Expert Systems With Applications 271 (2025) 126666

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Review

NtNDet: Hardware Trojan detection based on pre-trained language models

Shijie Kuang [a], Zhe Quan [a], Guoqi Xie [a], Xiaomin Cai [b,*], Xiaoqian Chen [c], Keqin Li [d]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[b] School of Computer Science and Technology, Hunan University of Finance and Economics, Changsha, China
[c] Academy of Military Sciences, Beijing, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Gate-level netlists
Hardware Trojan detection
Large language model
Netlist-to-natural-language
Transfer learning

ABSTRACT

Hardware Trojans (HTs) are malicious modifications embedded in Integrated Circuits (ICs) that pose a significant threat to security. The concealment of HTs and the complexity of IC manufacturing make them difficult to detect. An effective solution is identifying HTs at the gate level through machine learning techniques. However, current methods primarily depend on end-to-end training, which fails to fully utilize the advantages of large-scale pre-trained models and transfer learning. Additionally, they do not take advantage of the extensive background knowledge available in massive datasets. This study proposes an HT detection approach based on large-scale pre-trained NLP models. We propose a novel approach named NtNDet, which includes a method called Netlist-to-Natural-Language (NtN) for converting gate-level netlists into a natural language format suitable for Natural Language Processing (NLP) models. We apply the self-attention mechanism of Transformer to model complex dependencies within the netlist. This is the first application of large-scale pre-trained models for gate-level netlists HT detection, promoting the use of pre-trained models in the security field. Experiments on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks demonstrate that our approach outperforms existing HT detection methods. The precision increased by at least 5.27%, The True Positive Rate (TPR) by 3.06%, the True Negative Rate (TNR) by 0.01%, and the F1 score increased by about 3.17%, setting a new state-of-the-art in HT detection.

Contents

* Corresponding author.
E-mail addresses: ksj@hnu.edu.cn (S. Kuang), quanzhe@hnu.edu.cn (Z. Quan), xgqman@hnu.edu.cn (G. Xie), caixiaomin@hufe.edu.cn (X. Cai), chenxiaoqian@nudt.edu.cn (X. Chen), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.eswa.2025.126666
Received 4 December 2024; Received in revised form 18 January 2025; Accepted 22 January 2025
Available online 1 February 2025

1. Introduction

1.1. Background

Hardware Trojans (HTs) are malicious modifications embedded in Integrated Circuits (ICs), posing a significant security threat. These Trojans are designed to disrupt, alter, or secretly monitor device operations. HTs hide in the ICs until triggered, then activate their payload and begin destructive behavior. The miniaturization and complexity of ICs mean that HTs are very small and hidden. The hidden nature of HTs makes them difficult to detect during standard testing and verification processes.

Furthermore, IC production is globally distributed and involves multiple stages across different regions and companies. The complexity of the supply chain makes it difficult to maintain consistent safety oversight and integrity checks. The IC production process includes design, manufacturing, and testing stages, and security vulnerabilities may exist at each stage (Ashok, Turner, Walsworth, Levine, & Chandrakasan, 2022). These vulnerabilities create opportunities for HT implantation, leading to serious consequences such as data theft, system paralysis, and device performance degradation (Bhunia, Hsiao, Banga, & Narasimhan, 2014; Dhavlle, Hassan, Mittapalli, & Dinakarrao, 2021). Therefore, developing practical methods to detect HTs is crucial to protect ICs from hidden threats and maintain their security and operational integrity.

Traditional methods for detecting HTs have primarily relied on functional testing, structural analysis, and side-channel analysis (Narasimhan et al., 2012; Pan & Mishra, 2021). These approaches have been widely used to identify anomalies in ICs by examining their behavior, structure, and side-channel signals. In recent years, deep learning techniques have emerged as a promising avenue for HT detection. Deep learning methods aim to learn intricate patterns and features from data automatically. These methods include models based on Graph Neural Networks (GNNs) (Wu et al., 2020; Zhou et al., 2020) and Natural Language Processing (NLP) (Otter, Medina, & Kalita, 2020; Qiu et al., 2020), which can capture complex relationships within circuit designs and enhance the detection capabilities for HTs.

1.2. Motivation

Despite advancements in detection techniques, the hidden nature of HTs and the complexity of modern ICs continue to make HT detection a significant challenge. Researchers have explored deep learning methods to address this issue, with approaches primarily based on GNNs and NLP.

- GNN-based HT detection methods focus on the topology of the circuit and capture physical connections (Alrahis, Patnaik, Shafique, & Sinanoglu, 2022; Cheng et al., 2023; Chowdhury, Yang, & Nuzzo, 2021; Muralidhar, Zubair, Weidler, Gerdes, & Ramakrishnan, 2021). While effective in modeling structural information, they may have limitations in fully capturing the semantic information of the circuit, such as the functional roles of different components and the contextual relationships between them. These methods often require extensive feature extraction and a detailed understanding of the circuit structure.
- NLP-based HT detection methods are almost all based on architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), or Convolutional Neural Networks (CNNs). These methods rely on end-to-end training, cannot rely on the large amount of contextual information accumulated by pre-trained models, and may have limited generalization capabilities. In addition, these architectures usually ignore the attention mechanism and may not be able to effectively grasp global relationships.
- Transfer learning, which involves using knowledge gained from one task to solve related tasks (Niu, Liu, Wang, & Song, 2020; Zhu, Lin, Jain, & Zhou, 2023), opens up new avenues for HT detection. Both netlists and natural languages involve structured arrangements of elements with specific grammatical rules, hierarchical structures, and reliance on contextual understanding. Inspired by the similarities between netlists and natural language, we propose applying large-scale pre-trained NLP models to improve HT detection.

By leveraging the structural similarity between IC netlist files and natural language texts, we can transfer knowledge from natural language understanding to IC netlists. This approach reduces the dependence on large labeled datasets and enhances the generalization capability of HT detection models. The success of pre-trained language models in NLP has demonstrated the effectiveness of transfer learning (Ravimaram, Sathish, Vatchala, Rawat, TF, et al., 2023; Taneja & Vashishtha, 2022). Powerful models such as Bidirectional Encoder Representations from Transformers (BERT) (Devlin, Chang, Lee, & Toutanova, 2019), Generative Pre-trained Transformer (GPT) (Radford & Narasimhan, 2018), and Text-to-Text Transfer Transformer (T5) (Raffel et al., 2020) are adept at capturing syntactic, semantic, and contextual information from large amounts of textual data. Applying these models to HT detection can improve accuracy and efficiency.

1.3. Contributions

The main contributions of this study are as follows.

- This is the first study to apply transfer learning to gate-level netlists HT detection. We propose an HT detection approach based on large-scale pre-trained NLP models, including BERT, GPT-2 (Radford et al., 2019), and T5.
- We propose a novel HT detection approach named NtNDet, which includes a method called Netlist-to-Natural-Language (NtN) to convert gate-level netlists into a natural language format suitable for NLP models, improving the readability and comprehension of gate-level netlists.

- We use the Transformer's self-attention mechanism to effectively model relationships within gate-level netlists, which can capture local and global information and enhance the understanding of circuit structure.
- We treat HT detection as a logical gate classification problem without the need for golden model comparison. Experiments are conducted on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks (Salmani, Tehranipoor, & Karri, 2013; Trust-Hub, 2024), to demonstrate the generalizability and robustness of our NtNDet approach. The results indicate that our approach achieves state-of-the-art performance, improved the precision by at least 5.27%, the True Positive Rate (TPR) by 3.06%, the True Negative Rate (TNR) by 0.01%, and the F1 score increased by about 3.17%.

1.4. Organization of the paper

The remainder of this paper is structured as follows: Section 2 reviews related work in the field of hardware Trojan detection, focusing on machine learning approaches. Section 3 presents the preliminaries and provides an overview of our proposed method. Section 4 details the NtNDet approach for HT detection, including the Netlist-to-Natural-Language conversion method. Section 5 describes the experimental setup and results, demonstrating the effectiveness of our approach. Section 6 discusses our work's scalability, limitations, and future directions. Finally, Section 7 concludes the paper.

2. Related work

The rapid development of machine learning has spawned many deep network models, including HT detection models (Ketan, Shetty, et al., 2024; Rahimifar, Jahanirad, & Fathi, 2024; Samyukta & Ramesh, 2023; Sankaran, Mohan, & Purushothaman, 2021; Surabhi et al., 2022). Hasegawa et al.'s method marked the first application of machine learning in HT detection (Hasegawa, Oya, Yanagisawa, & Togawa, 2016). Subsequent machine learning-based HT detection methods are mainly based on GNNs or NLP.

2.1. GNN-based methods

GNNs are neural networks designed to process graph-structured data, capturing local and global information within graphs. They have been effectively applied to HT detection due to their ability to model circuit topologies. Yasaei et al. proposed a gold-free reference HT detection method for Register-Transfer Level (RTL) and gate-level netlists (Yasaei, Chen, Yu, & Al Faruque, 2022), leveraging GNNs to learn the behavior of hardware design circuits through data flow graphs. Hassan et al. introduced an IC topology and behavior-aware HT detection approach, extracting different structural features of the underlying IC along with behavioral information for HT detection (Hassan, Meng, Basu, & Dinakarrao, 2023). Yasaei et al. presented GNN4TJ (Yasaei, Yu, & Al Faruque, 2021), which converts Verilog designs into corresponding data flow graphs and uses GNNs to extract features and learn the underlying structure. Hasegawa et al. proposed an HT detection method, R-HTDetector, using adversarial training to improve robustness against modified HTs (Hasegawa, Hidano, Nozawa, Kiyomoto, & Togawa, 2022). Ma et al. proposed a method to transform RTL code into a data flow graph (Ma, Shang, et al., 2024), and node feature extraction and node label marking are carried out during the transformation process. Chen et al. proposed a two-stage graph neural network model GNN4HT for HT multi-function classification (Chen et al., 2025). In the first stage, HT is located and the positioning results are converted into HT information graphs; in the second stage, the multi-functions of HT are classified.

While these GNN-based methods effectively model structural information, they often require extensive feature extraction and detailed circuit knowledge, such as the number of logic gate fans, flip-flops, and primary input levels. Additionally, focusing primarily on the circuit's topology may not fully capture the semantic information or functional roles of different components.

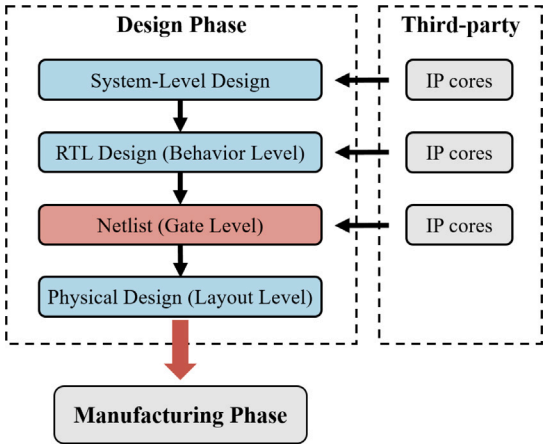

Fig. 1. Overview of the IC design and pre-manufacturing process.

2.2. NLP-based methods

NLP is an interdisciplinary subfield of computer science and linguistics, primarily concerned with enabling computers to process and analyze human language data. In the context of HT detection, NLP techniques have been applied to interpret circuit designs similarly to how language is processed.Shen et al. (2017) first proposed applying NLP techniques to HT detection, using the LMDet model to demonstrate the potential of language models in capturing circuit patterns through n-gram analysis. Lu et al. developed GramsDet (Lu et al., 2019), which uses gate embeddings of order-sensitive matrices to capture generic circuit representations better and reduce parameter overload. Ye et al. introduced SeGa (Ye et al., 2021), a method that extracts semantic information from gate-level netlists using feature vectors for neural network-based Trojan detection. Yu et al. proposed a data-driven HT detection system (Yu, Gu, Liu, & O'Neill, 2022) that uses a hybrid of LSTM and CNN architectures to improve detection accuracy. Ma et al. developed a circuit path sentence extraction algorithm based on signal propagation rules. They proposed PS-TextCNN for HT detection based on text convolutional neural network (Ma, Wang, & Wang, 2024). Dofe et al. proposed a supervised learning approach using RNNs for HT detection within FPGA configuration bitstreams (Dofe et al., 2024).

While these NLP-based methods have shown promise, many rely on end-to-end training approaches without utilizing large-scale pre-trained models, which often require large amounts of labeled data and may limit their generalization capabilities. Consequently, they may not benefit from the extensive contextual knowledge that has advanced NLP tasks in other domains. The related works are summarized in Table 1.

3. Preliminaries and overview

3.1. HT fundamentals and threat model

HTs are a significant security threat in the IC design and development process (Agrawal, Baktir, Karakoyunlu, Rohatgi, & Sunar, 2007; Hu et al., 2021). With the globalization of chip design, malicious attackers insert HTs into third-party modules and Intellectual Properties (IPs). HTs are implanted at critical moments since these IP cores are integrated at various stages, from system-level design to RTL design to gate-level netlists.

Gate-level netlists are particularly vulnerable when the circuit's logical design is finalized, making it a crucial point for potential HTs insertion (Dong et al., 2020; Xue, Gu, Liu, Yu, & O'Neill, 2020). Fig. 1 shows an overview of the IC design and pre-manufacturing process. The gate-level netlists are the basis for describing the hardware structure without chip layout information. These netlists become the critical

Table 1
Summary of Machine Learning-Based HT Detection Methods.

Reference	Key contribution	Datasets	Model	Abstraction level	Evaluation
Yasaei et al. (2022)	A golden reference-free pre-silicon HT detection based GNN	Trust-hub DES RC5	GNN	Gate-level netlist RTL	Precision:91% Recall:84% (netlist) Precision:92% Recall:97% (RTL)
Hassan et al. (2023)	A circuit-topology aware hardware Trojan detection method based GNN	Trust-hub	GNN	Gate-level netlist	Precision:93.5% TPR:91.38% F1-score:91.28%
Yasaei et al. (2021)	A golden reference-free HT detection method based GNN4TJ	Trust-hub	GNN	RTL	Precision:92% Recall:97%
Hasegawa et al. (2022)	A robust HT detection method using adversarial training; R-HTDetector	Trust-hub TRIT-TC	GNN	Gate-level netlist	TPR:83.5% TNR:94.6% (Trust-hub) TPR:94.9% TNR:85.3% (TRIT-TC)
Chen et al. (2025)	A two stage GNN model for HTs' multifunctional classification; GNN4HT	Trust-hub TRTC-IC	GNN	Gate-level netlist RTL	Precision:80.95% TPR:94.28% TNR:97.22% (netlist)
Ma, Shang, et al. (2024)	A variety of GNN classification models are built for detecting RTL HTs	Trust-hub	GNN	RTL	Precision:94.5% F1-score:96.7%
Shen, Tan, Li, Zhang, and Li (2017)	A scheme distinguishing the unnaturalness of HTs from the naturalness of normal circuits; LMDet	Trust-hub OpenCores	n-gram	Gate-level netlist	TPR:96.5% TNR:82.6%
Lu, Shen, Su, Li, and Li (2019)	A approach to detect HT through capturing suspicious circuit connection structure; GramsDet	Trust-hub	n-gram LSTM	Gate-level netlist	TRP:82.1% TNR:96.0%
Ye, Li, Shen, Li, and Li (2021)	A novel circuit gate embedding method; SeGa	Trust-hub	LSTM	Gate-level netlist	TPR:85.1% TNR:93.58%
Ma, Wang, and Wang (2024)	PS-TextCNN for HT detection	Trust-hub	TextCNN	Gate-level netlist	TPR:88.9% TNR:98.5%
Dofe, Danesh, More, and Chaudhari (2024)	A supervised learning approach using RNNs	ISCAS 85	RNN	Xilinx 7-series bitstream	Sequential acc:91% Combinational acc:93.5%

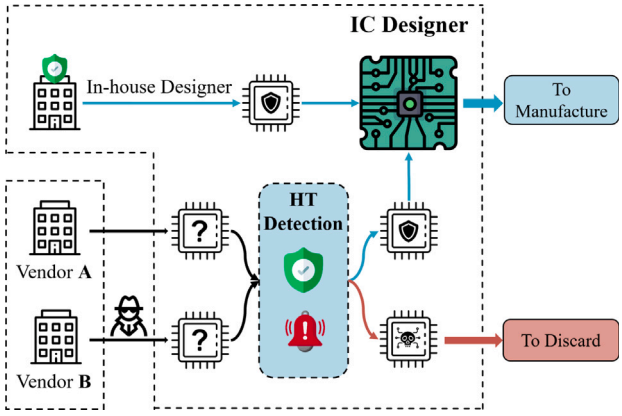

Fig. 2. HT detection in the IC design process.

stage in the design when HT may be inserted. Fig. 2 illustrates the HTs detection in the IC design process. Third-party vendors A and B provide IP cores that must pass HT testing before being integrated into the IC design. It is critical to perform a rigorous review process to ensure safety, and only verified safety IP cores enter manufactured.

This study focuses on detecting HT in gate-level netlists during the IC design phase. Gate-level netlists provide a basic description of the hardware structure and do not include the physical layout of the chip. Conducting a thorough circuit logic analysis can identify and remove hardware Trojans early, enhancing the preliminary design's overall security. The threat model for this research is based on several assumptions.

- Because of the globalization of chip design, attackers can insert HTs into third-party IP cores.
- The attacker aims to disrupt the IC product, cause damage, or leak information through logic rather than using power, electromagnetic, or other side-channel attacks.
- These HTs have a low probability of triggering, and the triggering conditions and payload information are usually hidden, making them challenging to detect during conventional logic testing.

3.2. Transfer learning

In the field of NLP, large datasets and increased computing power have led to the emergence of pre-trained models for various language tasks. After unsupervised pre-training and fine-tuning, these models have demonstrated excellent representation learning and capabilities (Hu et al., 2024). This study explores NLP techniques and transfer learning in HT detection.

NLP models are pre-trained on a large-scale corpus to accumulate universal knowledge. This knowledge is transferred to the specialized domain of gate-level netlists, which represent the complex logic of ICs. Fig. 3 illustrates this approach; model A is trained using big data from a large NLP corpus to acquire universal knowledge. This knowledge is then applied to model B, fine-tuned on a small dataset of gate-level netlists for effective HT detection. By converting circuit structure data into the format of NLP input, we can perform HT detection based on large-scale pre-trained models. This approach reduces the need for large labeled datasets, lowers computational costs, and improves the model's ability to capture complex relationships in ICs.

3.3. Overview of the NtNDet

Recognizing the similarities in structure and expression between gate-level netlists and natural language, we liken them to unique language frameworks for IC design. Our NtN method transforms netlists into natural language structures, making netlists related to natural language.

Our HT detection approach overview is shown in Fig. 4.

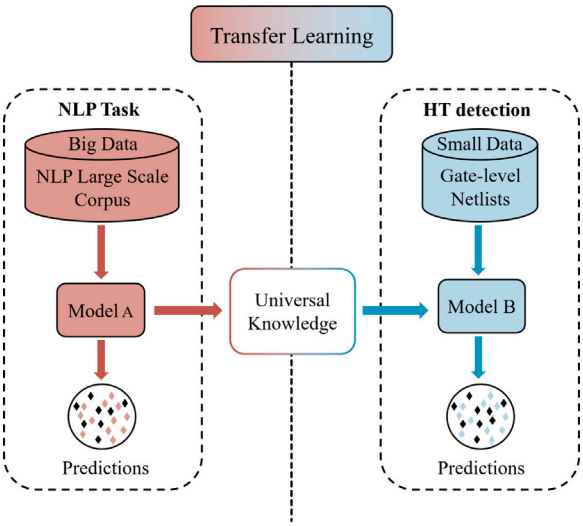

Fig. 3. Transfer learning for HT detection.

(1) Netlist-to-Natural-Language. The process begins by converting the original gate-level netlists into a natural language format using the NtN method. This conversion makes the netlists easily processable by the NLP model.

(2) Netlist encoding. Once the netlists are transformed into natural language format, they are encoded into a suitable representation that can be inputted into a pre-trained NLP model. This encoding step involves netlist embedding and positional encoding.

(3) Netlist Modeling. Transfer learning is applied to adapt the model to the HT detection task by leveraging a large-scale pre-trained NLP model. The pre-trained NLP model is fine-tuned using the encoded netlists to learn specific features related to HT detection.

The modeled netlists pass through the liner and softmax layers to output the prediction results and classify whether the netlists contain HTs. By integrating NLP techniques and transfer learning, our approach effectively leverages the strengths of pre-trained NLP models to address the challenges in HT detection.

3.4. Problem statement

In the process of IC design and manufacturing, the gate-level netlist is an important part of describing the logic structure of the circuit, and it is also the key location for attackers to implant HT. The NtN method converts the gate-level netlists into a natural language format suitable for the NLP model, improving readability and comprehension. We regard the converted gate-level netlist as text data and use a Transformer-based binary classification model to identify whether there is a Trojan in the netlists.

Formally, our dataset is represented as $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i denotes the textual representation of the i-th line in the gate-level netlist, and $y_i \in \{0, 1\}$ is a binary label indicating whether the i-th line belongs to a Trojan circuit ($y_i = 1$) or not ($y_i = 0$). We use a pre-trained Transformer-based model for fine-tuning, defined by the function $f(x_i) = \sigma(W \cdot \text{Transformer}(x_i; \theta_{\text{pre}}) + b)$, where $\text{Transformer}(x_i; \theta_{\text{pre}})$ represents the output from the pre-trained Transformer model for input x_i, with θ_{pre} being its parameters. Here, W and b are the weights and bias of the classification layer, respectively, and σ denotes the Sigmoid activation function, which outputs the probability $\hat{y}_i$ of the i-th line belonging to a Trojan circuit. The model is trained by minimizing the binary cross-entropy loss $\mathcal{L}$, optimizing the parameters $\theta = \{\theta_{\text{pre}}, W, b\}$ to accurately distinguish between lines that are part of Trojan circuits and those that are not. Further mathematical formulations and optimization details are provided in Section 5.

4. The NtNDet approach for HT detection

Our approach involves three main processes: First, we format gate-level netlists using the NtN method. Next, we encode them for input to neural networks. Finally, we apply transfer learning to model the netlists for HT detection.

4.1. Netlist-to-natural-language

We observe that gate-level netlists share some standard features with NLP, such as order, specific grammatical rules, hierarchical structure, and the need for contextual understanding. We additionally note that linguistic pre-training methods offer more than just a structural match to netlists. Large-scale language models inherently excel at capturing contextual dependencies over long sequences. By formatting gate-level netlists as if they were sentences, where a gate (e.g., *MUX21X1*) acts like a "verb", and signals (e.g., *WX11155, WX3442*) act like "nouns". The model can interpret how signals and gates relate to each other across distant parts of the circuit, much like tracking entities in text. This goes beyond the basic grammar or hierarchical rules and leverages the model's ability to learn subtle patterns from extensive training on large corpora. Consequently, when Trojan triggers or payloads are distributed across the netlist, the fine-tuned pre-trained model is better equipped to recognize suspicious dependencies or anomalies.

We employ a three-step process to convert these gate-level netlists into a format compatible with NLP models. For most netlists, the process concludes after the second step; the third step is only necessary for netlists containing certain characters, which account for only a small fraction of HT netlists. The example we provide below illustrates a special case requiring the third step.

(1) Natural Language Formatting. The first step is to define gate types, gate names, inputs, and outputs using a consistent grammar of natural language constructs. We use colon and comma punctuation to separate each attribute and quotes to explicitly define string values, such as gate names and port identifiers. For example, the original netlists "*MUX21X1 Trojan_Payload2 .IN1(WX11155), .IN2(WX3442), .S(Trojan_SE), .Q(WX11155_Tj_Payload);*" is a 2:1 multiplexer that selects one of two inputs based on a selection line and outputs it. These netlists are formatted as follows.

- Gate_type: *MUX21X1*
- Gate_name: *Trojan_Payload2*
- Input_ports: *"IN1:WX11155,IN2:WX3442,S:Trojan_SE"*
- Output_port: *"Q:WX11155_Tj_Payload"*

(2) Gate Name Elimination. In the second step, we need to eliminate the model's reliance on specific gate names, which can lead to overfitting. In the provided netlist example, the gate name *Trojan_Payload2* is removed, allowing the model to focus on gate types and circuit structure characteristics.

- Gate_type: *MUX21X1*
- Input_ports: *"IN1:WX11155, IN2:WX3442, S:Trojan_SE"*
- Output_port: *"Q:WX11155_Tj_Payload"*

For the vast majority of netlists, the process concludes after this step. However, in rare cases, if the netlist contains Trojan-related characters, we apply an additional third step to prevent the model from overfitting.

(3) Trojan-related Characters Treatment. When Trojan-related characters are present, we replace them with generic symbols to ensure the model generalizes well to all HTs. In the example, the Trojan-related characters *Trojan_SE* and *WX11155_Tj_Payload* are replaced, indicated by the symbol "*". In the experimental section, we discuss the methods for replacing Trojan-related characters. In practice, the identifiers do not appear; they are included in the benchmark to improve readability. The final format is as follows.

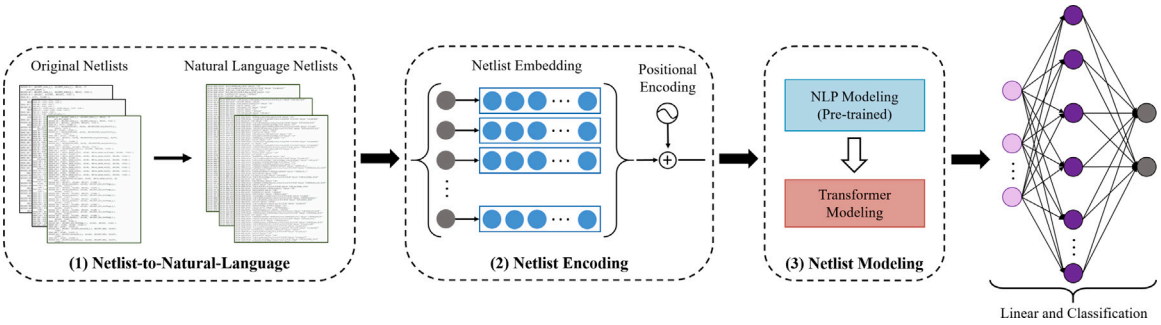

Fig. 4. Overview of the NtNDet.

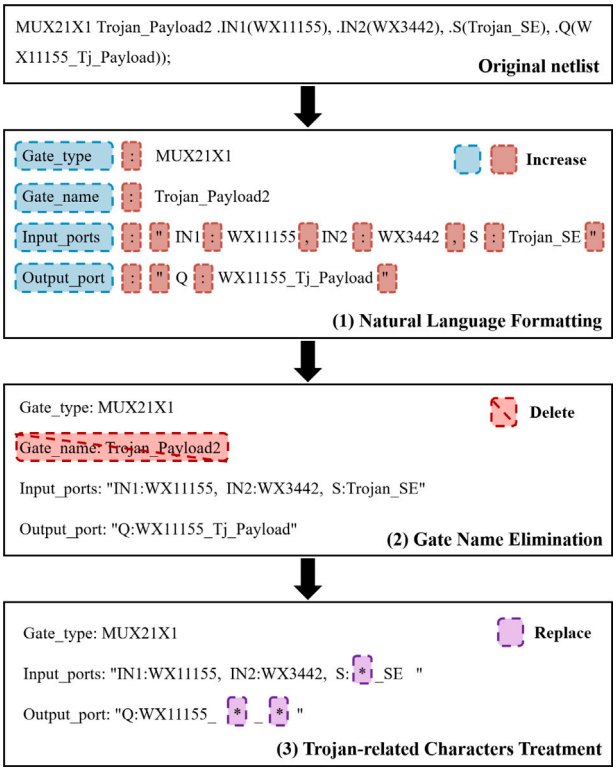

Fig. 5. Netlist-to-Natural-Language.

- Gate_type: *MUX21X1*
- Input_ports: *"IN1:WX11155, IN2:WX3442, S:*_SE"*
- Output_port: *"Q:WX11155_*_*"*

This processing helps avoid bias introduced by specific terms and maintains consistency and reliability across the detection process. The NtN process is shown in Fig. 5. The "Gate_type" in the figure represents the type of the gate, while "Input_port" and "Output_port" represent the connection relationship. This process eliminates the influence of the gate name and Trojan-related characters. The NtN method makes the netlist conform to grammatical conventions and improves interpretability, improving the model's understanding.

The previous example illustrated the transformation of a single gate in the netlist; the same principles can be applied to more complex circuits consisting of multiple gates. Each gate is processed individually, and their interconnections are maintained in the transformed netlist format. The following example demonstrates how the transformation process works for a netlist consisting of multiple gates.

Assume that this is part of a netlist consisting of three simple gates. The original netlist is as follows:

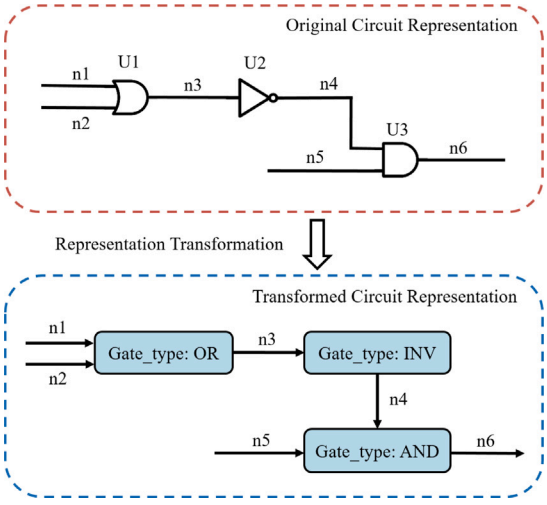

Fig. 6. Netlist representation transformation.

- *OR2X1 U1 (.A(n1), .B(n2), .Y(n3))* ;
- *INVX1 U2 (.A(n3), .Y(n4))* ;
- *AND2X1 U3 (.A(n4), .B(n5), .Y(n6))* ;

After natural language formatting:

- Gate_type: *OR2X1* Gate_name: *U1* Input_ports: "*A: n1, B: n2*" Output_port: "*Y: n3*" ;
- Gate_type: *INVX1* Gate_name: *U2* Input_port: "*A: n3*" Output_port: "*Y: n4*" ;
- Gate_type: *AND2X1* Gate_name: *U3* Input_ports: "*A: n4, B: n5*" Output_port: "*Y: n6*" ;

After gate name elimination:

- Gate_type: *OR2X1* Input_ports: "*A: n1, B: n2*" Output_port: "*Y: n3*" ;
- Gate_type: *INVX1* Input_port: "*A: n3*" Output_port: "*Y: n4*" ;
- Gate_type: *AND2X1* Input_ports: "*A: n4, B: n5*" Output_port: "*Y: n6*" ;

Fig. 6 shows the conversion of the netlist representation. The NtN process transforms the original circuit described by the netlist into a new format. In this format, gates correspond to nodes with "Gate_type" attributes, and edges represent signal connections defined by "Input_port" and "Output_port". This representation preserves meaningful gate types and their relationships, enabling NLP models to directly leverage semantic features for modeling.

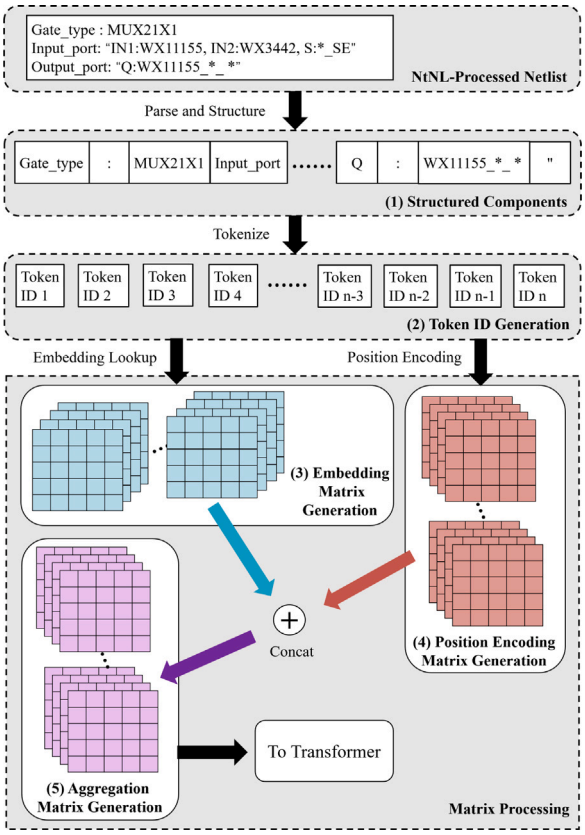

Fig. 7. Netlist encoding.

4.2. Netlist encoding

After the NtN step, the netlist is in the form of natural language, for example: *Gate_type: MUX21X1 Input_ports: "IN1:WX11155, IN2:WX3442, S:*_SE" Output_port:"Q:WX11155_*_*"*. The netlist must be encoded to convert the structured netlist data into a format that the NLP model can understand and process. The gate-level netlist encoding process is as follows.

(1) Structuring Components. This step involves processing the gate-level netlists pre-processed by the NtN method. The process aims to extract and identify basic components such as gate types, input ports, and output ports. This structured organization helps with subsequent processing.

(2) Token ID Generation. After the parsing step, the components are tokenized. Each component is converted into a discrete token and assigned a unique token ID. These token IDs are used as indices in the NLP model training vocabulary.

(3) Embedding matrix generation. Token IDs are converted into embedding vectors representing semantics in a multidimensional vector space. The conversion involves an embedding lookup operation, where each token ID is matched with its unique embedding vector. This process creates an embedding matrix, with each row representing the embedding vector of a token.

(4) Position encoding matrix generation. Positional encoding uses sines and cosines to preserve the position information of the tokens in the sequence and arranges them into a matrix. The positional encoding is calculated by

$$\begin{aligned} PE(i,2j) &= \sin\left(\frac{i}{10000^{2j/d}}\right), \\ PE(i,2j+1) &= \cos\left(\frac{i}{10000^{2j/d}}\right). \end{aligned} \tag{1}$$

PE stands for position encoding, i is the sequence position, d is the total dimension of the embedding, and j is the dimension index.

(5) Aggregation matrix generation. The embedding vector *TE* and the position encoding vector *PE* of each token are aggregated by element-wise addition. This operation forms a new vector that represents the integration of semantics and position. These vectors are then aggregated into a matrix

$$M_{agg} = \begin{bmatrix} TE(TI_1) + PE(TI_1) \\ TE(TI_2) + PE(TI_2) \\ \vdots \\ TE(TI_n) + PE(TI_n) \end{bmatrix}, \tag{2}$$

TI_i represents the token ID of the ith token. M_{agg} is an aggregate matrix containing token embeddings and positional encodings, which is finally pre-processed for input to the NLP model. This process includes adjusting the sequence length by padding or truncation and incorporating special tokens required by the Transformer model.

Fig. 7 illustrates the netlists encoding process. First, the gate-level netlists are converted into a series of tokens. These tokens are encoded as token IDs, and their corresponding embeddings are calculated. Positional encodings are added to enhance the model's understanding of the token order. The embeddings and positional encodings are combined and aggregated into a complete input matrix. This matrix is then processed by the Transformer model for further analysis.

4.3. Netlist modeling

The basis of our approach is to model the gate-level netlists using Transformer. The Transformer architecture, designed for relational data, constructs a hierarchical representation by layering multiple self-attention and feedforward neural networks. This structure aids in extracting both local and global structural information from gate-level netlists. The self-attention mechanism is

$$Attention(\mathbf{Q},\mathbf{K},\mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right)\mathbf{V}. \tag{3}$$

Q (Query), **K** (Key), and **V** (Value) are generated by applying linear transformations to the input sequence. The correlation between input elements is measured by calculating the dot product between the query and key vectors, then normalized into a probability distribution.

The Transformer architecture uses a multi-head attention mechanism that allows the model to compute self-attention representations simultaneously. In the multi-head attention mechanism, the input sequence representation is partitioned into multiple independent subspaces, each represented by a unique set of query, key, and value matrices. The self-attention representation of each subspace is calculated independently by

$$\mathbf{C}^{(i)} = Attention(\mathbf{Q}\mathbf{W}_q^{(i)}, \mathbf{K}\mathbf{W}_k^{(i)}, \mathbf{V}\mathbf{W}_v^{(i)}). \tag{4}$$

$\mathbf{W}_q^{(i)}$, $\mathbf{W}_k^{(i)}$, and $\mathbf{W}_v^{(i)}$ are respectively the weight matrices of query, key and value in the graph aggregate these representations to form multi-head attention **C** calculated by

$$\mathbf{C} = concat(\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \ldots, \mathbf{C}^{(r)}). \tag{5}$$

Multi-head attention aggregation integrates the relationship between different representation spaces and enhances the understanding of the input sequence structure and semantic information.

Fig. 8 illustrates the netlists Transformer modeling process. For example, this circuit contains an AND, OR, and NOT gate.

(1) Circuit-to-Local Attention. The original circuit is divided into subcircuits, each modeled as a local attention processing unit. Local Attention 1, Local Attention 2, and Local Attention 3, each unit being processed independently.

(2) Local Attention Processing. Each local attention unit processes its respective subcircuit, focusing on its local characteristics and capturing relevant features within each subcircuit.

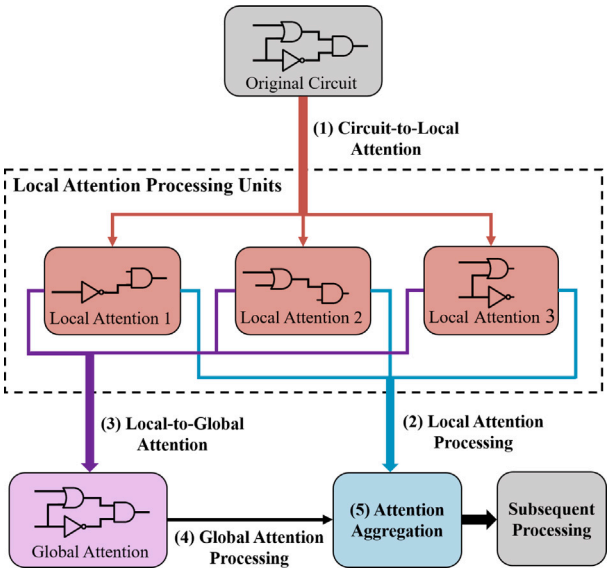

Fig. 8. Netlist modeling.

(3) Local-to-Global Attention. The local attention units are then integrated into global attention. This step ensures that local insights from each subcircuit are combined to understand the entire circuit topology comprehensively.

(4) Global Attention Processing. The global attention unit processes the integrated attention information, combining local and global insights to create a holistic representation of the circuit.

(5) Attention Aggregation. Finally, the local and global attention information is aggregated to form the aggregated attention, and forwarded to subsequent processing.

The gate-level netlists include input and output ports, various logic gate components, and interdependent relationships. The self-attention and feedforward neural network layers capture these dependencies and structural information. This capture ability enables the model to extract and integrate information from multiple levels of the gate-level netlist, enhancing its understanding and processing capabilities.

4.4. Advantages of approach

our approach leverages the advantages of large-scale pre-trained NLP models. These models are trained on extensive and diverse corpora, providing them with rich semantic understanding capabilities. The specific reasons and benefits of this approach are as follows.

- **Rich Semantic Understanding.** Large-scale pre-trained models are trained on a wide variety of textual data, including technical terminologies such as "Gate type", "Input ports", and "Output port". These models already understand what these terms mean in the context of natural language.
- **Handling of Punctuation.** The natural language format uses punctuation to separate and organize information. Pre-trained models can accurately interpret and process these punctuation marks, ensuring precise parsing of netlist information.
- **Potential Knowledge of Circuit-Related.** Pre-trained models may have learned circuit-related knowledge from the diverse technical documents on which they were trained. For example, a term like "MUX21X1" might be recognized as a 2:1 multiplexer. This inherent knowledge provides an advantage when processing circuit netlists.
- **Semantic graph parsing.** This approach can be viewed as semantic graph parsing, where each gate type is considered a node, and connections (input and output ports) are semantically meaningful edges, similar to how nodes and edges are interpreted in a graph. In addition, this approach incorporates semantic information beyond the typical structural representation in GNNs, providing a richer context for the analysis.

In summary, we can improve gate-level netlists parsing and processing by converting it into natural language format and using pre-trained models. This approach enhances the performance and reliability of HT detection.

Table 2
Benchmarks in our experiments.

Benchmarks		
Trust-Hub	TRIT-TC	TRIT-TS
RS232-T1000	c2670-T000	s1423-T400
RS232-T1100	c2670-T001	s1423-T401
RS232-T1200	c2670-T002	s1423-T402
RS232-T1300	c3540-T000	s13207-T400
RS232-T1400	c3540-T001	s13207-T401
RS232-T1500	c3540-T002	s13207-T402
RS232-T1600	c5315-T000	s15850-T400
s35932-T100	c5315-T001	s15850-T401
s35932-T200	c5315-T002	s15850-T402
s35932-T300	s1423-T000	s35932-T400
s38417-T100	s1423-T001	s35932-T401
s38417-T200	s1423-T002	s35932-T402
s38417-T300	s13207-T000	
s38584-T100	s13207-T001	
s38584-T200	s13207-T002	
s38584-T300		
s15850-T100		

5. Experiments

This section covers the experimental setup, analysis of Trojan-related characters processing, performance evaluation on different benchmarks, and robustness and generalization evaluation.

5.1. Experimental setup

(1) HT Detection Benchmarks. We use three benchmarks in the experiments: Trust-Hub, TRIT-TC, and TRIT-TS. Trust-Hub is a platform that provides hardware security and trust benchmarks, covering various design types, including combinational logic, sequential logic, and circuits of different complexity. TRIT-TC contains 580 benchmarks from 8 designs, and TRIT-TS contains 334 benchmarks from 4 designs. We selected some data from these three benchmarks for experiments, as shown in Table 2.

(2) Training Configuration and Evaluation Metrics. We are using four NVIDIA 3090 GPUs for distributed training alongside dual Intel Xeon E5-2666 v3 CPUs. The PyTorch framework was implemented on Ubuntu 20.04 LTS.

We use a batch size of 64 for training and run for 20 epochs. The maximum length of the text input is limited to 256 characters. We employ the CrossEntropyLoss function as the optimization objective and select the AdamW optimizer with a learning rate of 2e-5. These choices align with our use of large-scale pre-trained NLP models such as BERT, GPT2, and T5, which are typically trained using CrossEntropyLoss for their language modeling objectives. Additionally, AdamW is preferred for its stability and efficiency in optimizing transformer-based architectures, ensuring better convergence and generalization performance in our HT detection task. We employ Leave-One-Group-Out cross-validation to ensure that each data subset is used exactly once for validation. In each iteration, one group from the dataset is set aside as the validation set, while the remaining groups are used to train the model. This process is repeated until every group has been used once as the validation set.

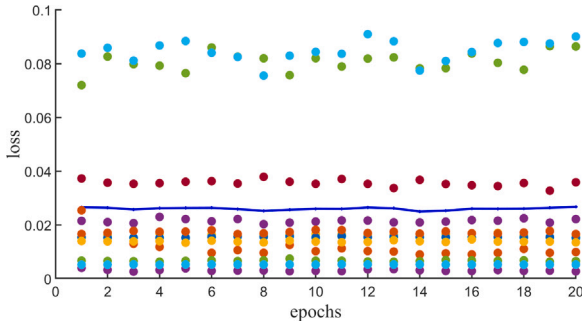

Fig. 9. The loss of GPT-2 per epoch on the TRIT-TS benchmark. The points of different colors represent different folds, and the blue curve represents the average loss.

We use TPR, TNR, and F1 score for experimental evaluation, and the metrics are calculated as follows:

$$\begin{aligned} TPR &= \frac{TP}{TP+FN}, \\ TNR &= \frac{TN}{TN+FP}, \\ F1 &= \frac{2\times P\times R}{P+R}. \end{aligned} \tag{6}$$

TP represents True Positive (correctly identified Trojan samples), *FP* represents False Positive (normal samples are incorrectly classified as Trojans), *TN* represents True Negative (correctly identified normal samples), and *FN* represents False Negative (Trojan samples are incorrectly classified as normal). *TPR* and *TNR* evaluate the model's accuracy in detecting Trojans and normal samples, respectively. *P* (Precision) is defined as *(TP / (TP + FP))*, and *R* (Recall) is defined as *(TP / (TP + FN))*. Precision measures the proportion of correct Trojan predictions, while recall measures the proportion of Trojans detected. The F1 score balances precision and recall to provide a comprehensive measure of model performance.

(3) Class Imbalance and Loss Function. To address the significant imbalance between the number of HT samples and normal samples, we adjusted the CrossEntropyLoss function with dynamically calculated class weights. The class weights are calculated by

$$w_{\text{pos}} = \frac{1}{\sqrt{N_{\text{pos}}}}, \quad w_{\text{neg}} = \frac{1}{\sqrt{N_{\text{neg}}}}. \tag{7}$$

N_{pos} and N_{neg} denote the number of samples in the positive and negative classes, respectively. The initial class weights w_{pos} and w_{neg} are determined by the inverse of the square root of the number of samples in each class. To normalize these weights so that their sum is equal to one, we use the following equations:

$$w'_{\text{pos}} = \frac{w_{\text{pos}}}{w_{\text{pos}}+w_{\text{neg}}}, \quad w'_{\text{neg}} = \frac{w_{\text{neg}}}{w_{\text{pos}}+w_{\text{neg}}}. \tag{8}$$

The normalized class weights w'_{pos} and w'_{neg} are then incorporated into the CrossEntropyLoss function, which is calculated by

$$L(y,\hat{y}) = -\sum_{i=1}^{C}[w'_{\text{pos}} y_i \log(\hat{y}_i) + w'_{\text{neg}}(1-y_i)\log(1-\hat{y}_i)]. \tag{9}$$

y represents the true label, $\hat{y}$ represents the predicted probability, w'_{pos} and w'_{neg} represent the weighting coefficients of the positive and negative classes respectively. This loss function solves the problem of class imbalance.

(4) Model Configuration and Selection. Since Leave-One-Group-Out is used, which iteratively trains on different dataset subsets, we need to select the best model based on a comprehensive performance metric that combines accuracy and loss. The metric for each model is calculated by

$$score_i = \alpha \cdot acc_i - (1-\alpha)\cdot loss_i, \tag{10}$$

acc_i and $loss_i$ represent the accuracy and loss of the ith model, respectively, and α is a balance parameter set at 0.5 to weigh accuracy and loss equally. We normalize these scores across all models to assign weights calculated by

$$w_i = \frac{score_i}{\sum_{j=1}^{n} score_j}, \tag{11}$$

n is the total number of models. The model with the highest normalized score is chosen for further analysis and use in subsequent experiments.

Fig. 9 provides an illustrative example of the epoch loss performance of GPT-2 on the TRIT-TS benchmark. Additionally, we evaluate the performance of BERT, GPT-2, and T5 models on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks to determine the best model for each dataset.

5.2. Character processing analysis

Experiment 1. The netlists in the experimental benchmarks are different from real scenarios. They include Trojan-related characters such as "Payload", "Trojan", "Tg", "Tj", and "Trigger" for easier identification, but these characters do not exist in reality. To resolve this difference, we must eliminate the impact of these characters. Instead of removing these characters, we explored several replacement methods to preserve data structure and context completeness.

We conduct experiments using BERT on the Trust-Hub benchmark to evaluate these character processing strategies. We then extend these experiments to GPT-2 and T5, and extend testing to the TRIT-TC and TRIT-TS benchmarks. We employ "BertForSequenceClassification", a variant of the BERT model designed for sequence classification tasks. It uses BERT's bidirectional encoder representation to capture contextual differences in the training text. We use "BertTokenizer", which combines the WordPiece algorithm and automatically adds special tokens: [CLS] is added to the beginning of each sequence, and [SEP] is added to the end. The model uses the relative representation of [CLS] tags to predict sequence classes. The model is initialized using "bert-base-uncased" and is case-insensitive.

We analyzed five processing methods, from method A to Method E. Table 3 displays the benchmark's performance under different character processing strategies. Below, we analyze each strategy.

- **Method A: using the original data**. Retains all Trojan-related characters. This method achieves high detection rates but risks the model focusing more on identifying these specific characters, rather than understanding the overall data structure and semantic content.
- **Method B: replacing with "unk"**. Replace Trojan-related characters with unknown. This method maintains the structural integrity of the data by preserving the positions of Trojan-related characters. However, it anonymizes its content, which may result in the loss of some contextual information.
- **Method C: replacing with "mask"**. Mask Trojan-related characters to reduce reliance on specific vocabulary while retaining their location context. Like *Method B*, this method treats all Trojan-related characters uniformly, which may limit the model's ability to distinguish them.
- **Method D: replace with "node X"**. Replace Trojan-related characters with "node X", where "X" represents random characters. This method performs well but may lead to overfitting if the character "node" mainly appears in Trojan samples.
- **Method E: Replace with random characters**. Replace Trojan-related characters with random sequences of corresponding length. This approach minimizes the reliance on specific terms while preserving the distinction between different Trojan-related characters. Allows the model to focus on structure and semantic content rather than particular characters.

Finally, we choose the "*Method E: Replace with random characters*" strategy because it can eliminate the influence of Trojan-related characters. This method effectively balances the generalization and specificity of the model. We will also adopt this method in subsequent experiments.

Table 3
Performance of BERT with different character processing strategies on the Trust-Hub Benchmark.

Benchmarks	Method A			Method B			Method C			Method D			Method E		
	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1
RS232-T1000	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1100	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1200	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1300	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1400	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1500	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1600	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T100	100.0	100.0	100.0	86.7	100.0	92.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T200	100.0	100.0	100.0	75.0	100.0	85.7	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T300	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T100	100.0	100.0	100.0	91.6	100.0	95.6	100.0	99.9	96.0	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T200	100.0	100.0	100.0	73.3	100.0	84.6	100.0	99.9	96.8	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T300	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.9	98.9	100.0	100.0	100.0	100.0	100.0	100.0
s38584-T100	100.0	100.0	100.0	88.8	100.0	94.1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s38584-T200	100.0	100.0	100.0	100.0	100.0	100.0	98.8	100.0	99.4	100.0	100.0	100.0	98.8	100.0	99.4
s38584-T300	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s15850-T100	100.0	100.0	100.0	92.5	100.0	96.1	100.0	100.0	100.0	100.0	100.0	100.0	97.6	100.0	98.8
AVG	100%	100%	100%	94.6%	100%	97.0%	99.9%	100%	99.5%	100%	100%	100%	99.8%	100%	99.9%

Table 4
Performance of GPT-2 and T5 on the Trust-Hub benchmark.

Benchmarks	GPT-2			T5		
	TPR	TNR	F1	TPR	TNR	F1
RS232-T1000	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1100	100.0	99.51	95.65	100.0	100.0	100.0
RS232-T1200	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1300	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1400	100.0	100.0	100.0	100.0	100.0	100.0
RS232-T1500	100.0	100.0	100.0	92.86	100.0	96.30
RS232-T1600	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T100	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T200	100.0	100.0	100.0	100.0	100.0	100.0
s35932-T300	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T100	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T200	100.0	100.0	100.0	100.0	100.0	100.0
s38417-T300	100.0	100.0	100.0	100.0	100.0	100.0
s38584-T100	100.0	100.0	100.0	100.0	100.0	100.0
s38584-T200	98.80	100.0	99.39	98.80	100.0	99.39
s38584-T300	100.0	99.98	99.93	100.0	99.98	99.93
s15850-T100	100.0	100.0	100.0	100.0	100.0	100.0
AVG	99.93%	99.97%	99.70%	99.51%	99.99%	99.74%

5.3. Benchmarks performance analysis

We evaluate the approach on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks. Furthermore, a comparative analysis with existing advanced approaches.

Experiment 2. Based on validating the character replacement strategy in *Experiment 1*, we extend the experiments on the Trust-Hub benchmark to GPT-2 and T5. We utilize the "GPT2LMHeadModel" and "T5ForConditionalGeneration" models to evaluate their performance on HT detection tasks.

Unlike BERT, which is primarily optimized for classification tasks, GPT-2 and T5 were originally designed for generative tasks. To adapt these models for HT detection, we added a classification head to each, enabling generative models to handle classification tasks.

We have previously analyzed BERT's performance under different character processing strategies, detailed in *Method E* of Table 3. For GPT-2, we employ the "GPT2LMHeadModel" variant and utilize "GPT2Tokenizer" with byte pair encoding to break down the text into subwords. For T5, we use the "T5ForConditionalGeneration" model along with "T5Tokenizer", applying the SentencePiece algorithm to optimize token granularity and sequence length. Both models were initialized with "gpt2" and "t5-small" using their respective tokenizers and pre-trained weights. The experimental results are presented in Table 4.

We evaluate the performance of three models on the Trust-Hub benchmark. Across all circuit datasets, these models show high effectiveness, achieving near-optimal TPR, TNR, and F1 score. GPT-2 achieved an average TPR of 99.93%, TNR of 99.97%, and F1 score of 99.7%. T5 performed equally well, with an average TPR of 99.51%, TNR of 99.99%, and F1 score of 99.74%. As shown in the "*Method E*" column of Table 3, BERT achieves an average TPR of 99.8%, a TNR of 100%, and an F1 score of 99.9%. While these models achieve high accuracy in most cases, there are some exceptions, such as the RS232-T1500 and s38584-T200 netlists, where the model performance degrades slightly. Overall, BERT, GPT-2, and T5 models performed well with equal-length random character replacement on the Trust-Hub benchmark.

Experiment 3. This experiment evaluates our approach to the TRIT-TC and TRIT-TS benchmarks. Unlike the Trust-Hub processing, in addition to the character processing in *Experiment 1*, we remove the characters introduced in the TRIT-TC and TRIT-TS benchmarks that affect HT detection, such as "troj" and "trig".

Table 5 shows the performance of BERT, GPT-2, and T5 on the TRIT-TC and TRIT-TS benchmarks. Performance on the TRIT-TC benchmark presents that the TPR, TNR, and F1 score of the three models are all over 99%, but the T5 model slightly outperforms the other two models in these indicators. T5 has a TPR of 99.33%, a TNR of 99.99%, and an F1 score of 99.98% on the TRIT-TC benchmark. These results present that T5 has a slight advantage in HT detection compared to BERT and GPT-2. Similarly, the evaluation of the TRIT-TS benchmark reached a similar conclusion. Since T5 achieved high scores in TPR, TNR, and F1, it achieved excellent performance on the TRIT-TC and TRIT-TS benchmarks. However, while GPT-2 performs well on both benchmarks, its F1 score (99.61%) on the TRIT-TC benchmark is slightly lower than that of BERT and T5. This suggests that GPT-2 may be less effective in balancing precision and recall on TRIT-TC.

These results suggest that the NtN method and Trojan-related character replacement methods contribute to HT detection performance.

Experiment 4. This experiment compares the latest netlists-based HT detection methods, focusing on performance metrics such as TNR, TPR, and F1 score. Precision (P) is added to the performance metrics for a more comprehensive comparison with other studies that use this metric.

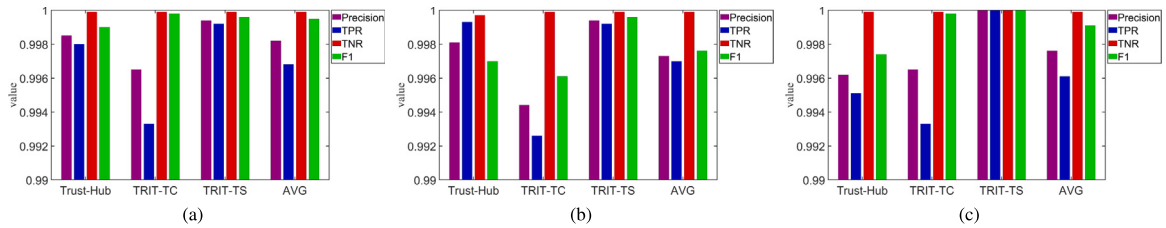

Fig. 10. The performance of our approach. (a) Performance of BERT, (b) Performance of GPT-2, (c) Performance of T5.

Table 5
Performance on the TRIT-TC and TRIT-TS Benchmarks.

Metrics	BERT	GPT-2	T5
TPR (TRIT-TC)	99.33%	99.26%	99.33%
TNR (TRIT-TC)	99.99%	99.99%	99.99%
F1 (TRIT-TC)	99.98%	99.61%	99.98%
TPR (TRIT-TS)	99.92%	99.92%	100%
TNR (TRIT-TS)	99.99%	99.99%	100%
F1 (TRIT-TS)	99.96%	99.96%	100%

Table 6
Comparison of different approaches.

	Precision	TPR	TNR	F1 score
LMDet (Shen et al., 2017)	–	82.8%	97.7%	–
GramsDet (Lu et al., 2019)	–	82.1%	96.0%	–
SeGa (Ye et al., 2021)	–	85.10%	93.58%	–
Netlist Info Ext. (Yu et al., 2022)	–	86.38%	99.98%	–
Topology-aware (Hassan et al., 2023)	93.5%	91.38%	–	91.28%
GNN (Alrahis et al., 2022)	91.2%	84.1%	–	86.0%
GNN4TJ (Yasaei et al., 2021)	92.3%	96.6%	–	94.0%
GNN classification (Ma, Shang, et al., 2024)	94.5%	–	–	96.7%
PS-TextCNN (Ma, Wang, & Wang, 2024)	–	88.9%	98.5%	–
GNN4HT (Chen et al., 2025)	80.95%	94.28%	97.22%	–
NtNDet (BERT)	99.82%	99.68%	99.99%	99.95%
NtNDet (GPT-2)	99.73%	99.70%	99.99%	99.76%
NtNDet (T5)	99.76%	99.61%	99.99%	99.91%
NtNDet (AVG)	**99.77%**	**99.66%**	**99.99%**	**99.87%**

Fig. 10 illustrates the individual performance of BERT, GPT-2, and T5 on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks and the combined average performance of all these models. We use average performance as a comparison. Table 6 summarizes the performance of different models on these benchmarks, showing that models based on our approach consistently outperform other excellent methods on all evaluation metrics. In our analysis, our average model demonstrates a substantial improvement over the best-performing model in the control group. The precision increased by at least 5.27%, TPR by 3.06%, and TNR by 0.01%. Additionally, the F1 score shows an increase of around 3.17%. However, it is essential to note that the datasets used in these studies vary; some use custom datasets while others use subsets of public datasets, which could potentially affect the comparative analysis.

Our approach has advantages in HT detection, such as higher accuracy, adaptability, no golden reference features, and automatic feature extraction. In particular, pre-trained models can obtain rich representations from many unlabeled data, which differs from traditional end-to-end strategies. While GNNs can handle structured data well, pre-trained models can capture complex dependencies and provide greater semantic richness. These features make our HT inspection very effective.

5.4. Generalization and robustness assessment

To assess our NLP-based approach's robustness and generalization capabilities, we examine how changes in netlist data representation impact detection performance. Since our model processes netlists similarly to language data, it is crucial to emphasize its robustness against variations in data representation. Unlike traditional HT detection methods, which focus on physical robustness to withstand environmental factors or perturbations, our approach prioritizes data-level robustness issues.

Considering that the TRIT-TC and TRIT-TS benchmarks have many netlists of the same type, they are more suitable for generalization and robustness experiments. We extracted evaluation samples with the same circuit names but different suffixes from the TRIT-TC and TRIT-TS benchmarks. We randomly selected 10 netlist files for each circuit, which means 50 and 40 netlist files were obtained from the TRIT-TC and TRIT-TS benchmark test sets, respectively. It is important to note that these selected netlists are independent of the training netlists; we excluded netlists T000, T001, and T002 to ensure an unbiased evaluation.

Experiment 5. This experiment evaluates the generalization ability of our approach to the TRIT-TC and TRIT-TS benchmarks. We use the selected 90 netlists as an independent evaluation dataset and use BERT, GPT-2, and T5 for generalization evaluation.

Fig. 11(a) exhibits the three models' commendable performance in the TRIT-TC benchmark testing involving 50 netlist files. BERT, on average, accomplished a TPR of 99.27%, a TNR of 99.99%, and an F1 score of 99.61%. Conversely, GPT-2 garnered slightly lower average scores, with a TPR of 99.26%, a TNR of 99.99%, and an F1 score of 99.6%. In stark contrast, T5 outperformed both by achieving near-perfect average scores of 99.99% across all metrics. The conversion to the TRIT-TS benchmark containing 40 netlist files, as shown in Fig. 11(b). T5 continued its lead with near-perfect average scores of 99.99% across TPR, TNR, and F1 score. BERT's average scores stood at 99.54% for TPR, 99.99% for TNR, and 99.77% for F1 score. GPT-2 closely trailed T5, with average scores of 99.97% for TPR, 99.99% for TNR, and 99.99% for F1 score.

Experiment 6. This experiment evaluates the robustness of our approach to data shuffling. We randomly shuffle each line of the netlist files and conduct experiments on the datasets of Trust-Hub, TRIT-TC, and TRIT-TS benchmarks using BERT, GPT-2, and T5.

This experiment simulates real-world scenarios where the order within the netlists does not follow any specific pattern, as shown in Table 7. Even though the order in the netlists changes, the model's performance on TPR, TNR, and F1 score metrics stays consistent with the original data. This consistency shows that the model can recognize and encode important patterns and structures, regardless of the input order. It demonstrates the robustness of our shuffle-disturbance method, which can effectively identify critical features within the netlists and ensure reliable HT detection, no matter where the HT is located.

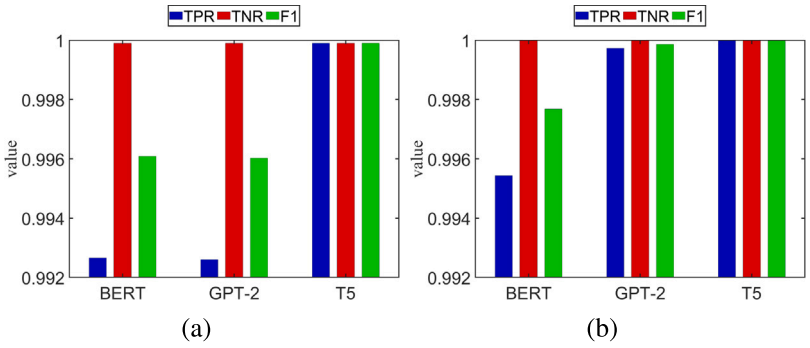

Fig. 11. Generalization performance of BERT, GPT-2, and T5 models on the TRIT-TC and TRIT-TS benchmarks. (a) Generalization on TRIT-TC, (b) Generalization on TRIT-TS.

Table 7
Benchmark Performance with Data Shuffling.

Metrics	BERT	GPT-2	T5
TPR (Trust-Hub)	99.80%	99.93%	99.51%
TNR (Trust-Hub)	100%	99.97%	99.99%
F1 (Trust-Hub)	99.99%	99.70%	99.74%
TPR (TRIT-TC)	99.33%	99.26%	99.33%
TNR (TRIT-TC)	99.99%	99.99%	99.99%
F1 (TRIT-TC)	99.98%	99.61%	99.98%
TPR (TRIT-TS)	99.92%	99.92%	100%
TNR (TRIT-TS)	99.99%	99.99%	100%
F1 (TRIT-TS)	99.96%	99.96%	100%

6. Discussion

6.1. Scalability and limitations

The NtNDet approach leverages a Transformer-based architecture that is inherently scalable due to its parallel processing capabilities and its ability to capture long-range dependencies. These properties enable our method to handle larger and more complex circuits without significantly degrading performance as circuit size increases.

While our primary focus is on gate-level HT detection, the underlying principles of NLP-based netlist analysis can be extended to a variety of emerging security challenges. Similar threats appear in automotive electronics (e.g., advanced driver assistance systems), data center accelerators (e.g., AI-specific chips), and consumer system-on-chip (SoC) designs, including Internet of Things (IoT) devices. NtNDet may detect malicious modifications early in the design process, proactively addressing increasingly sophisticated hardware attacks.

However, it must be acknowledged that our approach is computationally expensive and time-consuming, primarily due to the use of large-scale pre-trained models. Our current approach is limited to detecting known HTs in the training data. This means that novel or previously unseen Trojans may not be effectively identified, potentially limiting the model's applicability in real-world scenarios where new Trojans could emerge.

6.2. Future directions

Based on the current work, we plan to carry out the following research directions:

- **Integration with Lightweight Models:** Explore lightweight transformer variants such as DistilBERT (Sanh, 2019) and MobileBERT (Sun et al., 2020), which can be trained and fine-tuned for our HT detection task. These models aim to reduce computational resource requirements while maintaining high detection performance.
- **Extension to RTL-level:** Extend our approach to RTL-level, leveraging their structural similarities to gate-level designs. This will broaden the applicability of our model across different levels of circuit abstraction, enhancing its versatility in hardware security analysis.
- **Detection of Unknown or Novel Trojans:** Investigate techniques for identifying unknown or novel HTs not present in the training data. This will improve the effectiveness of our method in real-world situations where new types of Trojans may emerge, ensuring that our model remains robust against evolving threats.
- **Optimization Strategy Exploration:** Explore the potential of a two-stage fine-tuning approach, in which we transition from AdamW to SGD in the later stages of training. This strategy may help refine the model's performance, mitigate overfitting, and improve generalization, potentially leading to more stable and reliable accuracy levels.

By addressing these areas, we aim to enhance the scalability, efficiency, and practical applicability of our HT detection method, contributing to more robust and secure integrated circuit designs.

7. Conclusion

This study proposes a new approach to detecting HT using large-scale pre-trained NLP models. We introduce the NtN approach, which converts gate-level netlists into a format compatible with NLP models, leveraging Transformer models to capture the netlist's complex relationships. Our experiments on the Trust-Hub, TRIT-TC, and TRIT-TS benchmarks show that NtNDet outperforms existing HT detection methods. This approach shows great potential for advancing the field of HT detection and can serve as a foundation for further research.

CRediT authorship contribution statement

Shijie Kuang: Conceptualization, Methodology, Software, Investigation, Formal analysis, Writing – original draft. **Zhe Quan:** Visualization, Investigation. **Guoqi Xie:** Writing – original draft, Formal analysis. **Xiaomin Cai:** Conceptualization, Funding acquisition, Resources, Supervision, Writing – review & editing. **Xiaoqian Chen:** Software, Validation. **Keqin Li:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P., & Sunar, B. (2007). Trojan detection using IC fingerprinting. In *2007 IEEE symposium on security and privacy* (pp. 296–310). IEEE.

Alrahis, L., Patnaik, S., Shafique, M., & Sinanoglu, O. (2022). Embracing graph neural networks for hardware security. In *Proceedings of the 41st IEEE/ACM international conference on computer-aided design* (pp. 1–9).

Ashok, M., Turner, M. J., Walsworth, R. L., Levine, E. V., & Chandrakasan, A. P. (2022). Hardware trojan detection using unsupervised deep learning on quantum diamond microscope magnetic field images. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *18*(4), 1–25.

Bhunia, S., Hsiao, M. S., Banga, M., & Narasimhan, S. (2014). Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, *102*(8), 1229–1247.

Chen, L., Dong, C., Wu, Q., Liu, X., Guo, X., Chen, Z., et al. (2025). GNN4HT: A two-stage GNN-based approach for hardware trojan multifunctional classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *44*(1), 172–185.

Cheng, D., Dong, C., He, W., Chen, Z., Liu, X., & Zhang, H. (2023). A fine-grained detection method for gate-level hardware trojan base on bidirectional graph neural networks. *Journal of King Saud University-Computer and Information Sciences*, *35*(10), Article 101822.

Chowdhury, S. D., Yang, K., & Nuzzo, P. (2021). ReIGNN: State register identification using graph neural networks for circuit reverse engineering. In *2021 IEEE/ACM international conference on computer aided design* (pp. 1–9). IEEE.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American chapter of the association for computational linguistics*.

Dhavlle, A., Hassan, R., Mittapalli, M., & Dinakarrao, S. M. P. (2021). Design of hardware trojans and its impact on cps systems: A comprehensive survey. In *2021 IEEE international symposium on circuits and systems* (pp. 1–5). IEEE.

Dofe, J., Danesh, W., More, V., & Chaudhari, A. (2024). Natural language processing for hardware security: Case of hardware trojan detection in FPGAs. *Cryptography*, *8*(3).

Dong, C., Xu, Y., Liu, X., Zhang, F., He, G., & Chen, Y. (2020). Hardware trojans in chips: A survey for detection and prevention. *Sensors*, *20*(18).

Hasegawa, K., Hidano, S., Nozawa, K., Kiyomoto, S., & Togawa, N. (2022). R-HTDetector: Robust hardware-trojan detection based on adversarial training. *Institute of Electrical and Electronics Engineers. Transactions on Computers*, *72*(2), 333–345.

Hasegawa, K., Oya, M., Yanagisawa, M., & Togawa, N. (2016). Hardware trojans classification for gate-level netlists based on machine learning. In *2016 IEEE 22nd international symposium on on-line testing and robust system design* (pp. 203–206). IEEE.

Hassan, R., Meng, X., Basu, K., & Dinakarrao, S. M. P. (2023). Circuit topology-aware vaccination-based hardware trojan detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*, 2852–2862.

Hu, W., Chang, C.-H., Sengupta, A., Bhunia, S., Kastner, R., & Li, H. (2021). An overview of hardware security and trust: Threats, countermeasures, and design tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *40*(6), 1010–1038.

Hu, L., Liu, Z., Zhao, Z., Hou, L., Nie, L., & Li, J. (2024). A survey of knowledge enhanced pre-trained language models. *IEEE Transactions on Knowledge and Data Engineering*, *36*(4), 1413–1430.

Ketan, J., Shetty, A. A., et al. (2024). Hardware trojan detection in ISCAS-89 circuits using supervised machine learning algorithms. In *2024 international conference on automation and computation* (pp. 411–415). IEEE.

Lu, R., Shen, H., Su, Y., Li, H., & Li, X. (2019). GramsDet: Hardware trojan detection based on recurrent neural network. In *2019 IEEE 28th Asian test symposium* (pp. 111–1115). IEEE.

Ma, P., Shang, G., Liu, H., Shi, J., Pan, W., Zhang, Y., et al. (2024). GNN-based hardware trojan detection at register transfer level leveraging multiple-category features. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1–10.

Ma, P., Wang, Z., & Wang, Y. (2024). A pre-silicon detection based on deep learning model for hardware trojans. *Journal of Circuits, Systems and Computers*, *33*(08), Article 2450144.

Muralidhar, N., Zubair, A., Weidler, N., Gerdes, R., & Ramakrishnan, N. (2021). Contrastive graph convolutional networks for hardware trojan detection in third party ip cores. In *2021 IEEE international symposium on hardware oriented security and trust* (pp. 181–191). IEEE.

Narasimhan, S., Du, D., Chakraborty, R. S., Paul, S., Wolff, F. G., Papachristou, C. A., et al. (2012). Hardware trojan detection by multiple-parameter side-channel analysis. *Institute of Electrical and Electronics Engineers. Transactions on Computers*, *62*(11), 2183–2195.

Niu, S., Liu, Y., Wang, J., & Song, H. (2020). A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, *1*(2), 151–166.

Otter, D. W., Medina, J. R., & Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(2), 604–624.

Pan, Z., & Mishra, P. (2021). Automated test generation for hardware trojan detection using reinforcement learning. In *Proceedings of the 26th Asia and south Pacific design automation conference* (pp. 408–413).

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, *63*(10), 1872–1897.

Radford, A., & Narasimhan, K. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8), 9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*(1), 5485–5551.

Rahimifar, M. M., Jahanirad, H., & Fathi, M. (2024). Deep transfer learning approach for digital circuits vulnerability analysis. *Expert Systems with Applications*, *237*, Article 121757.

Ravimaram, S., Sathish, A., Vatchala, S., Rawat, R., TF, M. R., et al. (2023). Robust transfer learning based modelling for accelerating the learning of ai in the field of NLP. *1*, In *2023 9th international conference on advanced computing and communication systems* (pp. 1026–1030). IEEE.

Salmani, H., Tehranipoor, M., & Karri, R. (2013). On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st international conference on computer design* (pp. 471–474). IEEE.

Samyukta, K., & Ramesh, S. (2023). Detection of hardware trojan horse using unsupervised learning approach. In *2023 IEEE international conference on distributed computing, VLSI, electrical circuits and robotics* (pp. 77–82). IEEE.

Sanh, V. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Sankaran, S., Mohan, V. S., & Purushothaman, A. (2021). Deep learning based approach for hardware trojan detection. In *2021 IEEE international symposium on smart electronic systems* (pp. 177–182). IEEE.

Shen, H., Tan, H., Li, H., Zhang, F., & Li, X. (2017). Lmdet: A "naturalness" statistical method for hardware trojan detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(4), 720–732.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., & Zhou, D. (2020). Mobilebert: a compact task-agnostic bert for resource-limited devices. arXiv preprint arXiv:2004.02984.

Surabhi, V. R., Krishnamurthy, P., Amrouch, H., Henkel, J., Karri, R., & Khorrami, F. (2022). Trojan detection in embedded systems with FinFET technology. *Institute of Electrical and Electronics Engineers. Transactions on Computers*, *71*(11), 3061–3071.

Taneja, K., & Vashishtha, J. (2022). Comparison of transfer learning and traditional machine learning approach for text classification. In *2022 9th international conference on computing for sustainable global development* (pp. 195–200). IEEE.

(2024). *Trust-hub*, http://www.trust-hub.org.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.

Xue, M., Gu, C., Liu, W., Yu, S., & O'Neill, M. (2020). Ten years of hardware trojans: a survey from the attacker's perspective. *IET Computers & Digital Techniques*, *14*(6), 231–246.

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

Yasaei, R., Yu, S.-Y., & Al Faruque, M. A. (2021). Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 design, automation & test in europe conference & exhibition* (pp. 1504–1509). IEEE.

Ye, Y., Li, S., Shen, H., Li, H., & Li, X. (2021). SeGa: A trojan detection method combined with gate semantics. In *2021 IEEE 30th Asian test symposium* (pp. 43–48). IEEE.

Yu, S., Gu, C., Liu, W., & O'Neill, M. (2022). Deep learning-based hardware trojan detection with block-based netlist information extraction. *IEEE Transactions on Emerging Topics in Computing*, *10*(4), 1837–1853.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., et al. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57–81.

Zhu, Z., Lin, K., Jain, A. K., & Zhou, J. (2023). Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(11), 13344–13362.