SAND: A Self-supervised and Adaptive NAS-Driven Framework for Hardware Trojan Detection

Zhixin Pan, Ziyu Shu, Linh Nguyen, and Amberbir Alemayoh

Abstract

The globalized semiconductor supply chain has made Hardware Trojans (HT) a significant security threat to embedded systems, necessitating the design of efficient and adaptable detection mechanisms. Despite promising machine learning-based HT detection techniques in the literature, they suffer from ad hoc feature selection and the lack of adaptivity, all of which hinder their effectiveness across diverse HT attacks. In this paper, we propose SAND, a self-supervised and adaptive NAS-driven framework for efficient HT detection. Specifically, this paper makes three key contributions. (1) We leverage self-supervised learning (SSL) to enable automated feature extraction, eliminating the dependency on manually engineered features. (2) SAND integrates neural architecture search (NAS) to dynamically optimize the downstream classifier, allowing for seamless adaptation to unseen benchmarks with minimal fine-tuning. (3) Experimental results show that SAND achieves a significant improvement in detection accuracy (up to 18.3%) over state-of-the-art methods, exhibits high resilience against evasive Trojans, and demonstrates strong generalization.

1 Introduction

The rapid advancement of System-on-Chip (SoC) technology has led to increasing design scales, necessitating the globalization of the semiconductor supply chain to meet time-to-market constraints. However, the inevitable integration of third-party Intellectual Property (IP) vendors in the supply chain introduces security risks, exposing SoCs to Hardware Trojans (HTs) attacks. HTs pose severe threats, including information leakage and erroneous execution. Therefore, mitigating HT attacks is crucial to ensuring the trustworthiness of modern SoCs.

This paper focuses on trigger-activated HTs, which are the predominant type in current literature. Their trigger conditions are often designed using a combination of rare events (signals, transitions) to evade detection, posing the efficient defense against such HT attacks a serious challenge.

Conventional HT detection methods fall into two main categories: *formal verification* and *test generation*. Formal verification methods aim to mathematically prove the correctness of a circuit by exhaustively checking predefined security properties through a SAT solver [11]. However, they are often impractical for large-scale SoCs due to the high computational cost caused by exponential time complexity. Test generation methods, instead, apply carefully crafted test vectors to trigger anomalous behaviors or side-channel information, indicating the presence of HTs [7]. While effective performance was demonstrated in existing works, they are time-consuming to detect well-hidden Trojans [22].

Given the above limitations, machine learning (ML) has emerged as a promising approach for HT detection, demonstrating superior performance over traditional techniques. However, existing ML-based approaches also suffer from several critical limitations. First, these methods depend largely on manually selected feature vectors without a standardized guideline. Additionally, most ML models exhibit poor generalizability when applied to unseen variants of HTs, necessitating expensive efforts for model retraining [8]. This is especially problematic in the context of HTs detection, as Trojans may vary significantly across different devices, thus requiring frequent model updates.

To address these challenges, we propose SAND, a Self-supervised and Adaptive Detector for HT. The proposed framework enables an efficient integration between self-supervised learning (SSL) and neural architecture search (NAS). Specifically, the proposed work offers three major contributions:

- **Automated Feature Embedding via SSL**: Unlike existing works relying on manually selected features, SAND employs SSL to pre-train an encoder that captures meaningful representations from hardware benchmarks.
- **Adaptive Classifier Optimization via NAS**: SAND leverages NAS to dynamically search for an optimal neural network architecture tailored to specific detection tasks with minimum retraining overhead.
- **Comprehensive Experimental Evaluation**: Experimental results show that SAND achieves a significant improvement in detection accuracy (up to 18.3%), adaptivity, and stability over state-of-the-art methods.

2 Related Work and Background

2.1 ML-based HT Detection

As discussed in Section 1, ML-based HT detection approaches have demonstrated superior performance over traditional techniques. The majority of these methods rely on supervised learning [12], where the ML model is trained to identify HTs through pattern recognition of pre-defined circuit features. For example, a support vector machine (SVM)-based model was proposed in [4] to detect anomalous behaviors associated with HT activations in simulation-based environments. Similarly, deep learning-based techniques have been employed for HT detection in [24]. Other ML models, such as random forests [9], convolutional neural networks (CNNs)[23], and reinforcement learning[1, 14], have also been explored, obtaining promising classification accuracy. Despite these advancements, ML-based HT detection still suffers from critical limitations. First, most existing methods rely on **manually selected features** based on human expert knowledge, without a universally applicable strategy. For example, Hasegawa et al. [6] identified five key features from gate-level netlists to train a classifier for HT detection. Similarly, Zhou et al. [26] developed a feature extraction algorithm by analyzing the distribution of rare signals within IP cores. While effective within specific contexts, these domain-specific methods struggle to be applied universally.

This paper was published in the 2025 IEEE International Conference on Computer Design

Second, these methods often exhibit **poor generalizability**, struggling to detect unseen HT variants and requiring frequent retraining [8]. According to a recent study [3], the detection performance of several existing algorithms fluctuates across different benchmarks, often dropping below 60%, which is no better than a random guess. This highlights the urgent need for more efficient and adaptive HT detection frameworks.

2.2 Self-Supervised Learning (SSL)

The limitations of existing ML-based HT detection methods, particularly their reliance on manually designed feature representations, have highlighted the need for more adaptive approaches. A promising direction to address this challenge is through **self-supervised learning (SSL)**, which has demonstrated remarkable success in automated feature embedding.

There are different categories of SSL algorithms. In this paper, we focus on **contrastive learning**, a method that learns representations by distinguishing between similar and dissimilar data points. Specifically, the goal of contrastive learning is to map data into a feature space where similar sample pairs (positives) are pulled closer, while dissimilar ones (negatives) are pushed apart. This approach enables the model to capture essential patterns without requiring explicit feature engineering, shown in Figure 1.

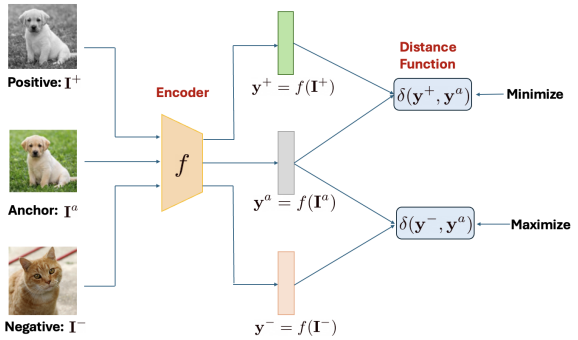


Figure 1: Illustration of contrastive learning. Given an anchor input I^a, a positive example I^+ is generated through data augmentation, while a negative example I^- is selected from a different class. The model learns a feature representation such that the distance $\delta(I^a, I^+)$ is minimized, while the distance $\delta(I^a, I^-)$ is maximized.

Compared to traditional supervised learning, where models rely on labeled data, SSL learns directly from the intrinsic structure of the dataset without the reliance on expert knowledge.

2.3 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) refers to techniques that automate the design of deep neural networks (DNNs) by searching for optimal architectures within a predefined space. Unlike manually designed AI models, NAS explores a large set of possible network architectures to identify configurations that yield superior performance, while also balancing the computational efficiency and model complexity. Typically, the NAS framework consists of three key components:

- **Search Space**: Defines the set of possible architectures, including choices for layer types, connections, and other hyperparameters.
- **Search Strategy**: Defines how architectures are explored within the search space.
- **Performance Estimation**: Evaluates the quality of candidate architectures, provides guidelines for subsequent updating.

There are many different strategies to implement the NAS framework, existing works include but are not limited to reinforcement learning (RL)-based NAS [25], evolutionary algorithm (EA)-based NAS [13], and gradient-based NAS [16]. While NAS has been widely applied in fields such as image classification and natural language processing, its adoption in hardware security and HT detection remains limited. We will demonstrate how NAS was integrated into our framework to enable adaptive HT detection, details are discussed in Section 3.4.

3 Proposed Methodology

3.1 Overview

To address these challenges described in Section 2, we propose SAND, a self-supervised learning (SSL)-based framework enhanced with neural architecture search (NAS) for adaptive and efficient HT detection. Figure 2 presents an overview of the proposed workflow. The framework consists of an **upstream encoder** and a **downstream classifier**, with the implementation divided into the following primary tasks:

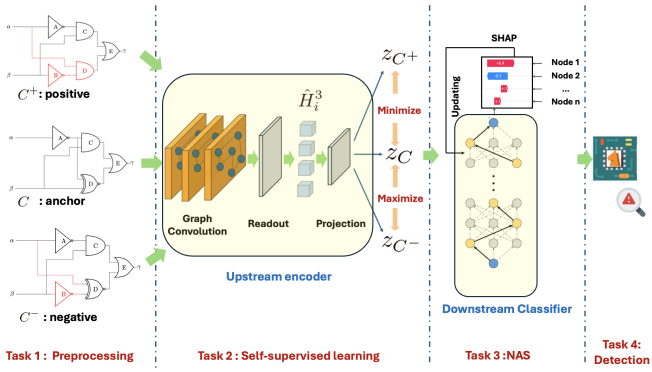


Figure 2: The overview framework of SAND. The main module consists of an upstream encoder, implemented through self-supervised learning (SSL) and adownstream classifier crafted by neural architecture search (NAS).

- **Data preprocessing:** In this task, we implement the specific data preprocessing technique for hardware circuits, and further transform circuit information into a graph format for subsequent contrastive learning. (Section 3.2)
- **Contrastive learning based SSL:** We adopt the contrastive learning approach to build a Graph Convolutional Network (GCN) as the upstream encoder, which learns the transferable feature representations of hardware circuits without relying on manually designed features. (Section 3.3)

- **SHAP-based NAS:** The features are fed into the downstream classifier to enable adaptive HT detection. Specifically, we integrate NAS to automatically optimize its architecture, where Shapley value analysis (SHAP)[21] is incorporated to refine the optimization process. (Section 3.4)
- **HT detection:** The finalized models are deployed in real-world HT detection tasks with enhanced adaptivity.

3.2 Data Preprocessing

Effective contrastive learning relies on crafting both positive and negative samples. For hardware benchmarks, negative samples can be crafted by injecting Trojans into the anchor samples. *However, generating positive samples for circuits is more challenging.* Directly using other benign circuits is impractical, as variations in scale and type lead to structural and functional disparities. Additionally, unlike image-based tasks, where simple augmentations like cropping or rotation preserve semantics, even a single gate modification in a circuit can alter its behavior. To address this, given a circuit C as anchor input, we obtain its positive counterpart C^+ using the following transformations:

- **Logic-equivalent modification:** We introduce structural variations while maintaining logical equivalence. For example, using DeMorgan's theorem to replace an AND gate with a NAND gate followed by a NOT gate.
- **Subcircuit extraction:** We adopt the algorithm in [18] to extract subcircuits from the given circuit, maintaining the partial structural patterns while reducing complexity.
- **Relocation:** We also randomly alter the locations of independent sub-modules within the same benchmark, introducing structural variations while preserving functionality.

Meanwhile, we transform circuits into graph representations and apply Graph Convolutional Networks (GCNs) in the subsequent contrastive learning stage for effective feature embedding. Basically, we model the input circuit C as a directed graph $G(\mathcal{V}, \mathcal{E})$. Nodes $\mathcal{V}$ represent circuit cells, encoding their fundamental information (cell-type, fan-in/outs, etc.). Edges $\mathcal{E}$ denote connection wires between cells, capturing the circuit's structural information.

3.3 SSL-based Upstream Encoder

In this section, we describe how the generated data is leveraged within our SSL framework. Specifically, we design the **upstream encoder** based on GCNs, followed by one readout layer, and one projection layer. The training is done by minimizing a *hybrid contrastive loss function*. The entire framework consists of the following key components:

3.3.1 Graph Convolution Layers (GCNs).

We adopt a *three-layer* GCN to capture hierarchical information from circuit graphs. Given the graph G as defined in the previous section, we assume each node $v \in \mathcal{V}$ has the initial attribute vector $\mathbf{h}_v^{(0)}$ encoding its fundamental information. Then the forward propagation in each GCN layer is defined as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops, $\tilde{\mathbf{D}}$ is the corresponding degree matrix. The term $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ normalizes the adjacency matrix by the degree matrix. This ensures that the information passed between nodes is scaled by their degree, allowing the network to focus on the relative importance of neighbors. This graph convolution operation can be seen as a message-passing operation, where each node aggregates information from its neighbors (and itself). At each iteration of graph convolution, each node embedding $\mathbf{h}_v^{(L)}$ captures both local connectivity and hierarchical circuit patterns, while propagating to the output layer to formalize a valid feature encoding for the given circuit.

3.3.2 Readout Layer.

To obtain a standard representation among circuits with different scales, we apply a readout function aggregating node embeddings into a single vector using *global sum pooling:*

$$\hat{\mathbf{H}}^{(1)} = \sum_{v \in \mathcal{V}} \mathbf{h}_v^{(L)} \tag{2}$$

3.3.3 Projection Layer.

We also apply a projection layer to map the aggregated feature into a fixed-length vector using a fully connected dense layer:

$$\mathbf{z} = \sigma(\mathbf{W}_p\hat{\mathbf{H}}^{(l)} + b_p) \tag{3}$$

Where σ is the non-linear activation function, $\mathbf{W}_p$ is the weight matrix of the projection layer, and b_p is the bias term. The projection allows flexibility in adapting to different problem sizes.

3.3.4 Hybrid Contrastive Loss Formulation.

Once the projected feature vectors are obtained, we can train our proposed model to enable contrastive learning. To effectively learn meaningful representations for HT detection, we design a **hybrid contrastive loss** consisting of three complementary components:

- a *Positive Contrastive Loss* to separate anchor circuits from the positive samples
- a *Negative Contrastive Loss* to separate anchor circuits from the negative samples.
- a *Global Clustering Loss* to encourage circuits of the same category to form well-separated clusters.

Positive Contrastive Loss. The positive contrastive loss aims to minimize the distance between the anchor sample C and all its corresponding positive samples C_i^+: $\mathcal{L}_P = \sum_i \|z_C - z_{C_i^+}\|^2$.

Negative Contrastive Loss. Similarly, the negative contrastive loss is defined as: $\mathcal{L}_N = \sum_i \max(0, m - \|z_C - z_{C_i^-}\|^2)$,

note that m is a margin hyperparameter that ensures a minimum distance between an original circuit and its Trojan-inserted variant.

Global Clustering Loss. For conventional contrastive learning, $\mathcal{L}_P$ and $\mathcal{L}_N$ suffice to define the total loss. However, in HT detection, circuits within the same category (benign/malicious) can exhibit significant variations in structural composition and basic functionality. Moreover, the structural difference between an original circuit and its Trojan-injected variant is typically much smaller than that between it and another irrelevant benign circuit. As a result, if we merely rely on positive/negative contrastive loss, training an effective classifier becomes extremely challenging. Figure 3(b) illustrates a representative example of this phenomenon.

To address this challenge, we introduce a *global clustering loss* into our framework, which enforces category-level coherence by

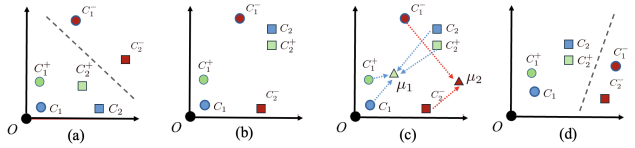


Figure 3: Illustration of global clustering. Two benign circuits (C_1, C_2) and their associated positive/negative samples are shown. (a) Without global structure, contrastive loss ($\mathcal{L}_P$, $\mathcal{L}_N$) may succeed if C_1 and C_2 are close, (b) but fails when they are far apart. (c) Global clustering loss pulls samples toward their class centroid (μ_k), (d) enhancing inter-class separation and reducing intra-class variance.

softly grouping circuits of the same class. Specifically, in addition to preserving pairwise relationships, we define class-specific centroids μ_k and encourage each circuit embedding to be close to its corresponding centroid, as shown in Figure 3(c). This ensures circuits of the same category (benign/malicious) form tight clusters in the feature space, balancing the intra-class variations:

$$\mathcal{L}_{\mathrm{G}} = \sum_{C \in \mathcal{D}} \|z_C - \mu_k\|^2, \quad \text{where } \mu_k = \frac{1}{|\mathcal{D}_k|} \sum_{C \in \mathcal{D}_k} z_C \tag{4}$$

where $\mathcal{D}_k$ denotes the set of all circuits belonging to class k, in our case $k \in \{0, 1\}$ since HT detection is a binary classification.

The final hybrid loss function integrates all three components, and we outline the complete training procedure in Algorithm 1. The encoder after training is expected to extract meaningful structural and functional representations of given circuits without a manual feature engineering step.

Algorithm 1: Contrastive Learning based SSL

Input: Circuit graphs $G = \{G_i\}$, encoder parameters θ
Output: Trained upstream encoder

`/* Graph Convolution */`
for *each epoch* **do**
  for *each mini-batch* **do**
    for *each GCN layer* $l = 1, 2, 3$ **do**
      $H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$
    $\hat{H}^{(L)} = \sum_{v \in \mathcal{V}} h_v^{(L)}$
    $z = \sigma(W_p\hat{H}^{(L)} + b_p)$
  `/* Loss Computation and Backpropagation */`
  $\mathcal{L}_P = \sum_{i,j} \|z_{C_i} - z_{C_{ij}^+}\|^2$
  $\mathcal{L}_N = \sum_{i,j} \max(0, m - \|z_{C_i} - -z_{C_{ij}^-}\|^2)$
  $\mathcal{L}_{\mathrm{G}} = \sum_i \|z_{C_i} - \mu_k\|^2$
  Total loss: $\mathcal{L} = \lambda_1\mathcal{L}_P + \lambda_2\mathcal{L}_N + \lambda_3\mathcal{L}_{\mathrm{G}}$
  Backpropagation: $\theta = \theta - \nabla_\theta\mathcal{L}$;
return θ ;

3.4 NAS-based Downstream Classifier

In this section, we describe how the upstream encoder's outputs are utilized to train a downstream classifier for HT detection.

As discussed in Section 2.1, another key challenge in ML-based HT detection is the limited adaptivity across circuit and Trojan variants. Manually designed architecture with fixed hyperparameters often lacks flexibility, leading to expensive retraining costs.

To address this issue, we employ Neural Architecture Search (NAS) to identify the optimal network architecture. By leveraging NAS, we aim to dynamically construct the classifier's architecture tailored to a focused HT detection task. Specifically, we apply the one-shot NAS training strategy consisting of two major steps: (i) SuperNet Training, and (ii) SHAP-based Pruning.

3.4.1 SuperNet Training. We begin with a large, over-parameterized network (*SuperNet*) that encompasses all potential architectural configurations. At each layer, the SuperNet incorporates a diverse set of network components, including fully connected layers, convolutional layers, pooling layers, and activation functions. We first train this SuperNet to establish its core classification capability.

3.4.2 SHAP-based Pruning. The next step is to apply evaluation metrics to identify the most effective components inside the SuperNet, extracting them as a streamlined sub-network optimized for the current HT detection task. We achieve this by identifying and pruning redundant components using Shapley value analysis (SHAP) [21], which assigns each component an importance value based on its contribution to a model's output. Higher SHAP values suggest that a component plays a crucial role in classification, while lower values indicate redundancy and will be pruned, leading to a more compact and task-specific model while preserving classification performance. Figure 4 demonstrates the NAS process.

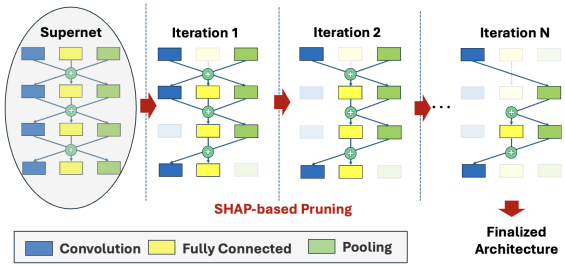


Figure 4: The workflow of SHAP-based NAS for optimizing the downstream classifier architecture. The process begins with an over-parameterized SuperNet, components with low SHAP values are progressively pruned (faded).

4 Experimental Evaluation

4.1 Experiment Setup

4.1.1 Implementation: To enable a comprehensive evaluation of proposed framework, the experimental is conducted on a host machine with an Intel i7 3.70GHz CPU, 32 GB RAM, and an NVIDIA RTX 3090 GPU. The proposed HT detection framework is implemented in Python, with model training and evaluation using the PyTorch deep learning library. Circuit netlists are transformed into graph representations using CircuitGraph [17], while SHAP-based NAS is implemented with libraries from [10].

4.1.2 Dateset: We use a diverse of benchmarks to ensure comprehensive evaluation:

- ISCAS85 & ISCAS89 [5]: Standard combinational and sequential benchmark suites.
- Trust-Hub [15]: Open repository with HT-infested designs across various circuit types.
- Open Designs [20]: Open designs based on RISC-V and MIPS architectures.
- Synthetic: Custom circuits with varied structures to simulate different attack scenarios.

Trojans are crafted and injected into these benchmarks using the tool provided from [2].

4.1.3 Methods: In this paper, we consider and compare the performance of the following four ML-based HT detection schemes:

- SVM [4]: Support vector machine (SVM) based method.
- AdaTest [1]: Reinforcement learning based adaptive method.
- GATE-Net [19]: State-of-the-art deep learning based method.
- SAND: Our proposed SSL-based and NAS-driven adaptive HT detection framework.

4.2 Case Study: Detection Accuracy

Table 1 compares the performance of our proposed method, SAND, with other approaches for HT detection using accuracy (Acc), recall (Rec), precision (Pre), and F1-score (F1) across various benchmarks. Each row corresponds to a benchmark, with average values summarized in the last row.

SVM exhibits the lowest overall performance, as expected due to its lightweight architecture and limited capacity, which leads to overfitting and poor generalization for large or synthetic benchmarks. For example, despite **100%** accuracy on c2670 and c5315, it drops to **50.8%** on RISC-V designs, comparable to random guessing. AdaTest, a reinforcement learning-based method, improves over SVM with more consistent results. However, its low precision and high recall suggest a tendency to over-detect Trojans, majorly due to its adaptive feedback-driven strategy. This behavior helps catch unseen threats but also raises false positives by misclassifying benign instances. As a comparison, GATE-Net, based on deep learning, achieves stronger performance than these two methods, but slightly degrades on larger benchmarks, indicating scalability limitations. It also shows higher precision and lower recall, reflecting a more conservative prediction style that aims to avoid false alarms but ends up missing threats. In contrast, SAND utilizes SSL to outperform the others with up to **18.3%** gain in detection accuracy and consistently strong performance across benchmarks.

4.3 Case Study: SSL-based Feature Embedding

To evaluate the effectiveness of the proposed SSL framework for automatic feature encoding, we analyze the evolution of feature embeddings over training epochs visualized by 2-D Principal Component Analysis (PCA). Figure 5 illustrates the incremental learning process. An ablation test of the global clustering loss is also performed to address its necessity for HT detection.

At the initial stage (Epoch = 0, Figure 5(a)), the initialized values are scattered randomly in space. As training progresses to Epoch = 50 (Figure 5(b)), initial clusters begin to emerge, but the distances between clusters are not sufficiently far. By Epoch = 150 (Figure 5(c)), the feature embeddings are well-formed, malicious samples are clustered together and pushed far from the benign and positive samples, making it a valid feature encoding. This result highlights the effectiveness of the proposed SSL framework, where meaningful feature representations emerge automatically through the learning process, without requiring predefined expert-driven heuristics or manual feature selection.

Notably, if we disable the global clustering loss and solely rely on contrastive loss, the learned representations exhibit a drastic difference (Figure 5(d)). As we can see, instead of forming compact clusters per category, multiple local clusters exist within each category, leading to intermixing between different categories, reducing the model's ability to generalize across diverse benchmarks.

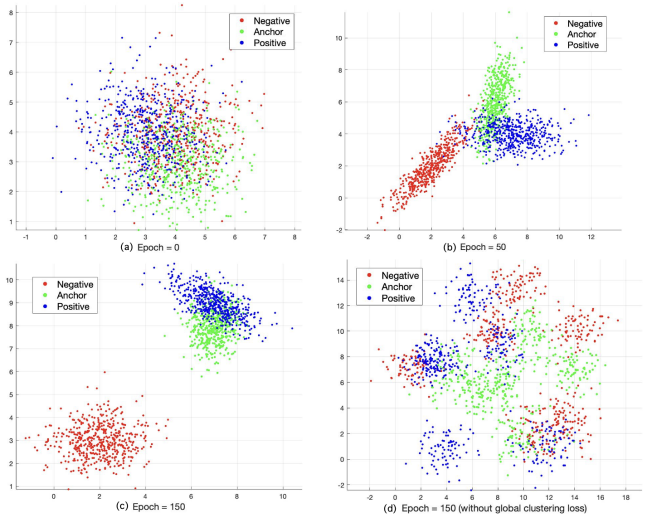


Figure 5: 2D feature embeddings learned by SAND. (a) Epoch 0: Features are randomly scattered. (b) Epoch 50: Early clustering with some category overlap. (c) Epoch 150: Clear, well-separated clusters. (d) Without global clustering loss: fragmented intra-class clusters lead to intermixing.

4.4 Case Study: NAS-based Adaptability

In this section, we evaluate the performance of our NAS-based framework. First, our framework enables automated model design by analyzing the contributions of different cell types (e.g., convolution, fully connected, pooling) within a 16-layer SuperNet, where each layer contains 6 cells. As shown in Figure 6(a).

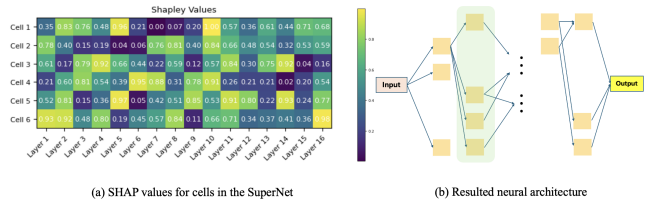


Figure 6: The automatically generated model architecture through SHAP-based NAS.

By computing the 2D SHAP values for all cells and applying thresholding, the final architecture in our experiment is shown

Table 1: Performance Comparison using accuracy (Acc), recall (Rec), precision (Pre) and F1 score (F1).

	SVM [4]				AdaTest [1]				GATE-Net [19]				SAND (Proposed Approach)			
Bench	**Acc**	**Rec**	**Pre**	**F1**	**Acc**	**Rec**	**Pre**	**F1**	**Acc**	**Rec**	**Prec**	**F1**	**Acc**	**Rec**	**Pre**	**F1**
c2670	100.0%	1.0	1.0	1.0	93.2%	0.94	0.90	0.92	100.0%	0.94	0.97	0.96	100.0%	1.0	1.0	1.0
c5315	100.0%	1.0	1.0	1.0	97.6%	0.99	0.95	0.96	100.0%	0.91	0.92	0.92	100.0%	1.0	1.0	1.0
c6288	88.5%	0.88	0.83	0.84	94.5%	0.98	0.89	0.93	98.1%	0.85	0.89	0.87	100.0%	0.99	0.99	0.99
c7552	87.2%	0.81	0.92	0.86	84.9%	0.86	0.81	0.83	100.0%	0.89	0.91	0.90	100.0%	1.0	1.0	1.0
s13207	78.5%	0.77	0.79	0.78	90.0%	0.92	0.89	0.90	95.7%	0.94	0.95	0.95	100.0%	1.0	1.0	1.0
s15850	78.8%	0.75	0.79	0.77	82.0%	0.84	0.79	0.81	90.0%	0.84	0.97	0.90	98.8%	0.99	0.99	0.99
s35932	73.1%	0.80	0.70	0.74	79.5%	0.80	0.77	0.78	80.5%	0.75	0.86	0.80	98.8%	0.97	0.99	0.98
AES-T100	85.9%	0.84	0.86	0.85	100.0%	1.0	1.0	1.0	96.5%	0.96	0.96	0.96	100.0%	1.0	1.0	1.0
AES-T200	79.3%	0.77	0.80	0.78	96.2%	0.97	0.95	0.96	95.1%	0.92	0.97	0.94	96.4%	1.0	1.0	1.0
AES-T1000	86.2%	0.86	0.87	0.86	90.5%	0.93	0.88	0.90	90.4%	0.85	0.95	0.90	98.8%	1.0	1.0	1.0
MIPS	66.7%	0.69	0.64	0.66	88.8%	0.89	0.87	0.88	73.5%	0.72	0.75	0.74	94.0%	0.95	0.93	0.94
RISCV	50.8%	0.53	0.50	0.51	82.0%	0.85	0.80	0.83	77.2%	0.77	0.77	0.77	90.4%	0.89	0.92	0.91
Synthetic	73.2%	0.70	0.75	0.72	85.4%	0.88	0.80	0.84	76.9%	0.74	0.79	0.77	91.0%	0.90	0.91	0.91
Average	**81.3 %**	**0.80**	**0.80**	**0.80**	**90.6%**	**0.91**	**0.88**	**0.90**	**92.2%**	**0.89**	**0.92**	**0.90**	**98.1%**	**0.98**	**0.98**	**0.98**

in Figure 6(b). While seemingly intuitive, this architecture was obtained automatically by NAS, without the need for expert-driven feature engineering or heuristic assumptions.

However, the NAS framework also incur higher overhead compared to the baseline approaches, shown in Figure 7. Specifically, SAND introduces increased training time and memory usage due to the need to explore and maintain a supernet during architecture optimization, and also exhibits elevated power consumption during training. Despite this additional cost, the NAS-based architecture grants strong adaptability to our proposed method. For example, we have conducted experiments using unseen benchmarks to assess performance degradation and retraining overhead. The results are shown in Table 2.

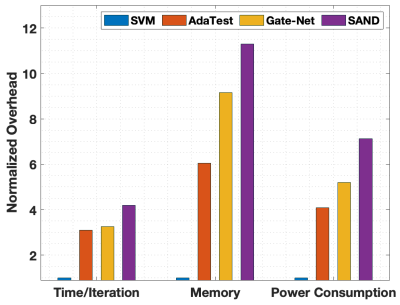


Figure 7: Training overhead comparison across methods.

As we can see, while traditional methods experience significant drops in accuracy, our approach retain strong performance. This robustness can be attributed to the utilization of self-supervised learning (SSL), which captures fundamental netlist properties without relying on manually crafted features, further improving adaptability. Additionally, the proposed approach minimizes retraining overhead, since we only need to fine-tune the downstream classifier encoded within the SuperNet, making adaptation to new benchmarks efficient. In contrast, all the other methdos requires longer epochs, leading to higher total costs for retraining.

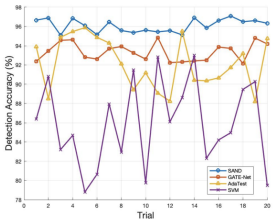
Table 2: Performance of adaptive HT detection.

Method	**Seen**	**Unseen**	**Perf-Drop**	**Retraining (Epochs)**
SVM [4]	81.3%	65.2%	-16.1%	38
AdaTest [1]	90.6%	70.8%	-18.3%	32
GATE-Net [19]	92.2%	75.4%	-16.8%	22
SAND (Proposed)	98.1%	94.2%	-3.9%	7

4.5 Case Study: Stability

Stability is another crucial factor for real-world HT detection tasks. To evaluate the stability of our proposed method, we conducted 20 trials to compare the detection accuracy of all methods.

As shown in Figure 8, we observe that both SAND and GATE-Net exhibit the highest stability, maintaining consistent and stable performance among multiple trials. In contrast, SVM demonstrates significant instability, frequently overfitting to specific patterns, resulting in large fluctuations in accuracy. AdaTest and GATE-NET shows a more stable performance compared to SVM, but still lags behind SAND. Our framework achieves superior stability due to its combination of three core design choices: (1) the data processing step ensures balanced supervision and mitigates bias from imbalanced datasets; (2) the global clustering loss enforces consistent inter-class separation, reducing variance in decision boundaries; and (3) the NAS-selected architecture is tailored to maintain performance across varying circuit complexities, minimizing performance fluctuation during deployment.



Method	**Average Accuracy (%)**	**Standard Deviation**
SAND	95.5	1.2
GATE-Net	92.6	1.5
AdaTest	89.4	7.3
SVM	84.1	10.4

Figure 8: Detection accuracy comparison over 20 trials

5 Conclusion

Hardware Trojans (HTs) pose a critical threat to embedded systems. Existing ML-based HT detection methods suffer from reliance on ad-hoc feature engineering and limited adaptivity. To address this, this paper presents a self-supervised learning (SSL) and Neural Architecture Search (NAS) based framework for efficient HT detection. The proposed method integrates an upstream encoder trained with contrastive learning to automatically perform feature extraction, followed by an adaptive downstream classifier optimized by NAS for efficient deployment. Experimental results demonstrate that our approach significantly outperforms existing methods in detection accuracy, while maintaining decent adaptivity to unseen HT variants.

References

[1] Huili Chen et al. 2023. AdaTest: Reinforcement learning and adaptive sampling for on-chip hardware Trojan detection. *ACM TECS* 22, 2 (2023), 1–23.

[2] Jonathan Cruz et al. 2018. An automated configurable Trojan insertion framework for dynamic trust benchmarks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1598–1603.

[3] Matthew G Gaber, Mohiuddin Ahmed, and Helge Janicke. 2024. Malware detection with artificial intelligence: A systematic literature review. *Comput. Surveys* 56, 6 (2024), 1–33.

[4] Kevin Immanuel Gubbi et al. 2023. Hardware trojan detection using machine learning: A tutorial. *TECS* 22, 3 (2023), 1–26.

[5] M. Hansen, H. Yalcin, and J. P. Hayes. 1989. ISCAS Benchmarks. https://filebox.ece.vt.edu/~{}mhsiao/iscas89.html.

[6] Kento Hasegawa, Masao Yanagisawa, and Nozomu Togawa. 2017. A hardware-Trojan classification method using machine learning at gate-level netlists based on Trojan features. *IEICE* 100, 7 (2017), 1427–1438.

[7] Guyue Huang et al. 2021. Machine learning for electronic design automation: A survey. *TODAES* 26, 5 (2021), 1–46.

[8] Zhao Huang, Quan Wang, Yin Chen, and Xiaohong Jiang. 2020. A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access* 8 (2020), 10796–10826.

[9] Tatsuki Kurihara and Nozomu Togawa. 2021. Hardware-trojan classification based on the structure of trigger circuits utilizing random forests. In *IOLTS*. IEEE, 1–4.

[10] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 4765–4774.

[11] Adib Nahiyan et al. 2017. Hardware trojan detection through information flow security verification. In *2017 IEEE International Test Conference (ITC)*. IEEE, 1–10.

[12] Vladimir Nasteski. 2017. An overview of the supervised machine learning methods. *Horizons. b* 4, 51-62 (2017), 56.

[13] Chao Pan and Xin Yao. 2021. Neural architecture search based on evolutionary algorithms with fitness approximation. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[14] Zhixin Pan and Prabhat Mishra. 2021. Automated test generation for hardware trojan detection using reinforcement learning. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 408–413.

[15] Hassan Salmani et al. 2015. TrustHub.org: Trust-HUB,. http://trust-hub.org/benchmarks/trojan.

[16] Yao Shu, Zhongxiang Dai, Zhaoxuan Wu, and Bryan Kian Hsiang Low. 2022. Unifying and boosting gradient-based training-free neural architecture search. *Advances in neural information processing systems* 35 (2022), 33001–33015.

[17] Joseph Sweeney, Ruben Purdy, Ronald D Blanton, and Lawrence Pileggi. 2020. CircuitGraph: A Python package for Boolean circuits. *Journal of Open Source Software* 5, 56 (2020), 2646.

[18] Narayanan Vijaykrishnan and N Ranganathan. 1996. SUBGEN: A genetic approach for subcircuit extraction. In *Proceedings of 9th International Conference on VLSI Design*. IEEE, 343–345.

[19] Sheng Wan, Shirui Pan, Jian Yang, and Chen Gong. 2021. Contrastive and generative graph convolutional networks for graph-based semi-supervised learning. In *AAAI*, Vol. 35. 10049–10057.

[20] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. 2011. The risc-v instruction set manual, volume i: Base user-level isa. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62* 116 (2011), 1–32.

[21] Eyal Winter. 2002. The shapley value. *Handbook of game theory with economic applications* 3 (2002), 2025–2054.

[22] Hasini Witharana, Yangdi Lyu, and Prabhat Mishra. 2021. Directed test generation for activation of security assertions in rtl models. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 4 (2021), 1–28.

[23] Rozhin Yasaei et al. 2022. Golden reference-free hardware trojan localization using graph convolutional network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 10 (2022), 1401–1411.

[24] Shichao Yu, Chongyan Gu, Weiqiang Liu, and Máire O'Neill. 2021. Deep learning-based hardware trojan detection with block-based netlist information extraction. *IEEE Transactions on Emerging Topics in Computing* 10, 4 (2021), 1837–1853.

[25] Tong Zhang et al. 2021. AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning. *IEEE Transactions on Evolutionary Computation* 25, 5 (2021), 830–841.

[26] Er-Rui Zhou et al. 2016. A novel detection method for hardware trojan in third party ip cores. In *2016 International Conference on Information System and Artificial Intelligence (ISAI)*. IEEE, 528–532.