insertion [107], and power estimation in simulation [189]. In this work, we leverage a state-of-the-art machine learning model, GCN, to model the hardware for security purposes.

3.5 Methodology

In this work, we propose an automated pipeline to locate Trojan circuit at RTL that includes several steps, as depicted in Figure 3.2; i) converting hardware design to graph, ii) extracting the node attributes, iii) labeling Trojan nodes, iv) node classification, and v) HT localization in HDL. In the following sections, we define our problem formulation and threat model, and

then we elaborate on the aforementioned steps for localization.

3.5.1 Problem Formulation and Threat Model

The main target of our methodology is to locate the Trojan circuit inside the hardware design at RTL. The model's input is an HDL code which is later converted to a graph. The graph representation is further processed, and the graph learning model classifies the graph nodes as Trojan or benign. Eventually, the model outputs a list of malicious signals and operations in the HDL code corresponding to Trojan nodes in the graph.

The graph learning model, GCN, is trained on a dataset of graphs derived from HT benchmarks in which the labels of the nodes are known. Our dataset only includes the HT-infested designs, not any Trojan-free designs. Our approach is golden reference-free and able to perform HT localization on unknown HTs. To demonstrate these characteristics, we train our model on a set of circuits and test it on the circuits not observed by the model before in the training stage. Therefore, the model locates Trojan nodes in the circuit under test while it has not seen its golden reference or HT benchmark. Moreover, we make no assumption about the HT payload or trigger type, and the fundamental features of Trojan nodes are automatically aggregated and learned by convolutional layers in our GCN.

An attacker may manipulate the hardware design at any pre-silicon stage of the IC supply chain in our threat model (refer to Figure 3.1), but eventually, the HDL code should be available for our methodology to perform HT localization. Therefore, multiple attack scenarios are feasible. The attacker can be a rogue in-house designer, an untrusted 3PIP design company, or a 3P-EDA tool provider who tampers with the HDL code. The adversary may alter the design in the low level of abstraction, such as netlist and physical layout. In this case, we assume that the RTL code is obtained by reverse engineering.

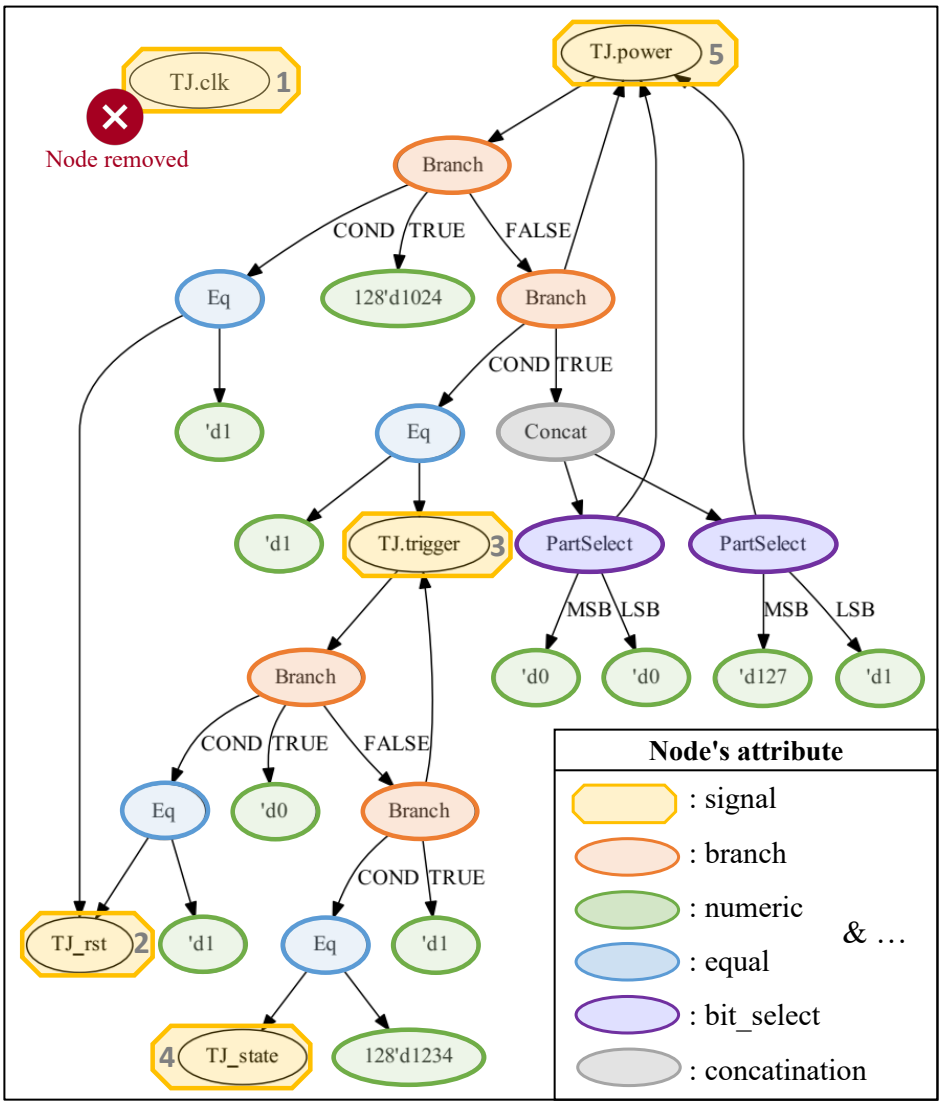


Figure 3.3: The data-flow graph and node attributes of AES-T1800 Trojan benchmark, shown in Figure 3.4.

3.5.2 Hardware Design Conversion to Graph

A circuit is described using Hardware Description Languages (HDL) at the design stage, such as Verilog and VHDL. The HDL code has a textual format with predetermined syntax and cannot be directly used as data for machine learning. Thus, we convert the HDL code to a graph that embeds the design features and preserves the topological information.

HDL code comprises modules, signals, and operations. Modules are used to cluster parts

of circuits and better express the hierarchy in the hardware design, but they do not affect the design specification. On the other hand, signals and operations fundamentally describe the hardware design. A signal can be a register or wire in HDL code, and it carries a value that is changed through an operation or assignment. For instance, the Verilog code for the AES-T1800 HT benchmark is shown in Figure 3.4 with its corresponding data-flow graph in Figure 3.3.

To convert HDL to graph, we combine the modules to have a single HDL code for the whole design. Afterward, we parse it and extract the data dependency subgraphs for all signals in HDL code using a hardware design tool called PyVerilog [154]. Each signal subgraph is a tree that expresses how the value of root signals depends on the operations and other signals in the design. We connect the common nodes in the subgraphs to construct one data-flow graph per hardware. The extracted graph $G = (V, E)$ is a directed homogeneous graph in which each node is named after its corresponding signal/operation in the HDL code, and an edge e_{ij} indicates that node v_i depends on node v_j. Lastly, the graph is processed to remove standalone nodes such as *clk* node in the data-flow graph of AES-T1800 HT benchmark, depicted in Figure 3.3.

3.5.3 Node Attribute Extraction

The initial data-flow graph $G = (V, E)$ expresses the flow of information and connections between components in the circuit, but it does not differentiate between the nodes. Therefore, we develop a node analyzer to extract the type of nodes from their name, which Pyverilog generates during graph generation. Then, the analyzer assigns an attribute vector to each node which is further used as an input feature to GCN. Nodes can be categorized as *constant*, *signal*, or *operation*. The *constant* nodes represent numbers and are tagged as numeric regardless of their value. The *signal* nodes are derived from a wire or register in the HDL code

```
module TJ (
  input clk, 1
  input rst, 2
  input trigger, 3
  input [127:0] state 4 );
  reg   [127:0] power; 5
  always @(clk) begin
    if (rst == 1) power <= 128'd1024;
    else if (trigger == 1) power<={power[0],power[127:1]};
  end
  always @(*) begin
    if (rst == 1) trigger <= 0;
    else if (state == 128'd1234) trigger <= 1;
  end
endmodule
```

Figure 3.4: The Verilog code of AES-T1800 Trojan benchmark.

and tagged as input, output, or signal based on their position in the circuit. The *operation* nodes are related to the operands and conditional statements in the HDL code. They have a wide variety, including gates (not, and, or, xor, etc.), branches, conditional operands (equal, less than, greater than, etc.), Part select, and concatenation. We have detected 28 different types of *operation* nodes, which sums up the total number of the node types to 32. The tags are independent of the circuit design, and they represent all the possible types of nodes that can be generated by our graph generation pipeline for any HDL code. Some examples of node tags are demonstrated in Figure 3.3. After tagging nodes, we generate an attribute vector for each node by performing one-hot encoding on tags. Therefore, the new directed graph with N nodes and F different tags is defined by $A \in R^{N \times N}$ and $X \in R^{N \times F}$ where A is an asymmetric adjacency matrix, and X is the matrix of node attributes.

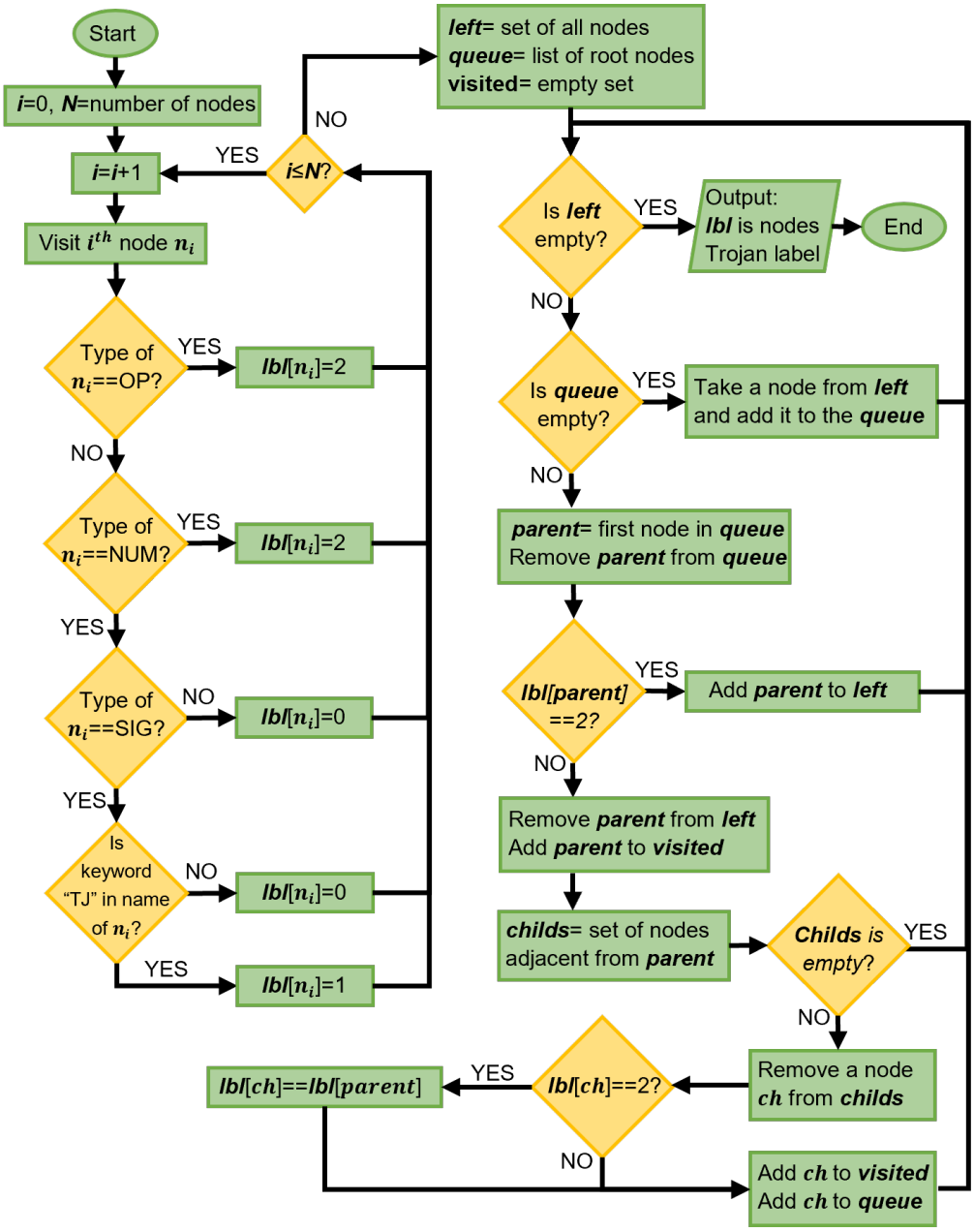


Figure 3.5: The flowchart of Trojan labeling algorithm.

3.5.4 Trojan Labeling Algorithm

Although the HT circuit is known in HDL codes used for training the GCN, the graph representation of the circuit does not have any notion of Trojan. Therefore, we develop an algorithm to determine the HT nodes in the graph representation of HT benchmarks. The algorithm has an HDL processing step in which a keyword is added to the signals and modules of the HT circuit. This keyword will be visible in the name of *signal* nodes of the HT circuit, but the *constant* and *operation* nodes will not be affected. So, after graph generation, the labeling algorithm iterates among the nodes and flags the *operation* and *constant* nodes as 2 (can be Trojan), the *signal* node with the keyword as 1 (definitely Trojan), and the rest of the *signal* nodes as 0 (not Trojan). Thus, the flag of *signal* nodes is known to be Trojan or benign. The flag of *constant* and *operation* nodes are modified based on the rules that the *operation* nodes applied to Trojan nodes are part of the HT circuit and the *constant* nodes inherit the flag of their parent *operation* node. Algorithm 1 and Figure 3.5 show how the algorithm traverses the graph starting from root nodes and modifies the number 2 flags based on these rules until there is no flag of 2 left and all nodes are marked either as Trojan or benign. The algorithm results in a Trojan label vector $Y \in [0,1]^N$ for each graph with N nodes in which the malicious nodes are marked as 1. This label vector is further used as the classification label of the training dataset to train the GCN model. Note that this step is only once performed for the training dataset in the training stage, and it is not required in the inference stage when the model is already trained and ready to locate HT in a circuit.

3.5.5 Graph Convolutional Networks

Traditionally, deep learning models often use an array/stack of trainable filters, such as convolutional neural networks to extract meaningful features for grid-like structured data. Inspired by those works, we adopt the GCN layer as our trainable filter from [80]. GCN

Input: Nodes name list *name*, Nodes type list *type*.
Input: Signal nodes tag SIG, Operation nodes tag OP, Numeric node tag NUM.
Output: Trojan label vector *lbl*.
Initialize *queue* = a list of root node.
Initialize $left$ = a set of all nodes.
Initialize *visited* = an empty set.
foreach *node* n_i *in* $left$ **do**
    if *(type[*n_i*] is* OP *or* NUM*)* **then**
        $lbl[n_i] = 2$
    else if *(type[*n_i*] is* SIG*) and (TJ in name[*n_i*])* **then**
        $lbl[n_i] = 1$
    else
        $lbl[n_i] = 0$
while $left$ *is not empty* **do**
    if *queue is empty* **then**
        node = $left$.pop()
        add *node* to *queue*
    else
        parent = *queue*.pop()
        if *lbl[parent] != 2* **then**
            remove *parent* from $left$
            add *parent* to *visited*
            foreach *child* c_i *of parent* **do**
                if c_i *not in visited* **then**
                    if *lbl[*c_i*] == 2* **then**
                        $lbl[c_i] = lbl[parent]$
                    add c_i to *visited*
                    add c_i to *queue*
        else
            add *parent* to $left$
return Trojan label vector *lbl*

Algorithm 1: Trojan Nodes Labeling Algorithm

is devised to embed nodes with different features while taking the topological information in non-euclidean data into account. The input of the GCN model is a graph $G = (V, E)$ represented by adjacency matrix $A \in N \times N$ and node attribute matrix $X \in N \times F$ where N is the number of nodes. F is the length of each node attribute vector in our model. Each graph convolution layer aggregates information from immediate node neighbors and updates nodes through a process called *message passing* based on the following formula:

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W^{(l)}) \tag{3.1}$$

Here, l denotes the layer number, and $H^{(0)}$ is the initial node features that equal to X, the node attributes matrix. $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(.)$ denotes the activation function that is Rectified Linear Unit (ReLU) in our model. To perform graph convolution, the normalized adjacency matrix $\tilde{A}$ is computed by:

$$\tilde{A} = \widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}} \tag{3.2}$$

Where $\widehat{D}$ is the diagonal degree matrix to solve the problem of scale change of the feature vectors after multiplication by the matrix A. It is calculated by $\widehat{D} = \sum_j \widehat{A}_{ij}$ and $\widehat{A}$ is derived from $\widehat{A} = A + I_N$ where I_N is the identity matrix that adds a self-loop connection to A, the adjacency matrix of graph G, to make sure each node embeds its previous value from last iteration as well as new data from its neighbors.

Stacking the graph convolution layers, we create a GCN that is able to integrate information from a larger set of neighbors. Our model architecture is illustrated in Figure 3.6. It comprises three convolution layers with a ReLU activation function and one last convolution layer connected to a layer of Softmax units to classify each node as Trojan or benign and generate the predicted node label Y. It concludes the computations of our GCN model as

below:

$$Z = Softmax(\tilde{A}\sigma(\tilde{A}\sigma(\tilde{A}\sigma(\tilde{A}XW^{(0)})W^{(1)})W^{(2)})W^{(3)}) \tag{3.3}$$

where $Z \in [0,1]^2$ indicates the predicted node labels in which $[1,0]$ denotes Trojan and $[0,1]$ denotes benign node.

Training: Figure 3.2 summarize the training process and units. The first step in training is preparing a training dataset based on the GCN model requirement. Thus, the HDL codes are converted to data-flow graphs in which the nodes have been assigned an attribute vector and a label (Trojan/benign). The attribute vectors feed information about each node's characteristics to the model, and the labels are used to calculate the error due to node misclassification. Training is an iterative optimization process that modifies the weights in GCN to minimize its classification error, anointed as the loss. We use Adam optimizer [79], a conventional optimization technique for efficient gradient descent to minimize the loss L. We utilize the cross-entropy loss function to calculate the error over all nodes in a graph using this formula:

$$L = -\sum_{i \in V}\sum_{j=0}^{C} Y_{ij} ln(Z_{ij}) \tag{3.4}$$

where C is the number of classes which is two (Trojan and benign), and the j indicates the dimension of the output vector. V is the set of nodes in a graph and i iterates over them. Y is the actual label of nodes obtained from the Trojan node labeling algorithm, and Z is the predicted label. Note that L is node classification loss for one graph, and total loss is the summation of all graphs' loss.

Inference: At the inference stage when the model is trained, we use it to test new hardware design, as shown in Figure 3.2. After label prediction, the node labels are passed to HT

localization in the HDL unit. It employs the mapping between graph nodes and HDL signals to mark malicious signals in HDL code based on Trojan nodes in the graph. We also perform HT detection by counting the number of nodes predicted to be Trojan and label the circuit as Trojan-free if the number of Trojan nodes is lower than a threshold and, basically, negligible compared to the size of the design. The user can set the threshold depending on their target sensitivity.

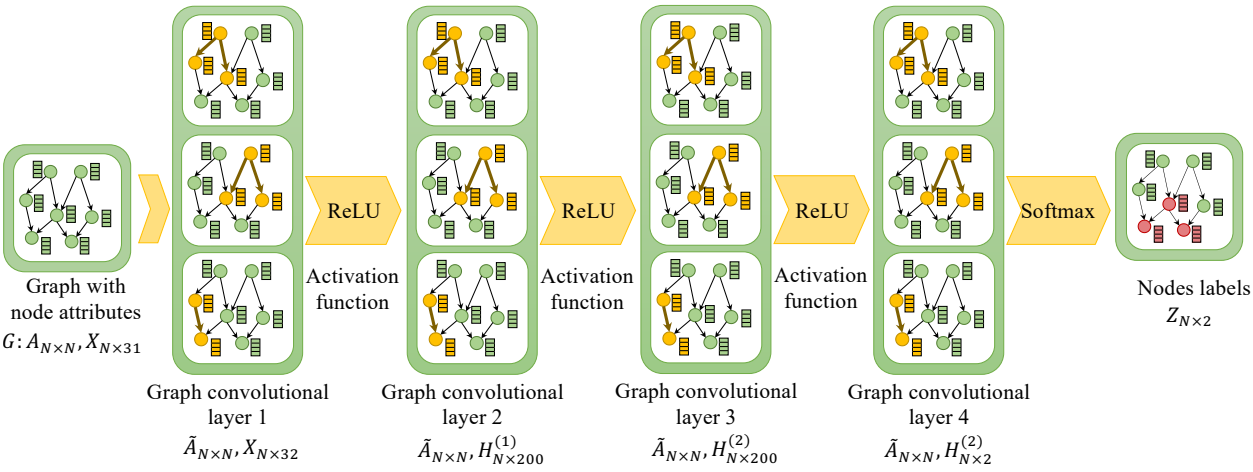


Figure 3.6: The architecture of our GCN model for node classification.

3.6 Evaluation

3.6.1 Experimental Setup

We construct and assess our GCN model on the graph representation of a dataset, consisting of 49 Trojan-infested RTL codes that are listed in Tables 3.2 and 3.3. The limited number of graphs in our dataset is not problematic since our machine learning model is for node classification, and each graph contains thousands of nodes, refer to Table 3.1. An extensive dataset enhances the model's performance and capability to learn a generic knowledge of HT and the learning-based model is easily adaptable by adding new circuits and Trojans to training for further generalization. Our dataset comprises three base circuits that contain

various HTs. Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Rivest Cipher 5 (RC5) are encryption cores with different algorithms that get an input number as plaintext along with a secret key and output the encrypted number known as ciphertext. The AES samples are derived from the TrustHub benchmark [2] which is the most popular open hardware Trojan dataset used in the literature. The RC5 and DES are open-source designs in which we insert the Trojan circuits extracted from AES-Txx benchmarks. However, some of TrustHub HT benchmarks are specific to AES and cannot be inserted in RC5 or DES circuits due to dependency on the internal signals of AES.

Table 3.1: The performance of HT detection.

Circuit	AES	DES	RC5
Classified as Trojan node	1	2	0
Classified as benign node	13437	10210	2106
Total nodes	13438	10212	2106
Classified as Trojan/total	7.44e-5	1.95e-4	0
HT detection accuracy	100%		

In Tables 3.2 and 3.3, the first part of the benchmark name represents the base circuit, and the second part, shows the type of Trojan inserted in the base circuit. For example, DES-T100 shows a DES circuit infected with T100 Trojan from the TrustHub dataset. All the algorithms and models are implemented in the Python language. We use PyTorch and the Geometric extension library to build the graph learning model. The GCN model training and testing are performed by NVIDIA GeForce GTX 1080 graphics card. We use the leave-one-out approach for evaluation. We report test results on a circuit infected with an HT benchmark while the model is trained on other circuits and HTs. We change the test circuit and repeat training on the rest again. The process is repeated until all samples are tested. In this scenario, the circuit under test and its HT are not seen by the model in training which indicates the capability of the model to locate HT in unknown circuits and HTs. In all evaluations, we define the positive sample as the Trojan node class and the negative sample as the benign node class. For example, true positive represents the Trojan nodes that are correctly classified as Trojan.

3.6.2 HT Detection and Localization Performance

After finding the best model and architecture which is elaborated in Sections 3.6.3 and 3.6.4, we construct our final model. The evaluation results per benchmark are reported in Tables 3.2 and 3.3. We also include the number of Trojan nodes and the ratio of Trojan nodes to total nodes in the graph to reflect the effect of HT size. We consider several evaluation metrics to assess the performance from different perspectives. The most common metric for classification is accuracy which expresses the correctly classified nodes over all nodes. Accuracy is intuitive but does not suffice since class distribution between nodes is not uniform. Thus, we look into the F1-score, the weighted average of recall and precision.

Recall expresses the ability to find all Trojan nodes in a design. On the other hand, precision is an indicator of False Positive (FP) and expresses the proportion of the nodes our model labels as Trojan, actually are Trojans. The combination of precision and recall metrics examines the model's performance in detecting Trojan nodes while avoiding mislabeling benign nodes as Trojan. We count True Positive (TP), False Negative (FN), and FP and calculate these metrics as follows:

$$P = \frac{TP}{TP + FP}, \; R = \frac{TP}{TP + FN} \tag{3.5}$$

$$F_\beta score = \frac{(1 + \beta^2) * P * R}{\beta^2 * P + R} \tag{3.6}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{3.7}$$

$$\text{HT nodes} = TP + FN \tag{3.8}$$

$$\text{HT/total ratio} = \frac{TP + FN}{TP + TN + FN + FP} \tag{3.9}$$

We provide a summary of results in Table 3.4 in which the average of metrics are calculated for each circuit as well as the average node classification time. It can be observed that high accuracy and F1-score in HT localization are maintained for all circuits regardless of size. The computation and timing of HT localization depend on the size of the circuit. Studying the timing in diverse designs, it is observed that HT localization time scales linearly with the number of nodes in the graph representation of the circuit, which makes it scalable for large designs. In conclusion, our GCN model exhibits high performance in locating the HT nodes with low false positives (below 0.009%) in less than 1 second. Further, we study the performance of our model in HT detection by testing it for the HT-free circuits of AES, DES, and RC5. The number of nodes classified as Trojan/benign is mentioned in Table 3.1. These results show that our model can determine if the design is healthy as it finds only a few false Trojan nodes in a design graph with thousands of nodes that are negligible.

3.6.3 The Best Graph Neural Network Architecture

There are plenty of graph neural network candidates with various hyper-parameters to choose from as our node classification model. Thus, we devise an experiment in which we construct and test different models to find the best model and architecture for our application. In this experiment, we implement 3 different graph learning models, including GCN [80], graph

Table 3.2: HT localization performance, number of Trojan nodes, and their ratio to total nodes for all AES and DES benchmarks.

Benchmark	Acc [1]	F1 score	Prec [2]	Recall	HT nodes	HT/total ratio
AES-T100	100%	99.4%	100%	98.8%	481	3.46%
AES-T200	100%	99.4%	100%	98.8%	486	3.40%
AES-T300	99.5%	94.7%	98.6%	91.0%	635	5.33%
AES-T400	99.9%	94.2%	100%	89.1%	110	0.71%
AES-T500	99.8%	85.4%	100%	74.5%	94	0.69%
AES-T600	99.9%	89.9%	100%	81.6%	87	0.64%
AES-T700	99.8%	97.8%	100%	95.7%	562	3.98%
AES-T800	99.8%	97.6%	100%	95.2%	628	4.42%
AES-T900	99.8%	97.8%	99.8%	96.0%	569	4.03%
AES-T1000	99.9%	98.4%	100%	96.8%	503	3.73%
AES-T1100	99.9%	97.7%	100%	95.6%	568	3.21%
AES-T1200	99.9%	98.3%	100%	96.7%	509	3.51%
AES-T1300	99.1%	87.5%	100%	77.8%	688	3.87%
AES-T1400	99.5%	94.1%	98.8%	89.9%	723	4.05%
AES-T1500	99.5%	92.1%	98.6%	86.4%	664	3.15%
AES-T1600	99.9%	92.2%	100%	85.5%	179	0.82%
AES-T1700	99.9%	89.0%	100%	80.2%	86	0.45%
AES-T1800	99.9%	83.3%	100%	71.4%	27	0.17%
AES-T1900	100%	82.8%	100%	70.6%	34	0.17%
DES-T100	99.9%	99.2%	99.8%	98.5%	481	4.50%
DES-T200	99.9%	99.2%	99.8%	98.6%	486	4.54%
DES-T400	99.9%	94.3%	98.0%	90.9%	110	1.07%
DES-T500	99.8%	84.8%	98.6%	74.5%	94	0.91%
DES-T600	99.8%	90.0%	98.6%	82.8%	87	0.84%
DES-T700	99.8%	97.6%	99.8%	95.6%	562	5.22%
DES-T800	99.6%	96.7%	98.2%	95.2%	628	5.79%
DES-T900	99.7%	97.4%	99.6%	95.3%	569	5.28%
DES-T1000	99.8%	98.1%	99.6%	96.6%	503	4.69%
DES-T1100	99.7%	96.9%	98.2%	95.6%	568	5.27%
DES-T1200	99.8%	98.2%	99.6%	96.9%	509	4.75%
DES-T1600	99.7%	91.6%	98.7%	85.5%	179	1.72%
DES-T1700	99.7%	83.6%	87.3%	80.2%	86	0.84%
DES-T1800	99.9%	80.2%	90.0%	72.3%	27	0.26%
DES-T1900	99.9%	81.4%	96.0%	70.6%	34	0.33%

[1]Accuracy [2]Precision

Table 3.3: HT localization performance, number of Trojan nodes, and their ratio to total nodes for all RC5 benchmarks.

Benchmark	Acc [1]	F1 score	Prec [2]	Recall	HT nodes	HT/total ratio
RC5-T100	99.8%	99.4%	100%	98.8%	481	18.59%
RC5-T200	99.8%	99.4%	100%	98.8%	486	18.76%
RC5-T400	99.5%	94.2%	100%	89.1%	110	4.96%
RC5-T500	98.9%	85.4%	100%	74.5%	94	4.27%
RC5-T600	99.3%	90.0%	98.6%	82.8%	87	3.97%
RC5-T700	99.1%	97.9%	100%	95.9%	562	21.06%
RC5-T800	98.9%	97.5%	99.7%	95.4%	628	22.96%
RC5-T900	99.0%	97.5%	99.6%	95.4%	569	21.27%
RC5-T1000	99.3%	98.2%	100%	96.4%	503	19.28%
RC5-T1100	99.1%	97.7%	100%	95.6%	568	21.24%
RC5-T1200	99.3%	98.1%	99.8%	96.5%	509	19.46%
RC5-T1600	98.8%	91.9%	99.4%	85.5%	179	7.83%
RC5-T1700	99.2%	88.5%	98.6%	80.2%	86	3.93%
RC5-T1800	99.6%	83.3%	95.2%	74.1%	27	1.27%
RC5-T1900	99.5%	82.8%	100%	70.6%	34	1.59%

[1]Accuracy [2]Precision

Table 3.4: The summary of dataset and HT localization performance.

Benchmark	All	AES-Txx	DES-Txx	RC5-Txx
Accuracy	99.6%	99.8%	99.8%	99.2%
F1-score	93.1%	93.2%	92.2%	93.1%
Precision	99.0%	99.8%	97.5%	99.4%
Recall	88.0%	88.0%	87.9%	88.0%
# of nodes	2000-14000	13438	10212	2106
Time	$<$ 500ms	222ms	162ms	37ms

attention network (GAT) [159], and local extrema convolution (LEC) [132] with different architectures (2-layer to 4-layer). The evaluation results are illustrated in Figure 3.7. F1-score is the main evaluation metric for comparison because it is the average of precision and recall and represents the two key expected qualities; detecting all Trojan nodes and having low false positives. The LEC model shows the worst performance, and by increasing the number of layers, its performance drops. On the contrary, the GCN and GAT models are improved by stacking more layers, while GCN exhibits relatively better performance. Therefore, The GCN model with four layers is chosen for node classification.

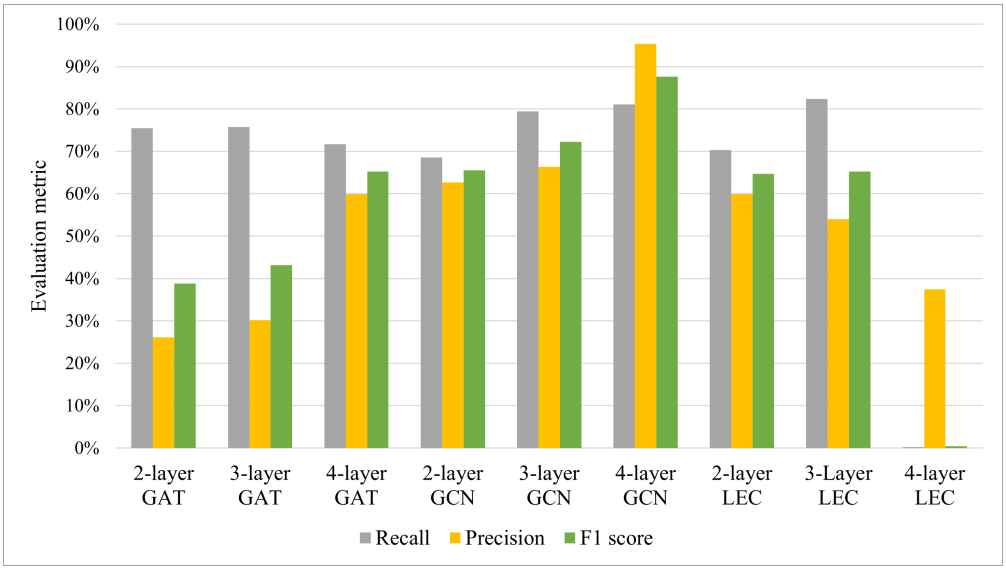


Figure 3.7: Performance of various graph neural network models and architectures.

3.6.4 Compensation for Unbalanced Dataset

A standard step of developing machine learning models is to find the best settings for the model based on the problem. One of the challenges of Trojan localization is the small size of the HT circuit, which results in an unbalanced dataset for machine learning. In our dataset, the ratio of HT nodes to total nodes is between 0.001-0.020 (refer to Tables 3.2 and 3.3), which means the distribution of node classes is not uniform, and one class of nodes is more common. The unbalanced dataset can affect the model's performance and push it to label all nodes as the dominant class, the benign node class. To tackle this problem, we assign a higher weight to the Trojan class in loss calculation that compensates for the minority of Trojan nodes and forces the model to label more nodes as Trojan. We devise an experiment to find the optimum value for class weight by altering the relative weight of the Trojan class to benign class among these values: 1:1 (none), 3:1 (low), 6:1 (high), and 21:1 (super). In the evaluation results in Figure 3.8, we notice that increasing the weight of the Trojan continuously increases the recall as more Trojan nodes are found. Still, after some point, it deteriorates the overall performance (F1-score) as the false positive sample increases, and consequently, the precision drops. Therefore, The best class weight with the highest F1-score

is 6:1 and we use this value for further evaluations.

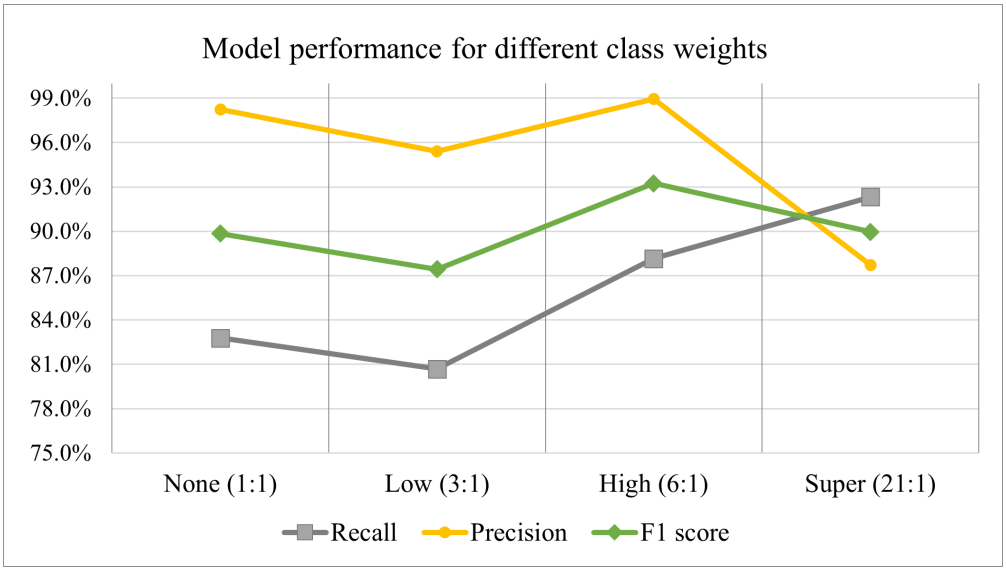


Figure 3.8: Performance of GCN model with different class weights.

3.6.5 Comparing HT Localization Methods

In this section, we compare our model with other HT localization methods in the literature that are elaborated in Section 3.4. u A quantitative comparison is challenging due to a couple of reasons. Firstly, the dataset and experiment conditions are very varied among different works. For example, [68, 160, 185, 52] papers propose various ideas to locate the HT nodes in the circuit, and they demonstrate promising results on their limited sets of benchmarks. However, each one reveals the shortcomings of the former method against distinct Trojans. Secondly, diverse techniques are used for localization with varied evaluation metrics that are not comparable. For example, [75] reports the error in activity estimation, which is further used for marking low-activity regions as vulnerable to HT. On the other hand, [74], our approach demonstrates accuracy in finding the Trojan nodes. Although the direct quantitative comparison is not feasible, we provide a numeric evaluation of different methods' performance in terms of how successful they were in locating HT (using accuracy, recall, and error metrics), how many benign nodes were mislabeled as HT (using false positive

rate and precision metrics) in Table 3.5. The comparison shows the superior performance of our model in successfully detecting HT nodes with very low false positives. The metrics definition is elaborated in Section 3.6.2.

We study them from qualitative aspects such as pre-silicon or post-silicon HT localization, golden reference-free, automated feature extraction/property definition, localization resolution, and the ability to detect various types of Trojans. According to Table 3.5, our compelling model surmounts the shortcomings of the state-of-the-art. The post-silicon techniques postpone the HT localization until after fabrication when the HT removal is very time-consuming and expensive. Therefore, it is crucial to locate and remove Trojans inserted in the design stage early before manufacturing. On the other hand, pre-silicon HT localization approaches mostly suffer from low resolutions because they cannot detect the Trojan nodes specifically. Instead, they mark the suspicious areas that are prone to HT insertion. Thus, they require further manual revision of circuit partitions to check for Trojan nodes, and their localization process is not automated.

3.7 Chapter Concluding Remarks

In this work, we create a novel, golden reference-free HT localization methodology that converts the hardware design to a graph, performs node classification on it using GCN, and outputs the malicious circuit corresponding to Trojan nodes. Our methodology is fully automated without any need for manual feature extraction or code inspection. Our evaluation demonstrates that it locates Trojan with 99.6% accuracy, 93.1% F1-score, and a false positive rate below 0.009%.

Table 3.5: Comparing the HT localization methods in the literature.

Method	Stage	Golden chip-Free	Auto-mated	Local-ization Resolu-tion	HT diversity	Performance
GCN (ours)	Pre-S[2]	Yes	Yes	High	High	HT localization with 99.6% accuracy and 98.9% precision
Social network [74]	Pre-S	Yes	No	High	Low	HT localization with 97.3% accuracy and less than 2% false positive
Code analysis [75]	Pre-S	Yes	No	Low	Low	Activity estimation with less than 2% error to flag low-activity as HT
VeriTrust [185]	Pre-S	Yes	No	Low	Low	HT localization with 100% recall and 11.5% precision
FANCI [160]	Pre-S	Yes	No	Low	Low	HT localization with 100% recall and less than 8% false positive
UCI [68]	Pre-S	Yes	No	Low	Low	HT localization with 100% recall and 7.5% precision
Symbolic algebra[52]	Pre-S	No	No	High	High	HT localization with 100% recall and 74% precision
Thermal map [155]	Post-S[3]	No	No	Low	High	Successfully locates the HTs with less than 20 gates
Path delay [137]	Post-S	No	No	High	High	HT localization with 100% recall and 0.56% false positive rate

Chapter 4

Hardware IP Piracy Detection

4.1 Introduction

The integrated circuits (IC) manufacturing industry has developed significantly and scaled down to 7nm technology that has made the integration of numerous transistors possible. However, the hardware engineers could not keep up with rapid advancement in fabrication technology and failed to use all of the available transistors in the die. To close this productivity gap under time-to-market pressure, hardware Intellectual Property (IP) core design has grabbed substantial attention from the semiconductor industry and has dramatically reduced the design and verification cost [30]. The globalization of the IC supply chain poses a high risk of theft for design companies that share their most valuable assets, IPs, with other entities. IP piracy is a serious issue in the current economy, with a drastic need for an effective detection method. According to the U.S. Department of Commerce study, 38% of the American economy is composed of IP-intensive industries [5] that lose between $225 billion to $600 billion annually because of Chinese companies stealing American IPs mainly in the semiconductor industry, based on the U.S. Trade Representative report [4]. Hardware IP

is considered as any stand-alone component of a system-on-chip design that is classified into three categories based on the level of abstraction: Soft IP (i.e., synthesizable HDL source code), Firm IP (i.e., netlists and placed RTL block), and Hard IP (i.e., GDSII and physical layout) [24].

Conventionally, the IP protection techniques fall into preventive (i.e., logic encryption, camouflaging, metering, and split manufacturing) and detective (i.e., digital signature) methods. All these methods add excessive implementation overhead to the hardware design that limits their applications in practice. Moreover, they mainly focus on security at the IC level, while many commercial IPs comprise the soft IPs due to flexibility, independence of platform technology, portability, and easy integration with other components. The high level of abstraction makes IP protection more challenging since it is easier for an adversary to slightly change the source code and redistribute it illegally at the lower levels of abstraction. Although the existing preventive countermeasures deter IP theft, they cannot guarantee IP security as the adversaries keep developing more sophisticated attacks to bypass them. **Therefore, an effective IP piracy detection method is crucial for IP providers to disclose the theft.** To this end, the state-of-the-art piracy detection method embeds the signature of the IP owner, known as a watermark, and the legal IP user, known as a fingerprint, in the circuit design to assure authorship and trace legal/illegal IP usage. IP watermarking and fingerprinting are prone to removal, masking, or forging attacks that attempt to omit the watermark, distort its extraction process, or embed another watermark in IP [24].

In this chapter, we apply the idea of modeling hardware design to a graph representation from Chapters 2 and 3 and propose a novel methodology for IP piracy detection that, instead of insertion and extraction of a signature to prove the ownership, models the circuits and assesses the similarity between IP designs. Therefore, our method does not require additional hardware overhead as the signature and is not vulnerable to removal, masking,

or forging attacks. It also effectively exposes the infringement between two IPs when the adversary complicates the original IP to deceive the IP owner. Modeling the hardware design is challenging since it is a structural non-Euclidean data type, despite most modeling techniques. Thus, similar to Chapter 2, we represent the circuit as a data-flow graph (DFG) due to similar data types and properties. Afterward, we model it using a state-of-the-art graph learning method.

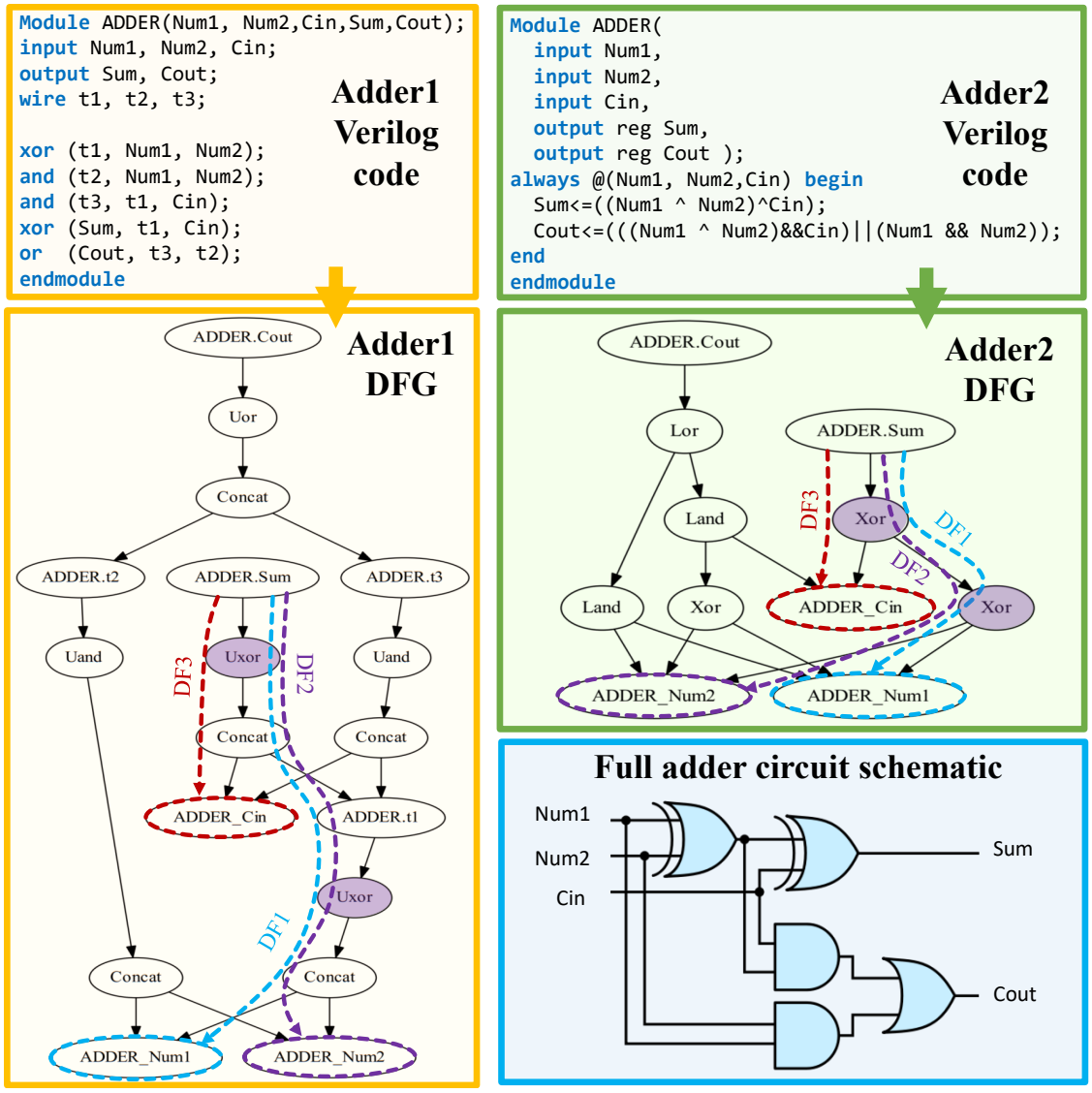


Figure 4.1: The circuit schematic, Verilog codes, and DFGs of full adder circuits.

4.1.1 Motivational Example

We study the concept of piracy and similarity among hardware designs in a test case of two different variations of the full adder circuit. As shown in Figure 4.1, although the Verilog codes for adder 1 and 2 are different, they both have fundamentally the same design, as depicted in the schematic figure. We unveil the similarity between two adders using DFG, which expresses the signals dependency and computational structure. At first glance, the generated DFGs for adders seem varied, but a deep look into data flows (DF) indicates the same signal relations. For instance, the output signal *Sum* depends on *Num1*, *Num2*, and *Cin* input signals through the DF1, DF2, and DF3, respectively. Suppose we focus on critical nodes in the flow (XOR nodes) and ignore the excessive nodes related to concatenation and internal signals. In that case, the DFs in both DFGs represent the same operations.

4.1.2 Research Challenges

The development of an effective IP piracy detection method poses paramount research challenges as follows:

- **Hardware overhead**: All existing piracy detection methods add hardware overhead to IP design.
- **Attacks**: Signatures-based countermeasures are vulnerable to removal, forging, and masking attacks.
- **Same behaviors, different topologies**: As the case study exemplified, varied HDL codes generate different DFGs even if they represent the same hardware design. The different typologies in DFGs can easily fool the standard graph similarity algorithms, and behavioral analysis of graphs is required to learn circuit design.

- **Scalability**: The manual review of hardware design is not feasible in practice. Graph similarity is an NP-complete problem, and existing algorithms [55] suffer from high complexity and are not scalable to large designs and industrial-level IPs with thousands of code lines.

4.1.3 Chapter Contributions

We propose a novel methodology based on graph learning to surmount research challenges and propose these contributions:

- To overcome the shortcomings of current IP piracy detection methods, we propose a novel countermeasure based on hardware design analysis that does not require adding any signature and overhead to IP design.
- We develop a scalable, automated framework called *hw2vec* that generates the DFG for hardware designs and assigns an embedding to them such that the proximity in the embeddings indicates similarity between circuits.
- We construct a Graph Neural Network (GNN) model to learn the circuit's behavior and assess the similarity between a pair of IPs according to graph embeddings.
- We gather a dataset of hardware designs in RTL and gate-level netlist to develop and assess our methodology.

4.2 Backgrounds and Related Works

4.2.1 Hardware IP Security

The hardware is susceptible to security threats such as IP piracy (unlicensed usage of IP), overbuilding, counterfeiting (producing a faithful copy of circuit), reverse engineering, hardware Trojan (malicious modification of circuit) [51, 49, 176], and side-channel attacks [15]. The IP protection methods proposed in the literature can be classified as follows:

Watermarking and fingerprinting [127]: The IP owner and legal IP user's signatures, known as watermark and fingerprint, are added to the circuit to prove infringement.

Hardware metering [82]: The designer assigns a unique tag to each chip, which can be used for chip identification (passive tag) or enabling/disabling the chip (active tag).

Obfuscation [30]: There are two obfuscation methodologies; **logic locking** (encryption) [168] and **IC camouflaging** [129]. In logic locking, additional gates such as XOR are inserted in non-critical wires. The circuit would be functional only if the correct key is provided which is stored in a secure memory out of reach of the attacker. Camouflaging modifies the design such that cells with different functionalities look similar to the attacker and confuses the reverse engineering process.

Split manufacturing [122]: IP house split the design to separate ICs and have them fabricated in different foundries. Thus, none of the foundries have access to the whole design to overbuild, reverse engineer, or perform malicious activities. The existing defenses suffer from a large overhead on area, power, and timing that restrict their application. As the new

countermeasures are developed, the attacks are advanced to bypass them. SAT attack is a powerful method used to formulate and solve a sequence of SAT formulas iteratively to unlock the encrypted circuit, reverse engineer the Boolean functionalities of camouflaged gates [42] or reconstruct the missing wire in 2.5D split manufactured ICs [162]. Anti-SAT [167], and AND-tree insertion [90] obfuscation techniques are proposed to mitigate SAT attacks. However, signal probability skew attack, AppSAT guided removal attack, and sensitization guided SAT attack [177] break them. Proximity attack [128] is another attack against 2.5D split manufacturing that iteratively connects the inputs to outputs in two IC partitions until a loop is formed. Removal, masking, and forging attacks bypass watermarking by eliminating, distorting, or embedding a ghost watermark [24]. There is a rising trend in machine learning-based defenses [176, 174] and the recent advances made the models even resistant against adversarial attacks [14].

4.2.2 Graph Neural Networks

In *GNN4IP*, we leverage GNN, a deep learning methodology that tackles graph data [166]. Several works in the literature have shown the effectiveness of GNN in identifying software clones and detecting binary code similarity [45, 169]. Our architecture is inspired by the Spatial-based Graph Convolution Neural Network, which defines the convolution operation based on a node's spatial relations with the following phases: (i) *message propagation phase* and (ii) the *read-out phase*. The *message propagation* phase involves two sub-functions: **AGGREGATE** and **COMBINE**, given by,

$$a_v^{(k)} = \mathbf{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in N(v)\}), \tag{4.1}$$

$$h_v^{(k)} = \textbf{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k)}), \tag{4.2}$$

where $h_v^{(k)} \in R^{C^k}$ denotes the node embedding after k iterations for the v_{th} node. Essentially, the **AGGREGATE** function collects the features of the neighboring nodes to extract an aggregated embedding $a_v^{(k)}$ for the layer k, and the **COMBINE** function combines the previous node features $h_v^{(k-1)}$ with $a_v^{(k)}$ to output next embedding $h_v^{(k)}$. This message propagation is carried out for a pre-determined number of iterations k. Next, in the *read-out* phase, the overall graph-level embedding extraction is carried out by either summing up or averaging up the node embeddings in each iteration. The graph-level embedding is denoted as $h_G^{(k)}$ and is defined as,

$$h_G^{(k)} = READOUT(\{h_v^{(k-1)} : v \in G\}) \tag{4.3}$$

In our work, we use $h_G^{(k)}$ as the hardware design embedding to assess the similarity between circuits and discover piracy.

4.3 Methodology

In this work, we formulate the problem of IP piracy detection as finding the similarity between two hardware designs. We assume the existence of a feed-forward function f that outputs whether two circuits p_A and p_B are subject to piracy or not through a binary label y as given in Equation 4.4.

$$y = f(p_A, p_B) = \begin{cases} (1,0) & \text{if piracy in } p_A, p_B \\ (0,1) & \text{if no-piracy in } p_A, p_B \end{cases} \tag{4.4}$$

To approximate f, we extract the DFGs G_A and G_B from circuits pair (p_A, p_B) using the *DFG generation pipeline* and pass it to a graph embedding layer, *hw2vec*, to acquire embeddings (h_{G_A}, h_{G_B}). Lastly, our model infer the piracy label, $\hat{Y}$, by computing the cosine similarity between (h_{G_A}, h_{G_B}).

4.3.1 Threat Model

In our threat model, we examine the IP designs in RTL or gate-level netlist to discover piracy. We assume that the design is a soft IP, firm IP, or derived by reverse engineering a hard IP or IC. The adversary can be a hardware designer, competitor company, or the fabrication foundry that presents the stolen IP as their genuine design and sell it as an IC or an IP at the same or lower level of abstraction. The attack scenario may involve modification of IP design to tamper with piracy detection. The attacker can get access to the original IP through one of these means: I) purchase the IP for limited usage, II) leak through a rogue employee in the design house or III) reverse engineer the physical layout or IC.

4.3.2 Hardware Data Flow Graph Extraction

Hardware design is non-Eulicidian structural data that shares similar properties with a graph. We generate DFGs from either RTL code or gate-level netlist as the first step to model it. The DFG is a rooted directed graph illustrating the computation structure and the data flow from the circuit's output signals (the root nodes) to the input signals (the leaf nodes).

It is defined as graph $G = (V, E)$ where $V = \{v_1, v_2, ..., v_n\}$ is the vertices set and each node v_i represents a signal, constant value, or operations such as concatenation, branch, Boolean operators, etc. We define a set of directed edges $E = e_{ij}$ for all i, j such that $e_{ij} \in E$ if the operation v_j is applied on v_i or the value of v_i depends on the value of v_j.

To extract DFG, we develop an automated framework using a hardware design toolkit called Pyverilog [153]. Figure 4.2 demonstrates our DFG generation pipeline that is consisted of five phases: preprocess, parser, data flow analysis, merge, and trim. The procedure begins with preprocessing the RTL code or gate-level netlist in Verilog format to flatten the modular codes and resolve incompatibilities and syntax errors. Afterward, the parser scans the code and produces the corresponding abstract syntax tree used by the data flow analyzer to generate a data flow tree per signal. Next, the signal trees are merged to construct one main DFG for the whole design. Eventually, the redundant nodes and disconnected subgraphs are trimmed, and the final DFG is generated.

4.3.3 Hardware IP Piracy Detection Algorithm

Our IP piracy detection algorithm is shown in Algorithm 2. In the algorithm, *GNN4IP* refers to approximating function f, which can yield the inference of whether two circuits p_1 and p_2 are subject to IP piracy. Applying typical machine learning methodologies to hardware designs, which are non-euclidean in nature, usually requires feature engineering and immense expert knowledge in hardware design. Thus, we propose our scalable, automated IP piracy detection framework, *hw2vec* with an architecture depicted in Figure 4.3.

The *hw2vec* uses the DFG generation pipeline and acquires the corresponding graph G for circuit p in the form of $(\mathbf{X}, \mathbf{A})$ where $\mathbf{X}$ represents the initial list of node embeddings and $\mathbf{A}$ stands for the adjacency information of G. Next, the *hw2vec* begins the message propagation phase, denoted as Graph_Conv in the algorithm, which is Graph Convolution

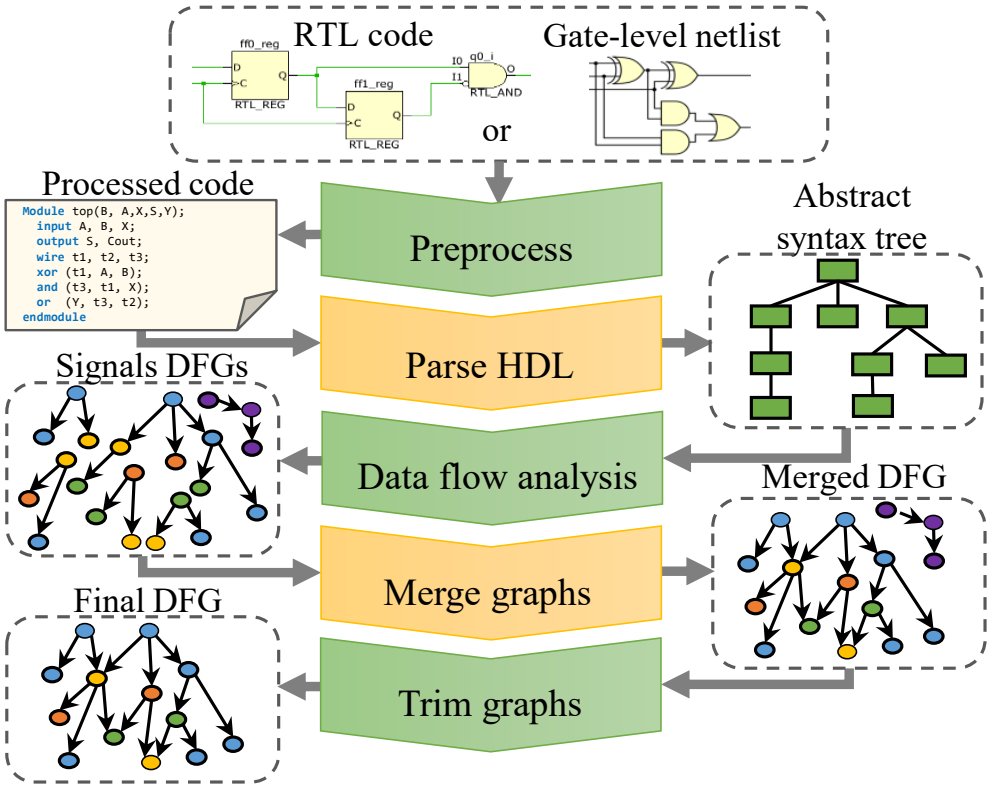


Figure 4.2: Data flow graph generation pipeline for RTL code and netlist.

Network (GCN) [80]. In each iteration l of message propagation, the node embeddings $\mathbf{X}^{l+1}$ will be updated as follows,

$$\mathbf{X^{(l+1)}} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}\mathbf{X^{(l)}}W^{(l)}) \tag{4.5}$$

where W^l is a trainable weight used in the GCN layer. $\widehat{A} = A + I$ is the adjacency matrix of G used in the layer for aggregating the feature vectors of the neighboring nodes where I is an identity matrix that adds the self-loop connection to make sure the features calculated in the previous iteration will also be considered in the current iteration. $\widehat{D}$ is the diagonal degree matrix used for normalizing $\widehat{A}$. $\sigma(.)$ is the activation function such as Rectified Linear Unit (ReLU). Here, we denote the initial node embedding as $X^{(0)}$ and initialize each node embedding $\mathbf{X}_i^{(0)}$, $\forall i \in V$, by directly converting the node's name to its corresponding one-hot vector. We denote the final propagation node embedding $\mathbf{X}^{(l)}$ as $\mathbf{X}^{prop}$, and denote

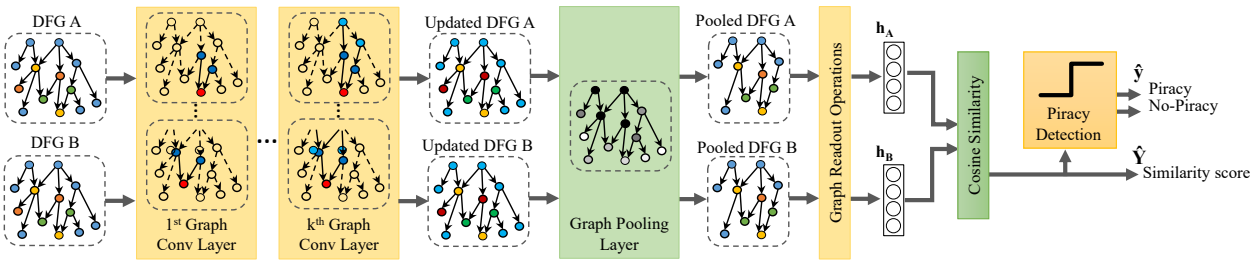


Figure 4.3: The overall architecture of *GNN4IP* for hardware IP piracy detection.

the corresponding adjacency matrix as $\mathbf{A}^{prop}$. Once propagated the information on G, the resultant node embedding $\mathbf{X}^{prop}$ is further processed with an attention-based graph pooling layer Graph_Pool. Denote the collection of all node embeddings of G after passing through L layers of GCN as $\mathbf{X}^{prop}$. The $\mathbf{X}^{prop}$ is passed to a self-attention graph pooling layer that learns to filter out irrelevant nodes from the graph, creating the pooled set of node embeddings $\mathbf{X}^{pool}$ and their edges $\mathbf{A}^{pool}$. In this layer, we use a graph convolution layer to predict the scoring $\alpha = \mathbf{SCORE}(\mathbf{X}^{prop}, \mathbf{A}^{prop})$ and use α to perform *top-k* filtering over the nodes in the DFG [85]. Then, the Graph_Readout in our algorithm aggregates the node embeddings $\mathbf{X}^{pool}$ to acquire the graph-level embedding $\mathbf{h}_G$ for the DFG G using this formula $\mathbf{h}_G = \mathbf{READOUT}(\mathbf{X}^{pool})$. The **READOUT** operation can be either summation, averaging, or selecting the maximum of each feature dimension over all the node embeddings, denoted as *sum-pooling*, *mean-pooling*, or *max-pooling* respectively. Lastly, *hw2vec* returns the embedding $\mathbf{h}_G$ of each hardware.

The *gnn4ip* utilizes *hw2vec* to transform p_1 and p_2 into the corresponding DFG embeddings, denoted as h_{p_1} and h_{p_2}. Then, it calculates the cosine similarity of h_{p_1} and h_{p_2} to produce the final IP piracy prediction, denoted as $\hat{Y} \in [-1, 1]$. The formula can be written as follows,

$$\hat{Y} = \text{Cosine\_sim}(h_{p_1}, h_{p_2}) = \frac{h_{p_1} \cdot h_{p_2}}{|h_{p_1}||h_{p_2}|} \tag{4.6}$$

Finally, our *gnn4ip* utilizes predefined decision boundary δ and $\hat{Y}$ to judge whether two

programs p_1 and p_2 are piracy to one another as described in Algorithm 2 and to return the results of IP piracy detection using a binary label (0 or 1).

Input: Hardware design programs p_1, p_2.
Output: A label indicating whether p_1, p_2 is piracy.
def *hw2vec*(p)**:**
  $X, A \leftarrow$ GraphExtraction(p)
  $X^{prop}, A^{prop} \leftarrow$ Graph_Conv(X, A)
  $X^{pool}, A^{pool} \leftarrow$ Graph_Pool(X^{prop}, A^{prop})
  $h_G \leftarrow$ Graph_Readout(X^{pool})
  return h_G
def *gnn4ip*(p_1, p_2)**:**
  $h_{p_1}, h_{p_2} \leftarrow$ *hw2vec*(p_1), *hw2vec*(p_2)
  $\hat{Y} \leftarrow$ Cosine_Sim(h_{G_1}, h_{G_2})
  if $\hat{Y} > \delta$ **then**
    return 1
  else
    return 0
gnn4ip(p_1, p_2) // run the GNN4IP check.

Algorithm 2: Hardware IP Piracy Detection Algorithm

As both *gnn4ip* and *hw2vec* include several trainable parameters, we need to train these parameters for IP piracy detection via computing the cosine embedding loss function, denoted as H, between true label Y and the predicted label $\hat{Y}$. The calculation of loss can be described as follows,

$$H(\hat{Y}, Y) = \begin{cases} 1 - \hat{Y}, & \text{if } Y = 1 \\ max(0, \hat{Y} - \text{margin}) & \text{if } Y = -1 \end{cases} \tag{4.7}$$

where the margin is constant to prevent the learned embedding to be distorted (always set to 0.5 in our work). Once the model is trained, our algorithm uses the $\hat{Y}$ and a decision boundary δ to make the final judgment of IP piracy.

4.4 Evaluation

In *hw2vec*, we use 2 GCN layers with 16 hidden units for each layer. For the *graph_pool*, we use the pooling ratio of 0.5 to perform top-k filtering. For the *graph_readout*, we use *max-pooling* for aggregating node embeddings of each graph. In training, we apply dropout with a rate of 0.1 after each GCN layer. We train the model using the batch gradient descent algorithm with batch size 64 and a learning rate to be 0.001.

4.4.1 Dataset

One of the significant challenges of machine learning model development is data collection. To construct *GNN4IP*, we gather RTL codes and gate-level netlists of hardware designs in Verilog format and extract their DFGs using our automated graph generation pipeline. Our collection comprises 50 distinct circuit designs and several hardware instances for each circuit design sums up 143 netlists and 390 RTL codes. As our model works on pairs of hardware instances, we form a dataset of 19094 similar pairs and 66631 different pairs, dedicating 20% of these 85725 pairs for testing and the rest for training.

4.4.2 IP Piracy Detection Accuracy and Timing

The *GNN4IP* examines a pair of hardware designs, label it as piracy (positive) or no-piracy (negative), and outputs a similarity score in the range [-1, +1] where the higher score indicates more similarity. We evaluate the model on RTL and netlist datasets, which results in the confusion matrices depicted in Figure 4.4(a). We compute the IP piracy detection accuracy as the evaluation metrics, which express the correctly labeled sample ratio, true positive (TP) plus true negative (TN), to all data. The accuracy and timing results in Table 4.1 show that our model pinpoints IP piracy with high accuracy rapidly, making it scalable

to large designs. The training and testing time depend on the graph size. The longer timing for netlists lies in the fact that in our dataset, the netlist DFGs with 3500 nodes on average are larger than RTL DFGs with 1000 nodes on average. We run the model on a computer with Intel Core i7-7820X CPU @3.60GHz with 16GB RAM and two NVIDIA GeForce GTX 1050 Ti and 1080 Ti GPUs and measure the timing for this computing platform.

Table 4.1: The *GNN4IP* performance for IP piracy detection.

Dataset	**Dataset size**	**# of graphs**	**Accuracy**	**Train time per sample**	**Test time per sample**
RTL	75855	390	97.21%	0.577 ms	0.566 ms
Netlist	9870	143	94.61%	5.999 ms	5.918 ms

4.4.3 Embedding Visualization

The *hw2vec* generates vectorized embedding for hardware designs and maps them to the points in the multi-dimensional space such that similar circuits are in close proximity. We visualize the *hw2vec* embeddings using dimensionality-reduction algorithms such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). Figure 4.4 (b,c) illustrate the embedding projection of 250 hardware instances for two distinct processor designs, pipeline MIPS and single-cycle MIPS, using PCA and t-SNE.

In the PCA plot, the first two principal components are depicted that express the two orthogonal directions, which maximize the variance of the projected data. t-SNE is a nonlinear machine learning algorithm that performs transformations on the data and approximate spectral clustering. We have deliberately chosen two MIPS processors with similar functionality for this experiment to harden the differentiation between them. The processors' contrast lies only in their design and specifications. According to the plots, two well-separated clusters of hardware instances are formed such that data points for the same processor design are close. It demonstrates that *hw2vec* is a compelling tool to distinguish between various hardware

designs. It not only considers the functionality and DFG structure but also recognizes the design.

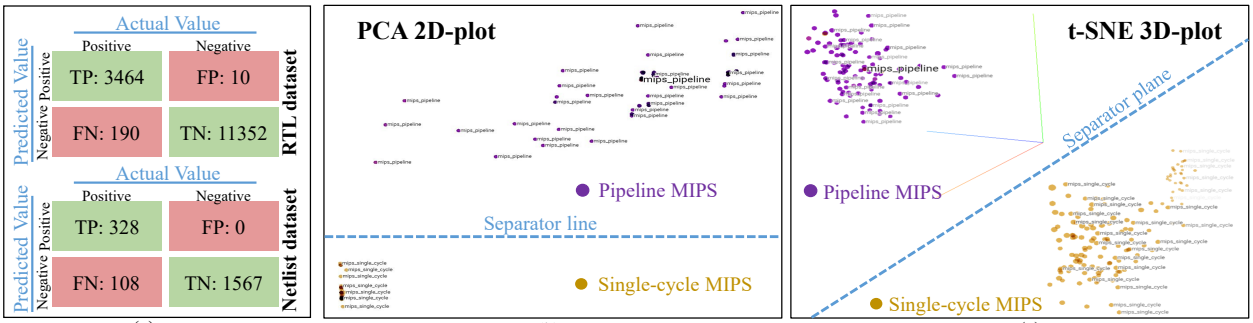


Figure 4.4: (a) Confusion matrices for IP piracy detection, (b) *hw2vec* embedding visualization using PCA, and (c) *hw2vec* embedding visualization using t-SNE.

4.4.4 Similarity Score Results

Our model identifies piracy based on its generated similarity score for two designs, and the decision boundary is controlled by a hyper-parameter δ. We have tuned the δ to achieve maximum accuracy, but the user can adjust it to decide how much similarity is considered piracy. We calculate the similarity score in 3 cases: 1) different designs, 2) different codes with the same design, and 3) a design and its subset. For each case, 4 examples and the mean score for 50 examples are mentioned in Table 4.2. As the results present, our model successfully discriminates hardware designs since the score is very low for different designs (case 1) and close to 1 for similar designs (case 2). In case 3, MIPS is a processor which comprises an ALU block. This relation is captured by the model and resulted in a score of approximately 0.5.

Table 4.2: The similarity score for a variety of hardware design pairs.

Case1		Case2		Case3	
Circuits pair	**Score**	**Circuits Pair**	**Score**	**Circuits pair**	**Score**
AES	-0.2020	AES_1	1	$P.MIPS_1$	+0.5106
FPA		AES_1		ALU_1	
AES	-0.5240	$P.MIPS_1$	+0.9939	$P.MIPS_2$	+0.4965
RS232		$P.MIPS_2$		ALU_2	
AES	-0.0250	$M.MIPS_1$	+0.8362	$P.MIPS_3$	+0.4949
MIPS		$M.MIPS_2$		ALU_3	
FPA	-0.0887	$S.MIPS_1$	+0.9982	$P.MIPS_4$	+0.5460
MIPS		$S.MIPS_2$		ALU_4	
Mean	-0.0831	Mean	+0.9571	Mean	+0.5342

*FPA: Floating Point Adder, P.MIPS: Pipeline MIPS, M.MIPS: Multi-cycle MIPS, S.MIPS: Single-cycle MIPS, X_i: i_{th} instance of hardware X.

4.4.5 Piracy Detection in Obfuscated Netlists

To further evaluate our model, we test it on a dataset of ISCAS'85 benchmarks, and their obfuscated instances in the gate-level netlist format, derived from TrustHub [3]. Obfuscation complicates the circuit and confuses reverse engineering but does not change the behavior of the circuit. Our model recognizes the similarity between the circuits despite the obfuscation because it learns the circuit's behavior. We test this capability in this experiment by comparing each benchmark with its obfuscated instances and computing each benchmark's average similarity score, presented in Table 4.3. In the experimental results, all the similarity scores are very close to 1. It means *GNN4IP* can identify the original IP in the obfuscated design 100% of the time and is resilient against attacks when the adversary manipulates the design to conceal the stolen IP. Furthermore, we assess our model on the pairs of different netlist instances, and the resultant average similarity is very low and closer to -1. It demonstrates that *GNN4IP* is potent in differentiating the varied designs at the netlist level.

Table 4.3: The similarity scores for obfuscated ISCAS'85 benchmarks.

Circuit	Circuit Function	# of circuits	Score
c432	27-channel interrupt controller	24	+0.9998
c499	32-bit single error correcting	23	+0.9928
c880	8-bit ALU	30	+0.9996
c1355	32-bit single error correcting	19	+0.9993
c1908	16-bit single/double error detecting	22	+0.9999
c6288	16 × 16 multiplier	25	+0.9945
Between benchmarks and their obfuscated instances			+0.9976
Between different benchmarks			-0.1606

4.4.6 Comparison with Rival Methods

The current state-of-the-art IP piracy detection method is watermarking. The concept of accuracy is not defined for it, and another metric called the probability of coincidence (P_c) is used. It declares the probability that a different designer inserts the same watermark and depends on the watermark signature size. Although the quantitative comparison with watermarking is not plausible, the false-negative rate provides similar intuition in machine learning. The state-of-the-art [127] outperforms its previous rival algorithms by reporting P_c= 1.11×10^{-87} with the cost of adding 0.13% to 26.12% overhead to design. Our model false-negative rate is zero for netlist and 6.65×10^{-4} for the RTL dataset which is very low and acceptable. Compared to [127], our model has the paramount advantages of zero overhead and resiliency over attacks against watermarking. Moreover, our model is powerful enough to recognize the similarity between designs despite obfuscation.

To the best of our knowledge, we are the first to model hardware as a graph for IP piracy detection. [55] utilizes a graph similarity algorithm to assess obfuscation, similar to Section 4.4.5. Due to different datasets, the exact comparison is not feasible. However, our similarity scores on the obfuscation assessment notably better identify the original IP in the obfuscated one and distinguish the different designs. Their computation time is in order of minutes and significantly slower due to the graph similarity algorithm's high complexity and lack of

scalability.

4.5 Chapter Concluding Remarks

In this chapter, we propose a novel IP piracy detection methodology, called *GNN4IP*, which does not have existing countermeasures shortcomings such as overhead and vulnerability to attacks. Our automated framework extracts the DFGs from RTL codes and gate-level netlist. Then, *hw2vec*, our graph neural network generates embeddings for graphs according to the similarity between designs. Based on embeddings, we infer IP piracy between circuits with 96% accuracy.

Chapter 5

Context-Aware Adaptive Anomaly Detection in IoT through Sensor Association

5.1 Introduction

Shifting from the low-level circuit in Chapters 2, 3, and 4 to the system-level, in this chapter we look into CPS security and apply the insights learned from previous chapters to IoT systems for security and reliability assurance. Over the last decade, IoT has grabbed substantial attention due to advancements in computation and communication, and it is utilized in many applications such as smart home, automotive, and medical aid. The rapid growth of IoT has raised concerns about the security and reliability of these systems. There is a tremendous amount of work in the literature that focuses on various aspects of IoT systems such as communication network [136, 93], hardware security [87, 19, 61, 86] or software security [11, 114, 142, 141]. However, the physical layer of IoT as a cyber-physical system (CPS)

is overlooked. To ensure the security of CPS systems, in addition to a bottom-up security attitude, a holistic approach is required [48, 33, 26, 35].

The ultimate goal of an IoT system is to control the environment and maintain it in the desired state. In order to explain the important role of sensors in fulfilling this goal, we categorize IoT systems under two categories, as depicted in Figure 5.1: (i) a *closed-loop control* system, and (ii) a *monitoring system*. On the one hand, a *closed-loop control system* consists of three major components: (i) sensors; (ii) controller; and (iii) actuators (see Figure 5.1(a)). The sensors monitor the system and send the status to the controller, which processes the sensor readings, decides how to react, and sends the control signals to the actuators to maintain the state of the system and environment.

On the other hand, *monitoring systems* mainly contain sensors that measure numerous parameters in the system and provide the user with information to take proper action (see Figure 5.1(b)). Although a *monitoring system* cannot directly manipulate the environment, it informs a supervising user of events that happen in the system, and the user controls the system manually. Thus, a monitoring system is eventually a part of a control loop.

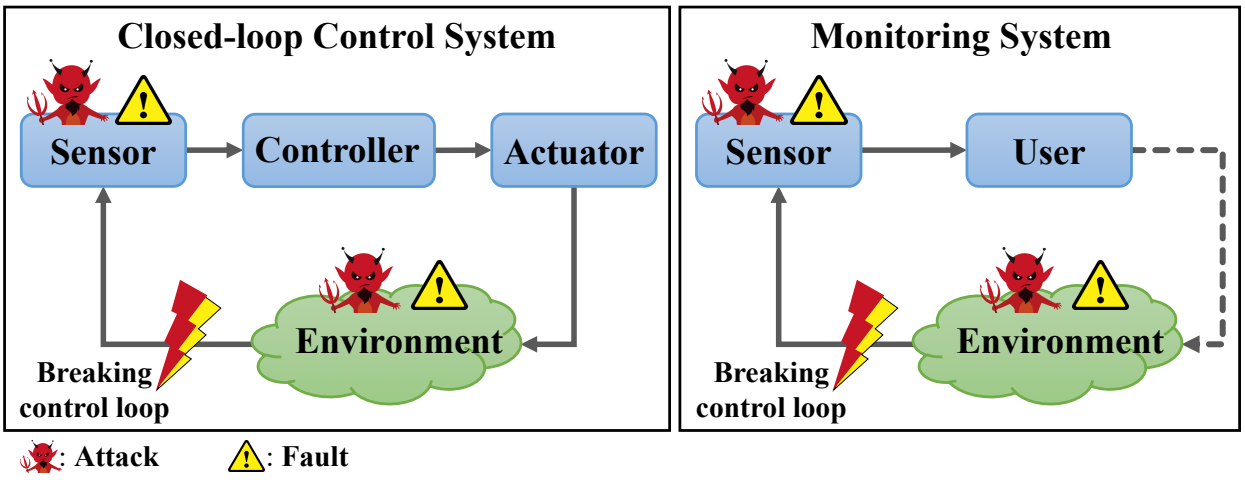


Figure 5.1: Two categories of IoT systems; (a) Closed-loop control system, and (b) Monitoring system.

In both categories, sensors are an essential component of the control loop since sensor mea-

surements determine the action that is needed to maintain the system in the desired state. Malfunction or manipulation of a sensor can break the control loop [16], and consequently, disrupt the services offered by the IoT system. Fault in a sensor device leads to the appearance of anomalous values in its readings, whereas not all anomalies in sensor measurements indicate sensor breakage because an unexpected event in the environment may cause an anomaly as well. Observing the possible anomalies in an IoT system, we present a classification of anomalies that facilitates the identification of the anomaly's source:

- **Environmental Anomaly (EA):** The environment is the area that surrounds the sensor, and the sensor measures its physical properties. Any anomaly in the environment affects the measurements of the sensor and disrupts it. An EA may occur as a result of malicious activities or unexpected incidents in the environment.
- **Sensing Device Anomaly (SDA):** When the operation of a sensor is corrupted, its measurements do not follow the same pattern, and an SDA is observed. This corruption occurs because of either security or reliability issues. For instance, [192, 8] discuss some attacks on the physical layer.

Current anomaly detection methods model the normal behavior of a device [113, 111, 54, 25] and label any deviation from expected behavior as an anomaly. Most of the works concentrate on anomaly detection in the network layer of IoT systems [71]. In spite of reasonable performance in network intrusion detection, these methods have a high rate of false alarms when used with sensor signals. They misinterpret the environmental variation in the sensors measurements as an SDA and disregard the potential information encoded in the relation between the system and the physical world, known as the **context** of the system (refer to Sec. 5.3.1 for the definition of context). Conventionally, context-aware methods are applied to a variety of applications [9], and recently, these methods are used to secure the authentication of co-located devices [116, 115, 161, 62].

We propose an adaptive data-driven model for unsupervised anomaly detection in IoT systems based on sensor measurements. The model monitors the system to detect anomalies, identifies the type of anomaly (SDA or EA), and locates them. To this end, we develop an algorithm to extract the patterns in sensor signals and generate the context of a system. Then, we associate the sensors from different modalities based on the context and cluster the sensors with similar behavior. We develop our customized Recurrent Neural Network (RNN), followed by a consensus algorithm to detect and localize anomalies. The consensus algorithm checks the consistency between sensors in each cluster and determines the type of anomaly. An IoT system has a dynamic structure that is open to changes, such as adding new nodes, removing the existing ones, or updating the framework and protocols. In order to address the variation in IoT systems over time, our model is designed to be adaptive and update itself.

5.1.1 Motivational Example

As a real-world IoT system, we study the environmental training center wastewater plant in Riccione [58]. The primary purpose of wastewater treatment is the elimination of nitrate. Nitrate contamination is a severe environmental problem because it can exhibit toxicity toward aquatic life, present a public health hazard, and affect the suitability of wastewater. In the treatment process, the wastewater is pumped into the tanks, which are equipped with sensors to monitor the concentration of oxygen, ammonia, and nitrate in the water. The actuators, such as blowers and valves, are controlled by a Programmable Logic Controller to adjust the level of chemicals (Figure 5.2(a)). Given the importance of the nitrate level, anomaly detection is applied to detect abnormal changes. Consider two scenarios with an anomalous rise in nitrate level; In the first scenario, environmental changes alter the water temperature, which affects the chemical reactions in the water tank (Figure 5.2(b), an example of EA). In the second scenario, the nitrate sensor is broken or manipulated by an

attacker. (Figure 5.2(c), an example of SDA). The current anomaly detection methods rely solely on nitrate sensor data, whereas the validity of its data is questionable. Thus, they can not find the source of the anomaly and discriminate between EA and SDA.

A recent study [58] analyzes the sensors of this wastewater plant and reveals the correlation between ammonia, oxygen, and nitrate sensor data. More specifically, when the rise in oxygen density reaches a certain threshold, the ammonia concentration decreases, and the nitrate concentration increases. Further investigation reveals the scientific rationale for this correlation; oxygen triggers the chemical reaction, which affects the ammonia and nitrate concentration. By considering this relationship, it is possible to validate sensor signals. In the first scenario, the incident affects all sensors. Despite irregularities in the sensor signals, they are consistent with each other. Thus, we can conclude that the integrity of the sensors' data is not compromised. In the second scenario, the anomaly in the nitrate sensor data is inconsistent with the patterns of other sensor signals. It indicates that the cause of the abnormality is fault or attack. This type of relationship between sensors is not limited to this wastewater plant and it is observed in many IoT systems due to the availability of many heterogeneous sensors.

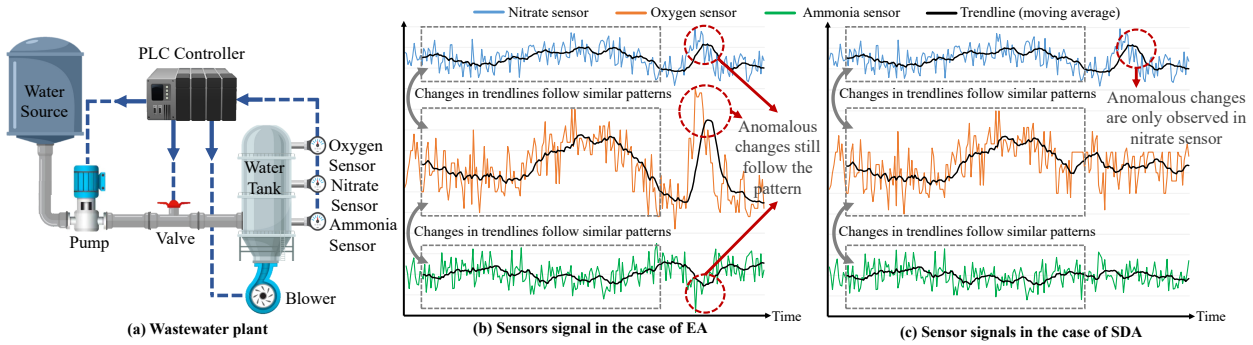


Figure 5.2: (a) Schema of wastewater plant, and synthetic sensors' signals in the (b) first scenario (EA), (c) second scenario (SDA).

5.1.2 Threat Model

The proposed methodology aims to detect SDA and EA, which occur due to an unexpected incident in the environment, reliability issue, or security breakage. Accidental damage, degradation, and defects are examples of plausible reliability problems that cause unintended device malfunctions. In contrast, the security breakage scenario involves an attacker who intentionally exploits the vulnerabilities in the system. In this threat model, the adversary has access to the sensor device and fiddles with it to inject fault, alter functionality, or deny its service. As another possible scenario, the attacker can control the communication channel and send faulty signals to the controller as sensor measurements. The model can detect anomalies in a standalone sensor, but to distinguish between SDA and EA in a sensor, it should be associated with at least two other sensors. To deceive this method, the attacker should be able to discover how sensors are clustered, learn the correlations and patterns in the sensors' signals, and manipulate them in a way that imitates the same correlation as before. It means that in addition to sensors, the attacker should have full access to the clustering layout of sensors and the trained anomaly detection model. It is assumed that the attacker does not have these privileges.

5.1.3 Research Challenges

Anomaly detection in the IoT sensors is challenging due to the following reasons [37]:

- The IoT data are multi-variant time-series data that are collected from a heterogeneous network of sensors with different modalities, data dimensions, sampling rates, specifications, and locations.
- Low-cost and resource-constrained sensors are usually sensitive to noise, and deployment of them in IoT systems affects the quality of data.

- Due to a lack of prior knowledge about possible anomalies and scarcity of anomalous observations, there is not enough labeled anomalous data available, and conventional supervised machine learning techniques are not applicable.
- IoT systems have dynamic characteristics that may be altered over time because of environmental changes, human interaction, mobility of devices, and updating firmware or software. Consequently, a static model fails to imitate the system in the long term.

5.1.4 Chapter Contributions

To the best of our knowledge, this is the first context-aware anomaly detection method for IoT systems. Our novel contributions to address the aforementioned challenges are summarized below:

- **Context-aware sensor association algorithm**: We develop a multi-modality clustering method to associate sensors that experience similar contextual variation.
- **Consensus-based strategy for unsupervised anomaly detection**: We design a methodology to pinpoint the anomalies without reliance on prior knowledge about possible anomalies.
- **Adaptive data-driven model**: Our proposed anomaly detection model is periodically updated at run-time to adapt itself to new states caused by variations in the system.

5.2 Related Works

Anomaly detection algorithms can be classified into the following main categories [58]:

Statistical or Probabilistic Methods: These methods create a statistical or probabilistic model based on historical data, which represents normal behavior [63, 140]. Upcoming observation is then compared with this model, and it is marked as an anomaly if it is statistically unlikely, or the probability of such observation is low.

Proximity Methods: These methods compute distances between data points to differentiate between anomalous and normal data. Two well-known techniques that fall in this category are the *Local Outlier Factor* [20] and *clustering* [66] methods.

Predictive Methods: In these methods, the anomaly detection problem is converted to obtaining an accurate sequence prediction algorithm that captures the recent and long-term trends in data sequences and reproduces them to predict future measurements. Afterward, the predictions are compared with the new observations to spot deviations from expected normal behavior. Recurrent Neural Networks (RNN) are capable of capturing the relationship between measurements over time because the feedback loops in the hidden layer of RNN can imitate memory.

Long-Short Term Memory (LSTM) layer was introduced in 1997 by [70] to overcome the shortcomings of RNN. It has gained a lot of attention lately because of its high accuracy in sequence prediction [113, 111, 25]. Conv-LSTM encoder-decoder is one of the neural network architectures that is used in the literature to enhance sequence prediction performance [92, 96, 179, 165, 99]. It contains convolutional layers to extract the essential features of input sequences and LSTM layers to perform the sequence prediction based on the features. Then, the anomaly is identified based on the reconstruction error of the model. LSTM-LSTM encoder-decoder [112, 125, 193] is another popular architecture that follows a similar strategy but it utilizes LSTM layers instead of convolutional layers for feature extraction.

Our methodology inherits the advantages of both probabilistic and predictive methods. We implement and compare the Conv-LSTM and LSTM-LSTM encoder-decoder as our predic-

tive models. Then, the reconstruction error, derived from the difference between real and predicted values, is modeled by a Multivariate Gaussian Estimator to detect the anomaly.

5.3 Anomaly Detection Methodology

Our proposed methodology (see Figure 5.3) detects SDA and EA in an IoT system to ensure sensing devices operate as they are expected.

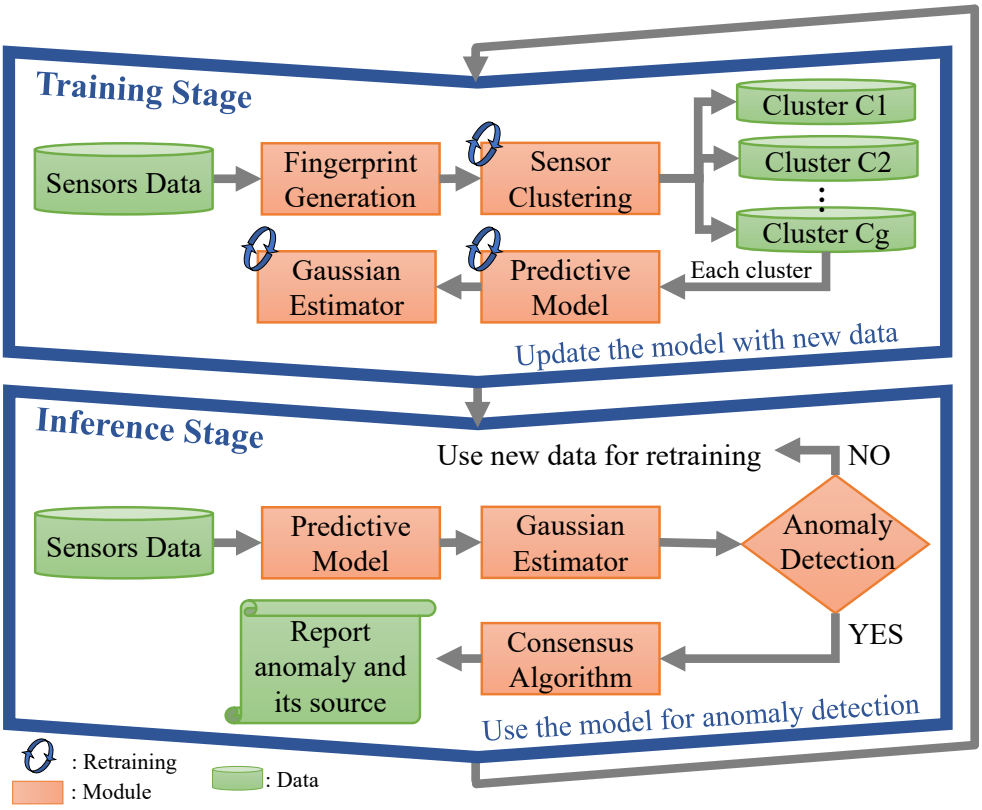


Figure 5.3: The architecture of our methodology in the training and inference stage.

5.3.1 Context Generation

The context of a system is defined as an abstraction formed by extracting features from system circumstances and individual element constructs[98]. It describes the condition in which the system is operating and affects the outcome of the system. The first step for obtaining our context-aware data-driven model is to generate the context of the system by encoding its physical properties. Understanding and transforming this information such that it can be mathematically described is called **context generation**. Following the strategy presented by Sadeghi et al. in [115], we convert all sensor signals to binary fingerprints regardless of their modality. The procedure of fingerprint generation has the following steps:

Step1: Each sensor continuously monitors the environment by taking a measurement each z seconds. The value z depends on the sampling rate of the sensor and may vary for different sensors. In a time window of q seconds from timestamp t, the sensor records $v = [q/z]$ measurements and forms a snapshot vector $S_t = (s^t, s^{t+z}, \ldots, s^{t+z(v-1)})$.

Step2: The ϵ_{rel} is a pre-defined threshold that controls the amount of variation that is said to conform to a change. The values obtained in a snapshot are averaged and the variation bit $b(t)$ is calculated as follows:

$$\overline{S}_t = \frac{1}{v}\sum_{s \in S_t} s,\ b(t) = \begin{cases} 1, & \text{if } \left|\frac{\overline{S}_{t+z} - \overline{S}_t}{\overline{S}_t}\right| > \epsilon_{rel} \\ 0, & \text{o.w.} \end{cases}$$

Step3: Finally, a sequence of k+1 consecutive snapshots $seq(t, t+kz) = (\overline{S}_t, \overline{S}_{t+z}, \ldots, \overline{S}_{t+kz})$, has an associated fingerprint $F(seq(t, t+kz)) = (b(t), b(t+z), \ldots, b(t+(k-1)z))$. The fingerprints of all the sensors with different sampling rates have the same length because each snapshot is the average of sensor measurements in a particular time interval. Figure 5.4 illustrates the process of generating the fingerprint of a temperature sensor.

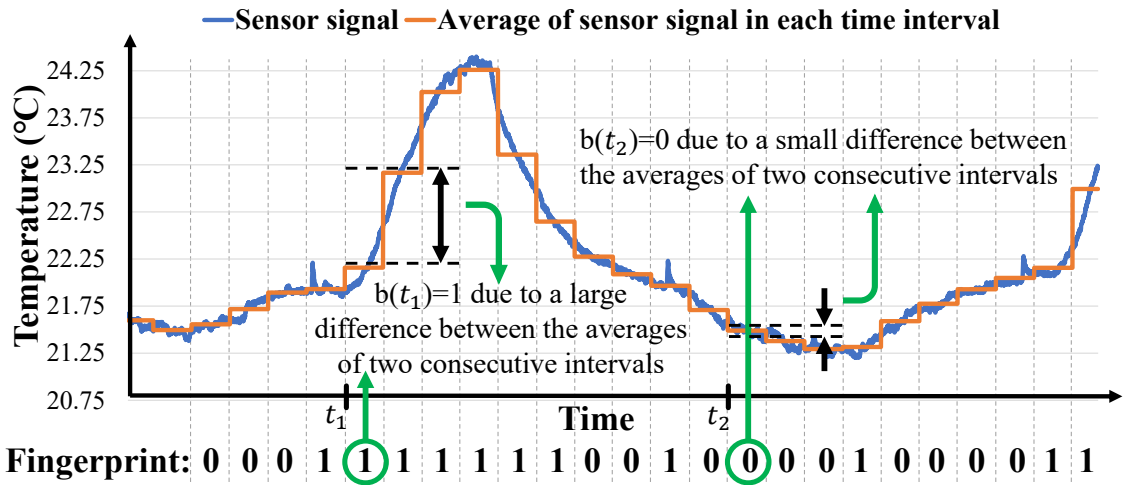


Figure 5.4: Extracting the fingerprint of a temperature sensor.

5.3.2 Sensor Association

Although each sensor's measurements differ based on its modality and physical location, the sensors that are affected by the same event follow similar patterns in their fingerprints. Based on this observation, we develop a sensor association algorithm that comprises two primary steps: I) pattern extraction and II) sensor clustering.

In the first step, we split each fingerprint into smaller sub-sequences, and cluster the sub-sequences of different sensors that have a similar binary pattern. For simplicity, assume that $F_i = F(seq(t, t + zk)_i)$ represents the fingerprint of the sensor i, which is split into d smaller sub-sequences ${f_i}^j$ as follows:

$$F_i \longrightarrow ({f_i}^1, {f_i}^2, \ldots, {f_i}^d),\ d = \frac{k-o}{l-o}$$

where l and o are the hyperparameters that determine the sub-sequences' length and their overlap accordingly. Afterward, our clustering algorithm is performed on the sub-sequences of index j ($j \in [1, d]$) of all sensors $({F_1}^j, {F_2}^j, \ldots, {F_n}^j)$ to group the ones with similar binary

patterns. Hence, it assign a pattern number $p_i^j \in \{0, 1, ..., p_{max}^j\}$ to $f_i{}^j$. Next, the clustering is repeated for index $j+1$, and after d iterations, all sub-sequences are clustered. Notice that p_{max}^j which represents the number of clusters for index j may vary for different index values. Eventually, for each sensor the pattern numbers form a *pattern history vector* $P_i = (p_i{}^1, p_i{}^2, \ldots, p_i{}^d)$.

In the second step, one final clustering is performed on the given set of sensors pattern history P_i to determine the sensors cluster layout $C = c_1, c_2, ...c_g$ where c_i represents a cluster and g is the number of sensor clusters. The sensors with similar contextual variations exhibit the same patterns in many sub-sequences and we cluster them together. An example of the sensor association procedure is demonstrated in Figure 5.5. In this example, the final clustering group the first and third sensors are grouped together.

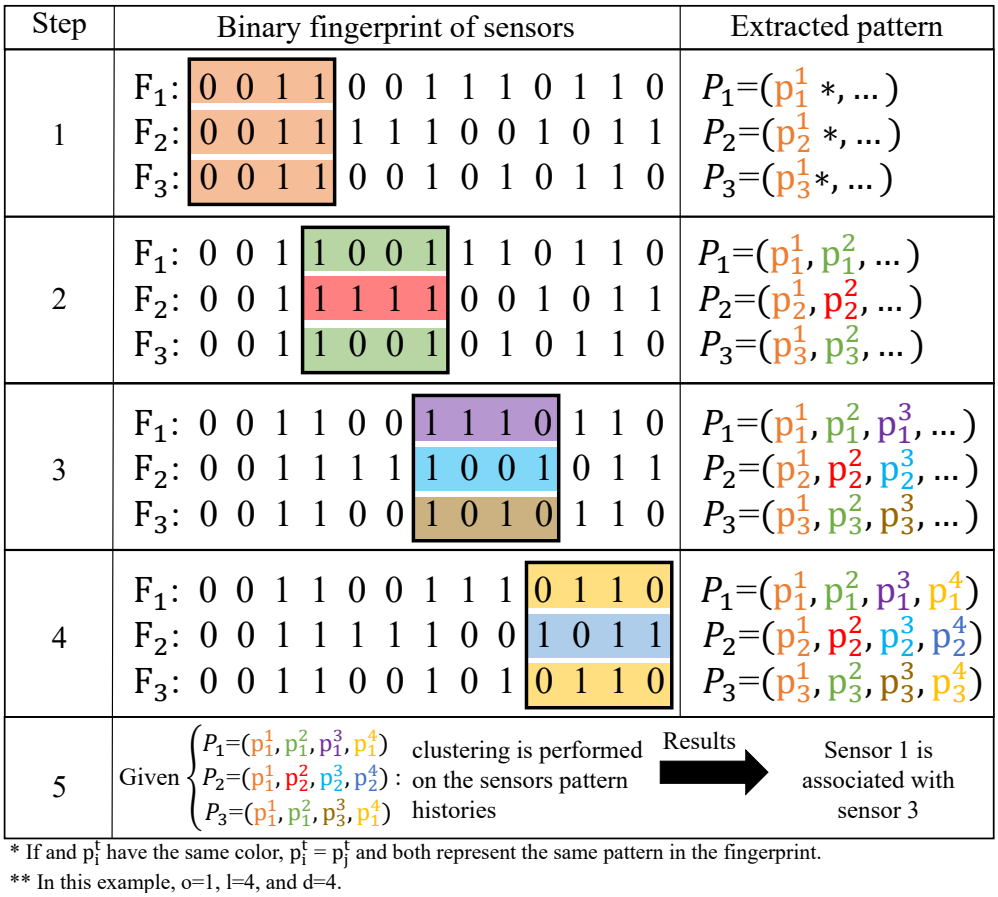


Figure 5.5: The procedure of extracting the patterns in sensor signals and clustering them.

All the mentioned clustering processes are done using our customized clustering algorithm which minimizes the distance between data points in the same cluster, the *intra-cluster distance (IC)*, and maximizes the distance among data points of one cluster from other cluster data points, the *inter-cluster distance (OC)*. The distance metrics are defined as follows:

$$IC = \overline{IC_i},\ IC_i = \frac{1}{|c_i|} \sum_{m,k \in c_i} Hamming(P_m, P_k)$$

$$OC = \overline{OC_{i,j}},\ OC_{i,j} = \min_{m \in c_i, k \in c_j} \{Hamming(P_m, P_k)\}$$

Where c_i and P_m represent a cluster and the pattern history of sensor m respectively. Algorithm 3 is a Pseudo-code that elaborates on our clustering algorithm. Our clustering has the following properties:

- It can be applied to data with string type because the distance metrics are based on the Hamming distance function, which calculates the number of non-matching bits.
- The number of clusters is automatically tuned. Initially, clustering is performed with an upper bound of the number of clusters. Afterward, the algorithm automatically removes the nodes which are not close to any cluster and eliminates clusters with two nodes to reach the optimum value for the number of clusters.

After sensor association, we evaluate the system to ensure that there is no standalone sensor that is not clustered. A standalone sensor is vulnerable because it is not related to any group of sensors that can verify its proper operation. In this case, anomaly detection can still be applied to the independent sensor individually, but the SDA and EA are indistinguishable.

The user is warned about this vulnerability in sensors and can resolve the issue by adding more sensors to the system.

Input: Fingerprints: $F \in IR^{n \times k}$, number of sensors: n, sub-sequence length: l, overlap: o
Output: Cluster layout $C = \{c_1, c_2, \ldots, c_g\}$
Initialize $d = \frac{k-o}{l-o}$
Initialize p_{max} = max # of patterns in sub-sequence
Initialize $iter_{max}$ = max # of iterations
Initialize the center of clusters randomly
foreach $j \in \{1, 2, \ldots, d\}$ **do**
  foreach $i \in \{1, 2, \ldots, n\}$ **do**
    Split fingerprint F_i to obtain sub-sequences ${f_i}^j$;
  Clustering:
  foreach $iter \in \{1, 2, \ldots, iter_{max}\}$ **do**
    foreach $i \in \{1, 2, \ldots, n\}$ **do**
      $p_i^j = Argmin_{x \in C} Hamming(f_i^j, center(x))$;
    foreach $x \in \{1, 2, \ldots, p_{max}\}$ **do**
      $center(x) = mean(\{f_i^j | p_i^j = x\})$;
    if *no changes in* $center(x)$ **then**
      break;
  foreach $x \in \{1, 2, \ldots, p_{max}\}$ **do**
    Calculate inter-cluster (OC) metrics;
    if $Hamming(F_i^j, center(p_i^j) > OC$ **then**
      Remove F_i^j from cluster p_i^j;
      Add F_i^j in *unclustered* nodes;
      Update p_{max}^j;
    if $|c_i| < 3$ **then**
      Remove cluster x;
      Update p_{max}^j;
    Add clusters to pattern histories P_i;
Perform the clustering again on pattern histories P_i, $i \in [1, n]$ to associate sensors;
return Sensors Cluster layout $C = \{c_1, \ldots, c_g\}$

Algorithm 3: Customized clustering algorithm for extracting patterns in sensor fingerprints and sensor association.

5.3.3 Predictive Model

The next module of our methodology is the predictive model that predicts the future measurements of sensors according to the clustering layout and history of measurements. We construct a Recurrent Neural Network (RNN) for each cluster of sensors as the predictive model. As it is depicted in 5.6, our RNN comprises LSTM encoder-decoder and dense layers, which encode the features of input sequences of length l_i and predict the future sequences of length l_o based on the encoded features. Sequences of data are derived from the input time-series signals using the sliding window technique. Afterward, the sequences are scaled through a **Min-Max Scaler** before being treated by the encoder because input signals come from multi-modality sensors with different signal ranges. Eventually, we have a set of predictive models $DT = \{M_1, M_2, \ldots, M_g\}$ where g is the number of clusters in the system and M_i represents the model for cluster c_i. Given cluster c_i that contains n_i nodes, the model M_i takes as input a matrix $X_i \in \mathbb{R}^{l_i \times n_i}$ to predict another matrix $Y_i \in \mathbb{R}^{n_i \times l_o}$. On top of predictive models, **Multivariate Gaussian Estimators** are trained to learn the probability of finding a particular error vector. This probability is used to ascertain whether the errors between predictions and real measurements correspond to the system's normal behavior or an anomaly has occurred. A multivariate Gaussian distributor $G_i = \mathcal{N}(\mu_i, \sigma_i)$ is fitted on the reconstruction error matrix E_i, which is the difference between the real values and predicted values. The parameters μ_i and σ_i are computed using **Maximum Likelihood Estimation**.

$$\mu_i = \frac{1}{m}\sum_{k=1}^{m} e_{i_j}^k = \overline{e}_{i_j},\ \sigma_i = \frac{1}{m}\sum_{k=1}^{m}(e_j^k - \mu_j)(e_j^k - \mu_j)^T$$

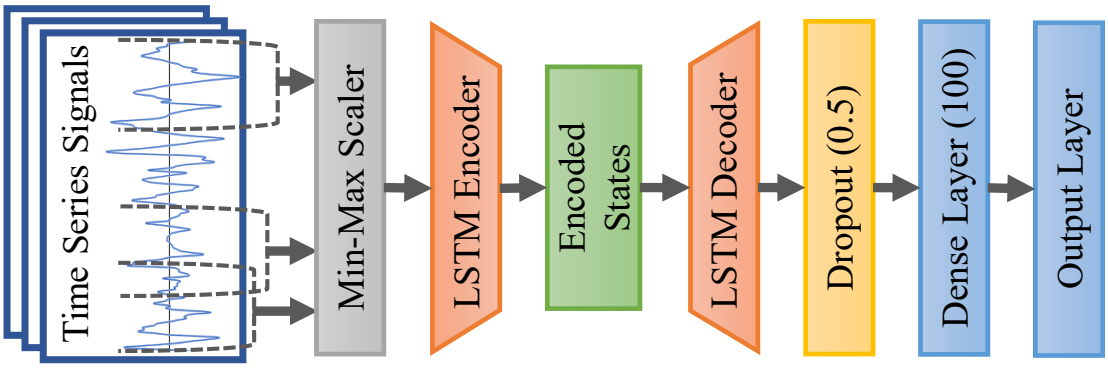


Figure 5.6: The architecture of RNN used as a predictive model.

5.3.4 Anomaly Detection

In the training stage, the predictive models and estimator modules are periodically used at run-time to infer anomalies. The frequency in which anomaly detection is performed can vary depending on the system specifications. At the run-time, an input measurement x^t is compared with model prediction y^t, and the reconstruction error e^t is calculated. Then, x^t classified as anomalous if $p^t < \alpha$, where p^t is the probability of obtaining the error vector given by the Gaussian estimator G. α is a predefined threshold value, and it is tuned to maximize the F-score of the model.

When anomalous data is discovered, we utilize our consensus algorithm to differentiate between EA and SDA. EA occurs as a result of an incident in the environment. If the EA causes an anomaly in a sensor signal, the correlated sensors are affected by the event and show abnormal changes in their signals. In contrast, SDA influences the sensors individually and results in an anomaly in one or some of the sensors in a cluster. For each cluster, the consensus algorithm inspects the consistency of the sensor behaviors. It uses a voting mechanism to check if all sensors in a cluster agree on the occurrence of an environmental incident. To account for inertia in the physics of the system, we check the consensus in the time intervals instead of data points.

5.3.5 Model Adaptation

Due to the high variation in the IoT system and environment, we add the property of **aliveness** to our method, which means the model automatically gets updated to adapt to the system alteration and make more accurate predictions. As Figure 5.3 demonstrates, the sensor association, predictive model, and estimator modules are trainable. There are two levels of updating the model; i)*complete update*, which retrains all trainable modules in order, and ii) *partial update*, which only retrains the predictor model. These update processes are triggered under three circumstances:

- Change in the number of sensors in the system (either added or removed) triggers *complete update*.
- Each time the sensors send data, the anomaly detection model first validates the new data. Afterward, *partial update* is triggered using the new anomaly-free data.
- If *complete update* is not provoked during a fixed interval of time $t_{retrain}$, it is triggered automatically. This way, the model accounts for changes in the environment, location, and placement. This parameter $t_{retrain}$ can be tuned by the user, depending on how frequently the system layout is changed.

5.4 Results and Evaluation

5.4.1 Fog Computing Architecture

Cloud servers are the common and potent available computation resource in IoT systems. However, the bandwidth of network and data transmission become a bottleneck due to the rapid expansion of IoT nodes and the quantity of data. As a result, fog computing has

emerged, which provides storage, computation, and application services closer to end-user with dense geographical distribution [102]. In the fog architecture (Figure 5.7), the bottom layer comprises a heterogeneous network of edge nodes with limited resources. The fog nodes in the middle layer collect and process the data from edge devices and communicate to the cloud via the internet.

Our methodology is fog-empowered, and the developed model for our target IoT system is implemented on a fog node. For IoT systems with a high density of devices and a massive volume of data, our method is scalable, and it still supports fog computing. Basically, the LSTM encoder-decoder networks are responsible for most of the computation in our method. Thus, instead of training an extensive network for the whole system, we construct a small network for each cluster of associated sensors that can be distributed between fog nodes. Furthermore, we perform several optimizations to meet resource constraints. In the sensor association, we use the binary fingerprint instead of time-series signals, which lowers storage usage and complicity. The sliding window technique in the LSTM network contributes to reducing storage usage as well.

5.4.2 Experimental Setup

To build and evaluate our methodology, we implement an IoT testbed in our laboratory. Our experimental setup consists of an Ad-Hoc network of multi-modality IoT sensors, a Software-Defined Radio (SDR) connected to an edge computing device, a gateway, and a laptop as a fog node. For this particular research, we have used 62 sensors that measure 13 different physical parameters (see Table 5.1). The acoustic sensor is a wide-range microphone with two right and left channels that captures the sound of the space and its output is amplified and recorded by the handy recorder ZOOM-H6. The raspberry pi board, which is directly connected to ZOOM-H6, collects its data and transmits it over the Internet and this part

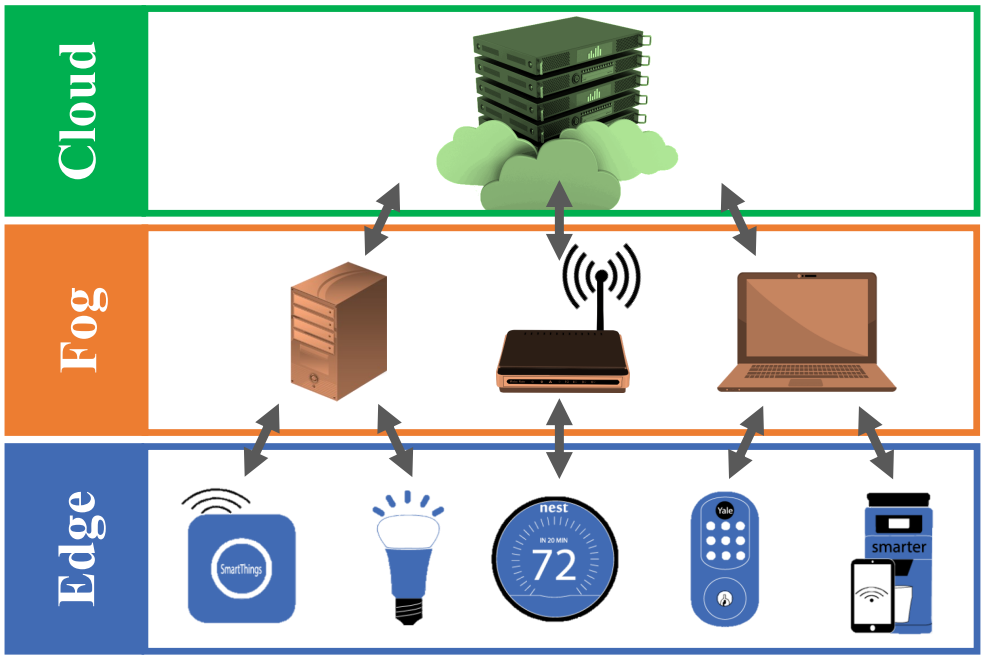


Figure 5.7: Fog computing hierarchy in IoT systems.

of the system simulates devices such as Google Home or Alexa. The other sensors are on the low-power embedded boards operated by TinyOS which are equipped with a wireless communication module based on IEEE 802.15.4 standard. We have implemented the IEEE 802.15.4 standard in the SDR device (USRP-B210) and created a wireless network of sensors in which SDR collects the sensor's data and send commands to them. SDR is connected to an edge computing device, a raspberry pi board, which works as a base station and gathers all data. The base station contains a Wi-Fi module and links the local network of IoT devices to the Internet through a router. It provides the system with the capability to be monitored on any device which is connected to the internet by looking up the base station and logging in using the password. The algorithms and anomaly detection model are implemented on a Laptop with 8Gb DDR4 RAM and the Intel(R)Core(TM) i5-6300HQ 2.3GHz processor, which receives the data from the base station and do the computations as a fog node in the IoT system. A powerful router such as Qotom Mini PC Q500G6 has similar capabilities and is capable of running the model at the gateway level. Figure 5.8 demonstrates the components of our experimental setup and their connections.

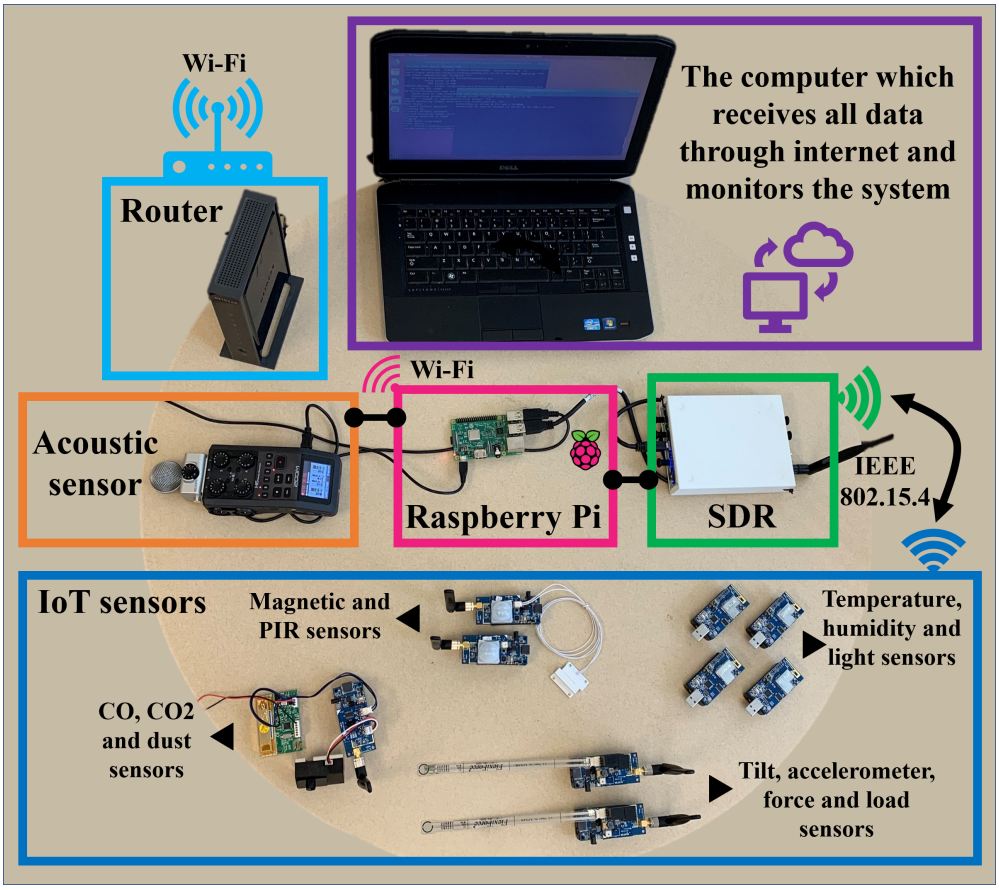


Figure 5.8: The scaled-down version of the experimental setup.

5.4.3 Sensor Association Evaluation

One of the contributions of our clustering algorithm is the capability to automatically tune the number of clusters and remove the ones which lack a sufficient number of sensors or have sensors that are far apart regarding the hamming distance between their fingerprints. Initially, we set the number of clusters to 20 in our system under test, and the algorithm reduces the number to 6. In order to assess the performance of the sensor association method, *Inter-cluster* and *Intra-cluster* distances are calculated for all clusters and plotted in Figure 5.9. The notable difference between the inter-cluster distance and the intra-cluster distance indicates that related sensors are clustered together, and the clusters are well separated from each other.

Table 5.1: List of sensors in our experimental setup.

Sensor	**Sensor board**	**# of sensors**
Temperature	MTS-CM5000	12
Humidity	MTS-CM5000	12
Visible light	MTS-CM5000	12
Infrared light	MTS-CM5000	12
Force and load	MTS-CO1000	2
Tilt	MTS-CO1000	2
Accelerometer	MTS-CO1000	2
Presence detector	MTS-SE1000	2
Magnetic	MTS-SE1000	1
CO_2	MTS-AR1000	1
CO	MTS-AR1000	1
Dust	MTS-SH3000	1
Acoustic	ZOOM-H6	2

Another validation method used is physical intuition, which explains the relationships among the associated sensors. For example, co-located sensors experience a similar context. Therefore, they are expected to be associated with each other. This intuition supports the result of our algorithm in which co-located humidity, temperature, and light sensors are clustered together, as it is shown in Figure 5.11. Another intuition behind the fact is that any physical process may have multi-modality emissions, and the sensors which capture the emission of one incident should be clustered together. It explains the clustering of PIR, vibration sensor (accelerometer and force), magnetic door switch, and acoustic sensor since they all capture the event of entrance through the door. These observations indicate that this strategy is capable of finding relations between sensors with similar contextual variations, further confirmed by the anomaly detection results in the next section.

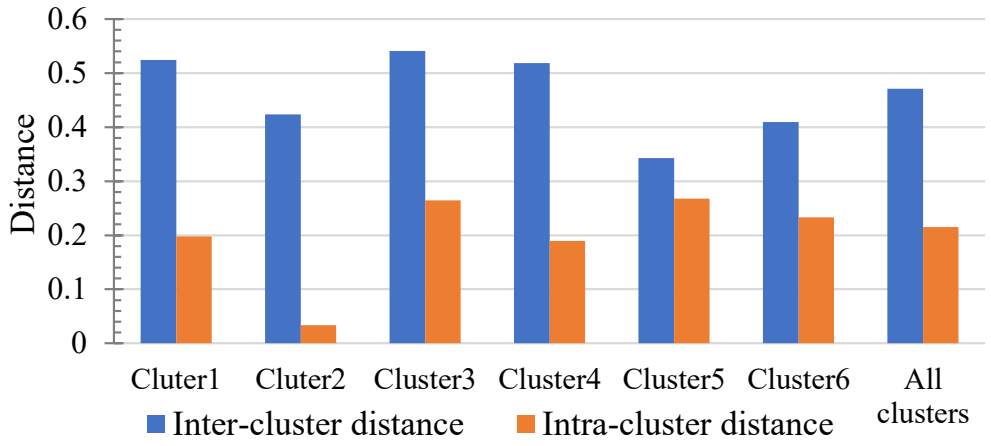


Figure 5.9: The inter-cluster and intra-cluster distances of sensor clusters.

5.4.4 Anomaly Detection Evaluation

The anomaly detection model is unsupervised, and it is trained only on normal data and evaluated using a validation dataset with synthetic anomalies. To analyze the results, True Positives (TP), False Positives (FP), and False Negatives (FN) are counted in the results to compute the validation scores. Although the most intuitive performance measure is accuracy, which is the ratio of correctly predicted observations to the observations, it is not appropriate for unbalanced datasets such as anomaly detection, where one category represents the overwhelming majority of the data points. Therefore, we use the *Precision(P), Recall(R)* and F_β *score* as performance metrics.

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F_\beta score = \frac{P \times R \times (1 + \beta^2)}{\beta^2 \times P + R}$$

Recall expresses the ability to find all anomalous observations in a dataset, while precision expresses the proportion of the observations our model labels as an anomaly, actually anomalous. F_β *score* is the weighted average of precision and recall which provides a better intuition toward both key important capabilities of the model. We implement the current state-of-art methods for anomaly detection in time-series data. Due to the importance of

precision, $F_{0.5}$ *score*, which favors precision over recall, is calculated for evaluation in addition to F_1 *score*. According to the results in Table 5.2, our methodology has the best performance with the highest F *scores* and precision.

Table 5.2: Comparison with the state-of-art methods

Method	**Base Model**	**Context Aware**	**Precision**	**Recall**	$F_{0.5}$ **score**	F_1 **score**
IoT-CAD	LSTM	Yes	%92	%56	%81	%70
[111]	LSTM	No	%64	%44	%58	%52
[100]	Conv LSTM	No	%51	%95	%56	%66
[104]	One Class SVM	No	%89	%25	%60	%39

5.4.5 Robustness

We evaluate the robustness of our methodology by adding three different types -pink, Gaussian, and uniform- of noise signals to the sensor measurements and observing the performance of the model. As Figure 5.10 indicates, although the precision of anomaly detection is decreased as the noise power increases in all models, our model is more resilient to noise and maintains high precision.

5.4.6 Case Study

As a case study, we analyze a cluster of associated sensors, which includes three humidity, temperature, and light sensors located in close proximity. As shown in Figure 5.11, the predicted values are very close to the real measurements which indicates the competency of our method to learn the normal behavior of sensors and predict the future measurements precisely. Furthermore, we observe that the pattern of changes in the sensor signals is similar. As the marked areas of Figure 5.11 highlight, any drop in the trend of humidity sensors comes with an increase in the trend of other sensors. It confirms the correlation among the sensors

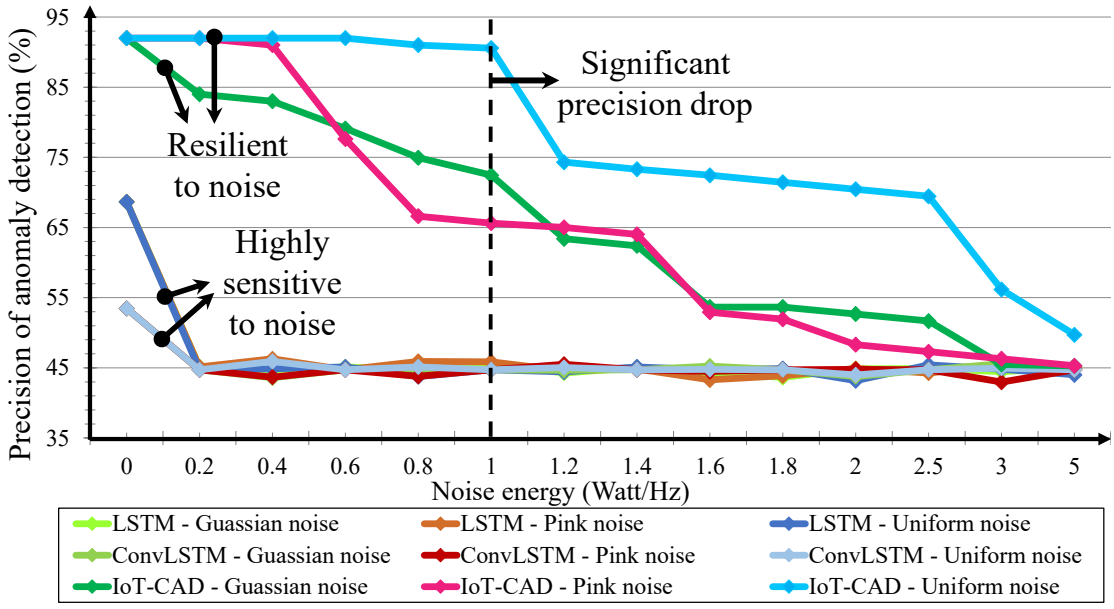


Figure 5.10: Evaluating the resilience of different models to pink, Gaussian, and uniform noise signals.

as the sensor association algorithm suggests. We simulate a fire incident in the environment as an EA, and the measurements from all sensors show an anomaly.

5.4.7 Timing Analysis

The timing of the method depends on the number of sensors, length of time-series signals, and computing platform which is used to implement the model. We implement our methodology on a fog computing platform and train it on data collected from 62 heterogeneous sensors for 8 days (roughly, 2.3 million data measurements). The training stage starts with the fingerprint generation process, which is repeated for all sensors (62 times). The sensor association process involves 604 times performing clustering to cluster the patterns and then sensors. Eventually, the clustering layout and sensor measurements are used for training the predictive model in an iterative process until the convergence of the model. Although the initial training is time-consuming, it occurs once, and the process of anomaly detection on

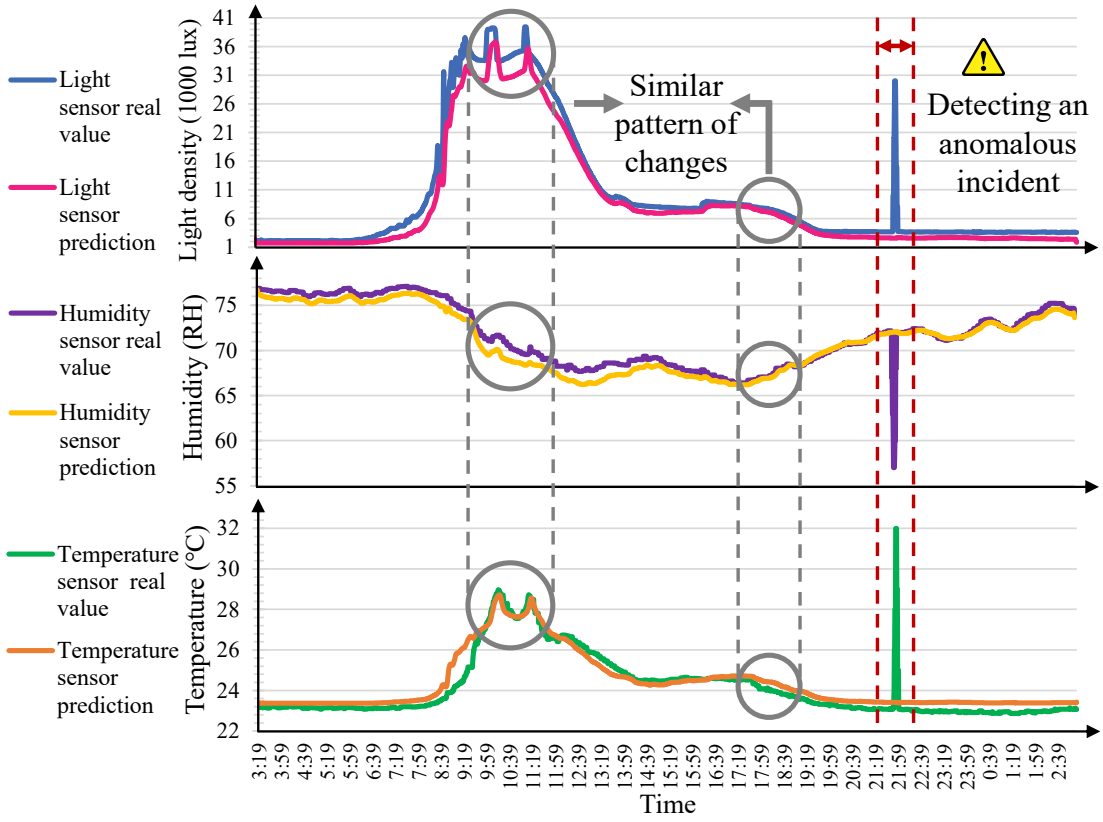


Figure 5.11: The real and predicted values of three correlated sensors; light, humidity, and temperature sensors.

the new measurements using the trained model only takes 0.532 seconds, which means it is real-time in our system under test. As mentioned in section5.3.5, the retraining process is triggered under some conditions, but it is faster than initial training since it is limited to new data and does not interrupt the anomaly detection (Refer to Table 5.3).

Table 5.3: Timing results.

Process	Recurrence	Time (seconds)
Fingerprint generation	62	2.98
Training sensor association	604	10.54
Training predictive model	1	2472.20
Anomaly detection	periodic	0.532

5.4.8 Aliveness Assessment

To assess whether updating the model is beneficial for maintaining the model's performance over time, we test it in three scenarios. The first and second scenarios simulate the effect of system degradation or environmental variation over time. In this regard, the measurements of temperature sensors are increased 5^oC in case 1, and 15^oC in case 2 for a day. The third scenario simulates changes in the layout of the IoT system by eliminating a sensor.

The model is initially trained on the original data before the occurrence of scenarios and tested with synthesized data from the cases. In the tests, we examine the effect of the partial update, complete update, and no update on the precision of the model, refer to Figure 5.12. According to the results, the variation in cases 1 and 2 led to a significant drop in the precision of the model without updating while updating the model, effectively preserving the high performance because of retraining the predictive model. The third case highlights the advantage of the complete update. Any alteration (add or remove) in the number of sensors in the IoT system changes the input layer dimension of the neural network. Thus, the model cannot perform anomaly detection in case 3 unless the sensor association is retrained to update the layout of sensors, and the model is reconstructed on the new layout. Results confirm that the complete update is successful in maintaining the performance despite removing a sensor. Based on this experiment, it can be concluded that being adaptive is crucial for the models used for IoT.

5.5 Chapter Concluding Remarks

This section presents a novel context-aware adaptive data-driven model for anomaly detection in IoT systems. It generates context information by encoding the relations among the IoT sensors and clusters the correlated sensors based on similar patterns, and contextual

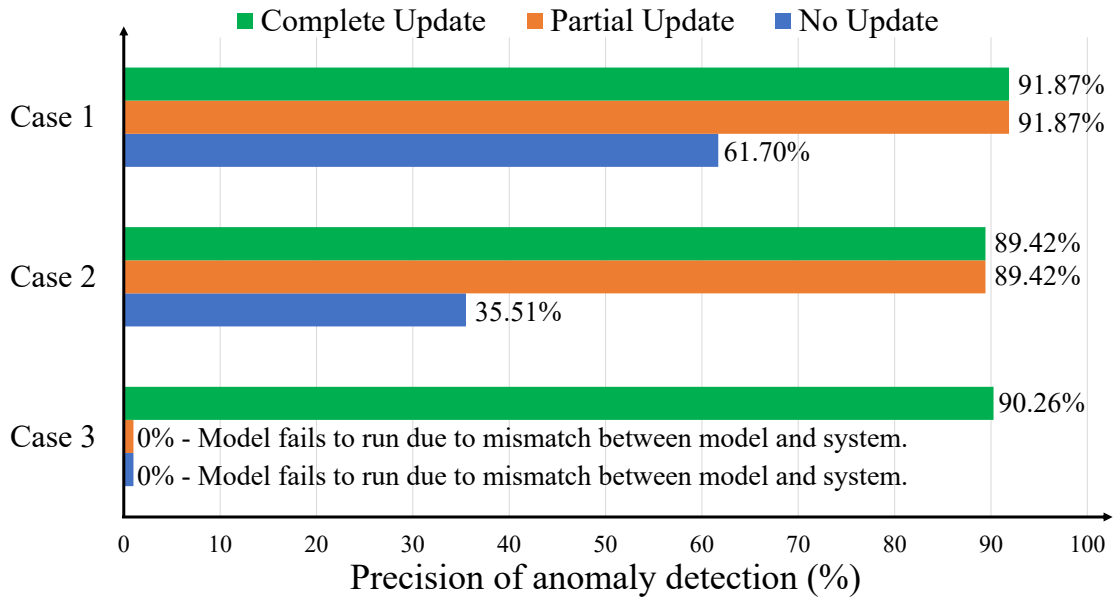


Figure 5.12: Analysis of the effect of the partial and complete update on preserving the performance of the model.

variation. According to the extracted context, a predictive model detects the anomalies, and a consensus-based algorithm determines the type of detected anomalies and pinpoints their source. Our proposed methodology can identify the anomalies with a **92%** precision in real-time on a fog computing platform. Compared with other methods, it has higher performance and the capability to update itself to account for variations in the system and environment.

Chapter 6

Multi-Modal Data Fusion for Anomaly Detection in IoT Physical and Network Layers

6.1 Introduction

The Internet of Things (IoT) presents an interconnected network of sensors and devices that can communicate, observe their internal and external environments and interact with the environment independently of humans. We are surrounded by numerous IoT devices as they are extensively deployed in various applications such as industrial systems, health care, energy management, smart home, and traffic control. Due to the technological advancement in manufacturing cost-effective smart devices, the IoT is undergoing rapid expansion with estimates of a global economic impact of up to $11.1 trillion per year by 2025 [44]. Since IoT systems often embody inexpensive devices with low power and computational capability, the system becomes vulnerable to attack and fault. Consequently, the widespread usage of IoT

and its critical application accentuates the importance of ensuring its security and integrity.

IoT architecture (Figure 6.1) comprises the physical, network, and application layers that handle the interaction with the physical environment, internal communication between devices, and cloud servers.

Chapter 5 introduces a context-aware anomaly detection model tailored to IoT sensors, underscoring the critical role of context and the relationships between components in discerning system behavior. Yet, the multidisciplinary nature of IoT extends beyond the sensor layer, complicating system state monitoring. As such, a comprehensive solution that traverses multiple domains is necessitated. In the literature, considerable approaches have been proposed to detect anomalies in time-series sensor data and many others to perform intrusion detection by network analysis, but the shared context among the sensors and network is overlooked, and a holistic approach is missing to detect and differentiate attacks toward communication network as well as anomalies in data. To close the gap, we propose a multi-modal sensor and communication data fusion methodology, which observes the sensor data and communication data to detect anomalies and attacks in real-time. The methodology proposed in this chapter integrates the understanding gleaned from graph learning as explored in Chapters 2, 3, and 4, along with insights from the context-aware anomaly detection approach discussed in Chapter 5.

6.1.1 Motivational Example

It is plausible for people who use wireless devices in outdoor environments to have different connectivity experiences in the same location on different days of the year. As a matter of fact, multiple pieces of research in the literature study the effect of environmental conditions, such as temperature, humidity, etc., on wireless communication. Notably, in [101], the authors show that temperature has a significant negative impact on the received signal

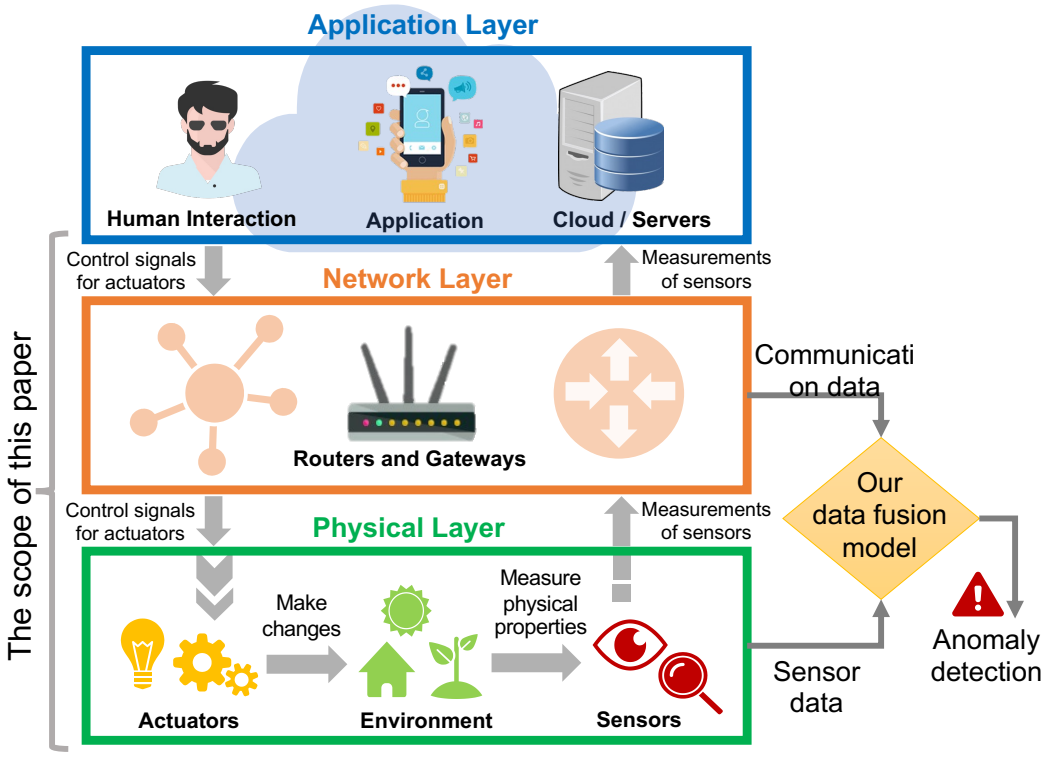


Figure 6.1: The IoT system architecture comprises physical, network, and application layers.

strength indicator (RSSI) while relative humidity is more impactful for below $0°C$ temperatures. Furthermore, their results show that "between the individual channels, there can be large variations in RSSI behavior, both within and between individual links" which they faintly relate to possible multipass propagation in the network. The findings of this paper prove that although building an exact model of the relationship between the environmental factors and RSSI might not be practical for a given environment, a statistical model (e.g., linear regression) can create an explanatory correlation between these two seemingly unrelated measurements, particularly for low power wireless communication networks.

While the correlation between RSSI and temperature/humidity has never been directly used for anomaly detection, the general concept of multi-modal data fusion and anomaly detection has been investigated, which we cover in section 6.2.2. Inspired by this example, in this paper, we introduce a methodology that creates the opportunity to connect/relate different

collectible modality values in an automated fashion without the field expert investigation requirement. Then, we address the challenges of data fusion and use state-of-art machine learning to pinpoint the occurrence of anomalous in an IoT network.

6.1.2 Research Challenges and Opportunities

Several studies have proposed solutions to anomaly detection in IoT, specifically for time-series sensors. Since IoT devices like sensors generate large volumes of time-series data, dimensionality reduction is an essential part of data processing. Therefore, one challenge researchers face is implementing a model that reduces the size of data without a great deal of loss with respect to feature information. Potential problems that are encountered include dimensional explosion and concept drift [29]. As time progresses, more data is collected, which includes noise (e.g., due to system failures). Noisy data can impact the prediction of anomalies, leading to inaccurate results. Moreover, as time goes on, the baseline for what is considered normal or an anomaly may change (concept drift), and this can also affect the accuracy of predicting abnormalities. Anomaly detection becomes even more difficult with multivariate time-series data since the number of features increases the complexity of the system.

As a Cyber-Physical System (CPS), IoT has unique characteristics that arise from intelligent devices' interaction with each other and the physical environment. As a result of these interactions, a mutual knowledge among system nodes exists, known as context. In addition to internal communication, the environment affects the components of the system, and some context stems from shared physical properties. For example, the sensors located close to or monitoring similar physical phenomena tend to be correlated. The context provides invaluable information for comprehending the system's behavior and identifying abnormal incidents. However, infusing the topology and context knowledge into the model is compli-

cated. To tackle its challenges, we employ a graph representation to model the IoT system and perform graph learning using the Graph Neural Network (GNN) for automatic feature extraction and predicting the normal state of the system. Further, we flag the observations that deviate from the normal state as anomalous incidents and trace the anomaly back to its source to identify the underlying attack or fault that has threatened the system's security and integrity.

6.1.3 Chapter Contributions

In summary, our contributions are listed as follows:

- We propose a novel holistic approach to the security and integrity of IoT systems. It performs selective sensor and network data fusion to detect anomalies and attacks on the communication network and physical layout of the system.
- We extract the shared context among multi-modal components of the system through time-series data analysis, and we selectively fuse only related data in the data fusion and machine learning processes.
- We propose a heterogeneous graph representation for IoT systems that embodies sensing devices and communication network parameters as nodes and correlation between component pairs as a connection.
- We leverage the state-of-the-art machine learning model, graph neural network, to automatically extract key features of the system from its graph representation and detect anomalous activities.

6.2 Related Works and Background

Our proposed methodology in this paper closely correlates with three well-established research areas: Network Intrusion Detection, Sensor Data Fusion, and GNN for anomaly detection. In the following subsections, we discuss the related works pertaining to these research areas.

6.2.1 Network Intrusion Detection

Network Intrusion Detection (NID) is an abnormality detection method that analyzes network data to identify deviations from normal traffic patterns. Deep learning approaches in NID have been on the rise due to their ability to take advantage of the most relevant features of the input data. Over the years, various implementations of deep learning models have been employed to overcome a diverse set of challenges for NID. D-PACK [73] proposes using a Convolutional Neural Network (CNN) and an Auto Encoder (AE) for categorizing traffic patterns and identifying anomalies with only the first few packets per flow. While D-PACK achieves nearly 100% accuracy and precision in detecting malicious anomalies, the approach fails to consider memory consumption, particularly with an extensive network. Kim et al. [77] forms a CNN-based model for denial-of-service (DoS) detection. Compared to a previously used Recurrent Neural Networks (RNN) model, their model is able to achieve higher accuracy. [78] uses a local outlier factor and AE to detect anomalies caused by malware intrusion. In [78], the system behavior versus network behavior is modeled separately in order to increase the model accuracy.

Most recently, E-GraphSage [97] proposes a GNN model to capture the flow of network data in addition to the topological information showing the flow's interrelationships endpoints in the graph. IP addresses were the graph nodes, and the network flows were the edges. It

uses the GraphSage algorithm to create node embeddings and also creates edge embeddings. Considering both the node and edge features is what makes E-GraphSage outperform other state-of-the-art classifiers. However, an area of concern is its space and time complexity.

Regarding cyberattacks at the physical layer, such as spoofing and wormhole attacks, [126] proposes an Adaptive Neural Network (ANN) that detects attacks based on changes in the channel characteristics. The model consists of a data-adaptive matrix that records temporal features that are later fed to a CNN to detect spoofing attacks. The authors exploit Received-Signal-Strength (RSS) vectors to train and test the model. They also perform anomaly detection by observing changes in the Signal-to-Noise Ratio (SNR) values. Liao et al. [91] leverage a mixture of machine-learning techniques for physical layer authentication in wireless sensor networks that can resist spoofing attacks. Their approach includes deep neural networks, CNN, and convolution preprocessing neural networks that act per each sensor data node separately and therefore do not leave a significant effect on the communication resources.

6.2.2 Data Fusion

Machine learning has grabbed substantial attention in numerous applications in the last decades, and due to its high potential, it is currently the most approach for data fusion and time-series data analysis. In the literature, CNN and Recurrent Neural Networks (RNN) have been utilized for anomaly detection since both spatial and temporal information are needed to create the model and generate predictions. Yin et al. [178] create a model that integrates a CNN and a recurrent autoencoder to detect anomalies in time-series data. A two-stage sliding window is used as a preprocessing step to extract temporal features so that the later stages could have more temporal relevance. While their results outperformed many other deep learning approaches, their model only works for univariate time-series

data. Forecasting multivariate time-series data is more intricate and complicated with a larger size. This challenging problem calls for a different approach that assesses data from various modalities and considers inter-relation information.

To unravel anomalies arising in periodic multivariate time-series data, Zhang et al. [187] propose Dual-Window RNN-CNN. The purpose of the Dual-Window is to capture the periodicity of the time-series data. An outer window detects anomalies between inner windows according to their time dependencies. Due to the high data acquisition frequency, the multi-head Gate Recurrent Unit (GRU) is used to compress the data and learn the temporal features of the data. The CNN-based Autoencoder then discovers the temporal and spatial dependencies of the features extracted by the GRU. Despite the high accuracy of their model and suitability for high acquisition frequencies in IoT, their datasets, namely Yahoo Benchmark and Numenta Anomaly Benchmark, have a small number of anomalies in the sample sets, and further studies are required [29].

[22] handles multivariate time-series anomaly detection in sensors with a deep convolutional clustering-based model. It uses a deep one-dimensional CNN autoencoder. The authors split the AE latent space into discriminative and reconstructive latent variables. A K-means clustering loss was used for the discriminative latent variables to increase the model anomaly detection performance.

6.2.3 GNN for Anomaly Detection

The typical deep learning approaches for anomaly detection are reconstruction models (i.e., autoencoders- AE or variational autoencoders- VAE) or forecast models (e.g., LSTMs). When they are used for multivariate time-series anomaly detection, they either treat each time-series variable independently or use stacks of CNN and GNN layers to correlate multivariate time-series signals to enrich the data used by the anomaly detection models. While

CNN is mostly used for handling Euclidian data, Graph Neural Networks have been getting attention as a promising method for anomaly detection due to their ability to model interdependencies among data and work with non-structured data.

MTAD-GAT [191] combines a forecasting-based model with a reconstruction-based model. It models the correlations between each univariate time-series feature and keeps track of the temporal dependencies within each signal. The method uses two graph attention layers - a feature-oriented attention layer and a time-oriented graph attention layer. A GRU network is later used to capture long-term dependencies.

GDN [41], an attention-based GNN approach, also focuses on anomaly detection in multivariate time series, specifically sensor data. One of its main goals is to learn the complex non-linear relationships between sensors. The graph structure is constructed by finding the cosine similarity between nodes and creating the edges in the graph by choosing the Top-K highest values. Graph attention-based feature extraction is used to train the model. While the approach is one of the better-performing GNN approaches (high precision, recall, and F1-score), it does not consider temporal dependencies, which can provide crucial information about the behavior of the system.

Combining a transformer with a GNN, GTA [28] uses a transformer-based architecture to learn a graph structure, performs graph convolution, and models temporal dependency for multivariate time-series anomaly detection. GTA places emphasis on learning the topological structure of the graph. There is an anomaly score for each time stamp. In the end, the final anomaly result is determined by multiple different thresholds. Instead of using the sensor correlation, this approach uses the Gumbel-Softmax Sampling strategy to determine if there is a connection between two nodes in the graph. Then, dilated convolutions are applied to extract information from each node to get the long-term temporal context. The outputs are then fed to the transformer. While the results for recall and F1-score outperform many of the state-of-the-art approaches, the model seems to have a tendency for false positives as

the precision tends to be low.

In general, data on a network can be massive and multi-dimensional, which slows down the training of the neural network. This becomes even more challenging with time-series data since the data volume grows over time. Dimensionality reduction is the key to combating this issue. However, this must be done in a way that extracts necessary features from the data so as to avoid compromising the accuracy of the model. Recently, there have been works that use knowledge graphs in order to facilitate feature extraction and anomaly detection. A knowledge graph stores data in RDF format (a standard data format for metadata), meaning its nodes are entities, and the edges represent the relationships between them. They are useful in machine learning because of their ability to capture information between entities in a system. Comparing unforeseen data to the pre-existing data in the graph unravels the anomalies.

Garrido et al. [57] use machine learning with knowledge graphs in order to detect anomalies in industrial automation systems. Their approach constructs a knowledge graph from the database that continuously stores the data collected from their industrial automation prototype. The knowledge graph combines information about the automation system, observations at the network level, and observations at the application level. Data is collected to form a baseline and is then used to perform unsupervised training on the link-prediction algorithm. The anomaly detection is evaluated by introducing events that would not occur during normal operation. While the model does seem to make predictions that are in alignment with what is expected, there was not much statistical information provided that could give a better evaluation of the model in such a way that it could be compared to similar approaches.

Yang et al. [171] also use knowledge graphs to address the problem of relevant feature extraction regarding the semantic relationships between network features of network requests. These network requests include both normal and anomalous requests, the attack types in-

cluding DoS, Probe, R2L, etc. In addition to a knowledge graph, the model uses statistical analysis feature-based extraction. An attention-based CNN-BiLSTM then learns the traffic features. While their approach beats other state-of-the-art methods in the precision, recall, and F1-score, it does perform the slowest.

Although the current methods in the literature have demonstrated promising results in anomaly detection in multivariate time-series sensor data, a holistic model is missing to fuse multi-modal data from communication networks and sensors. This deep data integration in our approach equips the model with the system context shared between the physical layer and network layer, which is further leveraged for anomaly and attack detection.

6.3 IoT Security and Data Analysis

We aim to integrate physical and network layers of IoT systems and detect security threats and malfunctions in the system, which appear as anomalies in the data. To achieve this goal, we first analyze the multi-modal IoT data and examine the potential security and integrity breaches. Next, we simulate these incidents and imitate their impact on the data as anomalies. In the following, we describe the details of our study and the anomaly injection process.

6.3.1 IoT Network Security Analysis

In this section, we discuss the most common attacks against Long Range Wide Area Network (LoRaWAN), which is deployed in IoT systems, such as the greenhouse monitoring system [146]. Moreover, we study the impact and signature of different attacks and further leverage these signatures on communication metadata, such as RSS and SNR, to unravel the attack.

LoRa is a wireless modulation technique upon which LoRaWAN is built. LoRaWAN is a low-power, wide-area networking protocol that allows IoT applications to use LoRa to transmit and receive messages. IoT networks lack robust security due to low power and low cost, so they cannot afford to protect against cyberattacks [60] such as spoofing, jamming attacks, replay attacks, and wormhole attacks.

Spoofing: Spoofing in the context of IoT attacks is when an attacker disguises themself as a trusted node to gain access to the system and modify or corrupt data. LoRaWAN is especially vulnerable to acknowledgment (ACK) spoofing, as demonstrated by [170]. Sheng et al. [144] analyze the Received Signal Strength (RSS) to distinguish between the attacker and the victim when the devices are substantially distanced. There is a correlation between RSS and transmission power and the distance between transmitter and receiver. Therefore, an extreme change in RSS values indicates a spoofing attack because devices do not frequently alter the transmission power.

Jamming attacks: Jamming attacks are a type of DoS attack that involves a malicious node that causes intentional interference. The most generic form of jamming involves the attacker continuously transmitting a powerful signal, which blocks communication along the channel [124]. While continuous jamming affects all devices in the network with the same frequency, selective jamming does not. In selective jamming, the attacker targets a specific device in the network and blocks all communication to and from it without impacting the other devices in the network. Selective jamming can be more difficult to detect because it is less evident whether or not there has been a malicious attack or technical issue.

[13] proposes a selective jamming architecture that uses an RFM95 radio module and a microcontroller to detect and receive LoRaWAN packets. Then, it performs the jamming attack if the software determines that the particular message must be jammed. It is important to note that the smaller the packet size and Spreading Factor (SF) and the lower the Received Signal Strength Indicator (RSSI) of the jammer, the more difficult it is for a

jammer to jam a device. The smaller packet size is easier to transmit; therefore, it spends less time on-air, so the packet is not vulnerable for a long time. The SF gauges the range of data transmission. A lower SF increases the data rate and decreases the time-on-air, making it harder for the jammer to capture the packets. Furthermore, a jammer must have a large enough RSSI to fulfill its goal by introducing enough noise into the channel to cause interference. Despite these potential issues, more than 98% of the jamming attacks produced by [13] were successful. Regardless of the type of jamming attack, the main goal of a jammer is to increase the SNR, which reflects whether or not a jamming attack has occurred. SNR is defined as P_{signal}/P_{noise}, where P_{signal} is the average power of the input signal and P_{noise} is the average power of noise. When the jammer broadcasts a signal, it introduces noise into the network, decreasing the SNR and complicating decoding packets. Hence, jamming often leads to packet loss [117].

Replay attack: A replay attack is when an attacker gains access to the data transmission, intercepts it, and then repeats or resends the message. This allows the attacker to manipulate the data and metadata. Sung et al. [152] present a method that protects the end device in LoRaWAN against replay attacks using RSSI and hand-shaking techniques. Looking at the RSSI value by itself was only useful when the end device and attacker have similar directions and distances. If the RSSI is continually changing, then that is an indication of a replay attack.

Wormhole attacks: In wormhole attacks, a malicious device captures a data packet and sends it to another malicious device through a low-latency tunnel. The device that receives the data packet can replay it multiple times. A specific kind of wormhole attack necessitates a sniffer and a jammer. A sniffer reads the data within the packet, after which a signal is sent to the jammer. The jammer prevents the packet from reaching the desired gateway. Wormhole attacks give attackers the opportunity to tamper with sensor metadata, such as RSSI, SNR, travel time of the packet, etc. In [67], the authors describe an adaptive data rate

spoofing attack that can be realized by an attacker sending messages through a wormhole, thereby changing their metadata. As a result, the end device's RSSI and SNR will increase.

6.3.2 Anomaly Implementation

In order to evaluate our approach and learn the signatures of various security threats, we simulate multiple incidents in the IoT system and their influence on the data gathered from sensors and networks. To imitate the incidents, we study the typical attacks on the IoT system and learn their marks on the system, presented in section 6.3.1. Based on our findings, we simulate spoofing, jamming, replay, wormhole, and physical sensor attacks and inject different synthetic anomalies into 10% of the data. In the following paragraphs, we elaborate on different anomaly injection procedures.

Network Anomalies: For each node, we randomly choose indices (timestamps) in the data where the anomalies would be added. Then, for each index, we randomly select a type of anomaly to inject. For example, to simulate spoofing, we increase the RSSI and add a data anomaly (the injection of data anomalies is discussed in greater detail later). We randomly assign a value for the RSSI between -40dBm and -30dBm inclusive. If the selected attack is a replay attack, we inject a data anomaly and simulate packet dropping by randomly choosing a time difference between and including the period and twice the period; this represents the gap in time caused by packet drops. We alter the RSSI, SNR, and humidity and temperature value for wormhole attacks and simulate packet dropping. First, we change the RSSI by randomly picking a high RSSI (i.e., randomly select a value between -40dBm and -30dBm inclusive) or a low RSSI (i.e., randomly select a value between -100 dBm and -90 dBm inclusive). Next, we lower SNR to show an upsurge in noise by randomly selecting a value between -30dB and -20dB inclusive. Eventually, the jamming attack is executed by dropping the SNR and simulating packet drops.

Sensor Anomalies: The anomalies in sensor measurement fall into two categories [83]: global outliers and contextualized outliers. Global outliers are any values that are significantly smaller or bigger than the rest of the values. It is important to note that the global outliers are still in the sensor output range and feasible to be recorded by the sensor. These become values for the smallest anomalous value, *smallest_global_value*, and largest anomalous value, *largest_global_value*. The upper bound for small global outliers is the *lower_outlier*, and the lower bound for large global outliers is the *upper_outlier*. The *lower_outlier* is computed by first multiplying 1.5 by the interquartile range and then subtracting that value from the first quartile. The *upper_outlier* is calculated similarly, but instead, we add the product to the third quartile. Ultimately, we pick a global outlier by randomly choosing a value from (*smallest_global_value*, *lower_outlier*) and another from (*upper_outlier*, *largest_global_value*) and inject it as an anomaly.

The other type of sensor anomaly is a contextualized outlier, a marked deviation relative to the average of a particular range of values. At a particular instance in time, X_t, the context is considered to be all values within the range (X_{t-l}, X_{t+l}), where l is the length of the context. The anomalous value, A, is computed as follows:

$$A = \overline{X}_{t-l,t+l} + (\lambda * \sigma) \tag{6.1}$$

where λ is the contextual threshold and σ is the standard deviation of the values in the sub-sequence.

6.4 Multi-modal data fusion

We present our methodology in this section and elaborate on the pipeline depicted in Figure 6.2. The data fusion process starts with context extraction from the sensor and commu-

nication data and unraveling the relation between different elements. Next, we integrate all the elements and the context of the system into a graph representation enriched with sensor and communication data, appearing as node embeddings. Eventually, our GNN model extracts the system's key features and learns its normal state for forecasting. It is later used to measure system deviation from the normal state and anomaly detection.

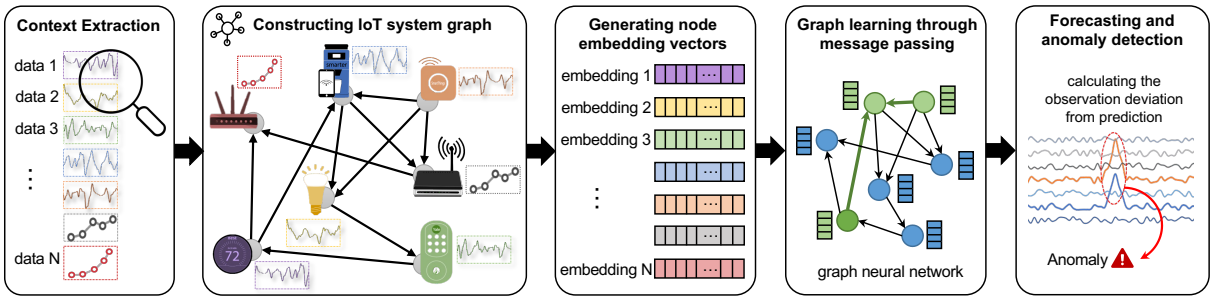


Figure 6.2: Multi-modal data fusion pipeline for anomaly detection in IoT systems.

6.4.1 Data Preprocessing

As part of the preprocessing of the multivariate time-series dataset, we ensure that data is periodic and continuous, and lastly, we synchronize the data. After analyzing the data, we found that the period of each sensor was the same. Some sensors had gaps in the data (i.e., there were missing data for two hours). We chose a threshold for the maximum allowable time difference in minutes. Any time differences less than or equal to the threshold but greater than the period would be made periodic by filling the missing data with the value at the previous timestamp.

In order to determine which sensors are synchronized, we needed to make the data continuous to ensure the number of timestamps for each sensor would be the same. This was done by removing the seconds from the timestamps and filling the gaps in minutes with the value at the previous timestamp so each timestamp would be a minute apart. We also replaced any outliers with a previous value. Then, we found the maximum start time between all the

sensors whose start times began on the same date and ensured all of them began at that time. The final step toward synchronizing the sensors with the same start time was to find the minimum end time between the sensors and cut all the other sensors at that time. As a result, all the sensors with the same start time were the same length.

6.4.2 Context Extraction and Graph Generation

GNN is a type of neural network model that learns information from data that is represented as a graph. The motivation behind GNNs is to be able to apply deep learning methods to non-Euclidean data. Graphs are defined as $G = (V, E)$, where V is the set of nodes, and E is the set of edges between nodes. We represent the relationship between sensors with a directed graph, where the nodes are sensors and the edges show which nodes are related.

We propose a graph structure representing the IoT system in which the components and their generated data (sensor or communication) are incorporated as nodes. The graph presentation of our greenhouse IoT system under study is depicted in Figure 6.3. As context is paramount in modeling an IoT system, we enclose the context as graph edges between nodes indicating the components' correlation. These edges influence the feature extraction in the graph learning process and steer data fusion selectively between related data instead of the whole network.

In our graph presentation, we use an adjacency matrix, A_{ij}, to store the relationships. The relations either come from the system expert, who inputs the potential correlation due to intuition and understanding of the system, or are extracted automatically through data analysis. We test two methods for data analysis: the cosine similarity and the correlation matrix.

By default, all sensors are assumed to be related, in which case A_ij is filled with 1's. However,

Figure 6.3: The graph representation greenhouse IoT system.

the user can also specify which sensors are related and which are not. In this case, if sensor i is not related to sensor j, then the adjacency matrix at row i and column j will be filled in with a 0.

To assess the similarity among component data, we calculate the cosine similarity of node embeddings. Inspired by [41], we find the cosine similarity between each node embedding, x_i and every other node, x_j, to produce the consequent embedding, e_{ji}. We construct an adjacency matrix, A_{ji}, from the cosine similarities and use it to construct the layers of the model:

$$e_{ji} = \frac{x_i^T x_j}{|x_i| \cdot |x_j|} \tag{6.2}$$

$$A_{ji} = \begin{cases} 1, & \text{if} j \in Top - K(\{e_{ki} : k \in S_i\}) \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

where S_i is the set of embeddings that does not include x_i. The Top-K indices of the normalized dot products are chosen, and they compose the learned graph and adjacency matrix. These Top-K indices are used during the forward propagation stage of the model.

As another alternative, we discover the similarity between the sensor embeddings by finding the correlation coefficient matrix, R::

$$R_{ji} = \frac{S_{ji}}{\sqrt{S_{jj}S_{ii}}} \tag{6.4}$$

where S is the covariance matrix. Then we assemble the adjacency matrix similar to the process done for cosine similarity.

6.4.3 Forecasting Graph Neural Network

Before we carry out anomaly detection, we must have a baseline with which to compare data that could be anomalous. We train our GNN model using a forecast-based approach to predict the values at a future time based on a window of values from the past. The input to the model, *x(t)*, NXw, where N is the number of nodes in the dataset, and w is the size of the sliding window:

$$x^t := s^{(t-W)}, s^{(t-W+1)}, \cdots, s^{(t-1)} \tag{6.5}$$

The output predicts the node values at time $s(t)$. The aforementioned learned graph is then used to perform graph attention-based feature extraction. It aggregates a node's features

with the information of its neighboring node through a process called message passing. We aggregate sensor i's input feature vector, $x_{i(t)}$ with all of the features of its neighboring nodes to produce z_i:

$$\mathbf{z}_i^{(t)} = ReLU\left(\alpha_{i,i}\mathbf{W}\mathbf{x}_i^{(t)} + \sum_{j \in N_{(i)}} \alpha_{i,j}\mathbf{W}\mathbf{x}_j^{(t)}\right), \tag{6.6}$$

where $N_{(i)} = \{j \ | \mathrm{A}_{ji} > 0\}$ is the neighborhood of sensor i. $W \in \mathrm{R}_{d \times w}$. α_{ij} are attention coefficients and can be found with the following calculations:

$$\mathbf{g}_i^{(t)} = \mathbf{v}_i \oplus \mathbf{W}\mathbf{x}_i^{(t)} \tag{6.7}$$

$$\pi\left(i,j\right) = LeakyReLU\left(\mathbf{a}^T\left(\mathbf{g}_i^{(i)} \oplus \mathbf{g}_i^{(j)}\right)\right) \tag{6.8}$$

$$\alpha_{i,i} = \frac{exp\left(\pi\left(i,j\right)\right)}{\sum_{k \in N_{(i)} \cup \{i\}} exp\left(\pi\left(i,k\right)\right)} \tag{6.9}$$

W is a trainable weight matrix that performs a linear transformation on a node's input feature vector, $\mathbf{x}_i^{(t)}$. $\mathbf{g}_i^{(t)}$ concatenates the sensor embedding and the transformed result. LeakyReLU is used to calculate the attention coefficient, which is normalized with a softmax function. Each aggregated sensor, z_i, is multiplied element-wise with its embedding, v_i. The

final results for each sensor are concatenated and represent the vector of predicted values, $\hat{s}^{(t)}$ for each sensor at time t:

$$\hat{s}^{(t)} = f_\theta\big(\big[\boldsymbol{v_i} \circ \boldsymbol{z_i^{(t)}}, \ldots, \boldsymbol{v_N} \circ \boldsymbol{z_N^{(t)}}\big]\big) \tag{6.10}$$

6.4.4 Anomaly Detection

To train and test the performance of our model, we split the data into training, testing, and validation datasets. The training dataset contains no anomalies and is used to develop a baseline for normal behavior. The trained model then makes predictions for each node in the testing dataset. Unlike many models in the literature, our model does anomaly detection in real-time on the validation dataset. In other words, we monitor observations made by sensors and other devices and continuously examine for anomalies to detect, whereas deep learning models often perform prediction and anomaly detection in batches for more effortless and rapid testing, but their approach is not practical for run-time monitoring in real-time.

After predicting the normal state, the next challenge is determining the level of deviation from normal, which should be considered an anomaly. Detection of all the anomalies often raises considerable false warnings, meaning mislabeling normal data as an anomaly. There is a trade-off between high anomaly detection precision and low false positives. Thus, we deploy different strategies and pick the most promising to compute thresholds that we use to determine whether or not sensor readings at time t are anomalous. We execute the following techniques:

The first method uses the error scores between the actual sensor values and the predicted sensor values and then uses a Gaussian Estimator to determine the optimal threshold that

can be used to detect an anomaly on the validation dataset. Taking into consideration the seasonality of time-series data assists in making better predictions. Hence, we make sure to assign a different threshold for each time of day. Quantities such as temperature and humidity tend to vary depending on the time of day, so we divide the hours into four categories: morning, afternoon, evening, and night.

We then create a Gaussian distribution from the mean and standard deviation of the losses in the test dataset for each time of day in order to calculate the probability of each data point occurring in each distribution by using the probability density function. For each time of day, we compute the threshold by iterating over an array of threshold candidates and choosing the one that results in the highest F1 score. The array of potential candidates ranges from the smallest probability, p_{min}, to the largest probability of the datapoints, p_{max}, with a step size of pmax-pmin/1000. As we iterate over each probability, ϵ, we compare each value in the threshold of candidates to ϵ to create an array of predictions. If the value is smaller than ϵ, then it is labeled as an anomaly with a value of 1; otherwise, it is considered normal with a value of 0. We compare each prediction with the actual label and compute the F1-score. The ϵ that yields the highest F1-score is considered the threshold for that time of day.

Using the thresholds computed for each time of day, we make anomaly predictions on the validation dataset. In order to satisfy the real-time criterion, we make predictions for each successive timestamp by using a batch size of 1. We then compute the loss of the prediction and calculate its probability of occurring in the distribution of values for the timestamps corresponding time of day. If the probability is less than the threshold, then it is labeled as an anomaly with a value of 1. Normal values are labeled with a 0.

The second method calculates the mean μ and standard deviation σ for the test results and uses the following rule to define anomaly.

$$state(v_t) = \begin{cases} 1, & \text{if } v_t < \mu - 2.5 * \sigma \\ 1, & \text{else if } v_t > \mu + 2.5 * \sigma \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

where v_t is a node value at the time t and $state(v_t)$ indicates the data state which can be anomalous (1) or normal (0). Even if one of the data exceeds the normal range of values that node at t, then that data point is labeled as an anomaly.

The third method calculates the error scores at time t for each test data:

$$Err_i(t) = |s(t)_i - \hat{s}(t)_i| \tag{6.12}$$

The error scores are then normalized so that if one node has drastic deviations, its value will not dominate.

$$a_i(t) = \frac{Err_i(t - \widetilde{\mu}_i)}{\widetilde{\sigma}_i} \tag{6.13}$$

where $\widetilde{\mu}_i$ and $\widetilde{\sigma}_i$ are the median and inter-quartile range of each sensor's array of errors, across t. The anomaly score at time t is the maximum $a_i(t)$ over all the nodes. The score that results in the highest F1 score is considered the threshold. During each iteration of experimenting with the validation dataset, the model computes the prediction at a timestamp, and then we calculate the error score. Unlike the errors for the test dataset, we do not smooth them for the validation dataset. If any of the sensors at a particular time, t, has an error score higher than the threshold, then we label that timestamp as an anomaly.

The last strategy is to perform the same calculation as the previous technique except that

we take the maximum of all the node prediction scores at t, and if it is greater than the threshold, then that timestamp is labeled as anomalous.

6.5 Evaluation

6.5.1 Experiment Setup

To evaluate our multi-modal data fusion methodology, we analyze the data generated by a LoRaWAN-based greenhouse monitoring IoT system that supervises tomato crops in Belgium. We target this dataset because it is one of the few public datasets that contain both communication and sensor data. As summarized in Table 6.1, there are 22 devices in the system, each recording the temperature, humidity, RSSI, and SNR over the course of five months. We extracted the communication delay from the time stamps as well, which resulted in 110 components. Further, the graph representation of the system is generated, with 110 nodes, each retaining a sequence of 12661 data instances. The dataset is split to 80% for training on normal data and 20% for the testing and validation, which are infected with 10% anomalies. Our primary methodology is the multi-modal model, developed on a complete greenhouse dataset with 110 nodes. We also construct single-modal models for each data type to investigate the impact of fusing data from different modalities and integrating communication and sensor data. The single-modal method results in 5 models, each developed on one sensor type with 22 nodes. Another public dataset used for further analysis is the Secure Water Treatment data (SWaT) dataset which comprises days of continuous normal operation data and attack scenarios. The details of SWaT are reported in Table 6.1. It is the base dataset to compare the performance GNN model for anomaly detection with other methods in the literature. We implement our method and its variants in PyTorch version 1.5.1 with CUDA 10.2 and the PyTorch Geometric library.

Table 6.1: Statistics of the Greenhouse and SWaT dataset

Dataset	#devices/ features	Train	Test	Anomalies
Greenhouse	110	10128	2533	10%
SWaT	51	47515	44986	12%

6.5.2 Forecasting Model Performance

The initial stages in our pipeline are data preprocessing, context extraction, and graph generation, which prepare the data and transform it into a graph representation. Our GNN model follows afterward, which is supposed to learn the normal state of the system and predict its expected behavior. We train this forecasting model on an anomaly-free training dataset to develop a baseline for what is considered normal. Training is an iterative learning process to minimize prediction errors. We calculate the error with the Mean Squared Error (MSE) loss function, calculated as follows:

$$L_{MSE} = \frac{1}{T_{train} - w} \sum_{t=w+1}^{T_{train}} \left(\hat{s}^{(t)} - s^{(t)}\right)^2 \tag{6.14}$$

where T_{train} is the training dataset, w is the sliding window size, $\hat{s}^{(t)}$ is the predicted output, and $s^{(t)}$ is the actual output. Figure 6.4 represents that loss decreases as the epochs of training pass and the model converge to the target until the changes in loss become negligible and the learning process is stopped.

The loss indicates the model's capability to learn the patterns of time-series data in a normal state and model performance to predict the future value of each node. For instance, Figure 6.5 illustrates the actual recordings and predictions for a humidity sensor and exemplifies how the forecast follows a similar pattern as the actual measurement and captures the essence of the system. Another visualization of prediction performance is presented in

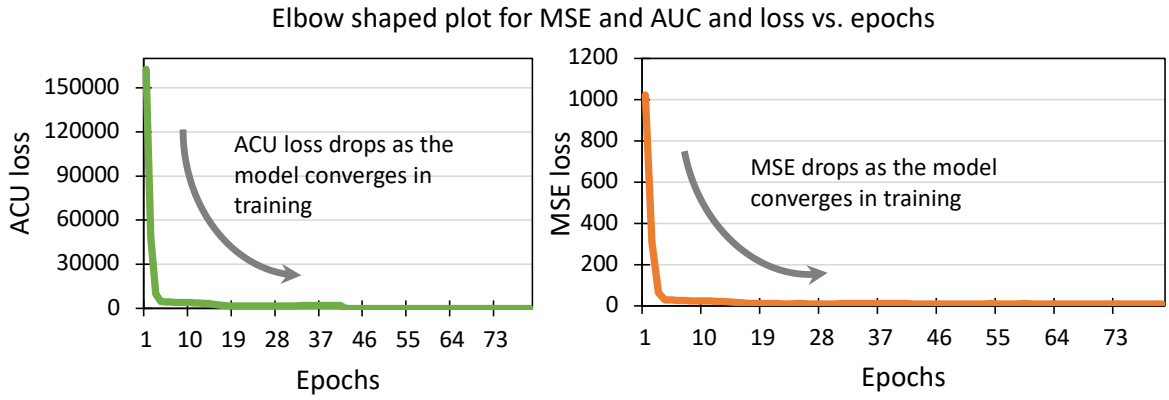


Figure 6.4: The plot MSE and AUC loss vs. the number of epochs.

Figure 6.6. It shows various data instances of a temperature sensor that are depicted in a plot with actual value as the y-axis and predicted value as the x-axis. The closer the data points to the y=x line, the lower the prediction error. Examination of this figure indicates a low error and high performance in modeling the system and predicting its behavior.

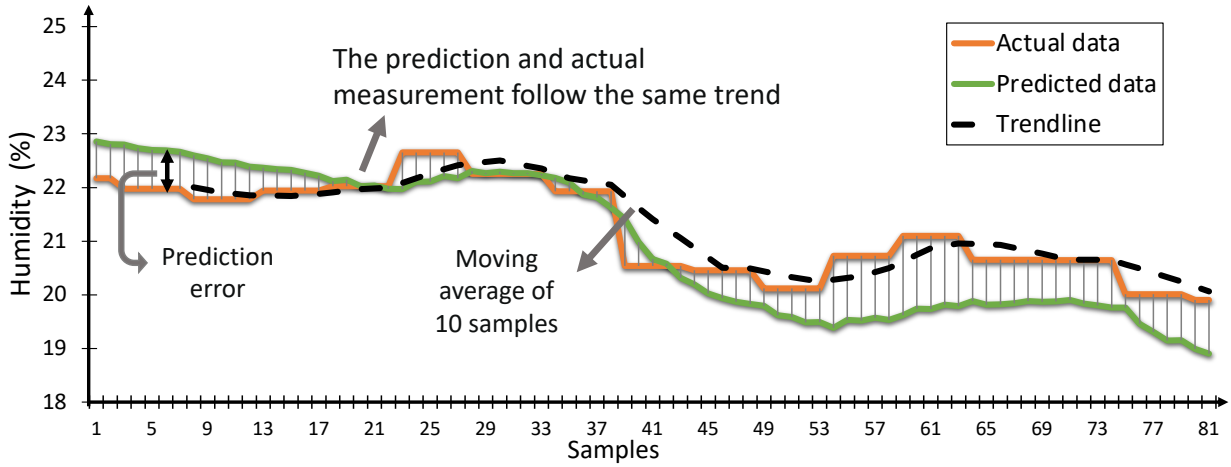


Figure 6.5: The predicted values, actual recordings, and actual data trendline for a humidity sensor follow a similar pattern.

Table 6.2 displays the MSE loss for different models. The multi-modal data fusion demonstrates better performance in prediction as it has a lower average loss per node compared to other single-modality methods. It indicates the potential of our GNN-based methodology to integrate data from distinct layers of the system with a different modality for enhanced learning and a superior context-aware model.

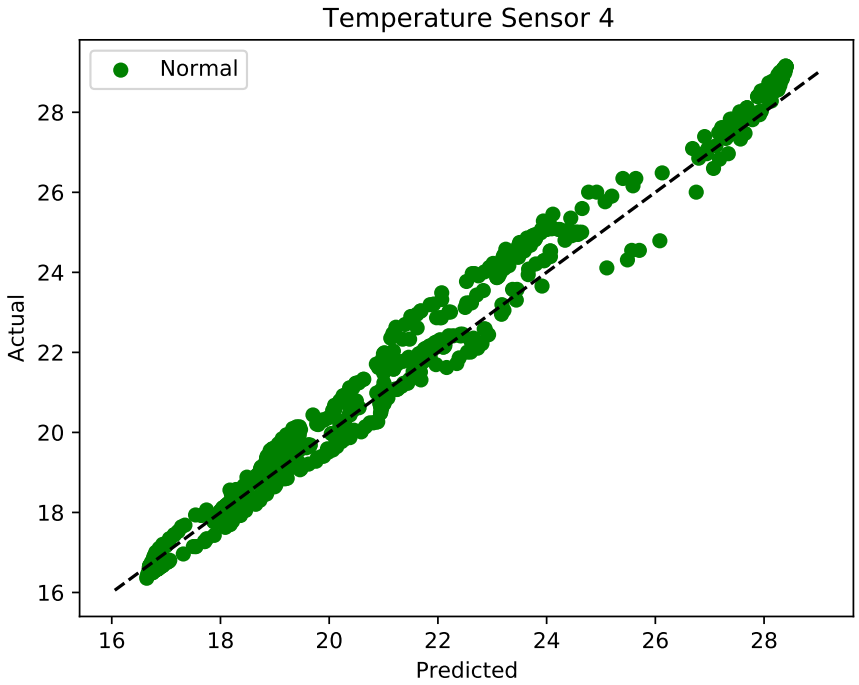


Figure 6.6: The prediction vs. actual data for a temperature sensor.

6.5.3 Anomaly Detection Performance

The forecasting model predicts the upcoming data instance based on historical data, which is the expected normal value. The deviation from expectation is analyzed upon observation of actual data to determine whether an anomaly has occurred and label the data instance. We use F1-score, precision, and recall as evaluation metrics to assess the performance of anomaly detection, which are calculated as follows:

$$Prec = \frac{TP}{TP + FP} \ , \ Recall = \frac{TP}{TP + FN} \tag{6.15}$$

Table 6.2: Data forecasting error and graph characteristics for different models

Methodology	**Total loss (MSE)**	**#nodes**	**#edges**
Multi-modal	0.29	110	5500
Only Temperature	0.16	22	132
Only Humidity	12.12	22	132
Only SNR	0.25	22	220
Only RSS	0.78	22	51
Only Delay	0.84	22	51

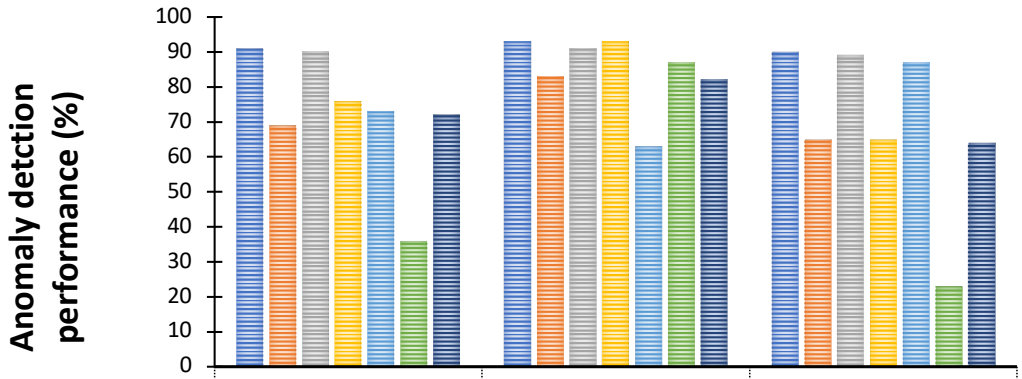
$$F1 - score = \frac{2 \times Prec \times Recall}{Prec + Recall} \tag{6.16}$$

Where a positive value denotes an anomaly, and TP, FP, and FN are the total numbers of true positives, false positives, and false negatives. We construct and fine-tune models to maximize the F1 score.

Single-modal vs. Multi-modal Anomaly Detection: Testing the single-modal and multi-modal models on anomaly-infested testing datasets resulted in the performance reported in Figure 6.7. The results indicate that the multi-modal approach outperforms the single-modal approach on average by 22%. Although for some data types, such as temperature, the single-modal model has comparable results to multi-modal, standalone models fail to detect all the various types of anomalies because different attacks have varied impacts and signatures, as discussed in section 6.3.1. All the parameters are required to capture diverse types of anomalies and on average anomaly detection on a single data modality performs inferior.

Our model can detect anomalies reasonably well, but it misses a few data points that should have been labeled as anomalous. The underlying reason the multi-modal dataset performs much better than the single-modal datasets is that its graph is more comprehensive. Hence, it provides more context during attention-based forecasting. The graph density in Table 6.2

is the total number of edges for each dataset, which ends up being the total number of nodes used for training multiplied by the top k values of the normalized dot products that were computed during graph generation. The multi-modal method graph has about five times as many edges as the single-modal datasets, which could challenge the training process but embeds more detailed information about nodes and their relation.



	F1 score	Precision	Recall
Multi-modal	91	93	90
Ave. single-modal	69	83	65
Temperature	90	91	89
Humidity	76	93	65
SNR	73	63	87
RSS	36	87	23
Delay	72	82	64

Figure 6.7: The anomaly detection performance of multi-modal and single-modal models.

Context Extraction Evaluation:

We investigate the impact of the initial similarity extraction algorithm on the model performance and compare the performance of the multi-modal method using the cosine similarity versus the correlation coefficient matrix to generate the adjacency matrix. In Table 6.3, the results show that cosine similarity supersedes the correlation coefficient matrix with higher F1-score, precision, and recall.

Comparing Anomaly Detection Methods in the Literature:

Table 6.3: Comparing context extraction methods: cosine similarity vs. correlation coefficient matrix

Algorithm	**F1-score**	**Precision**	**Recall**
Cosine Similarity	91%	93%	90%
Correlation Coefficient Matrix	83%	87%	79%

We compare the GNN-based approach for anomaly detection in time-series data with other popular techniques in the literature. We base the comparison on the SWaT dataset and gather the results presented by different methods. Table 6.4 elaborates on the various models' performance as well as the characteristics of each model. The comparison reveals that GNN outperforms the other models.

Table 6.4: Comparing anomaly detection performance of state-of-the-art methods for SWaT dataset.

Method	**F1-score**	**Precision**	**Recall**
GNN	81%	99%	68%
MAD-GAN [88]	77%	99%	63%
LSTM-VAE [121]	74%	96%	60%
AE [7]	61%	52%	72%
DAGMM [194]	39%	27%	70%
PCA [145]	23%	25%	22%
FB [84]	10%	10%	10%
KNN [12]	8%	8%	8%

6.5.4 Timing Analysis

IoT systems supervision requires continuous monitoring and fast anomaly detection in real-time, whereas computation resources are restrained in IoT systems. On the other hand, model execution on cloud servers significantly adds an enormous amount of data transmission overhead to the network, sending raw measurements all the way from edge devices to the fog layer and then to the cloud. Therefore, we optimize our model to perform anomaly detection in real-time while running on a fog platform. In addition to higher performance, one of

the advantages of our multi-modal data fusion approach is that all the individual machine learning models running for each data type are fused into a single model. Although the multi-modality model is larger than each single-modality model, it is smaller than all single-modality models combined, decreasing total computational overhead and memory footprint.

Table 6.5 indicates training, testing, and validation time per data, all executed on an X86-64 CPU. It is noteworthy that training and testing time refers to the model development process, which will be performed once for an IoT system. Afterward, the validation timing refers to run-time anomaly detection delay. Testing on the validation dataset is not done in batches but per data instance to satisfy continuous real-time monitoring requirements; The average time it takes for a single data point should be less than the system's frequency of change and data collection. In conclusion, the anomaly detection time for our multi-modal data fusion approach is 3.7ms which is very fast and satisfies the real-time requirement for our greenhouse system.

Table 6.5: Timing of train, test, and validation for each data point.

Methodology	**Training time**	**Testing time**	**Validation time**
Multi-modal	150ms	5.9ms	3.7ms
Only Temperature	67ms	3.8ms	2.4ms
Only Humidity	66ms	6.0ms	2.7ms
Only SNR	67ms	4.0ms	2.5ms
Only RSS	65ms	3.7ms	2.7ms
Only Delay	67ms	3.7ms	2.6ms

6.5.5 Attack Analysis

Deep integration of system communication nodes and sensing nodes through multi-modal fusion opens the opportunity to analyze anomalous incident further and trace it back to the source for system recovery. We study the impact of typical cyber or physical attacks in section 6.3.1 and apply our findings in a case study. After an anomaly is flagged in the system, we identify the type of attack associated with the anomaly. A physical attack against

sensors aims to inject fault data, causing abnormal sensor measurements. Denial of service can be a consequence of a physical attack or system failure and leads to missing data or acquiring a constant one.

On the other hand, attacks through communication channels influence readings from the cyber domain, such as SNR or RSSI, as well as the physical layer. For example, Figure 6.8 displays the actual and predicted recording of a pair of temperature and humidity sensors and their communication channel SNR. In this case, abnormal activity is detected concerning the drastic changes in these data while the rest of the system is working as expected. Further data analysis can determine our subsequent reaction. Based on the signature of the attack discussed before, further study implies a spoofing attack. Similarly, it can be applied to other adversaries, such as jamming, reply, and wormhole attacks.

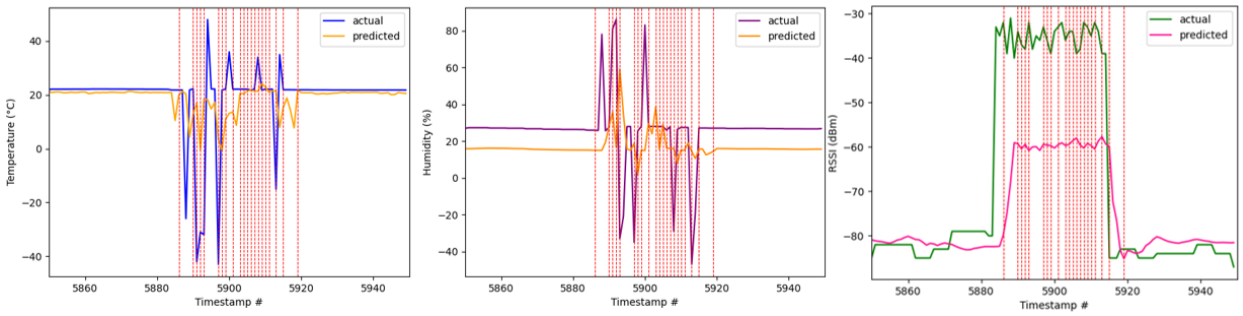


Figure 6.8: The impact of a spoofing attack on temperature, humidity, and RSSI readings (left to right).

6.6 Chapter Concluding Remarks

In this work, we propose a methodology to selectively fuse sensor and communication data and model an IoT system to detect reliability and security issues through anomaly detection in real-time. Our approach integrates multi-modal data into a graph representation for an IoT system and learns the system behavior using GNN. Analyzing diverse network and sensor attacks, we identify the attack signatures and trace the anomaly back to its origin and

the attack type. Attack understanding facilitates the system recovery, knowing the specific security measures to implement. To the best of our knowledge, this is the first work to fuse sensor and communication data using GNN, which inspires deeper integration of system components and data fusion to construct a precise digital model of real-world applications.

Bibliography

[1] Semiconductor intellectual property (ip) market with covid-19 impact analysis. Accessed: 2021-04-20.

[2] Trusthub. *Available on-line: https://www.trust-hub.org*, 2016.

[3] Trusthub. *Available on-line: https://www.trust-hub.org*, 2016.

[4] Special 301 report. *the United States Trade Representative*, 2017.

[5] Copyrights and patents, piracy and theft. *The Washington Times*, 2018.

[6] S. Adee. The hunt for the kill switch. In *IEEE Spectrum*, 2008.

[7] C. Aggarwal. Outlier analysis. data mining, 2015.

[8] M. M. Ahemd, M. A. Shah, and A. Wahid. Iot security: A layered approach for attacks amp; defenses. In *2017 International Conference on Communication Technologies (ComTech)*, pages 104–110, April 2017.

[9] U. Alegre, J. C. Augusto, and T. Clark. Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, 117:55–83, 2016.

[10] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu. Gnn-re: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[11] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, and M. Rossi. Low power link layer security for iot: Implementation and performance analysis. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 919–925, July 2013.

[12] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, pages 15–27. Springer, 2002.

[13] E. Aras, N. Small, G. S. Ramachandran, S. Delbruel, W. Joosen, and D. Hughes. Selective jamming of lorawan using commodity hardware. In *Proceedings of the 14th EAI*

International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, pages 363–372, 2017.

[14] M. Ashrafiamiri, S. Manoj Pudukotai Dinakarrao, A. H. Afandizadeh Zargari, M. Seo, F. Kurdahi, and H. Homayoun. R2ad: Randomization and reconstructor-based adversarial defense on deep neural network. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, pages 21–26, 2020.

[15] M. AshrafiAmiri, A. H. A. Zargari, S. M.-H. Farzam, and S. Bayat-Sarmadi. Towards side channel secure cyber-physical systems. In *2018 Real-Time and Embedded Systems and Technologies (RTEST)*, pages 31–38. IEEE, 2018.

[16] A. Barua and M. Al Faruque. Hall spoofing: A non-invasive dos attack on grid-tied solar inverter. *29th Usenix Security*, 2020.

[17] A. Barua and M. A. Al Faruque. Hall spoofing: a noninvasive dos attack on grid-tied solar inverter. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 1273–1290, 2020.

[18] S. Bhunia and M. Tehranipoor. *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018.

[19] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl. Tytan: tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[20] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[21] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 1986.

[22] G. S. Chadha, I. Islam, A. Schwung, and S. X. Ding. Deep convolutional clustering-based time series anomaly detection. *Sensors*, 21(16):5488, 2021.

[23] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia. Mero: A statistical approach for hardware trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 396–410. Springer, 2009.

[24] C.-H. Chang et al. Hardware ip watermarking and fingerprinting. In *Secure System Design and Trustable Computing*. 2016.

[25] S. Chauhan and L. Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7. IEEE, 2015.

[26] H. C. Chen, M. A. A. Faruque, and P. H. Chou. Security and privacy challenges in iot-based machine-to-machine collaborative scenarios. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2016.

[27] X. Chen, Q. Liu, S. Yao, J. Wang, Q. Xu, Y. Wang, Y. Liu, and H. Yang. Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[28] Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng. Learning graph structures with transformer for multivariate time series anomaly detection in iot. *IEEE Internet of Things Journal*, 2021.

[29] Z. Chen, Z. Peng, X. Zou, and H. Sun. Deep learning based anomaly detection for muti-dimensional time series: A survey. In *China Cyber Security Annual Conference*, pages 71–92. Springer, Singapore, 2021.

[30] J. Chen et al. Decoy: Deflection-driven hls-based computation partitioning for obfuscating intellectual property. In *Design Automation Conference (DAC)*, 2020.

[31] S. R. Chhetri, A. Barua, S. Faezi, F. Regazzoni, A. Canedo, and M. A. Al Faruque. Tool of spies: Leaking your ip by altering the 3d printer compiler. *IEEE Transactions on Dependable and Secure Computing*, 18(2):667–678, 2019.

[32] S. R. Chhetri, S. Faezi, and M. A. Al Faruque. Information leakage-aware computer-aided cyber-physical manufacturing. *IEEE Transactions on Information Forensics and Security*, 13(9):2333–2344, 2018.

[33] S. R. Chhetri, S. Faezi, A. Canedo, and M. A. A. Faruque. Quilt: quality inference from living digital twins in iot-enabled manufacturing systems. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, pages 237–248. ACM, 2019.

[34] S. R. Chhetri, S. Faezi, N. Rashid, and M. A. Al Faruque. Manufacturing supply chain and product lifecycle security in the era of industry 4.0. *Journal of Hardware and Systems Security*, 2:51–68, 2018.

[35] S. R. Chhetri, N. Rashid, S. Faezi, and M. A. Al Faruque. Security trends and advances in manufacturing systems in the era of industry 4.0. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[36] S. D. Chowdhury, K. Yang, and P. Nuzzo. Reignn: State register identification using graph neural networks for circuit reverse engineering. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[37] A. Cook, G. Mısırlı, and Z. Fan. Anomaly detection for iot time-series data: A survey. *Internet of Things Journal*, 2019.

[38] F. Courbon, P. Loubet-Moundi, J. J. Fournier, and A. Tria. A high efficiency hardware trojan detection technique based on fast sem imaging. In *2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 788–793. IEEE, 2015.

[39] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra. Hardware trojan detection using atpg and model checking. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, 2018.

[40] F. Demrozi, R. Zucchelli, and G. Pravadelli. Exploiting sub-graph isomorphism and probabilistic neural networks for the detection of hardware trojans at rtl. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, 2017.

[41] A. Deng and B. Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[42] M. El Massad et al. The sat attack on ic camouflaging: Impact and potential countermeasures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[43] R. Elnaggar and K. Chakrabarty. Machine learning for hardware security: Opportunities and risks. *Journal of Electronic Testing*, 34(2), 2018.

[44] J. M. et al. The internet of things: Mapping the value beyond the hype. *McKinsey Glob. Inst., New York, NY, USA. [Online]. Available: https://www.mckinsey.com/businessfunctions/mckinsey-digital/our-insights/the-internet-of-things-thevalue-of-digitizing-the-physical-world*, 2015.

[45] W. W. et al. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020.

[46] S. Faezi. Data-driven modeling and analysis for trustworthy cyber-physical systems. page 133, 2021.

[47] S. Faezi, S. R. Chhetri, A. V. Malawade, J. C. Chaput, W. Grover, P. Brisk, and M. A. Al Faruque. Oligo-snoop: a non-invasive side-channel attack against dna synthesis machines. In *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019.

[48] S. Faezi, S. R. Chhetri, A. V. Malawade, J. C. Chaput, W. H. Grover, P. Brisk, and M. A. Al Faruque. Oligo-snoop: A non-invasive side channel attack against dna synthesis machines. In *NDSS*, 2019.

[49] S. Faezi, R. Yasaei, and M. Al Faruque. Htnet: Transfer learning for golden chip-free hardware trojan detection. *IEEE/ACM Design Automation and Test in Europe Conference (DATE'21)*, 2021.

[50] S. Faezi, R. Yasaei, and M. A. Al Faruque. Htnet: Transfer learning for golden chip-free hardware trojan detection. In *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[51] S. Faezi, R. Yasaei, A. Barua, and M. A. Al Faruque. Brain-inspired golden chip free hardware trojan detection. *IEEE Transactions on Information Forensics and Security*, 16:2697–2708, 2021.

[52] F. Farahmandi, Y. Huang, and P. Mishra. Trojan localization using symbolic algebra. In *Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, 2017.

[53] N. Fern and S. Carlson. A complete system-level security verification methodology. (white paper from cadence and tortuga logic). 2019.

[54] P. Filonov, A. Lavrentyev, and A. Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model, 2016.

[55] M. Fyrbiak et al. Graph similarity and its applications to hardware security. *IEEE Transactions on Computers*, 2019.

[56] M. Fyrbiak et al. Graph similarity and its applications to hardware security. *IEEE Tran. on Computers*, 2020.

[57] J. S. Garrido, D. Dold, and J. Frank. Machine learning on knowledge graphs for context-aware security monitoring. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 55–60. IEEE, 2021.

[58] F. Giannoni, M. Mancini, and F. Marinelli. Anomaly detection models for iot time series data. *arXiv*, 2018.

[59] X. Guo, R. Dutta, P. Mishra, and Y. Jin. Scalable soc trust verification using integrated theorem proving and model checking. *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

[60] R. Gurunath, M. Agarwal, A. Nandi, and D. Samanta. An overview: security issue in iot network. In *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on*, pages 104–107. IEEE, 2018.

[61] B. Halak, M. Zwolinski, and M. S. Mispan. Overview of puf-based hardware security solutions for the internet of things. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, Oct 2016.

[62] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague. Do you feel what i hear? enabling autonomous iot device pairing using different sensor types. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 836–852. IEEE, 2018.

[63] M. L. Han, J. Lee, A. R. Kang, S. Kang, J. K. Park, and H. K. Kim. A statistical-based anomaly detection method for connected cars in internet of things environment. In C.-H. Hsu, F. Xia, X. Liu, and S. Wang, editors, *Internet of Vehicles - Safe and Intelligent Mobility*, Cham, 2015. Springer International Publishing.

[64] T. Han et al. Hardware trojans detection at register transfer level based on machine learning. In *Symposium on Circuits and Systems(ISCAS)*, 2019.

[65] K. Hasegawa, M. Yanagisawa, and N. Togawa. Hardware trojans classification for gate-level netlists using multi-layer neural networks. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017.

[66] Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[67] F. Hessel, L. Almon, and F. Álvarez. Chirpotle: a framework for practical lorawan security evaluation. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 306–316, 2020.

[68] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *2010 IEEE symposium on security and privacy*, 2010.

[69] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *2010 IEEE Symposium on Security and Privacy*, 2010.

[70] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[71] E. Hodo, X. Bellekens, A. Hamilton, P. Dubouilh, E. Iorkyase, C. Tachtatzis, and R. Atkinson. Threat analysis of iot networks using artificial neural network intrusion detection system. In *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, May 2016.

[72] T. Hoque, S. Narasimhan, X. Wang, S. Mal-Sarkar, and S. Bhunia. Golden-free hardware trojan detection with high sensitivity under process noise. *Journal of Electronic Testing*, 33(1):107–124, 2017.

[73] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen. An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access*, 8:30387–30399, 2020.

[74] S. A. Islam, F. I. Mime, S. Asaduzzaman, and F. Islam. Socio-network analysis of rtl designs for hardware trojan localization. In *2019 22nd International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2019.

[75] S. A. Islam, L. K. Sah, and S. Katkoori. A framework for hardware trojan vulnerability estimation and localization in rtl designs. *Journal of Hardware and Systems Security*, 2020.

[76] J.-Y. Jou and C.-N. J. Liu. Coverage analysis techniques for hdl design validation. *Proc. Asia Pacific CHip Design Languages*, 1999.

[77] J. Kim, J. Kim, H. Kim, M. Shim, and E. Choi. Cnn-based network intrusion detection against denial-of-service attacks. *Electronics*, 9(6):916, 2020.

[78] S. Kim, C. Hwang, and T. Lee. Anomaly based unknown intrusion detection in end-point environments. *Electronics*, 9(6):1022, 2020.

[79] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[80] T. N. Kipf et al. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[81] B. Knyazev et al. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[82] F. Koushanfar. Active hardware metering by finite state machine obfuscation. In *Hardware Protection through Obfuscation*. 2017.

[83] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu. Revisiting time series outlier detection: Definitions and benchmarks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[84] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, 2005.

[85] J. Lee et al. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

[86] C. Lesjak, H. Bock, D. Hein, and M. Maritsch. Hardware-secured and transparent multi-stakeholder data exchange for industrial iot. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 706–713, July 2016.

[87] C. Lesjak, N. Druml, R. Matischek, T. Ruprechter, and G. Holweg. Security in industrial iot – quo vadis? *e & i Elektrotechnik und Informationstechnik*, 2016.

[88] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International conference on artificial neural networks*, pages 703–716. Springer, 2019.

[89] H. Li, Q. Liu, and J. Zhang. A survey of hardware trojan threat and defense. *Integration*, 55:426–437, 2016.

[90] M. Li et al. Provably secure camouflaging strategy for ic protection. *Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[91] R.-F. Liao, H. Wen, J. Wu, F. Pan, A. Xu, Y. Jiang, F. Xie, and M. Cao. Deep-learning-based physical layer authentication for industrial wireless sensor networks. *sensors*, 19(11):2440, 2019.

[92] T. Lin, T. Guo, and K. Aberer. Hybrid neural networks for learning the trend in time series. pages 2273–2279, 2017.

[93] W.-C. Lin, S.-W. Ke, and C.-F. Tsai. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-based systems*, 78:13–21, 2015.

[94] S. Liu, J. H. Park, and S. Yoo. Efficient and effective graph convolution networks. In *Proceedings of the 2020 SIAM International Conference on Data Mining*. SIAM, 2020.

[95] Y. Liu, K. Huang, and Y. Makris. Hardware trojan detection through golden chip-free statistical side-channel fingerprinting. In *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014.

[96] Y. Liu, H. Zheng, X. Feng, and Z. Chen. Short-term traffic flow prediction with conv-lstm. In *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, Oct 2017.

[97] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann. E-graphsage: A graph neural network based intrusion detection system. *arXiv preprint arXiv:2103.16329*, 2021.

[98] B. Longueville, M. Gardoni, et al. A survey of context modeling: approaches, theories and use for engineering design researches. In *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*, pages 437–438, 2003.

[99] J. Lu, Q. Zhang, Z. Yang, M. Tu, J. Lu, H. Peng, et al. Short-term load forecasting method based on cnn-lstm hybrid neural network model. *Automation of Electric Power Systems*, 43(8):131–137, 2019.

[100] W. Luo, W. Liu, and S. Gao. Remembering history with convolutional lstm for anomaly detection. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 439–444, July 2017.

[101] J. Luomala and I. Hakala. Effects of temperature and humidity on radio signal strength in outdoor wireless sensor networks. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1247–1255. IEEE, 2015.

[102] L. Lyu, J. Jin, S. Rajasegarar, X. He, and M. Palaniswami. Fog-empowered anomaly detection in iot using hyperellipsoidal clustering. *Internet of Things Journal*, 2017.

[103] Y. Lyu and P. Mishra. Automated trigger activation by repeated maximal clique sampling. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[104] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *International Joint Conference on Neural Networks*, 2003.

[105] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu. Understanding graphs in eda: From shallow to deep learning. ISPD '20. Association for Computing Machinery, 2020.

[106] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu. Understanding graphs in eda: From shallow to deep learning. In *ISPD*, 2020.

[107] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu. High performance graph convolutional networks with applications in testability analysis. In *ACM/IEEE Design Automation Conference (DAC'19)*, 2019.

[108] A. V. Malawade, N. D. Costa, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Neuroscience-inspired algorithms for the predictive maintenance of manufacturing systems. *IEEE Transactions on Industrial Informatics*, 17(12):7980–7990, 2021.

[109] A. V. Malawade, S.-Y. Yu, B. Hsu, H. Kaeley, A. Karra, and M. A. Al Faruque. Roadscene2vec: A tool for extracting and embedding road scene-graphs. *Knowledge-Based Systems*, 242:108245, 2022.

[110] A. V. Malawade, S.-Y. Yu, B. Hsu, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Spatiotemporal scene-graph embedding for autonomous vehicle collision prediction. *IEEE Internet of Things Journal*, 9(12):9379–9388, 2022.

[111] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[112] P. Malhotra, V. TV, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Multi-sensor prognostics using an unsupervised health index based on lstm encoder-decoder, 2016.

[113] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[114] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. Trustvisor: Efficient tcb reduction and attestation. In *2010 IEEE Symposium on Security and Privacy*, pages 143–158. IEEE, 2010.

[115] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani. Context-based zero-interaction pairing and key evolution for advanced personal devices. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 880–891. ACM, 2014.

[116] M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan. Revisiting context-based authentication in iot. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[117] M. M. R. Monjur, J. Heacock, R. Sun, and Q. Yu. An attack analysis framework for lorawan applied advanced manufacturing. In *2021 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–7. IEEE, 2021.

[118] T. Mortlock, D. Muthirayan, S.-Y. Yu, P. P. Khargonekar, and M. A. Al Faruque. Graph learning for cognitive digital twins in manufacturing systems. *IEEE Transactions on Emerging Topics in Computing*, 10(1):34–45, 2021.

[119] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor. Hardware trojan detection through information flow security verification. In *2017 IEEE International Test Conference (ITC)*, 2017.

[120] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor. Avfsm: A framework for identifying and mitigating vulnerabilities in fsms. *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.

[121] D. Park, Y. Hoshi, and C. C. Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

[122] S. Patnaik et al. Raise your game for split manufacturing: Restoring the true functionality through beol. In *Design Automation Conference (DAC)*, 2018.

[123] L. Piccolboni et al. Efficient control-flow subgraph matching for detecting hardware trojans in rtl models. *ACM Transactions on Embedded Computing Systems (TECS)*, 2017.

[124] H. Pirayesh and H. Zeng. Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2022.

[125] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction, 2017.

[126] X. Qiu, J. Dai, and M. Hayes. A learning approach for physical layer authentication using adaptive neural network. *IEEE Access*, 8:26139–26149, 2020.

[127] S. Rai et al. Hardware watermarking using polymorphic inverter designs based on reconfigurable nanotechnologies. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.

[128] J. Rajendran et al. Is split manufacturing secure? In *Design, Automation & Test in Europe Conference (DATE)*, 2013.

[129] J. Rajendran et al. Security analysis of integrated circuit camouflaging. In *ACM conference on Computer & communications security*, 2013.

[130] J. Rajendran et al. Detecting malicious modifications of data in third-party intellectual property cores. In *ACM/IEEE Design Automation Conference (DAC)*, 2015.

[131] J. Rajendran et al. Formal security verification of third party intellectual property cores for information leakage. In *International Conference on VLSI Design and Embedded Systems (VLSID)*, 2016.

[132] E. Ranjan, S. Sanyal, and P. Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[133] N. Rashid, M. Dautta, P. Tseng, and M. A. Al Faruque. Hear: Fog-enabled energy-aware online human eating activity recognition. *IEEE Internet of Things Journal*, 8(2):860–868, 2020.

[134] N. Rashid, B. U. Demirel, and M. A. Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, 9(15):13041–13051, 2022.

[135] N. Rashid, B. U. Demirel, M. Odema, and M. A. Al Faruque. Template matching based early exit cnn for energy-efficient myocardial infarction detection on low-power wearable devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(2):1–22, 2022.

[136] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad hoc networks*, 11(8):2661–2674, 2013.

[137] M. Sabri, A. Shabani, and B. Alizadeh. Sat-based integrated hardware trojan detection and localization approach through path-delay analysis. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[138] S. Saha, R. S. Chakraborty, S. S. Nuthakki, D. Mukhopadhyay, et al. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015.

[139] H. Salmani, M. Tehranipoor, and R. Karri. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE, 2013.

[140] H. Sedjelmaci, S. M. Senouci, and M. Al-Bahri. A lightweight anomaly detection technique for low-resource iot devices: A game-theoretic methodology. In *2016 IEEE International Conference on Communications (ICC)*, May 2016.

[141] A. Seshadri, M. Luk, and A. Perrig. Sake: Software attestation for key establishment in sensor networks. In S. E. Nikoletseas, B. S. Chlebus, D. B. Johnson, and B. Krishnamachari, editors, *Distributed Computing in Sensor Systems*, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[142] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. K. Khosla. Scuba: Secure code update by attestation in sensor networks. In *Workshop on Wireless Security*, 2006.

[143] B. Shakya, M. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 2017.

[144] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell. Detecting 802.11 mac layer spoofing using received signal strength. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1768–1776. IEEE, 2008.

[145] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, 2003.

[146] R. K. Singh, M. H. Rahmani, M. Weyn, and R. Berkvens. Joint communication and sensing: A proof of concept and datasets for greenhouse monitoring using lorawan. *Sensors*, 22(4):1326, 2022.

[147] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 23–40. Springer, 2012.

[148] C. Sturton, M. Hicks, D. Wagner, and S. T. King. Defeating uci: Building stealthy and malicious hardware. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011.

[149] C. Sturton, M. Hicks, D. Wagner, and S. T. King. Defeating uci: Building stealthy and malicious hardware. In *2011 IEEE symposium on security and privacy*, 2011.

[150] P. Subramanyan and D. Arora. Formal verification of taint-propagation security properties in a commercial soc design. In *Design, Automation & Test in Europe Conference (DATE)*, 2014.

[151] T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, and T. Fujino. Reversing stealthy dopant-level circuits. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 112–126. Springer, 2014.

[152] W.-J. Sung, H.-G. Ahn, J.-B. Kim, and S.-G. Choi. Protecting end-device from replay attack on lorawan. In *2018 20th International conference on advanced communication technology (ICACT)*, pages 167–171. IEEE, 2018.

[153] S. Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *International Symposium on Applied Reconfigurable Computing*, 2015.

[154] S. Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *International Symposium on Applied Reconfigurable Computing*. Springer, 2015.

[155] Y. Tang, L. Fang, and S. Li. Activity factor based hardware trojan detection and localization. *Journal of Electronic Testing*, 2019.

[156] M. Tehranipoor, H. Salmani, and X. Zhang. Integrated circuit authentication. *Switzerland: Springer, Cham. doi*, 2014.

[157] K. Vatanparvar, S. Faezi, I. Burago, M. Levorato, and M. A. Al Faruque. Extended range electric vehicle with driving behavior estimation in energy management. *IEEE transactions on Smart Grid*, 10(3):2959–2968, 2018.

[158] K. Vatanpavar and M. A. Al Faruque. Acqua: Adaptive and cooperative quality-aware control for automotive cyber-physical systems. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 193–200. IEEE, 2017.

[159] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[160] A. Waksman et al. Fanci: identification of stealthy malicious logic using boolean functional analysis. In *ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[161] J. Wan, A. Lopez, and M. A. A. Faruque. Physical layer key generation: Securing wireless communication in automotive cyber-physical systems. *ACM Transactions on Cyber-Physical Systems*, 3(2):13, 2018.

[162] W.-C. Wang et al. Reverse engineering for 2.5 d split manufactured ics. *Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[163] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, 2020.

[164] C. Wolf. Yosys open synthesis suite. `http://www.clifford.at/yosys/`, 2022.

[165] Y. Wu and H. Tan. Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework, 2016.

[166] Z. Wu et al. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[167] Y. Xie and A. Srivastava. Mitigating sat attack on logic locking. In *conference on cryptographic hardware and embedded systems*, 2016.

[168] Y. Xie et al. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *Design Automation Conference (DAC)*, 2017.

[169] X. Xu et al. Neural network-based graph embedding for cross-platform binary code similarity detection. In *SIGSAC Conference on Computer and Communications Security*, 2017.

[170] X. Yang. Lorawan: Vulnerability analysis and practical exploitation. *Delft University of Technology. Master of Science*, 2017.

[171] X. Yang, G. Peng, D. Zhang, and Y. Lv. An enhanced intrusion detection system for iot networks based on deep learning and knowledge graph. *Security and Communication Networks*, 2022, 2022.

[172] R. Yasaei, L. Chen, S.-Y. Yu, and M. A. Al Faruque. Hardware trojan detection using graph neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[173] R. Yasaei, S. Faezi, and M. A. Al Faruque. Golden reference-free hardware trojan localization using graph convolutional network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(10):1401–1411, 2022.

[174] R. Yasaei, F. Hernandez, and M. A. Al Faruque. Iot-cad: Context-aware adaptive anomaly detection in iot systems through sensor association. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[175] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque. Gnn4ip: Graph neural network for hardware intellectual property piracy detection. In *Design, Automation Conference (DAC'21)*. IEEE, 2021.

[176] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1504–1509, 2021.

[177] M. Yasin et al. Removal attacks on logic locking and camouflaging techniques. *IEEE Trans. on Emerging Topics in Computing*, 2017.

[178] C. Yin, S. Zhang, J. Wang, and N. N. Xiong. Anomaly detection based on convolutional recurrent autoencoder for iot time series. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(1):112–122, 2020.

[179] H. Yu, Z. Wu, S. Wang, Y. Wang, and X. Ma. Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. *Sensors*, 17(7), 2017.

[180] S.-Y. Yu, A. V. Malawade, S. R. Chhetri, and M. A. Al Faruque. Sabotage attack detection for additive manufacturing systems. *IEEE Access*, 8:27218–27231, 2020.

[181] S.-Y. Yu, A. V. Malawade, and M. A. A. Faruque. Multi-modal attack detection for cyber-physical additive manufacturing. *arXiv preprint arXiv:2110.02259*, 2021.

[182] S.-Y. Yu, A. V. Malawade, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Scene-graph augmented data-driven risk assessment of autonomous vehicle decisions. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):7941–7951, 2021.

[183] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque. Hw2vec: A graph learning tool for automating hardware security. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST'21)*, 2021.

[184] F. Zareen and R. Karam. Detecting rtl trojans using artificial immune systems and high level behavior classification. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2018.

[185] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu. Veritrust: Verification for hardware trust. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[186] J. Zhang, F. Yuan, and Q. Xu. Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. Association for Computing Machinery, 2014.

[187] S. Zhang, X. Chen, J. Chen, Q. Jiang, and H. Huang. Anomaly detection of periodic multivariate time series under high acquisition frequency scene in iot. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 543–552, 2020.

[188] Y. Zhang. Stealing deep learning model secret through remote fpga side-channel analysis. *ProQuest Dissertations and Theses*, page 52, 2021. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-03-08.

[189] Y. Zhang, H. Ren, and B. Khailany. Grannite: Graph neural network inference for transferable power estimation. In *ACM/IEEE Design Automation Conference (DAC'20)*, 2020.

[190] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque. Stealing neural network structure through remote fpga side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.

[191] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850. IEEE, 2020.

[192] K. Zhao and L. Ge. A survey on the internet of things security. In *2013 Ninth International Conference on Computational Intelligence and Security*, pages 663–667, Dec 2013.

[193] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

[194] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018.

Appendix A

Post-Silicon Hardware Trojan Detection

A.1 Introduction

Design and fabrication outsourcing has made integrated circuits (IC) vulnerable to malicious modifications by third parties known as Hardware Trojans (HT). This dissertation focuses mainly on mitigating HT threats in the design stage before fabrication. However, we have pursued hardware security research in the post-silicon stage as well (elaborated by the lead author dissertation [46]). In this research, we have the premise that the culprit can be in any entity in the IC supply chain, including the design vendors and foundries. We propose two novel golden chip-free methodologies based on circuit side-channel signals to detect HT in the run-time as the last line of defense.

A.2 Background and Research Challenges

Due to their stealthy nature, most HTs are designed to be minuscule and remain inactive with a negligible impact on the circuit specification until a rare specific event triggers them. Ideally, any drift from the original circuit design should be detectable by post-fabrication testing and verification. However, these methods fall short of detecting HT because the probability of triggering the HT is usually low. Therefore, some other methods are required to ensure circuit security. To this end, two major paths are pursued in literature: i) imaging techniques and ii) side-channel and covert-channel analysis. Destructive approaches mainly involve the following steps: de-layering the chip, imaging the die, reverse-engineering the image of the circuit, and conducting the element-by-element comparison. Although these methods are relatively capable of guaranteeing trust on the fabricated IC [151, 38, 43], they are destructive, impractical, time-consuming, expensive, and inapplicable to detect the contaminated third-party IPs. In contrast, the second approaches are non-destructive and assess the behavior of the chip through side-channel (e.g., power, temperature, electromagnetic, and timing) or embedded sensor measurements. In these methods, the parameters are measured and compared to the expected values to identify the presence of additional structure.

The fundamental shortcoming of the majority of side-channel based HT detection methodologies is the reliance on a trusted chip (a.k.a golden chip) to create a reference model of the expected side-channel values. Reliance on a golden chip is a problem since, in practice, a trusted supply chain to manufacture the golden chip is unavailable, or it would be too expensive and not affordable for most SoC designers [89]. A few works in the literature have tried to resolve this issue by using self-referencing techniques [72] or using accurate trusted simulations combined with embedded sensors in the chip to analyze the side-channel emissions [95]. However, similar to other side-channel based HT detection methodologies, they often perform poorly when HT is not triggered.

A.3 Methodologies

A.3.1 Transfer Learning for HT Detection

We develop a novel neural network design (i.e., HTNet) and a training methodology for HT detection in run time without the golden-chip requirement. We create a library of known HTs and collect electromagnetic (EM) and power side-channel signals for each case, and train HTnet to learn the best discriminative features based on this library. Then, during the testing phase, we fine-tune the HTnet to learn the behavior of the particular chip under test. We use HTnet followed by an anomaly detection algorithm in run-time to monitor the side-channel signals emitted from the chip and report malicious behaviors related to a triggered HT. We devise a custom neural network architecture that performs the same or better than any available approach in the literature for HT detection. Furthermore, we propose a methodology to transfer the knowledge learned based on known HT benchmarks to a new circuit that may or may not contain an HT. With this work, we publish our collected electromagnetic and power side-channel data for hardware implementation of AES infected by various forms of HTs.

This approach is evaluated using TrustHub [139] benchmarks, and the results provide evidence that it is possible to extract robust features that can be used for HT detection using our previous knowledge about HTs. However, running a neural network along with the main circuit will consume a considerable amount of energy which will make our approach most suitable for applications that do not rely on limited power resources. Here, we provided a proof of concept for golden chip-free HT detection using recent developments in artificial intelligence.

A.3.2 Brain-Inspired HT Detection using Hierarchical Temporal Memory

We construct a model based on the power consumption of the chip, which monitors the side-channel emissions during the testing and at the run-time. Our proposed model is hierarchical temporal memory (HTM), which mimics human brain behavior to interpolate the side-channel data and pinpoint anomalies that indicate the existence of an HT in the IC. We devise an HTM architecture that performs the same or better than state-of-the-art approaches for HT detection. Our proposed method is trained on side-channel data from the Device Under Test (DUT) rather than a golden chip. The process variation does not affect our proposed detection mechanism since it relies on DUT for training. HTM training has virtually no cost in terms of power consumption. Hence, sporadic retraining of the model is feasible to cope with aging/temperature effects on the side-channel data.

The high dimensional form of encoded input in SDRs followed by spatial and temporal pooling gives HTM the capacity to capture complex patterns in the signal with minimum required computations. The complexity of all the operations in the HTM is at most $n \log n$, where n is the size of the encoder. Moreover, the high dimensionality of the data flow in HTM makes it highly resilient against noise in the input signal compared to neural networks and other ML techniques. Furthermore, the majority of unsupervised ML techniques cannot adapt to changes in the signal since retraining the model in real-time is not feasible. However, HTM uses Hebbian Learning rules that are a light training methodology and can be applied in real-time. Last but not least, in our experiments, we notice that training HTM requires a much less number of samples than other unsupervised ML techniques. Our evaluation based on TrustHub benchmarks [2] shows that our proposed detection mechanism can detect 92.20% of triggered HT in five benchmarks while consuming less power compared to state-of-the-art machine learning techniques.

Appendix B

Stealing Neural Network Structure through Remote FPGA Side-channel Analysis

In this dissertation, we study the security across different layers of a system from low-level circuits up to high-level CPS. Software security and embedded board are two of the internment layers in a computing system and the security embedded boards and cybersecurity are studied separately in the literature. In this appendix, we assess the security across these layers by demonstrating a remote attack toward FPGA to steal the software IP executed on it [190] (elaborated in lead author master thesis [188]).

Deep Neural Networks (DNN) are gaining prominence in solving complex problems like image recognition, natural language processing, and hardware applications. With companies investing heavily in the development of customized machine learning models, these models become valuable intellectual properties. The uniqueness of DNN models lies in their layers and hyper-parameters, making them potential targets for side-channel attacks that seek

to compromise their confidentiality. Such attacks are feasible on traditional shared computing platforms [163], but their viability on emerging cloud platforms powered by Field Programmable Gate Arrays (FPGA) has not been studied.

Investigating attack vulnerability on FPGA-based systems is a significant concern as major cloud providers have begun offering FPGA-based instances to enhance DNN training and testing. FPGA virtualization is used for flexible resource allocation, enabling multiple users to concurrently deploy their logic on the same FPGA board. Despite the logical and physical separation of users' logic execution, potential security implications arise under multi-tenant FPGA environments.

Our research is focused on exploring remote power side-channel analysis on shared FPGA instances, gauging the feasibility of model-stealing attacks. We developed an attack method that allows remote theft of model secrets through power consumption analysis, requiring no physical access to the FPGA instance. Utilizing machine-learning-based inference models, the attack can identify individual layers and their hyper-parameters when a power sensor is deployed on the same FPGA board running the victim's DNN model.

We validated this attack on multiple DNN models, achieving over 90% inference accuracy for secret elements across all models. Importantly, our attack demonstrated generality, being effective even when training and testing models belong to different families. The results underscore the practical threat posed by FPGA-based model-stealing attacks, necessitating the development of new defense mechanisms. This pioneering work presents the first attack exploiting remote power side-channel for DNN model theft and contributes a method for the DNN model's architecture reconstruction without physical access to the FPGA platform. The intellectual property of machine-learning companies, or DNN model secret, should be cautiously protected when utilizing cloud computing resources. We urge the security community to devise cost-effective defense solutions against this threat.