Deep Learning-based Hardware Trojan Detection with Block-based Netlist Information Extraction

Shichao Yu, *Student Member, IEEE,* Chongyan Gu, *Member, IEEE,* Weiqiang Liu, *Senior Member, IEEE,* Máire O'Neill, *Senior Member, IEEE*

Abstract—With the globalization of the semiconductor industry, hardware Trojans (HTs) are an emergent security threat in modern integrated circuit (IC) production. Research is now being conducted into designing more accurate and efficient methods to detect HTs. Recently, a number of machine learning (ML)-based HT detection approaches have been proposed; however, most of them still use knowledge-driven approaches to design features and often use engineering intuition to carefully craft the detection model to improve accuracy. Therefore, in this work, we propose a data-driven HT detection system based on gate-level netlists. The system consists of four main parts: 1) Information extraction from netlist block; 2) Natural language processing (NLP) for translating netlist information; 3) Deel learning (DL)-based HT detection model; 4) HT component final voter. In the experiments, both a long short-term memory networks (LSTM) model and convolutional neural network (CNN) model are used as our detection models. We performed the experiments on the HT benchmarks from Trust-hub and K-fold crossing verification has been applied to evaluate different parameter settings in the training procedure. The experimental results show that the proposed HT detection system can achieve 79.29% TPR, 99.97% TNR, 87.75% PPV and 99.94% NPV for combinational Trojan detection and 93.46% TPR, 99.99% TNR, 98.92% PPV and 99.92% NPV for sequential Trojan detection after voting-based optimization using the LEDA library-based HT benchmarks (*logic_level*=4, upsampling, LSTM, 5 epochs).

Index Terms—Hardware Trojan detection, Deep learning (DL), Natural language processing (NLP), Word embedding, Long short term memory (LSTM), Convolutional neural network (CNN).

1 Introduction

The globalization of the semiconductor industry means that the production chain of integrated circuits (ICs) is now separated and distributed worldwide. In order to reduce production costs and shorten the time-to-market, more and more third-party entities are involved in the supply chain to provide outsourcing of design services, foundry services and off-the-shelf intellectual property (IP). However, a number of untrusted entities may also be involved which increases the security risk from HT attacks across the supply chain.

A HT can be a malicious circuit inserted into an IC design or a malicious modification of the circuits in order to change the normal behaviour of the chip or leak secret information from the chip. Trojans can be inserted into an IC at both the design and manufacture stages. They can be inserted through the attackers hidden in the design team, the design support from the untrusted third-party vendors, hacked electronic design automatic (EDA) tools, the foundry team and even during the procedure of design files transfer. According to their physical characteristics, HTs can be classified into two categories: parametric Trojans and functional Trojans. It is easier to insert functional Trojans at a design stage because there are more opportunities for third-party entities to accomplish Trojan insertion. Moreover, the logic of functional Trojans can be cleverly designed in size and activated stealthily to achieve its malicious purpose, which is challenging for detection approaches using traditional IC tests and verification solutions [1], [2].

To tackle these security threats, various functional Trojan detection techniques were proposed over the past decade [3]. However, conventional HT detection approaches based on simulation, side channel analysis (SCA), reverse engineering and logic testing have shortcomings. Both simulation and logic testing have difficulties in generating comprehensive test vectors. SCA approaches usually need a 'golden' circuit and are sensitive to process variation. Moreover, for both the reverse engineering and SCA attacks, the preparation cost of test platforms or the extra overhead of the integration of detection sensors in ICs could make the detection very expensive.

Static HT detection techniques, which can check Trojans without the need to run the circuit in design-time, have been proposed to prevent HT insertion before manufacturing and provide timely feedback to the design team. For example, machine learning (ML)-based and neural network (NN)-based HT detection methods have been proposed to detect and prevent HT-insertion at design-time without involving

Manuscript received January 28, 2021. This research is supported by RISE (ukrise.org - EP/R011494/1) and funded by the UK Engineering and Physical Sciences Council (EPSRC) and the National Cyber Security Centre (NCSC), and in part by the National Natural Science Foundation of China under Grant 62022041.

- *S. Yu, C. Gu and M. O'Neill are with the Centre for Secure Information Technologies, Institute of Electronics, Communications, and Information Technology, Queen's University Belfast, Belfast BT3 9DT, U.K. (e-mail: syu08@qub.ac.uk; cgu01@qub.ac.uk; m.oneill@ecit.qub.ac.uk).*
- *W. Liu is with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China (e-mail: liuweiqiang@nuaa.edu.cn).*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

any extra complicated pre-processing or introducing additional overheads. In these approaches, HT related features are directly extracted from circuit designs and fed into ML/NN models to train HT detection models [4], [5], [6], [7], [8], [9], [10], [11]. Existing ML/NN-based HT detection methods have to depend on the pre-understanding of the characteristics of a circuit netlist (called knowledge-driven) to extract effective features for training ML/NN models. Hence, knowledge of the circuits, including the circuit topology, types of components, and types of HTs, is essential for the detection and determines the accuracy of detection results. However, it is difficult to ensure an accurate feature extraction for different circuits that could include a variety of HTs in practice.

In this paper, to address the knowledge-driven based problem, we propose a HT detection system, which only depends on the circuit netlist without requiring any pre-knowledge of the circuits. The method, referred to as data-driven, only utilizes the circuit netlist from which the deep learning (DL) algorithm automatically extracts features and then learn models by itself. Therefore, the proposed HT detection system provides an extremely simplified detection process without the need for any pre-processing or extra circuit overheads and it is also effective for various types of circuits.

In addition, for ML/NN-based detection methods, the number and variety of Trojan samples used in experiments will affect the accuracy of the detection. In previous research only 26 gate-level samples using Synopsys 90nm library from the Trust-Hub benchmark [12] are used for evaluation, the size of which is too small to provide a comprehensive result and analysis. This research utilizes for the first time, the recently LEDA system released 250nm technology-based Trojan benchmark [13], which contains 914 samples from the Trust-Hub [12]. More specifically, the main contributions of this paper are summarized as follows:

- A data-driven HT detection system is proposed to effectively detect HTs without any pre-knowledge of the circuits. Compared to other works, the proposed system has significantly simplified the whole detection process with no additional hardware overheads.
- A Natural language processing (NLP) technique has been utilized for feature extraction from the circuit netlist for HT detection. To the best of the authors' knowledge, this is the first time NLP has been applied on raw gate-level netlist data for HT detection.
- Data-driven DL models, such as LSTM and CNN, are utilized for data training based on the extracted features using the NLP algorithm.
- The first time that the LEDA-based Trojan benchmark [13] containing 914 samples from Trust-Hub [12] is utilized for providing a comprehensive HT detection evaluation.
- The experimental results show that both the LSTM and CNN DL models achieve good HT detection performance for various Trojan netlists from publicly available HT benchmarks using our system and the proposed netlist information extraction strategy.

The remainder of this paper is organized as follows: Section 2 introduces HTs and previous research. Session 3 presents an architectural overview of the proposed HT detection system. Section 4 presents the NLP technique for netlist information extraction and the DL models for training. Section 5 evaluates the proposed HT detection system with different DL models and using different configuration parameters. Section 6 discusses a comparison with other existing research and presents the potential of this HT detection system for multi-type HT detection. Finally, Section 7 provides concluding remarks.

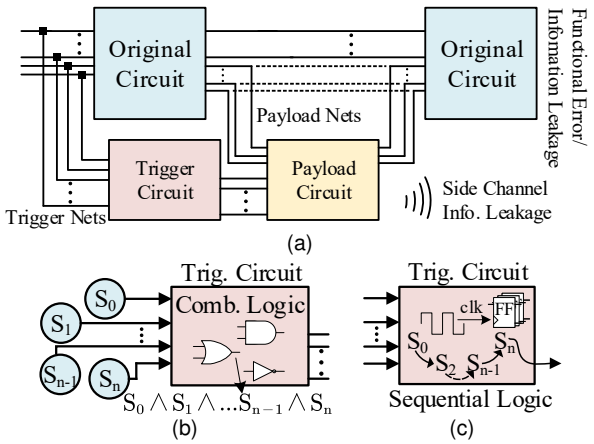

Fig. 1. The circuit models of HTs. (a) functional Trojans. (b) combinational Trojans. (c) sequential Trojans.

2 Background

2.1 Hardware Trojans

A HT can be a malicious circuit inserted into IC designs or a malicious modification of the circuits. Typically, based on the type of physical characteristics, HTs can be classified into two categories: parametric and functional Trojans. Parametric HTs are inserted into a target circuit by modifying wire thickness, dopant area and doping concentration of the predefined transistors or any other circuit parameter [14], while functional HTs are malicious logic implemented into a chip design by inserting new circuits or modifying the original circuits, components or wires. The implementation of parametric Trojans involves manipulating the IC layout during the manufacturing process, which is hard to achieve based on feedback we received from industry. Hence, this paper focus on functional HTs which can be inserted at gate-level during the design stage.

Fig. 1(a) shows a typical functional HT circuit, which consists of trigger logic and payload logic [1]. The trigger circuit is a group of sensing circuits monitoring a set of signals in order to activate the payload logic at a precise time, while the payload circuit works once the trigger is asserted and executes the predefined malicious logic functions, such as causing logic error or leaking key information [15]. Based on the trigger circuit, functional HTs can be classified as combinational Trojans and sequential Trojans, as shown in Fig. 1(b) and Fig. 1(c), respectively. The trigger circuit in combinational Trojans is typically built from combinational logic. Its trigger condition is designed to require the monitored signals to meet multiple conditions simultaneously. The trigger circuit in sequential HTs is designed to require a series of signals reaching specified values in sequence [16].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

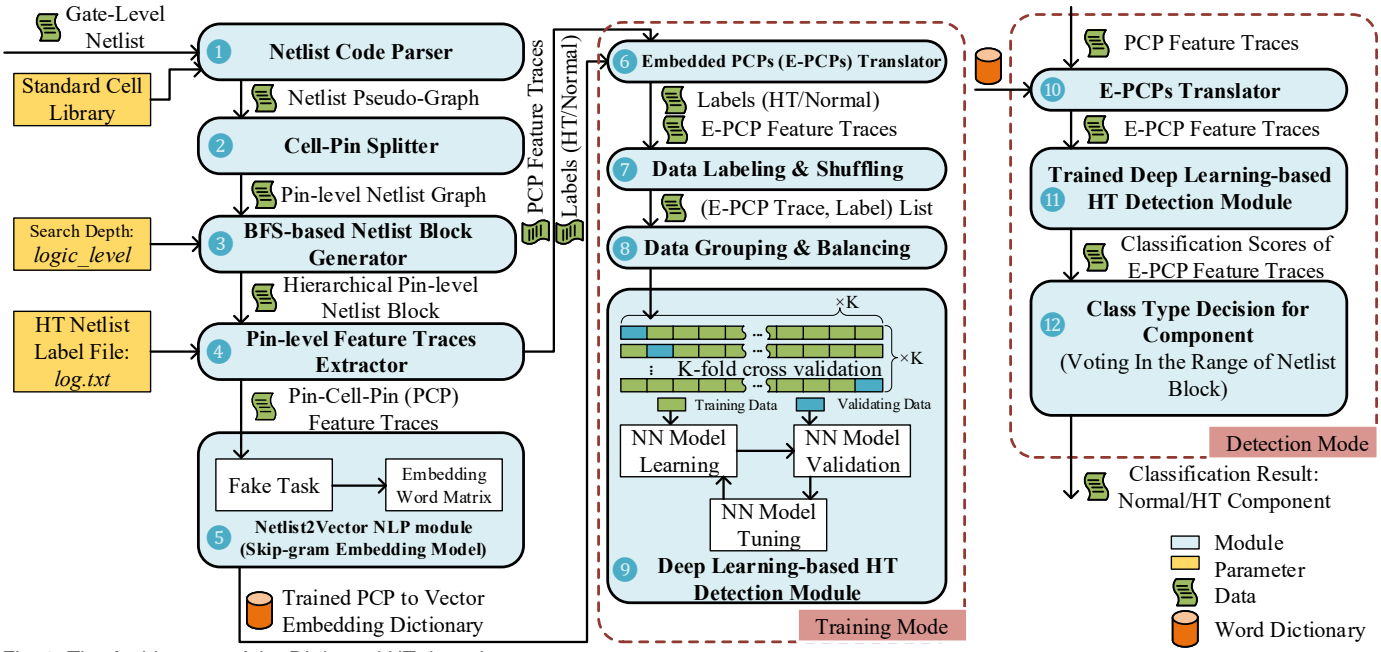

Fig. 2. The Architecture of the DL-based HT detection system.

2.2 Previous Research

Static HT detection approaches extract HT related circuit information without the need to run the circuit. Design sources, such as gate-level netlists, are usually used for detection in this scenario. Improved by ML, some new static detection approaches haven been developed to improve the efficiency of traditional ad-hoc approaches [17], [18], [19].

In [6], a K-means clustering-based HT detection approach that analyses controllability and observability features of nets has been proposed. [7] tested the same features in [6] on supervised ML algorithms and obtained a better classification result using a Bagged Trees model. However, with the limitation of computing power, the controllability and observability of the nets with deep depth in the netlist cannot be calculated correctly. Hence, the feature can not be generally applied on all netlists for HT detection. Although [8] enhanced the controllability and observability-based Trojan classifier by gate-level feature statistics (i.e., the number of primitives), the feature is specific for field programmable gate arrays (FPGA) netlists only. [4] identified 5 general circuit features for SVM-based HT detection. Moreover, [5] manually designed 51 circuit features for gate-level statistical analysis and screened the best 11 features for HT detection using the Random Forest model. Recently, an NN model, multi-layer perceptron (MLP), was the first to train for HT detection on gate-level netlists in [9]. However, the utilized features are the same as [5]. Similarly, [10] adopted an auto-encoder model for HT detection, but the features are still manually designed based on the prior knowledge of the circuits. Although [11] attempted to auto-encode the circuit structures as features, their proposed encoding method has to still use pre-knowledge of the netlist transition probability and the detection accuracy still highly depends on the manual adjustment of parameters.

In summary, most of the existing ML-based HT detection research still relies on the pre-understanding of the circuit characteristics to design effective features for training ML/NN-based HT detection models. Even with DL-based algorithms (auto-encoder and LSTM), features are still identified by knowledge-driven approaches without utilizing the self-learning ability of DL algorithms. Additionally, this previous research only used 26 gate-level samples from the SAED-based benchmark [12] for experiments, the size of which is too small to provide comprehensive results and analysis (especially for DL-based approaches).

3 Proposed HT Detection System

Fig. 2 shows the architecture of the proposed HT detection system, which can be divided into four components as follows:

1) Step 1 to 4: Information extraction from "netlist-block";
2) Step 5: NLP model for encoding netlist information;
3) Step 6 to 11: DL-based HT detection model;
4) Step 12: HT component final voter.

In steps 1 to 4, the Pin-Cell-Pin (PCP) feature traces are extracted using a previously developed platform [20], which will be explained in Section 4.1 to 4.3. The skip-gram prediction model from NLP is utilized in step 5 to train and derive word embedding dictionary which contains the distributed representations of each PCP word. When the system works in training mode, each PCP feature trace in the data set is translated in step 6 to an embedded-PCP (E-PCP) feature trace. In step 7, feature traces are labelled as *HT* or *normal* and then the data is shuffled. In step 8, all the feature traces are divided into K groups to prepare for K-fold cross-validation. To balance the amount of data in the K groups, data balancing methods are applied to the data set. Finally, K-fold cross-validation is implemented for training and validating the DL-based HT detection model in step 9. When the DL-based model is trained, the system will switch to a detection mode. The PCP feature traces from the netlist under test are first embedded and then classified

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

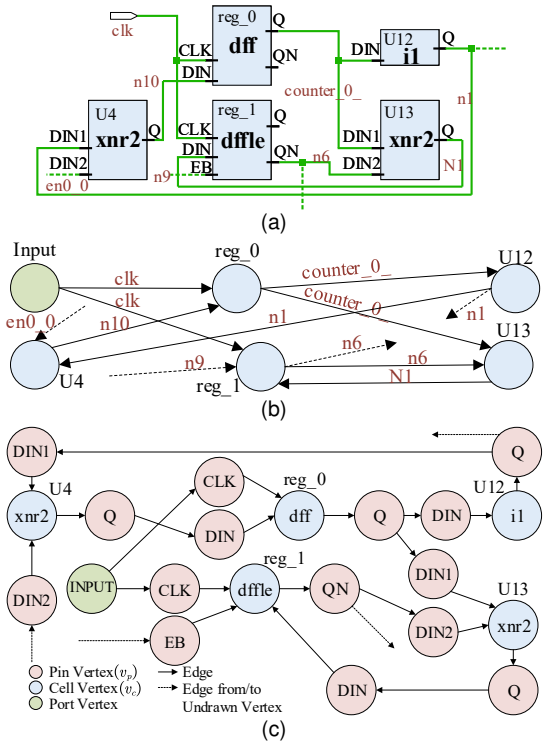

Fig. 3. Pin-level netlist graph generation. (a) The schematic of a gate-level netlist slice from a Trojan trigger. (b) The corresponding directed graph. (c) pin-level directed graph of the netlist slice.

by the trained DL-based detection model in step 10 and 11. In step 12, majority voting is applied to the classification results of the feature traces belonging to the same netlist block to determine their centre component's class types: HT or normal.

4 Proposed Netlist Information Extraction Method and HT Detection Method

In previous ML-based HT detection approaches the choice of specific features for classification is a crucial step to improve efficiency. Although a HT is like a hidden "block box" in traditional IC tests, rarely triggered and observed, it is still a normal circuit design. In order to learn the features of Trojan circuits using DL-based detection model, a netlist information extraction method working on a design's gate-level netlists is proposed.

4.1 Pin-level Netlist Graph

Generally, the topology of a netlist can be represented as a direct graph (containing loops and multiple edges), in which components (instantiations of the cells in library) and nets can be considered as nodes and edges, respectively. The graph of a netlist slice from the s15850_T436 Trojan [12] (Fig. 3(a)) is shown in Fig. 3(b). In order to present the netlist topology more generally and eliminate the random net names and component names generated by design compiler tools, the pins of each component are separated into independent vertices and extra direct edges are inserted between each component and its pins. A new direct pin-level graph is generated as shown in Fig. 3(c). As the edge between each pair of output and input pins is unique, any

Algorithm 1 Netlist block extraction for components in netlist.

Input: Pin-level netlist graph, $G(V, E)$ (V is a set of vertices, E is a set of directed edges, $E \subseteq \{(x, y) \mid x, y \in V \wedge x \neq y\}$); Searching depth, $logic\_level$;
Output: Dictionary to store all the netlist block, D_{nb};

Extract cell vertices list L_{vc} from G;
for each v_c in L_{vc} **do**
  Initialize Dict. D_t to store v_c's netlist block tree;
  $cnt \leftarrow logic\_level$
  if indegree(v_c) > 0 **then**
    $D_t['I'] \leftarrow$RECURSION(v_c, cnt, $'I'$)
  end if
  if outdegree(v_c) > 0 **then**
    $D_t['O'] \leftarrow$RECURSION(v_c, cnt, $'O'$)
  end if
  $key \leftarrow$ Instance Name of v_c
  $D_{nb}[key] \leftarrow D_t$
end for
return D_{nb};

procedure RECURSION(v_c, cnt, $Direction$) ▷ Recursive BFS on G
  Initialize List L_p to store the accessed nets;
  if $cnt > 0$ **then**
    $ll \leftarrow (logic\_level - cnt) + 1$
    if $Direction = 'I'$ **then**
      for each v_p in $\{v \mid (v, v_c) \in E\}$ **do**
        for each $v_p^{'}$ in $\{v \mid (v, v_p) \in E\}$ **do**
          $v_c^{'} \leftarrow$ parent($v_p^{'}$)
          L_p.push($[v_c^{'}, v_p^{'}, v_p, v_c, ll]$)
          L_p.push(RECURSION($v_c^{'}$, $cnt - 1$, $'I'$))
        end for
      end for
    else if $Direction = 'O'$ **then**
      for each v_p in $\{v \mid (v_c, v) \in E\}$ **do**
        for each $v_p^{'}$ in $\{v \mid (v_p, v) \in E\}$ **do**
          $v_c^{'} \leftarrow$ child($v_p^{'}$)
          L_p.push($[v_c^{'}, v_p^{'}, v_p, v_c, ll]$)
          L_p.push(RECURSION($v_c^{'}$, $cnt - 1$, $'O'$))
        end for
      end for
    end if
  end if
  return L_p;
end procedure

net edge in the original netlist graph can be determined by unique ordered pair of pin vertices (x, y), where x is the tail (output pin) of the edge and y is the head (input pin) of the edge. This procedure is carried out by the **Cell-Pin Splitter** module (step 2) in Fig. 2. Based on the pin-level netlist graph, a netlist information extraction method is designed for the NLP-based data encoding and DL-based HT detection, as described in the next section.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

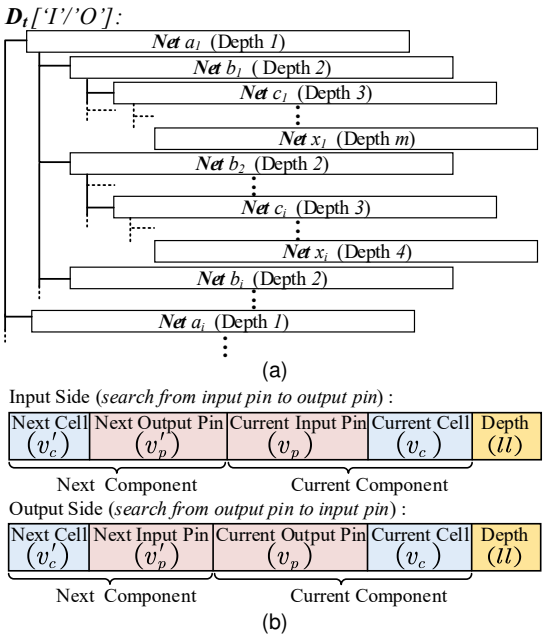

Fig. 4. Netlist block format. (a) The tree-like list for storing nets in netlist block(searching depth = m). (b) The vertex information of each accessed net stored in the netlist block.

4.2 Netlist Block and Its Searching Depth

4.2.1 Netlist-Block

The netlist-block of each component in a netlist is defined as the surrounding netlist topology found by breadth first searching (BFS) starting from this component on the design's pin-level netlist graph while this component is defined as the centre component of the netlist block.

Algorithm 1 shows how the netlist-block (D_t) of each component is extracted from the pin-level netlist graph (G). The searching depth of BFS is defined by *logic_level* (further explained in Section 4.2.2). The searching direction of the BFS procedure (RECURSION) is also controlled by parameter *Direction*. On the input side ($'I'$-side) of the component, the program searches the netlist topology from input pins to output pins, while on the output side ($'O'$-side), it searches from the output pins to input pins. Each accessed net is described by an array of vertices $[v_c^{'}, v_p^{'}, v_p, v_c]$ ($v_c^{'}$ is the next cell vertex, $v_p^{'}$ is the next pin vertex, v_p is the current pin vertex, v_c is the current cell vertex). The accessed nets on the $'I'$ and $'O'$ sides are recorded with a hierarchical structure in two lists ($D_t['I']$ and $D_t['O']$), as shown in Fig. 4(a). The nets are kept in a tree-like structure and their hierarchical dependencies are defined by depths. The vertex information of each accessed net is shown in Fig. 4(b), in which the next cell vertex defines the next level's starting node, the next and current pin vertices define the accessed net and the current cell vertex defines the net's father node. In a netlist block, all the structural information surrounding the centre component are retained. It contains all the nearby components within the range of the searching depth and the spatial dependence between each path. Therefore, the function of the component in its local netlist area can be reserved.

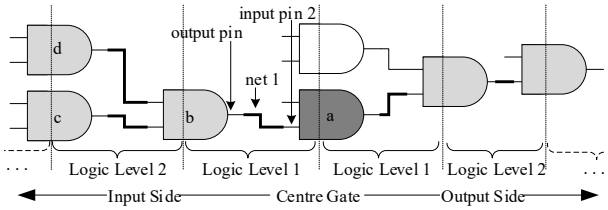

Fig. 5. The definition of logic level.

4.2.2 Searching Depth

As shown in Algorithm 1, the parameter *logic_level* is defined to limit the searching depth for the netlist topology. For example, when *logic_level* is 5, the **BFS-based Netlist Block Generator** module (step 3 in Fig. 2) will extract a netlist block within 5 logic levels of the centre component. The logic level, as shown in Fig. 5, is defined according to the number of gates encountered on the route when transmitting from the centre gate to the current component in netlist. In Fig. 5, a is the centre gate, and when d is the current gate, on the path from a to d, gates b and d are counted. The path from a to b is in logic level 1. The path from b to d is in logic level 2.

The logic level affects the precision of the circuit features in netlist block and there is a trade-off in the selection of the *logic_level* value. With a large *logic_level* value, the structural information contained in the netlist block will be more specific and more in depth features can be included. However, it can also include less relevant and redundant information. On the other hand, a smaller *logic_level* value will reduce the information retained in the netlist block, while ensuring the information is more relevant and smaller in size. In order to choose the more appropriate *logic_level* value, the number of components in the netlist block is estimated. The mutual information between the signals at the edges of the netlist block and the signals from the centre component is also estimated for different *logic_level* values. To simplify the estimation, reconvergent fan-out is not considered and an ideal model is used. The actual netlist may be more complicated than this ideal model, however, it provides a reference for *logic_level* selection.

Assuming there is a netlist block for which the components on both sides of the centre component are all two-input one-output gates, and on the output side each output pin is connected to 2 input pins at the next logic level (as shown in Fig. 6(a)), the relationship between the number of components and the *logic_level* can be represented in Fig. 6(b). Then, assuming the logic function of all components is AND or OR operations and applying random inputs to all the input ports in the netlist, the trend of mutual information between an input signal at the edge of the input side and the output signal of the centre component (identified by the blue arrows in Fig. 6(a)) can be demonstrated in Fig. 6(c). Similarly, Fig. 6(d) shows the relationship between an output signal at the edge of the output side and an input signal of the centre component (the brown arrows in Fig. 6(a)). Based on the estimation, the initial value of $logic\_level$ is recommended to be set to 3, 4, or 5, offering sufficient information while eliminating less relevant information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

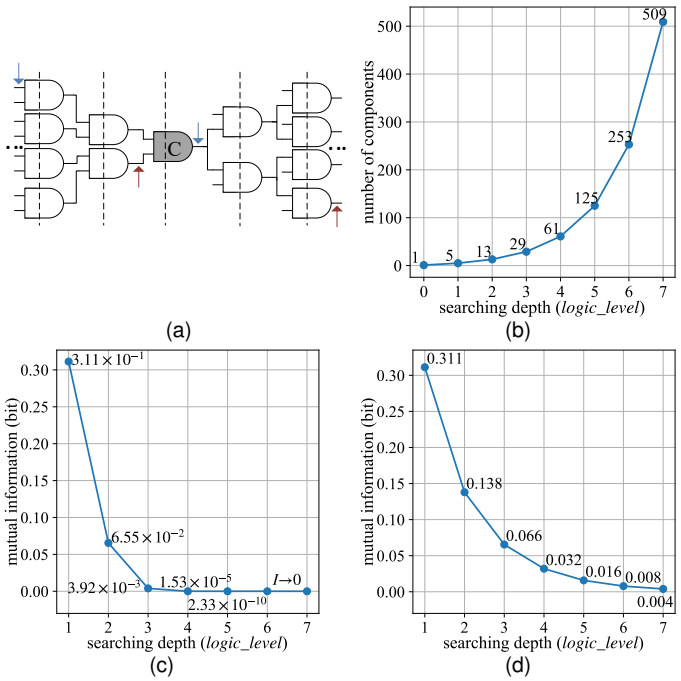

Fig. 6. Estimation for *logic_level* selection. (a) The assumed netlist block model. (b) The number of components in this netlist block. (c) The estimated mutual information between the edge of input side and the centre component. (d) The estimated mutual information between the edge of output side and the centre component.

4.3 Netlist Path Information

The netlist block translation to PCP feature traces is described in Algorithm 2. The netlist block contains the topology information surrounding the centre component, but the topology information needs to be extracted from the hierarchical structure. The FLATTEN procedure in Algorithm 2 utilizes a stack to unfold the nets in the netlist block. The Pin-Cell-Pin (PCP) feature trace is generated in Algorithm 2 to describe the netlist path. PCP, a basic element of the PCP feature trace, is a tuple of the input pin vertex (v_{in_p}), cell vertex (v_c) and output pin vertex (v_{out_p}) from the same component that connects two nets in adjacent logic levels. According to the net information (as shown in Fig. 4(b)), Equation 1 indicates the process of PCP generation from two adjacent nets (*logic_level* is $x-1$ and x) on the $'I'$ side and $'O'$ side.

$$\left.\begin{matrix}[v_c^{'}, v_p^{'}, v_p, v_c]_{x-1} \\ [v_c^{'}, v_p^{'}, v_p, v_c]_{x}\end{matrix}\right\} \rightarrow \begin{matrix} I\text{: } [v_{p,x}, v_{c,x}, v_{p,x-1}^{'}] or \\ O\text{: } [v_{p,x-1}^{'}, v_{c,x-1}^{'}, v_{p,x}], \end{matrix} \quad (1)$$

A PCP feature trace can be obtained by listing the PCPs on the route when traversing the entire netlist block from the input to the output. After the translation, for each netlist block, all its PCP feature traces are stored in a list. The data structure is shown in Fig. 7(a), where each list element is a PCP feature trace. Meanwhile, since the *logic_level* (*ll*) defines the searching depth for the netlist block, the number of PCP elements in each PCP feature trace is $2 \times ll - 1$. This procedure is carried out by a **Pin-level Feature Traces Extractor** module as shown in the system architecture (step 4 in Fig. 2).

Algorithm 2 PCP feature trace extraction from netlist block.

Input: Dictionary of netlist blocks, D_{nb};
Output: Dictionary to store PCP feature traces, D_{ft};

for each key, D_t pair in D_{nb} **do**
  Init. List L_{PI}, L_{PO} for storing netlist paths on 2 sides;
  Initialize List L_{ft} to store PCP feature traces;
  $L_{PI} \leftarrow$FLATTEN($D_t['I']$)
  $L_{PO} \leftarrow$FLATTEN($D_t['O']$)
  for each $path_I$ in L_{PI} **do**
    for each $path_O$ in L_{PO} **do**
      Initialize List L_{pcps} to store PCP feature trace;
      for $i =$length of $path_I - 1$; $i > 1$; $i - -$ **do**
        $net_a \leftarrow path_I[i-1]$
        $net_b \leftarrow path_I[i]$
        $PCP \leftarrow [net_b.v_p, net_b.v_c, net_a.v_p^{'}]$
        L_{pcps}.push(PCP) ▷ input path PCPs
      end for
      $net_a \leftarrow path_I[0]$
      $net_b \leftarrow path_O[0]$
      $PCP_0 \leftarrow [net_a.v_p, net_a.v_c, net_b.v_p]$
      L_{pcps}.push(PCP_0) ▷ centre PCP
      for $i = 1$; $i <$length of $path_O - 1$; $i + +$ **do**
        $net_a \leftarrow path_O[i-1]$
        $net_b \leftarrow path_O[i]$
        $PCP \leftarrow [net_a.v_p^{'}, net_a.v_c^{'}, net_b.v_p]$
        L_{pcps}.push(PCP) ▷ output path PCPs
      end for
    end for ▷ end of L_{PO} loop
  end for ▷ end of L_{PI} loop
  $D_{ft}[key] \leftarrow L_{pcps}$
end for
return D_{ft};

procedure FLATTEN(L) ▷ L, list of nets
  Initialize List L_S as a stack for unfolding nets;
  Initialize List L_P to store lists of nets (netlist paths);
  for each net in L **do**
    if L_S is not empty $\wedge$ L_S.top.$ll \geqslant net.ll$ **then**
      L_P.push(L_S)
      repeat
        L_S.pop
      until L_S is empty $\vee$ L_S.top.$ll < net.ll$
    else
      L_S.push(net)
    end if
  end for
  if L_S is not empty **then** ▷ clear stack
    L_P.push(L_S)
  end if
  return L_P;
end procedure

4.4 NLP-based Encoding for Extracted Information (Net2Vec)

Word embedding is a collective name for a set of language modelling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers [21]. As each unique PCP can be treated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

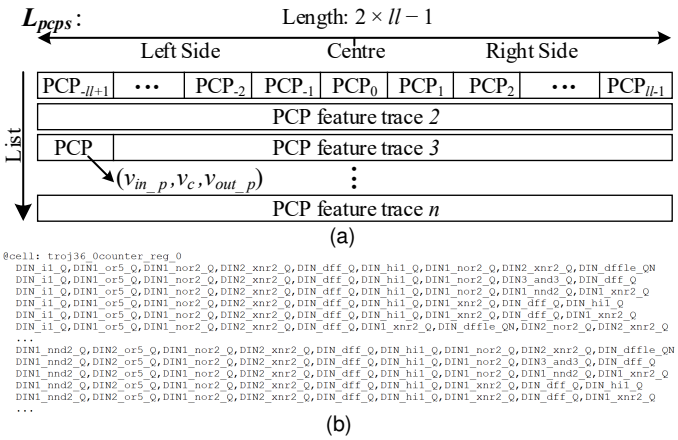

Fig. 7. The sentence-like netlist path information. (a) The data structure of the PCP feature trace list for each netlist block. (b) An example of a PCP feature trace list in text format for component 'troj36_0counter_reg_0' in benchmark netlist s15850_T436 ($logic\_level$ = 5).

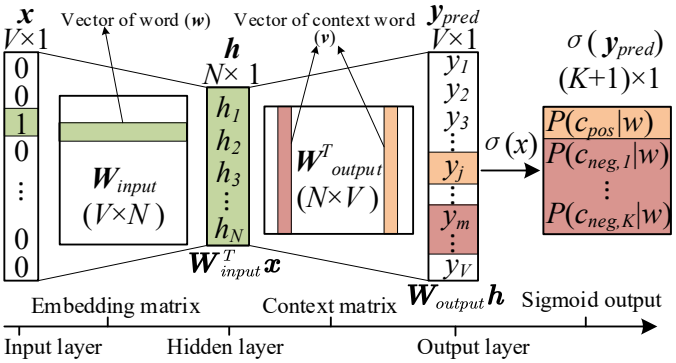

Fig. 8. The structure of Skip-gram word embedding model with negative sampling.

as vocabulary, the PCP feature trace is similar to sentences in natural language. Therefore, in this paper, we propose a Net2Vec methodology, in which a word embedding algorithm is utilized to encode PCP feature traces.

The PCP vocabulary (w) can be assembled according to Equation (2). An example of the rewritten PCP feature traces using PCP words is shown in Fig. 7(b). The characters related to cell size (like "s1" in "and2s1") are eliminated to make the PCP word more general.

$$w = v_{in\_p} + \text{"\_"} + v_c + \text{"\_"} + v_{out\_p} \tag{2}$$

In this research, a skip-gram neural network model [22] with negative sampling is utilized to embed PCP words. A PCP to numeric vector translation matrix is built after the model training. The skip-gram model builds a fake task that utilizes a given centre word (w_I) to predict its context words (C_{pos}) to train the weight matrix in its neural network and finally learns the meaningful vector representations of the words. The architecture of the skip-gram model is shown in Fig. 8. The model contains one input layer, one hidden layer and one output layer. The weight matrix from the input layer to the hidden layer, $\boldsymbol{W}_{input}$, is the word embedding matrix. Each line of this matrix represents a word vector. When the vocabulary size of PCP words is V, for a centre PCP word w_I, $\boldsymbol{x}$ is its one-hot encoded V-dim vector. Then $\boldsymbol{h}$ is the N-dim word vector from the corresponding row ($\boldsymbol{w}$) in $\boldsymbol{W}_{input}$, projected by $\boldsymbol{x}$ ($\boldsymbol{h} = \boldsymbol{W}_{input}^T \boldsymbol{x}$). At the output layer, $\boldsymbol{y}_{pred}$ is the V-dim matrix product of the context weight matrix ($\boldsymbol{W}_{output}$) and $\boldsymbol{h}$, which represents the unnormalized probability distribution of all possible context PCP words when the centre PCP word is w_I. Negative sampling is utilized to improve the computational efficiency for each centre word and context word (positive) pair (w_I, c_{pos}). K words are randomly drawn from a noise distribution $P_n(w)$ (equation (3)) as negative words (c_{neg}). $P_n(w)$ is a probability distribution related to the word (w) frequency.

$$P_n(w) = \frac{count(w)^{0.75}}{\sum_{i=1}^{V}(count(w_i))^{0.75}} \tag{3}$$

Therefore, for each w_I, only $K+1$ results y_{pos}, $y_{neg,1}, ..., y_{neg,K}$ in y_{pred} need to be calculated instead of the whole vocabulary size (V). Next, the sigmoid function $\sigma(x)$ is applied to get the normalized probabilities $p(c_{pos}|w_I)$. $\{p(c_{neg,i}|w_I)|i = 1, 2, ..., K\}$ is for loss calculation and backward propagation. The main purpose of native sampling is to optimize the weight matrices $\boldsymbol{W}_{input}^T$ and $\boldsymbol{W}_{output}^T$ by maximizing the probability of the observed positive pairs $p(c_{pos}|w_I)$ while minimizing the $p(c_{neg,i}|w_I)$ for negative pairs, $i = 1, 2, ..., K$. With the variables from the model, the cost function (L) of the model can be defined as equation (4), where $\boldsymbol{v} \in \boldsymbol{W}_{output}$, W_{neg}=$\{c_{neg,i}|i = 1, 2, ..., K\}$, $\boldsymbol{\theta} = [\boldsymbol{W}_{input}^T, \boldsymbol{W}_{output}^T]$. $\boldsymbol{\theta}$ represents the weight parameters in two matrices. $\boldsymbol{\theta}$ is updated by backward propagation per (w_I, c_{pos}, W_{neg}) pair.

$$L(\boldsymbol{\theta}) = -\log\sigma(\boldsymbol{v}_{c_{pos}} \cdot \boldsymbol{h}) - \sum_{c_{neg} \in W_{neg}} \log\sigma(-\boldsymbol{v}_{c_{neg}} \cdot \boldsymbol{h}) \tag{4}$$

Different from sentences in natural language, each component's netlist block has already extracted information on the surrounding netlist components around the centre component (window size = $logic\_level$). For each PCP feature trace, PCP_0 is the centre word w_I, and the remainder, $\{\text{PCP}_i|i \neq 0\}$ is the context words, c_{pos}. K negative samples are randomly drawn from the noise distribution $P_n(w)$ of each PCP word in the corpus for each $(\text{PCP}_0, \text{PCP}_i)$ pair, $i \neq 0$. The corpus here is the PCP feature traces extracted from the netlist samples compiled by the same process-specific standard cell library, whether it has Trojans or is Trojan-free. In our model, $K = 5$, each training batch contains 1 positive PCP word and 5 negative PCP words. $N = 100$, the hidden layer contains 100 neurons. After training, a $V \times 100$ trained PCP word embedding matrix, $\boldsymbol{W}_{input}$, can be obtained from the model. V is the vocabulary size of the corpus. The i-th row ($\boldsymbol{w_i}$) in $\boldsymbol{W}_{input}$ represents the trained 100-dim vector representation of the PCP_i word at index i in the vocabulary dictionary. The training procedure is completed by the **Netlist2Vector NLP** module (step 5).

4.5 Embedded PCP (E-PCP) Word Vectors

As an example, an embedded matrix containing 204 PCP word vectors is trained using the feature traces from 60 HT-infected netlists (Appendix, Table 11) randomly selected from the Trust-Hub benchmark library [12] when the *logic_level* is 4, which includes both combinational and sequential (FSM-based, counter-based) Trojans inserted in 8 different host designs. The top 10 word vectors close to and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 1
PCP list sorted by the distance to the vectors of "*DIN1_nnd4_Q*"

Top 10 Nearest	Distance	Top 10 Furthest	Distance
DIN2_nnd4_Q	0.215	*SDIN_sdff_Q*	3.187
DIN3_or5_Q	0.506	*DIN_dffle_Q*	2.903
DIN2_or5_Q	0.511	*CLK_sdff_Q*	2.630
DIN4_or5_Q	0.557	*SSEL_sdff_Q*	2.628
DIN1_or5_Q	0.605	*DIN_dffle_QN*	2.612
DIN3_nnd4_Q	0.627	*SDIN_sdff_QN*	2.539
DIN1_nnd3_Q	0.635	*output*	2.429
DIN4_nor6_Q	0.722	*None_I*	2.428
DIN2_or3_Q	0.746	*CLK_dsmxc31_Q*	2.427
DIN2_nnd3_Q	0.751	*DIN7_and9_Q*	2.424

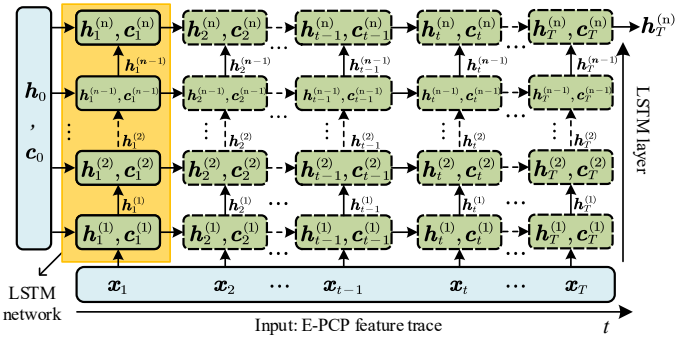

Fig. 9. The processing of each E-PCP feature trace in a stacked n-layer LSTM-based HT detection network.

far from the PCP word vector "*DIN1_nnd4_Q*" are listed in Table 1. "*None_I*" in the table is the placeholder for the vacant positions on the input side when the feature trace is short. "*output*" is the placeholder for the output ports. It can be found that the closer the word vectors, the more similar their semantics in the netlist. And the farther they are, the greater the difference between them. The semantics here are the function and role of the PCP in the netlist. The **E-PCPs Translator** module (step 6) in the system is utilized to translate the PCP feature traces into E-PCP feature traces based on the embedding matrix.

4.6 Deep Learning-based HT Detection on Embedded Feature Traces

With a large number of artificial neurons and multiple hidden neural network layers, the most significant advantage of the DL algorithm is that the neural network can progressively learn high-level features from the raw input. Utilizing the PCP feature traces explained in Section 4.3 and the PCP word embedding matrix generated by the the NLP model in Section 4.4, a DL-based HT detection method is designed and implemented in this section.

The **Deep Learning-based HT Detection Module** (step 9) in Fig. 2 implements the DL-based HT detection modules designed in this Section and carries out the model training and validation. According to the different interpretations of the PCP feature traces, two general DL models are considered—the long short term memory networks (LSTM) model and the convolutional neural network (CNN) model.

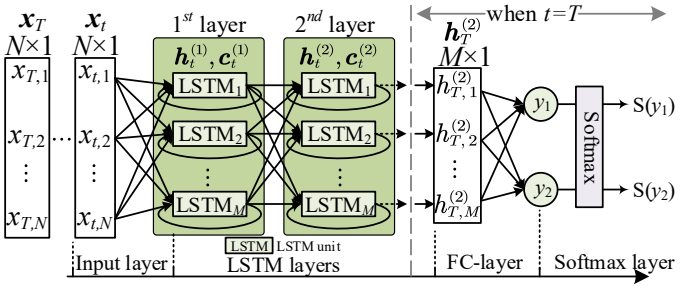

Fig. 10. The architecture of the LSTM network in Fig. 9.

4.6.1 LSTM Model

As discussed in section 4.3, a PCP feature trace contains the structural information from the corresponding netlist paths, and spatial dependency exists between PCP words in trace. Consequently, PCP feature traces can be treated as data sequences with spatial dependency. LSTMs [23], a special kind of Recurrent Neural Network (RNN), can learn long-term dependencies in entire sequences of data and hence are considered suitable for this HT detection task. Fig. 9 shows the processing of an embedded PCP (E-PCP) feature trace by an n-layer LSTM network. Each PCP is translated to the corresponding E-PCP word vector by looking up the embedding matrix. $t = 1, 2, 3, ...$ is the time step. According to the order of E-PCP vectors in the feature trace at each time step, there is one E-PCP vector (x_t) processed by the LSTM network, hence, $T = 2ll - 1$. The green block in Fig. 9 shows the state of each LSTM layer at different time steps. $\boldsymbol{h}$ is the hidden state and also the output of each LSTM layer, while cell state $\boldsymbol{c}$ is the memory of the current layer. For the first layer, at each time step, the input data is composed of ($\boldsymbol{x}_t$) at current time step t and $\boldsymbol{h}_{t-1}^{(1)}$, $\boldsymbol{c}_{t-1}^{(1)}$ from the previous time step $t-1$. For a middle layer n, the input comprises $\boldsymbol{h}_t^{(n-1)}$ from layer $n-1$ at t and $\boldsymbol{h}_{t-1}^{(n)}$, $\boldsymbol{c}_{t-1}^{(n)}$ at this layer from $t-1$. Thus, $\boldsymbol{h}$ at the last time step of the last layer can be regarded as the vector representation of the whole input sequence (E-PCP feature trace).

The architecture of the LSTM-network utilized in our system is composed of 2 LSTM layers. The dimension of hidden state M is taken to be 128, which is larger than the dimension of the word vector to retain the encoded information of the whole feature trace. As presented in Fig. 10, it contains 1 input layer, 2 LSTM layers, 1 fully connected (FC) layer and 1 softmax layer. $\boldsymbol{x}_t$ is the E-PCP word vector at time step t. The input layer maps the 100-dim x_t to the LSTM layer with 128-dim. Each dimension is handled by one LSTM unit [23]. After T time steps's encoding, a 128-dim feature vector ($\boldsymbol{h}_T^{(2)}$) for the whole E-PCP feature trace is generated. The FC-layer calculates the classification scores (y_1, y_2) and the softmax layer normalizes them to either the HT class probability $S(y_1)$ or normal class probability $S(y_2)$.

The adopted cost function for loss calculation and back-propagation is cross entropy. It estimates the difference between the probability distribution of the sample's actual class and that of the predicted classification result. The equation is shown in (5), where c_i is the class i. For each sample, $p(c_i)$ is the actual probability of class i and $q(c_i|\boldsymbol{\theta})$ is the predicted probability of class i under the current model ($\boldsymbol{\theta}$ represents all the parameters in the model). In our model

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

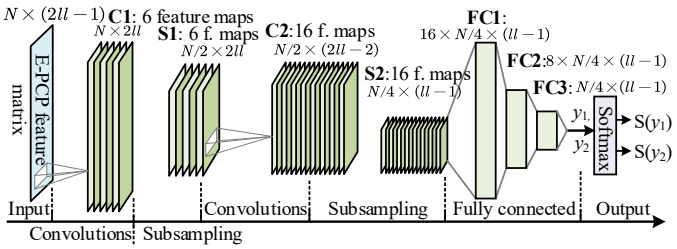

Fig. 11. The CNN network for HT detection.

there are two classes, c_0 is the normal class, while c_1 is the HT class. For a normal trace, $p(c_0) = 1, p(c_1) = 0$, while for a HT trace, $p(c_0) = 0, p(c_1) = 1$. The cross entropy can be calculated according to equation (6).

$$L(\boldsymbol{\theta}) = -\sum_i p(c_i) \log q(c_i|\boldsymbol{\theta}) \quad (5)$$

$$= -p(c_0) \log q(c_0|\boldsymbol{\theta}) - (1 - p(c_0)) \log(1 - q(c_0|\boldsymbol{\theta})) \quad (6)$$

4.6.2 CNN Model

Taking another perspective, the E-PCP feature traces can be treated as $N \times (2ll-1)$ matrices, in which each column is an E-PCP word vector, and the row width is equal to the length of the trace. Hence, instead of sequential processing, neural network (NN) models can process all the E-PCP vectors in one feature trace simultaneously as a matrix. The convolutional neural network (CNN) [24], which performs well in image processing, is suitable for building classification models from matrix data.

To evaluate its performance, we customized a CNN model based on LeNet-5 [25] and tested it in our HT detection system. The model has been trimmed to match the matrix size of the E-PCP feature trace. Its architecture is shown in Fig. 11. The model contains 2 convolution layers (**C1**, **C2**), 2 subsampling layers (**S1**, **S2**), 3 fully connected layers (**FC1**, **FC2**, **FC3**) and 1 softmax output layer. The function of the subsampling layers used in our model is max-pooling.

The convolution layers can automatically extract higher-level features from its input data. The subsampling layers perform a non-linear down-sampling on the input data to shrink the data size and reduce the exact positional features in data to enhance the robustness of the model. The subsequent FC-layers and softmax layer perform the same functions as those in the LSTM model. However, as a whole E-PCP feature is processed by the CNN model per time step, more neurons will be needed in FC-layers than in LSTM, the 3 stacked FC-layers can encode the multiple feature maps from the **S2** layer into a 2-dim score vector, (y_1, y_2).

The key parameters in our CNN model are listed in Table 2. The exact size of the output feature maps at each layer depends on the length of the E-PCP feature trace ($2ll-1$) and the number of dimensions (N) of the E-PCP word vector in the embedding matrix. The cost function is cross entropy. As the output layer is not changed, the form of the cost function is the same as that in the LSTM model (equation (6)).

4.7 Voting Method in HT Detection Mode

After model training, steps 10 to 12 (Fig. 2) utilize the trained model to detect HTs in the target netlist. A Trojan/normal component voter (**Class Type Decision for Component**, step 12 in Fig. 2) is designed to optimize the detection results and conclude the final detection results at component level.

TABLE 2
The Key features of Each layer in Our CNN Model

Layer	Kernels (num., size)	Stride (row, col.)	Padding (row, col.)	Output feature maps (num., size)
Input	—	—	—	$1, N \times (2ll-1)$
C1	$6, 3 \times 2$	$1, 1$	$1, 1$	$6, N \times 2ll$
S1	$1, 2 \times 1$	—	—	$6, N/2 \times 2ll$
C2	$16, 3 \times 3$	$1, 1$	$1, 0$	$16, N/2 \times (2ll-2)$
S2	$1, 2 \times 2$	—	—	$16, N/4 \times (ll-1)$
FC1	—	—	—	$-, 2N(ll-1)$
FC2	—	—	—	$-, N(ll-1)/4$
FC3	—	—	—	$-, 2$
Softmax	—	—	—	$-, 2$

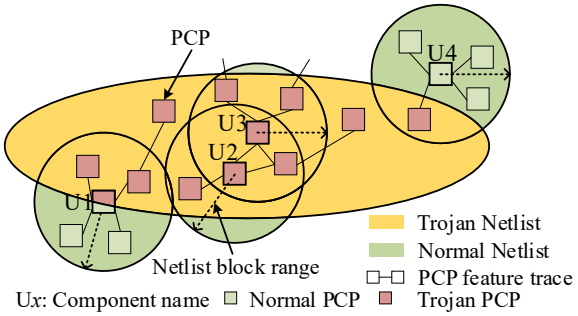

Fig. 12. The voting method in netlist block range for final classification.

As each PCP feature trace represents the structural information contained on the corresponding netlist path, some normal feature traces with partial HT-like structural features may be misclassified as HTs by the trained model and vice versa. Meanwhile, the information contained in a single PCP feature trace can not fully represent the logic of that netlist block. Hence, a voting strategy in the range of netlist blocks is adopted in this module for final HT detection. To reduce the impact of misclassification and improve the detection accuracy, instead of the PCP feature trace, components in the netlist become the object of final HT classification. The voting strategy decides the class type (HT/normal) of a component based on the voting results of all PCP feature traces in the network block with it as the centre component (Section 4.2).

According to the searching algorithm of for each netlist block (Algorithm 1), the topology of a netlist block is shared by multiple PCP feature traces. If a HT netlist exists in the range of the netlist block, its structural features should also be shared by multiple PCP feature traces. Therefore, with the classification results of all feature traces within the scope of each component's netlist block, the voting scheme can decide the class of the component by majority voting. In other words, outliers in the PCP feature trace classification results will be covered by voting. The voting procedure is shown in Fig. 12. Ux is the name of the centre component in a netlist block. If the number of PCP feature traces classified as HTs is higher than the number of normal ones in the scope of a component's netlist block, this component will be classified as a HT component, and vice versa. In this case,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

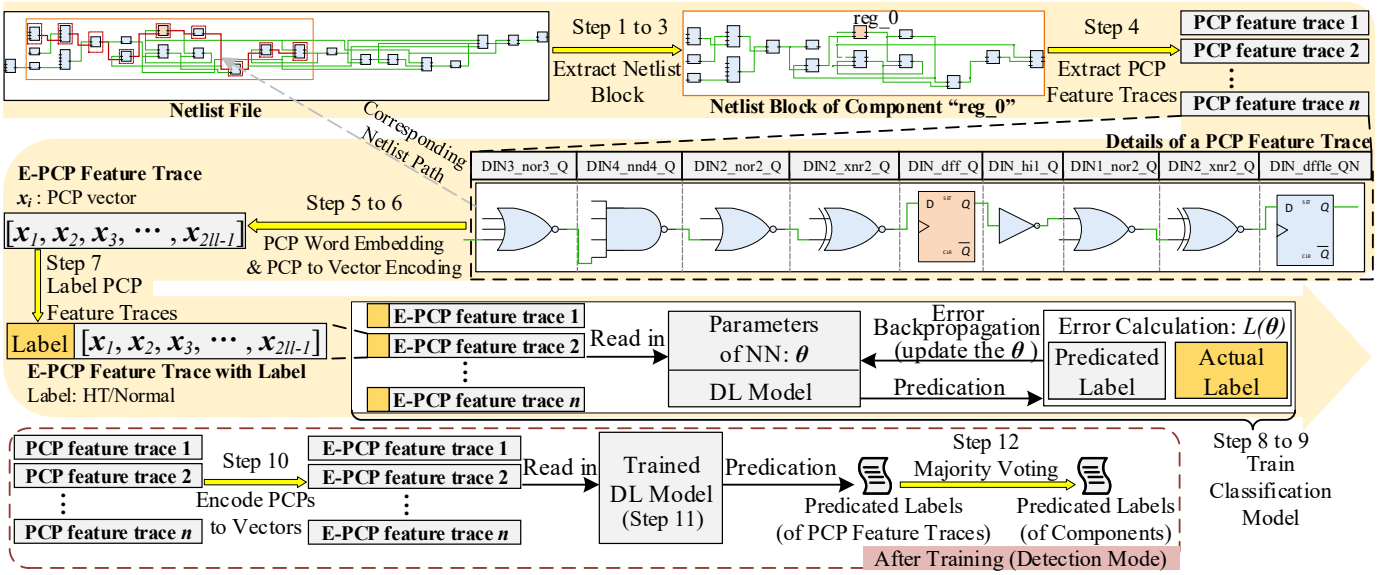

Fig. 13. Workflow of the proposed HT detection system. "reg_0" from netlist s15850_T436 [12] is the example component being processed ($logic\_level$ = 5).

U1, U2 and U3 are voted into the HT class, while U4 is classified to the normal class. For boundary conditions (as shown for U1), when the number of traces in the two classes is equal, the component will be classified to the HT class in the proposed system.

4.8 System Workflow

The detailed description of the algorithms and functional modules has been presented in the previous sections. To summarise, Fig. 13 shows the workflow of the system. The processing of component "troj36_0counter_reg_0_" ("reg_0" for short) from HT-infected benchmark netlist s15850_T436 [12] is presented as a running example to clarify the procedures. As shown in Fig. 13, steps 1 to 3 in the system (Fig. 2) extract the netlist block of "reg_0" within 5 logic levels, and then the netlist path information is extracted as PCP feature traces in step 4. The details of one PCP feature trace and the corresponding netlist path from the original netlist are provided as an example to illustrate how the topology information surrounding the centre component ("reg_0") is stored in the feature trace. After the NLP-based encoding (steps 5 and 6), the E-PCP feature traces are labelled as either HT or normal according to the class of their centre component (step 7). For example, as the centre component ("reg_0") of the extracted feature traces in Fig. 13 belongs to a Trojan circuit according to the netlist description file provided by the benchmark library [12], this feature trace is labelled as HT, and vice versa. The DL model is trained with these labelled feature traces to predicate their categories (either HT or normal class). During the training, the cost function continuously calculates the error values between the predicated labels and the actual labels, then the values will be back-propagated to the DL model to automatically update the parameters ($\boldsymbol{\theta}$) of its inner neural network based on gradient descent to reduce the predication error. In this procedure, the features (like the type, number and order of different components, used pins and specific structures, etc.) contained in the feature traces that can be utilized for classification are gradually extracted and learned by the DL model. After training, the DL model can perform HT detection based on its trained inner neural network (trained $\boldsymbol{\theta}$) without labelling.

5 Experimental Results

In this section, we evaluate the impact of system parameters on the detection results and test the performance of the proposed HT detection system.

5.1 Experimental Setup

Overall, the experimental setup consists of two parts: the platform setup, and benchmark preparation. For the platform, the modules in step 1 to 3 of the proposed HT detection system are implemented in Perl language, while all other modules are implemented in Python. An open source machine learning library, Pytorch, is utilized to build the neural network models in the system (Net2Vec, CNN, LSTM), and the detection system is running on a Linux machine with an AMD Ryzen 1700 CPU, a NVIDIA RTX 2060 GPU and 32GB memory. For the benchmarks, Trojan-infected gate-level netlists from the Trust-Hub Trojan benchmarks [12] are utilized as samples for the experiments. It should be noted that the netlist samples from Trust-Hub can be divided into two groups according to the standard cell libraries used. One is the Synopsys 90nm generic library (SAED), and the other is the LEDA system 250nm library. An overview of these benchmarks is presented in Table 3.

5.2 Parameter Control

The parameters that can be configured in our system are summarized in Table 4, and are classified into two groups: system parameters and neural network hyperparameters. For the system parameter *logic_level* defines the searching depth of a netlist block and directly influences the information contained in each feature trace. Therefore, its impact on HT detection results is investigated. For the neural network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 3
Overview of Publicly Available Trojan-infected Netlists on Trust-Hub

Category	Library	Num.	Details
Abstraction Level/ Gate/TRIT-TC, TRIT-TS	LEDA	914	580 Combinational HTs based on 8 host circuits; 334 Sequential HTs based on 8 host circuits.
Abstraction Level/ Gate/other benchamrks	SAED	26	11 Combinational HTs based on 6 host circuits; 15 Sequential HTs based on 6 host circuits.

TABLE 4
The Summary of the Parameters in the Proposed System

Category		Descriptions
System Parameter		*logic_level* (*ll*)
NN Hyper-parameters	Net2Vec Model	Embedding Dimension
	Detection Model	LSTM Networks Parameters
		CNN Parameters
	Training Procedure	Class Balancing Method(Up/Down Sampling, Weighted Loss Function); Batch Size; Number of Epochs.

hyperparameters, some of the parameters are fixed to control the total number of variables in the experiments and reduce the complexity. As we do not focus on optimizing the deep learning models, the parameter configurations of the PCP word embedding model (Net2vec) and the HT detection models are fixed as described in Section 4.4 and 4.6. The batch size is also set to 32 in all the experiments.

Meanwhile, as the feature trace numbers of the HT class and normal class in the training set are usually unbalanced, class balancing is used in the training procedure. This will change the composition of the training dataset. As such, 3 methods with 1 control group are evaluated in our experiments as follows:

1) Upsampling: This method up samples the feature traces in the smaller class by randomly duplicating samples to make the number of samples in both classes the same.
2) Downsampling: This method is similar to upsampling, but involves down sampling the larger class by randomly discarding samples.
3) Weighted Loss Function: This function deals with the imbalance by re-scaling the penalty on each class for the misclassification in training. The weights (W) are defined according to the number (N_i) of feature traces in each class. According to equation (5), the new cost function is rewritten as (7).

$$L(\boldsymbol{\theta}) = -\sum_i \frac{1}{\sum_i \frac{1}{N_i}} \frac{1}{N_i} p(c_i) \log q(c_i|\boldsymbol{\theta}) \tag{7}$$

4) Control Group: The HT detection model will be trained with the original training set without class balancing.

Overall, the impact of system parameter *logic_level* and the class balancing methods will be investigated. During training, the number of epochs indicates the number of times that the entire training dataset passes through the model.

TABLE 5
The evaluation metrics (confusion matrix)

		True Condition: Normal	True Condition: HT		
Predicted Condition	Normal	TN (True Negative)	FN (False Negative)	NPV	/
	HT	FP (False Positive)	TP (True Positive)	/	PPV (Precision)
		TNR	FNR=1-TPR		
		FPR=1-TNR	TPR (Recall)		

5.3 Evaluation

Considering the number of samples and the variants of inserted HTs, benchmarks from the LEDA library [13] are utilized as the source for the following evaluation. According to the parameter configuration in 5.2, in experiment 1, different class balancing methods are evaluated. In experiment 2, we evaluate the impact of the *logic_level* parameter swap on the detection results. Finally, in experiment 3, based on the parameter settings acquired from experiment 1 and 2, the performance of the proposed HT detection system when integrating with different DL models (LSTM and CNN) is evaluated. In the experiment 1 and 2, we evaluate the impact of parameter configurations on the detection results rather than evaluating the performance. As such, the evaluation datasets are smaller than that in experiment 3 to save training time and accelerate the multiple control group experiments.

The basic evaluation metrics are shown in Table 5, which is referred to as the confusion matrix. The true negative rate (TNR), true positive rate (TPR, Recall), negative predictive value (NPV) and positive predictive value (PPV, Precision) are calculated according to equations (8) to (11).

$$\text{TNR} = \frac{\sum \text{TNs}}{\sum \text{True Condition Normal}} \tag{8}$$

$$\text{TPR} = \frac{\sum \text{TPs}}{\sum \text{True Condition HT}} \tag{9}$$

$$\text{NPV} = \frac{\sum \text{TNs}}{\sum \text{Predicted Condition Normal}} \tag{10}$$

$$\text{PPV} = \frac{\sum \text{TPs}}{\sum \text{Predicted Condition HT}} \tag{11}$$

In addition, as the FN case is more critical than FP in HT detection, the evaluation metric F2-score is also used to evaluate the trained models. The calculation is presented in equation (12).

$$F_\beta\text{-score} = (1+\beta^2) \cdot \frac{\text{PPV} \cdot \text{TPR}}{\beta^2 \cdot \text{PPV} + \text{TPR}}, \quad \beta = 2 \tag{12}$$

5.3.1 Experiment 1: Different Balancing Methods

In this experiment, HT-infected netlist datasets are randomly picked from the benchmark library. One is a combinational Trojan-infected dataset, the other a sequential Trojan-infected dataset. For each dataset, there are 24 netlist samples with different combinational/sequential Trojans inserted to 8 host circuits (3 samples for each). The detection results are acquired when *logic_level* is configured to 4 and the DL model is LSTM.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 6
Experimental Results with Different Class Balancing Methods (DL model=LSTM, *logic_level*=4, 5 epochs)

HT Type	Balancing Method	After 5 Epochs												
		Validation Group: K_1				K_2				K_3				Average (95% CI)
		TNR	TPR	NPV	PPV	TNR	TPR	NPV	PPV	TNR	TPR	NPV	PPV	F2-score
Comb.	None	0.9999	0.4888	0.9994	0.9006	0.9999	0.6346	0.9997	0.8801	0.9999	0.7616	0.9998	0.9057	0.666 (+/- 0.251)
	Upsampling	0.9997	0.7821	0.9998	0.7163	0.9999	0.6213	0.9996	0.9010	0.9998	0.8296	0.9998	0.7933	0.751 (+/- 0.163)
	Downsampling	0.9820	0.9867	1.0	0.0572	0.9850	0.9496	1.0	0.0565	0.9895	0.9672	1.0	0.0838	0.257 (+/- 0.094)
	Weighted Loss	0.9994	0.7321	0.9997	0.5879	0.9966	0.6277	0.9996	0.1477	0.9988	0.6919	0.9997	0.3613	0.554 (+/- 0.324)
Seq.	None	0.9999	0.967	0.9995	0.9927	0.9996	0.9844	0.9997	0.9832	0.9996	0.9754	0.9994	0.9838	0.978 (+/- 0.012)
	Upsampling	0.9997	0.9778	0.9997	0.9821	0.9994	0.9900	0.9998	0.9701	0.9991	0.9860	0.9997	0.9642	0.982 (+/- 0.007)
	Downsampling	0.9993	0.9802	0.9997	0.9570	0.9989	0.9902	0.9998	0.9508	0.9985	0.9884	0.9997	0.9421	0.979 (+/- 0.007)
	Weighted Loss	0.9979	0.9732	0.9996	0.8743	0.9977	0.9845	0.9997	0.9001	0.9989	0.9818	0.9996	0.9548	0.965 (+/- 0.025)
Seq. (non -scan)	None	0.9998	0.9395	0.9963	0.9974	0.9992	0.9780	0.9981	0.9913	0.9987	0.9839	0.9983	0.9876	0.972 (+/- 0.036)
	Upsampling	0.9998	0.9506	0.9970	0.9968	0.9994	0.9742	0.9977	0.9925	0.9987	0.9863	0.9986	0.9870	0.975 (+/- 0.028)
	Downsampling	0.9994	0.9555	0.9973	0.9894	0.9976	0.9825	0.9985	0.9726	0.9977	0.9864	0.9986	0.9776	0.976 (+/- 0.024)
	Weighted Loss	0.9996	0.9458	0.9967	0.9938	0.9981	0.9856	0.9987	0.9790	0.9976	0.9874	0.9987	0.9772	0.975 (+/- 0.035)

In order to provide a fair evaluation for each balancing method, 3-fold cross-validation is used. Before training and validation, the netlist dataset is divided into 3 groups: K_1, K_2 and K_3. For each evaluation, 1 group (8 samples) is retained for validation and the other 2 groups (16 samples) are utilized for training. This cross-validation process is repeated 3 times to make sure each of the 3 groups is used exactly once as the validation set. Finally the F2-scores of the 3 groups are averaged to evaluate the configured model. (The list of chosen samples and their grouping are shown in Appendix, Table 12.)

Table 6 shows the detection results of the HT detection model, which is trained with different class balancing methods after 5 epochs, acquired from 3-fold cross-validation. From the average F2-scores with 95% confidence interval (CI), it can be found that the detection model with the upsampling method achieves the best performance for both combinational and sequential Trojan types. Using downsampling and weighted loss can not improve the performance when compared with the model trained without class balancing.

In particular, Table 6 shows that the HT detection performance for the sequential Trojan dataset is much higher than that for combinational Trojans. According to our investigation into the benchmark netlists, we found that the flip-flops (FFs) used in the sequential Trojans are all general FF cells while all the FFs used in the normal circuit part are scan-enabled. In order to eliminate the possibility that the scan-FFs affect the detection results, we replaced all the scan-FFs with corresponding general FF cells for the netlists in the sequential Trojan dataset and used it as a control group (seq.(non-scan)). From the table, it can be found that the performance difference between the original sequential Trojan dataset and non-scan dataset is almost negligible, which proves that our trained HT detection model does not overfit for scan FFs.

From Fig. 14(a), it can be seen that all the trained models have achieved a high F2-score after finishing 1 epoch of training. It shows that our HT detection model can be well trained for sequential Trojan detection within 1 epoch. On the other hand, as shown in Fig. 14(b), after 2 training epochs, the F2-scores of the model trained with the upsampling and weighted loss methods start to fluctuate, which means the convergence of the model with these two methods needs at least 2 epochs of training for combinational Trojan detection. From Fig. 14(b), it is also evident that the score of the model with downsampling keeps raising during all 5 epochs and model does not converge within this time frame. The reduction in the amount of training feature traces caused by the downsampling has a great impact on the convergence speed of the HT detection model. Overall, from the validation results, upsampling is selected as the class balancing method for subsequent experiments.

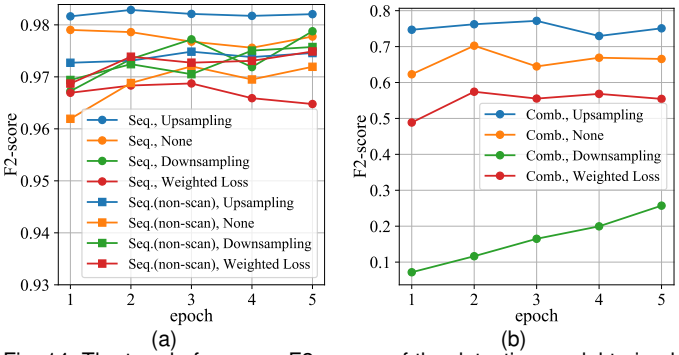

Fig. 14. The trend of average F2-scores of the detection model trained with different class balancing methods on the Trojan-infected datasets in 5 training epochs. (a) Sequential Trojans. (b) Combinational Trojans.

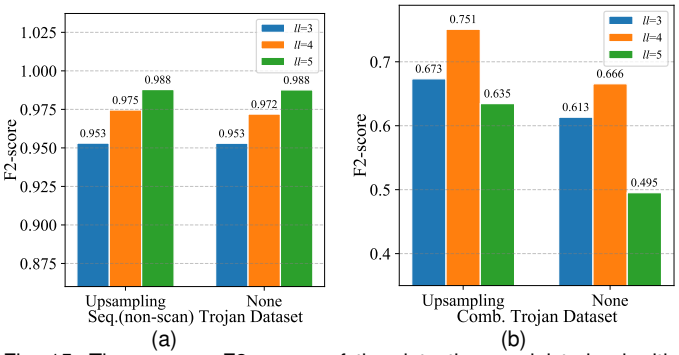

Fig. 15. The average F2-scores of the detection model trained with feature traces extracted under different *logic_level*s on the sequential/combinational Trojan-infected datasets after 5 training epochs. (a) Sequential Trojans. (b) Combinational Trojans.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 7
Experimental Results with Different *logic_level* Values (DL model=LSTM, 5 epochs)

HT Type	Balancing Method	*ll*	After 5 Epochs												
			Validation Group: K_1				K_2				K_3				Average (95% CI)
			TNR	TPR	NPV	PPV	TNR	TPR	NPV	PPV	TNR	TPR	NPV	PPV	F2-score
Comb.	Upsampling	3	0.9998	0.5908	0.9995	0.8150	0.9999	0.6202	0.9996	0.8267	0.9998	0.7256	0.9997	0.8152	0.673 (+/- 0.123)
		4	0.9997	0.7821	0.9998	0.7163	0.9999	0.6213	0.9996	0.9010	0.9998	0.8296	0.9998	0.7933	0.751 (+/- 0.163)
		5	0.9992	0.6679	0.9997	0.4526	0.9988	0.6775	0.9997	0.3833	0.9990	0.8294	0.9998	0.4435	0.635 (+/- 0.127)
	None	3	0.9999	0.4874	0.9993	0.8802	0.9998	0.5594	0.9996	0.7284	0.9999	0.6825	0.9997	0.9076	0.613 (+/- 0.190)
		4	0.9999	0.4888	0.9994	0.9006	0.9999	0.6346	0.9997	0.8801	0.9999	0.7616	0.9998	0.9057	0.666 (+/- 0.251)
		5	0.9998	0.5128	0.9995	0.7492	0.9999	0.2688	0.9992	0.8151	0.9999	0.5868	0.9996	0.8727	0.495 (+/- 0.332)
Seq. (non -scan)	Upsampling	3	0.9995	0.9322	0.9978	0.9838	0.9989	0.9462	0.9976	0.9751	0.9983	0.9654	0.9983	0.9650	0.953 (+/- 0.023)
		4	0.9998	0.9506	0.9970	0.9968	0.9994	0.9742	0.9977	0.9925	0.9987	0.9863	0.9986	0.9870	0.975 (+/- 0.028)
		5	0.9995	0.9787	0.9975	0.9957	0.9987	0.9855	0.9975	0.9928	0.9980	0.9954	0.9990	0.9911	0.988 (+/- 0.013)
	None	3	0.9996	0.9411	0.9981	0.9874	0.9996	0.9422	0.9974	0.9909	0.9993	0.9506	0.9976	0.9857	0.953 (+/- 0.008)
		4	0.9998	0.9395	0.9963	0.9974	0.9992	0.9780	0.9981	0.9913	0.9987	0.9839	0.9983	0.9876	0.972 (+/- 0.038)
		5	0.9997	0.9732	0.9969	0.9971	0.9994	0.9897	0.9982	0.9964	0.9988	0.9937	0.9986	0.9947	0.988 (+/- 0.017)

5.3.2 Experiment 2: Choice of logic_level Parameter

The netlist datasets used in Experiment 2 are the same as in Experiment 1, but from Experiment 1, the class balancing method is fixed to upsampling, and the non-scan version of the sequential Trojan dataset is used. The *logic_level* (*ll*) parameter is configured to 3, 4 and 5 in this experiment. The detection results of *ll*=4 comes from Experiment 1.

Fig. 15 shows the average F2-scores of the detection model trained with PCP feature traces extracted under different *ll* values. In Fig. 15(a), when evaluating the sequential Trojan-infected netlists, the F2-score increases as the value of *ll* increases, but the improvement is small compared with the baseline performance achieved by the trained detection model (F2-score=0.953, when *ll*=3, no class sampling method).

According to the detailed results in Table 7, this improvement is mainly due to the growth of the TPR, which means for sequential Trojan detection increasing *ll* can ensure more HT feature traces are classified correctly. But we do not push *ll* to a higher value due to the increase in the number of the extracted feature traces and training time (Table 9).

However, for combinational Trojan detection, the situation changes. As shown in Fig. 15(b), when *ll*=4, the detection model achieves the best performance, while when *ll*=5, there is a large performance drop. According to Table 7, the decrease in the TPR and PPV causes the large performance loss when *ll*=5. It means that when comparing with condition *ll*=4 (length of PCP feature trace = 7 PCP), HT feature traces under condition *ll*=5 (length=9 PCP) are more likely to be misclassified as normal.

This phenomenon is due to the fact that the structural features in combinational Trojans are more similar to normal circuits than sequential Trojans. In addition, a longer Trojan feature trace may include more information (that is irrelevant to the Trojan) related to the normal parts of the circuit, so that the DL model can not correctly specify the weights on the different features in the feature trace for classification during the training. The DL model learned more, but less accurate, Trojan features from the feature traces of length 9 than those from length 7, thus leading to misclassification.

Considering the overall performance, *ll*=4 will be configured as the default value for feature traces extraction for both combinational and sequential Trojan detection.

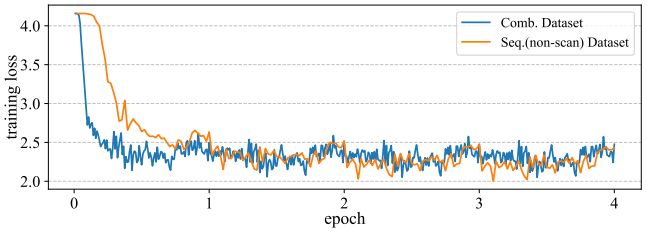

Fig. 16. The training loss of the Net2Vec model when generating each dataset's PCP word embedding matrix.

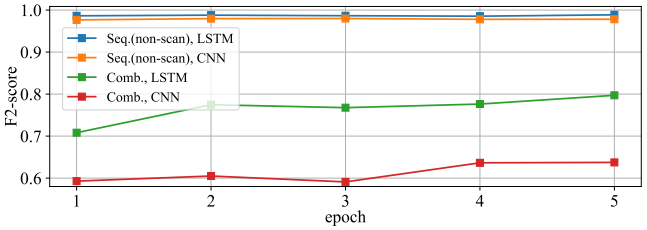

Fig. 17. The F2-scores of the trained LSTM/CNN-based HT detection model on each epoch.

5.3.3 Experiment 3: Overall System Performance

Based on these previous experiments, the influence of the balancing methods on model training and the impact of *logic_level*, which is the only user-defined parameter in the system, on detection results are investigated. Experiment 3 focuses solely on performance evaluation.

In this experiment, the performance of the overall HT detection system is evaluated. In order to provide a fair performance evaluation for the system, two common deep learning models (LSTM and CNN) are utilized in the test. Their structures are not modified. The parameters are only modified once before the test to fit the size of the feature data, then fixed during the test without further optimisation.

Two training and testing datasets are prepared, one is a combinational Trojan-infected netlist dataset, the other a sequential Trojan-infected netlist dataset. For each training and testing dataset, 80 netlist samples are randomly picked from the benchmark library with no overlapping, of which 50% are used as the training set and 50% are used as the test set. (The list of chosen samples and their grouping are shown in Appendix, Table 13.) According to the results in experiment 1 and 2, *logic_level* is configured to 4 for feature trace extraction, and upsampling is adopted to balance the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 8
HT Detection Performance with Different DL Models on Test Sets (*logic_level*=4, upsampling, 5 epochs)

HT Type	DL Model	Epoch	Num. of Feature Traces				TNR	TPR	NPV	PPV	F2-score	HT Type	DL Model	Epoch	Num. of Feature Traces				TNR	TPR	NPV	PPV	F2-score
			TN	FN	TP	FP									TN	FN	TP	FP					
Comb.	LSTM	1	18307889	5872	14483	6355	0.9997	0.7115	0.9997	0.6950	0.708	Seq.(non-scan)	LSTM	1	7514993	7638	541188	6966	0.9991	0.9861	0.9990	0.9873	0.986
		2	18310537	4746	15609	3707	0.9998	0.7668	0.9997	0.8081	0.775			2	7514658	6465	542361	7301	0.9990	0.9882	0.9991	0.9868	0.988
		3	18303430	3626	16729	10814	0.9994	0.8219	0.9998	0.6074	0.768			3	7515796	7689	541137	6163	0.9992	0.9860	0.9990	0.9887	0.987
		4	18309289	4480	15875	4955	0.9997	0.7799	0.9998	0.7621	0.776			4	7517216	8791	540035	4743	0.9994	0.9840	0.9988	0.9913	0.985
		5	18305138	3185	17170	9106	0.9995	0.8435	0.9998	0.6534	0.797			5	7515539	6169	542657	6420	0.9991	0.9888	0.9992	0.9883	0.989
	CNN	1	18275361	4172	16183	38883	0.9979	0.7950	0.9998	0.2939	0.593		CNN	1	7509821	13030	535796	12138	0.9984	0.9763	0.9983	0.9778	0.977
		2	18276665	3975	16380	37579	0.9979	0.8047	0.9998	0.3036	0.605			2	7508386	10596	538230	13573	0.9982	0.9807	0.9986	0.9754	0.980
		3	18275349	4227	16128	38895	0.9979	0.7923	0.9998	0.2931	0.591			3	7507471	10129	538697	14488	0.9981	0.9815	0.9987	0.9738	0.980
		4	18281621	3722	16633	32623	0.9982	0.8171	0.9998	0.3377	0.636			4	7505939	11131	537695	16030	0.9979	0.9797	0.9985	0.9711	0.978
		5	18286957	4473	15882	29287	0.9985	0.7803	0.9998	0.3680	0.637			5	7510272	12076	536750	11687	0.9984	0.9780	0.9984	0.9787	0.978

TABLE 9
The Number of Extracted Feature Traces and the Time for Training with Different *logic_level* Values (training set=K_2+K_3, 16 samples, 1 epoch)

ll	HT Type	Num. of Traces	Training Time	
			Upsampling	None
3	Comb.	2157804	18min	9min
	Seq. (non-scan)	1271965	11min	5min
4	Comb.	7332420	1h12min	36min
	Seq. (non-scan)	3330315	29min	16min
5	Comb.	25164366	3h35min	2h8min
	Seq. (non-scan)	9405796	1h18min	41min

feature traces in each class.

To train the PCP word embedding matrix, the training loss is shown in Fig. 16. It shows that the loss is stabilized at around 2.2 after 2 training epochs for both combinational and sequential Trojan-infected datasets, which also means the fake task generates stable embedding matrices after 2 epochs.

Table 8 presents the HT detection results when applying the trained DL-based HT detection models on test sets after each training epoch. Meanwhile, Fig. 17 shows the trend of F2-score for each model with increasing epochs. Both LSTM and CNN perform well in detecting sequential Trojans in test set. As sequential Trojans can be easily detected nearly in all experiments (different class balancing methods, different *logic_level* and different DL models), we can infer that the sequential Trojans in the benchmark library retain obvious structural features.

However, the detection of combinational Trojans is more challenging. After 5 training epochs, the TPR of the LSTM model applied to the combinational Trojan test set reaches 84.35% and the PPV is 65.34%. Meanwhile, the LSTM model performs 20% better than CNN after the first training epoch and 25% better after 5 training epochs.

Furthermore, as the results of LSTM outperform CNN for both the sequential and combinational Trojan test sets, we can infer that LSTM model is more suitable for processing the proposed sentence-like PCP feature traces in our HT detection system.

Finally, the Trojan/normal component voter (Section 4.7) performs voting on the class type of the components (in the range of the netlist block) according to the classification results of the feature traces. Table 10 presents the voting results based on results classified by the LSTM-based detection model. This table provides the detection results for each netlist sample in two test sets. In these samples, although some Trojan components are not detected (FN) and some normal components are misclassified into the HT class (FP), the number of detected Trojan components (TP) is typically much higher than the FN number and no Trojan sample evades detection (TP=0). The statistical results of all the components show that, after voting, the precision (PPV) of the detection increased from 65.34% to 87.75% for combinational Trojans and from 98.83% to 98.92% for sequential Trojans, but the price is a small TPR loss in both test sets (5.06% for combinational Trojans and 5.42% for sequential Trojans).

In conclusion, according to the final results in Table 10, the proposed HT detection system achieves 79.29% TPR, 99.97% TNR, 87.75% PPV and 99.94% NPV for combinational Trojan detection and 93.46% TPR, 99.99% TNR, 98.92% PPV and 99.92% NPV for sequential Trojan detection (*logic_level*=4, upsampling, LSTM, 5 epochs). It means the detection accuracy (TPR) for combinational Trojans is 79.29%, with 87.75% precision (PPV) and only 0.03% (TNR=1-TNR) Trojan components are undetected. The performance of sequential Trojan detection is even better, with 93.46% accuracy, 98.92% precision and the misclassification of Trojan components is only 0.01%. The system achieves high accuracy, high precision and a very low Trojan component misclassification rate for both combinational and sequential Trojan detection.

6 Discussion

6.1 Comparison

The performance of nearly all ML, NN and DL algorithms is positively related to the quality and size of the data set. Our survey of the publicly available HT netlist benchmark is shown in Table 3.

According to the number of Trojan variants, the types of host circuits and the number of samples, the LEDA library-based Trojan-infected benchmark [13] published in 2018 is selected as the source for the training and testing datasets in our experiments.

However, all previous research on gate-level HT detection, including ML-based and DL-based, utilizes the SAED library-based benchmark for experiments, in which the number of Trojan variants and the number of host circuits is too small to provide a comprehensive experimental result and evaluation. In particular, for DL-based approaches, the SAED benchmark can not provide sufficient samples to train a trustworthy detection model. Therefore, it is not possible

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3116484, IEEE Transactions on Emerging Topics in Computing

TABLE 10
HT Detection Results on Each Sample in Test Sets after Voting in Netlist Block (*logic_level*=4, upsampling, LSTM, at epoch 5)

Combinational Trojan-infected Dataset										Sequential (non-scan) Trojan-infected Dataset									
Netlist	Num. of Components				Netlist	Num. of Components				Netlist	Num. of Components				Netlist	Num. of Components			
	TN	FN	TP	FP		TN	FN	TP	FP		TN	FN	TP	FP		TN	FN	TP	FP
c2670_T093	776	4	5	0	s15850_T003	2984	4	3	1	s1423_T408	480	4	53	0	s15850_T417	2985	2	22	0
s15850_T012	2985	3	5	0	c6288_T041	2416	0	9	0	s15850_T439	2985	0	35	0	s13207_T462	2309	4	57	1
c2670_T016	775	1	6	1	c6288_T066	2416	0	5	0	s15850_T450	2985	3	30	0	s35932_T414	6839	7	76	0
c2670_T073	769	1	7	7	s1423_T008	480	3	4	0	s1423_T405	480	6	101	0	s13207_T440	2310	1	20	0
c2670_T054	776	0	6	0	s1423_T003	480	1	6	0	s1423_T429	479	5	84	1	s35932_T402	6836	5	68	3
c2670_T095	775	0	6	1	s15850_T009	2984	4	4	1	s1423_T418	478	5	61	2	s13207_T449	2310	0	18	0
c3540_T087	1134	4	6	0	s1423_T011	480	1	5	0	s1423_T412	480	1	41	0	s35932_T421	6836	0	32	3
c3540_T005	1133	0	9	1	s1423_T005	480	1	4	0	s15850_T468	2984	2	18	1	s13207_T484	2310	4	8	0
c3540_T015	1133	1	7	1	s1423_T014	480	0	5	0	s1423_T407	480	1	16	0	s35932_T413	6839	2	60	0
c3540_T012	1129	0	5	5	s13207_T002	2309	1	4	1	s1423_T411	480	1	19	0	s13207_T444	2310	1	16	0
c3540_T017	1133	3	6	1	s35932_T015	6838	4	4	1	s1423_T421	480	5	19	0	s35932_T408	6839	8	75	0
c5315_T004	2307	1	7	0	s13207_T013	2310	5	6	0	s1423_T422	480	1	19	0	s13207_T473	2310	2	10	0
c5315_T047	2306	0	8	1	s35932_T006	6838	2	5	1	s1423_T413	480	0	18	0	s15850_T406	2985	9	40	0
c5315_T064	2306	0	6	1	s13207_T014	2310	0	6	0	s15850_T434	2985	2	8	0	s35932_T430	6839	0	21	0
c5315_T057	2306	0	6	1	s35932_T005	6838	3	4	1	s13207_T425	2310	0	41	0	s35932_T435	6839	1	22	0
s15850_T014	2984	3	1	1	s13207_T005	2310	0	7	0	s13207_T468	2310	1	22	0	s15850_T429	2984	9	92	1
c5315_T063	2306	0	8	1	s35932_T018	6838	5	4	1	s15850_T475	2984	2	21	1	s35932_T427	6839	0	22	0
c6288_T049	2415	0	6	1	s13207_T011	2310	2	4	0	s13207_T461	2310	0	21	0	s15850_T443	2983	0	33	2
c6288_T048	2416	0	6	0	s35932_T016	6838	1	5	1	s15850_T433	2985	1	20	0	s35932_T411	6839	1	21	0
c6288_T082	2416	0	5	0	s15850_T002	2985	0	7	0	s13207_T450	2309	7	93	1	s35932_T434	6839	0	18	0
Final	TNR=0.9997, TPR=0.7929, NPV=0.9994, PPV=0.8775									Final	TNR=0.9999, TPR=0.9346, NPV=0.9992, PPV=0.9892								

to provide a meaningful comparison on detection results with other research based on the SAED library.

For future referencing and comparison against our work, in section 5, experiment 3, the lists of utilized samples (Appendix, Table 13), detailed performance data and the detection results for each netlist in the test sets are provided.

6.2 Multi-type Detection

Currently, the DL model in the proposed system is trained to detect combinational Trojans and sequential Trojans separately in order to simplify the experiments and show the performance difference. But multi-type HT detection for a mixed Trojan data set is realizable based on this system. A Softmax function that can output multi-class prediction scores is employed as the DL model's output layer. If different labels are assigned to combinational Trojan feature traces and sequential Trojan features traces, and then mixed into a single training set, a multi-type HT detection model can be trained. However performance optimization is much more complex and therefore, will be considered in further work.

7 Conclusion

In this paper, we propose a data-driven HT detection system for gate-level netlists using novel netlist information extraction technology. The proposed block-based netlist information extraction technology involves a netlist data preprocessing algorithm can automatically extract sentence-like netlist path information (PCP feature traces) from netlist topology for limited depth (*logic_level*). Next, an NLP model (Net2vec) is design to encode the netlist information traces by training an embedding matrix (PCP word embedding matrix). After that, the DL-based learning models are trained to classify the encoded feature traces (E-PCP feature trace) into either a HT class or normal class. Finally, a voting algorithm in the range of the netlist block is used to classify if a netlist component belongs to a HT circuit or not. The experimental results show that the proposed HT detection system can achieve 79.29% TPR, 99.97% TNR, 87.75% PPV and 99.94% NPV for combinational Trojans and 93.46% TPR, 99.99% TNR, 98.92% PPV and 99.92% NPV for sequential Trojans with voting-based optimization using the LEDA library-based HT benchmark (*logic_level*=4, upsampling, LSTM, 5 epochs).

References

[1] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware Trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, Mar 2017.

[2] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware Trojans: A survey from the attacker's perspective," *IET Computers Digital Techniques*, vol. 14, no. 6, pp. 231–246, 2020.

[3] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *IEEE Symp. on Security and Privacy*, May 2007, pp. 296–310.

[4] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists based on machine learning," in *Proc. IEEE 22nd Int. Symp. On-Line Testing and Robust System Design (IOLTS)*, Jul. 2016, pp. 203–206.

[5] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[6] H. Salmani, "COTD: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb. 2017.

[7] C. H. Kok, C. Y. Ooi, M. Moghbel, N. Ismail, H. S. Choo, and M. Inoue, "Classification of Trojan nets based on SCOAP values using supervised learning," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[8] X. Xie, Y. Sun, H. Chen, and Y. Ding, "Hardware Trojans classification based on controllability and observability in gate-level netlist," *IEICE Electronics Express*, vol. 14, no. 18, pp. 1–12, 2017.

[9] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks," in *Proc. IEEE 23rd Int. Symp. On-Line Testing and Robust System Design (IOLTS)*, Jul. 2017, pp. 227–232.

[10] R. Vishnupriya and M. Nirmala Devi, "Hardware Trojan detection using deep learning-deep stacked auto encoder," in *Proc. Int. Conf. Recent Trends in Machine Learning, IoT, Smart Cities and Applications*, Oct. 2020, pp. 345–353.
[11] R. Lu, H. Shen, Y. Su, H. Li, and X. Li, "GramsDet: Hardware Trojan detection based on recurrent neural network," in *Proc. IEEE 28th Asian Test Symp. (ATS)*, Dec. 2019, pp. 111–1115.
[12] H. Salmani and M. Tehranipoor, "Trust-Hub," accessed on 2020-10-02. [Online]. Available: https://trust-hub.org/home
[13] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable Trojan insertion framework for dynamic trust benchmarks," in *Proc. IEEE Design, Automation Test in Europe Conf. Exhib. (DATE)*, Mar. 2018, pp. 1598–1603.
[14] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans," in *Proc. Int. Conf. Cryptographic Hardware and Embedded Syst. (CHES)*, Aug. 2013, pp. 197–214.
[15] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan-free trusted ICs: Problem analysis and detection scheme," in *Proc. IEEE Design, Automation Test in Europe Conf. Exhib. (DATE)*, Mar. 2008, pp. 1362–1365.
[16] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojan detection," in *Proc. Int. Conf. Cryptographic Hardware and Embedded Syst. (CHES)*, Sep. 2009, pp. 396–410.
[17] 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 *Proc. IEEE Symp. Security and Privacy*, May 2010, pp. 159–172.
[18] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "Veritrust: Verification for hardware trust," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 7, pp. 1148–1161, 2015.
[19] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, Nov. 2013, pp. 697–708.
[20] S. Yu, C. Gu, W. Liu, and M. O'Neill, "A novel feature extraction strategy for Hardware trojan detection," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, Oct. 2020, pp. 1–5.
[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Information Processing Systems - Volume 2 (NIPS)*, Dec. 2013, pp. 3111–3119.
[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. on Learning Representations (ICLR)*, May 2013. [Online]. Available: http://arxiv.org/abs/1301.3781
[23] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. 13th Annu. Conf. Int. Speech Communication Association (INTERSPEECH)*, Sep. 2012, pp. 194–197.
[24] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. and Technol. (ICET)*, Aug. 2017, pp. 1–6.
[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

Shichao Yu (GS'19) received the B.Sc. degree in electrical and information engineering from Hangzhou Normal University, Hangzhou, China, in 2014, and the M.Sc. degree in electronic circuit and system from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2017. He is currently pursuing the Ph.D. degree in electrical and electronic engineering with Queen's University Belfast, Belfast, U.K. He is a research student in the Center for Secure Information Technologies, Institute of Electronics Communications and Information Technologies. His research interests mainly include hardware Trojan detection, secure hardware architecture, and the application of machine learning/deep learning in hardware security.

Chongyan Gu (S'14–M'16) received the Ph.D. degree from Queen's University Belfast, Belfast, U.K., in 2016. She is currently a Lecturer (Assistant Professor) in the School of EEECS at Queen's University Belfast, and a member of Center for Secure Information Technologies (CSIT) with in the Institute of Electronics Communications and Information Technologies (ECIT). Dr. Gu is an expert in hardware security. Her research into physical unclonable function (PUF) has been utilised as part of a security architecture for electronic vehicle (EV) charging systems, licensed by LG-CNS, South Korea, and was also licensed for evaluation by Thales, U.K.. Her team was the overall winner of INVENT 2015, a competition to accelerate the commercialisation of innovative ideas. She has successfully organised two special session conferences (IEEE APCCAS in 2018 and ACM GLSVLSI in 2020). She was invited to give tutorial/talks to international conferences, such as, IEEE ASP-DAC 2020 on the topic of practical PUF design on FPGA. Her current research interests include PUFs, security in/for approximate computing, true random number generator (TRNGs), hardware Trojan detection and machine learning attacks.

Weiqiang Liu (M'12-SM'15) received the B.Sc. degree in Information Engineering from Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China and the Ph.D. degree in Electronic Engineering from the Queen's University Belfast (QUB), Belfast, UK, in 2006 and 2012, respectively. In Dec. 2013, he joined the College of Electronic and Information Engineering, NUAA, where he is currently a Professor and the Vice Dean of the college. He has published one research book by Artech House and over 100 leading journal and conference papers. His paper was selected as the Highlight Paper of IEEE TCAS-I in the 2021 January Issue and the Feature Paper of IEEE TC in the 2017 December issue. He has been awarded the prestigious Excellent Young Scholar Award by NSFC in 2020. He serves as the Associate Editors for IEEE TCAS-I (2020.1-2021.12), IEEE TETC (2019.5-2021.4) and IEEE TC (2015.5-2019.4), an Steering Committee Member of IEEE TVLSI (2021.1-2022.12). He is the program co-chair of IEEE ARITH 2020, and technical program committee members for a number of IEEE conferences. His research interests include approximate computing, hardware security and VLSI design for digital signal processing and cryptography.

Máire O'Neill (M'03-SM'11) is Regius Professor of Electronics and Computer Engineering at Queen's University Belfast. She is Director of the Institute of Electronics Communications and Information Technologies (ECIT) and the Centre for Secure Information Technologies (CSIT) at Queen's. She is also Director of the £5M EPSRC/NCSC-funded Research Institute in Secure Hardware and Embedded Systems (RISE: www.ukrise.org) and recently led the €3.8M EU H2020 SAFEcrypto (Secure architectures for Future Emerging Cryptography: www.safecrypto.eu) project (2014-2018). She previously held a UK EPSRC Leadership Fellowship (2008-2014) and was a former holder of a UK Royal Academy of Engineering research fellowship (2003-2008). She has received numerous awards, which include a Blavatnik Engineering and Physical Sciences medal, 2019, a Royal Academy of Engineering Silver Medal, 2014 and British Female Inventor of the Year 2007. She has authored two research books, and over 170 peer reviewed international conference/journal publications. She has been an Associate Editor for IEEE TC and IEEE TETC and is Chair-Elect of the IEEE Circuits and Systems for Communications Technical committee. She is a member of the UK AI Council. She is a Fellow of the Royal Academy of Engineering, a member of the Royal Irish Academy and a Fellow of the Irish Academy of Engineering. Her research interests include hardware cryptographic architectures, lightweight cryptography, side channel analysis, physical unclonable functions, and post-quantum cryptography.