# Applying Neural Networks to Delay Fault Testing: Test Point Insertion and Random Circuit Training

Spencer K. Millican, Yang Sun, Soham Roy, and Vishwani D. Agrawal
Department of Electrical and Computer Engineering, Auburn University
341 War Eagle Way, Auburn, AL 36849-5201
millican@auburn.edu, yzs0057@auburn.edu, szr0075@auburn.edu, agrawvd@auburn.edu

**Abstract—This article presents methods of increasing logic built-in self-test (LBIST) delay fault coverage using artificial neural networks (ANNs) to selecting test point (TP) locations a method to train ANNs using randomly generated circuits. This method increases delay test quality both during and after manufacturing. This article also trains ANNs without relying on valuable third-party intellectual property (IP) circuits. Results show higher-quality TPs are selected in significantly reduced CPU time and third-party IP is not be required for ANN training.**

## I. Introduction

Many design-for-test (DFT) methods increase fault coverage and circuit reliability under pseudo-random stimuli, with one noteworthy method being test-points (TPs). Integrated circuits (IC) manufacturing is prone to defects [1] which cannot be tolerated in digital circuits, especially when such circuits are used in life-critical applications (e.g., self-driving cars and embedded medical devices). TPs are circuit modifications which are only enabled during test (and therefore do not change the function of the circuit outside of test) and allow more defects to be detected. Although TPs have many applications depending on context (e.g., analog TPs or TPs for easier ATPG), this article addresses TPs which improve pseudo-random tests: these TPs make pseudo-random stimuli (typically generated on-chip) more likely to detect random pattern-resistant (RPR) faults.

Choosing where to insert TPs, i.e. "TP insertion" (TPI), has a long history, but there are two modern challenges to address. First, design nuances unique to modern technologies are infrequently addressed in older TPI literature [2] and thus may insert less-than-ideal TPs: this article will show how previous TPI methods can mask the detection of delay faults. Second, TPI time must be improved. Optimal TPI (and many other DFT problems) is known to be NP-hard [3]: current TPI uses heuristics to overcome computational barriers, but established algorithms' CPU time scales worse than linearly with respect to circuit sizes, which is demonstrated by this article's data. This study performs TPI to increase delay fault coverage with significantly less CPU time compared to existing heuristics.

Artificial neural networks (ANNs) are computing methods which recently found success on previously unsolved problems, and their potential to be applied to DFT problems must be explored. ANNs were first introduced in the 1950's [4], but they were long neglected due to a lack of computational resources and training data. This was recently remedied, and ANNs have be demonstrated as effective methods to perform handwriting analysis, image classification, and other previously unsolvable problems. ANNs have recently been applied to EDA and DFT problems [5]–[8], both for increasing result quality and decreasing computational complexity. This work continues this trend and leverages ANNs to improve TPI quality and speed.

Obtaining ANN training data is difficult when data owners are unwilling to share. Training ANNs requires "training circuits", but circuit designers are reluctant to divulge their intellectual property (IP): leaked IP can be reproduced by malicious parties without compensation. Without training data (or with obsolete legacy circuits), ANN-training EDA companies will face difficulties training their ANNs for iterative TPI (or similar problems). This article explores generating training circuits and evaluates their ability to train ANNs compared to industrial benchmarks.

This article extends work performed in [7] and adds the following original contributions.

- A method using ANNs for iterative TPI increases delay fault coverage is presented and is compared against an equivalent heuristic algorithm.
- The computational complexity and scalability of using ANNs in lieu of established heuristics is analyzed.
- A potential source of unlimited training data is presented, and the effectiveness of its use is explored.

The remainder of this article is organized as follows. Section II presents previous literature which motivates this articles' contributions. Section III describes the ANN which evaluates a TP's impact on delay fault coverage, how it is trained, and how it is used in a TPI algorithm. Section IV introduces a method of generating training. Section V presents the experiments and results which evaluate this article's contributions. Section VI discusses the implications of results given in Section V and motivates future studies.

## II. Established Literature & Motivations

### A. *Test-points & delay fault coverage*

This article addresses TPs in the context of pseudo-random testing, which is a common testing method for modern circuits. Pseudo-random tests are typically generated on-chip using pseudo-random pattern generators (PRPGs), typically in the form of linear feedback shift registers (LFSRs) [9]. PRPGs apply stimuli to circuit inputs while circuit outputs are measured: outputs are either measured directly or are compacted into a signature using on-chip hardware (e.g., using a multiple-input signature register, or MISR [10]). PRPGs can also set and measure the state of scannable memory elements

2377-5386/19/$31.00 ©2019 IEEE
DOI 10.1109/ATS47505.2019.000-7

IEEE computer society

[11]. This method, often named logic built-in self-test (LBIST), is widely used in industry for several reasons. First, LBIST can obtain high fault coverage in small time and compliment ATPG tests, although certain circuit features can prevent this, which is addressed in the following paragraph. Second, LBIST tests are easily applied "in-the-field for post-manufacturing "reliability tests". This latter feature is desirable when continuous circuit reliability is paramount and is the focus of the emerging DFT sub-field of research, "automotive test".

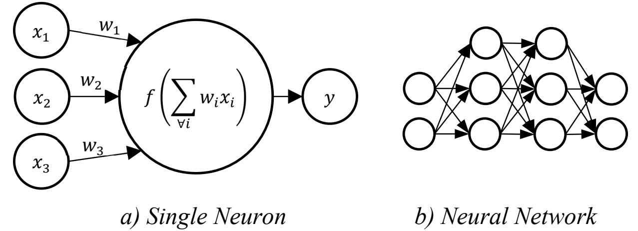
LBIST is widely-used in industry, but LBIST quality is known to suffer from RPR faults. RPR faults are tested by a small subset of all possible stimuli, and therefore the probability of detecting such faults under pseudo-random stimuli is negligible [9]. A prototypical example of an RPR fault is the output of a 32-bit AND gate stuck-at 0. This fault requires 32 logic-1's to be applied to the AND gate, which if all inputs are equally likely to be 0 or 1, the probability of this occurrence is $(50\%)^{32}$, i.e., negligible.

Many methods increase RPR faults coverage, with one method, TPI, being the subject of this article. In the context of LBIST, TPs are circuit modifications which fall into one of two categories: **1)** "control points" which force circuit lines to pre-determined values and **2)** "observe points" which route circuit lines to observable circuit outputs. TPs are controlled with a "TP enable" signal, which when disabled leaves the function of the circuit is unchanged. When TPs are inserted into key circuit locations, RPR faults become easy to excite and observe. In practice, TPs frequently compliment other methods of detecting RPR faults (e.g., deterministic seeding [12] and weighted random patterns [13]).

Although control points can increase stuck-at fault coverage, they are known to block other faults found in modern technologies [2]. Delay-causing defects become more common as semiconductor sizes scale [14], and the stuck-at fault model can no longer be the sole fault model for evaluating LBIST quality. [2] found typically-implemented control TPs (i.e., control-0 points using AND gates and control-1 points using OR gates [15]) mask the propagation and excitation of delay faults when active. Two alternatives can mitigate delay-masking detriments, but both have drawbacks. First, observe points can be used exclusively, but exciting RPR faults may require forcing values in circuits and implementing observe points require expensive latches or output pins. Second, circuits can be tested with TPs both on and off. This is normal industry practice, but disabling TPs mitigates their purpose: ideally, active TPs can test RPR and delay faults simultaneously. Therefore, this study's ANN has the following goal: insert TPs, both control and observe points, to detect RPR faults a without masking delay faults.

### B. *Artificial neural networks*

ANNs [4] are computer algorithms which are becoming popular due to growing training data availability. ANNs are models biological neural networks (i.e., brains) in software or hardware. ANNs excel at solving problems in which manually-developed "heuristic" algorithms falter [4]: they use problems with known solutions (i.e., "training data") and "train" themselves until correct/accurate results are achieved for each training problem. Historically, ANNs were impractical due the absence of training data [4], but today, ANNs are used in many industries.




*a) Single Neuron* *b) Neural Network*

*Figure 1: An example of a) a single neuron with input signals, input weights, and an activation function, and b) a neural network composed of multiple hidden layers.*

The process of creating and training ANNs can be found in other sources [4], [7] and is briefly summarized here.

**Creation (ANN structure Selection):** Many ANN structures exist in literature, but the prototypical ANN structure is illu strated in Figure 1. ANNs contain "neurons" with some neurons being input "feature" and output "label" neurons. Neurons are connected by "dendrites" which multiply ("weigh") neuron outputs. Neurons have "activation functions" which calculates neuron outputs using neuron inputs. Many activation function and neuron arrangements (the number of levels, neurons per level, connections, etc.) choices are available: these choices are optimized through trial-and-error.

**Training:** Training finds weights which matches inputs to desired outputs. Depending on the ANN configuration, the number of features and labels, and training data size, this can be computationally-intensive and is not an optimal process. Time permitting, this is repeated using different parameters to optimize weights and explore configuration choices.

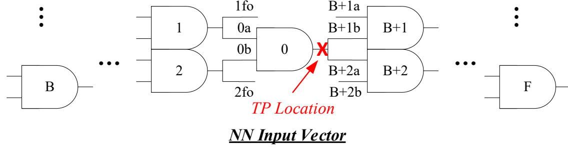
ANNs of various forms have recently been applied to DFT problems, including scan-chain diagnosis [5] and fault diagnosis [6], but using ANNs for TPI has not been thoroughly explored. Two studies have explored using ANNs for TPI [7], [8]. [8]'s scope is limited: only observe points are considered, only 4 benchmarks are analyzed, and no CPU time analysis is performed. Both [7] and [8] only consider stuck-at faults.

## III. ANN FOR TP DELAY FAULT QUALITY EVALUATION

### A. *Use in an iterative TPI algorithm*

This article's ANN "qualifies" TPs finds the "best" TP for use in a TPI algorithm. Most TPI algorithms iteratively insert TPs until a condition is met: either **1)** the number of TPs inserted meets a pre-designated limit (which represents a TP hardware overhead limit), **2)** the predicted fault coverage meets a pre-designated limit (i.e., no more TPs are needed), **3)** CPU time has reached a pre-designated limit, or **4)** no TP is predicted to increase fault coverage. Each iteration, the TP which is predicted to increase fault coverage the most is inserted. TPI algorithms calculate fault coverage in different ways, including fault simulation [15], probabilistic calculations [8], [16], and most recently ANNs [7], [8], but methods use the same iterative approach. All these methods attempt to balance calculating fault coverage accurately while keeping CPU time low.

In the context of this article, the "quality" of a TP is the predicted impact on delay fault coverage when activated. A TP's type and location can increase (or decrease) delay fault coverage significantly [2]. To the authors' knowledge, inserting increase delay fault coverage through TPI has little existing literature [17], [18]: [17] uses analog TPs to measure specific delay values, and [18] proposes a rudimentary heuristic to select




**NN Input Vector**

$CC_0, CC_{1fo}, CC_{0a}, CC_{0b}, CC_{2fo}, CC_1, CC_2, \ldots, CC_{Bb}, CC_{B+1a}, CC_{B+1b}, \ldots, CC_{B+F-1}, CC_{B+F},$
$CO_0, CO_{1fo}, CO_{0a}, CO_{0b}, CO_{2fo}, CO_1, CO_2, \ldots, CO_{Bb}, CO_{B+1a}, CO_{B+1b}, \ldots, CO_{B+F-1}, CO_{B+F},$
$Gate_0, Gate_1, Gate_2, \ldots, Gate_B, Gate_{B+1}, Gate_{B+2}, \ldots, Gate_{B+F}$

*Figure 3: The input features of the ANN are the controllability (CC) and observability (CO) values, as well as the gate types (Gate, represented as a one-hot encoding) of a sub-circuit centered around a TP's location. The more gates forward (F) and backwards (B) included in the sub-circuit, the more accurate the ANN will be at the expense of training time.*

TP locations. This article will use the TP-qualifying method from [17] for comparison (see Section V.A).

### B. Input features

Many input features and feature extraction methods of this article's ANN are adapted from [7] and briefly discussed here. First, since ANNs have pre-defined sizes and large ANNs require more training time and data to be useful, this article's ANN evaluates a TP's quality using a sub-circuit centered around a TP's location. Second, the ANN's input features are the probabilistic calculations of the lines in this sub-circuit. These calculations are COP controllability and observability values [19], which are widely used in TPI [16], [17]: these values estimate the probability that a circuit line is logic-1 (and inversely, logic-0) and the probability that a fault on a circuit line will be observed, respectively. Third, the gate types in sub-circuits are represented as one-hot binary strings. During TPI and training data generation, these features are calculated and passed to the ANN in a set order, as illustrated in Figure 2: first controllability values, then observability values, then gate types, all of which are given in a breadth-first order [7].

### C. Output labels

The output label of the ANN is the change in transition delay fault (TDF) coverage on a sub-circuit centered around the TP location (when the TP is active). Many delay fault models attempt to accurately model delay-causing defects [20], and the TDF model is known to model realistic defects.

The use of a sub-circuit allows training data to be efficiently collected (see Section III.D), but this creates a potential detriment: TP quality-calculating heuristics can calculate the effect a TP has on an entire circuit (as is the case of this article's comparison, [17]), and therefore may return more accurate results. The effect of this detriment will be explored and discussed in Sections V and VI, respectively.

### D. Training data generation

Training this article's ANN's requires the "true" TDF coverage of a sub-circuit when inserting an active TP: this is obtained through fault simulation. Fault simulation is a CPU-intensive process, which is why most heuristic TPI methods use estimation techniques [16], [17]. Such estimations can give inaccurate fault coverage results [19], and thus less effective TPs may be chosen for insertion. This article's ANN attempts to accurately predict the TDF coverage impact of a TP without performing fault simulation during TPI. Instead, fault

**Algorithm**: DAG Generator
**Inputs:** positive integers $I$, $O_{min}$, $R$, $N$, $f_{max}$
**Variables:** set of vertices in rank $S(r)$, flattened set of all vertices $M$, current set of vertices $S$, current rank $r$, current set of fan-ins $F$
**for** $r = 0 \ldots R$ **do**
  **if** $r = 0$
    $S \leftarrow \{ set\ of\ I$ vertices$\}$
  **else if** $r = R$
    $S \leftarrow \{ set\ of\ O$ vertices$\}$
  **else**
    $S \leftarrow \{ set\ of\ \frac{N-O}{R}$ vertices.$\}$
  **end if**
  **if** $r \neq 1$
    **for** $\forall s \in S$ **do**
      $F = \{random\ set\ of\ size\ 1 \ldots f_{max}\ from\ M\}$
      Set $F$ as inputs to $s$
    **end for**
  **end if**
  $S(r) \leftarrow S$
  $M \leftarrow \{M, S\}$
**end for**
**return** $\boldsymbol{M}$

*Figure 2: Random DAGs generated by this algorithm, given in the form of vertices and edges, can be converted into logic circuits.*

simulation is performed during training: this training CPU time is distributed across all TPI instances.

To obtain the training label (TDF coverage) from a sub-circuit, techniques are needed to reduce training data generation time. Presuming $N$ pseudo-random vector pairs (pairs are required for delay simulation) are applied to a sub-circuit with $I$ inputs, at least one vector pair will be applied more than once if $V > 2^{2 \cdot I}$, which is likely for significant values of $V$ and smaller sub-circuit sizes (which for this study, $I \leq 6$). Fault simulating a vector pair more than once wastes simulation time. This is exacerbated when sub-circuits are extracted from a circuit, since input vectors to the sub-circuit are not random: paths from circuit inputs to sub-circuit inputs will "weigh" circuit value probabilities and will make some vectors more likely to occur than others.

To prevent training vector pairs being applied more than once, the following technique uses heuristically-calculated circuit controllability calculations to conditionally simulate all vector pairs. First, for each vector $v$ among the $2^I$ sub-circuit input vectors, the probability of applying the vector is calculated using COP circuit controllability information [19]. The probability of the required logic-0 and logic-1 values occurring ("$C_i'$") can calculate this probability:

$$p(vector\ simulated) = p_v = \prod_{\forall i \in I} (1 - (1 - C_i')^V)$$

Second, a sub-set of all the $2^{2 \cdot I}$ sub-circuit vector pairs is chosen to be simulated using the previously calculated probabilities, i.e., a vector pair is simulated with probability $p_{v_1} \cdot p_{v_2}$. Third, each sub-circuit vector pair is fault simulated $T + 1$ times: once without any TPs and once for each of the $T$ TP types (in this study, $T = 3$: control-0, control-1, and observe points). Fault effects on output pins are probabilistically observed using observability calculations from COP [19]. Lastly, the controllability, observability, and gate types of the sub-circuit are used for training features and the change in fault coverage created by the $T$ TPs are used as training labels for $T$ separate ANNs.

TABLE 1
TRAINING BENCHMARKS

| Bench. | Training Samples | Inputs | Outputs | Gates |
|---|---|---|---|---|
| c17 | 2 | 5 | 2 | 13 |
| b02 | 2 | 5 | 5 | 32 |
| b06 | 2 | 11 | 15 | 65 |
| b08 | 8 | 30 | 25 | 204 |
| b10 | 9 | 28 | 23 | 223 |
| c499 | 12 | 41 | 32 | 275 |
| c1355 | 24 | 41 | 32 | 619 |
| b04 | 32 | 77 | 74 | 803 |
| b12 | 51 | 126 | 127 | 1197 |
| c2670 | 49 | 233 | 140 | 1566 |
| c6288 | 100 | 32 | 32 | 2480 |
| c7552 | 143 | 207 | 108 | 3827 |
| b15 | 445 | 485 | 519 | 9371 |
| b14 | 471 | 277 | 299 | 10343 |
| b20 | 950 | 522 | 512 | 20716 |
| b21 | 980 | 522 | 512 | 21061 |
| b22 | 1443 | 767 | 757 | 30686 |
| b17 | 1562 | 1452 | 1512 | 33741 |
| b18 | 5276 | 3357 | 3343 | 117941 |

## IV. RANDOM TRAINING DATA GENERATION

This section attempts to overcome a challenge of ANN training: the lack of training data. As discussed in Section 0, circuit developers are reluctant to share their circuits to ANN trainers since their IP will lose value if leaked to unauthorized users. Since ANNs may not be trained with industry-representative circuits if IP is protected (and training with obsolete benchmarks may degrade ANN quality), this section introduces a method to create training circuits, and its utility will be explored in Sections V and VI.

This circuit-generating algorithm creates directed acyclic graphs (DAGs), since DAGs can easily be converted into circuits. DAGs have directional edges and no path exists from any vertex to itself. A given DAG can be converted into a random binary circuit by **1)** replacing all vertices with no inputs with circuit inputs, **2)** replacing all other vertices with a randomly selected gate (or if it has 1 input, a buffer or inverter), and **3)** replacing dead-end vertices with circuit outputs.

The algorithm to generate DAGs (which are converted into circuits) is given in Figure 3. The inputs to this algorithm are **1)** the number of circuit inputs to create ($I$), **2)** the minimum number of circuit outputs to create ($O_{min}$), **3)** the maximum number of circuit levels ($R$), **4)** the maximum number of fan-ins per gate ($f_{max}$ ) and **5)** the desired number of nodes to create ($N$). The algorithm creates "ranks" of vertices: in the first rank, no vertex has inputs, in the last rank, no vertex has outputs. Every vertex in a rank $r$ is connected randomly to vertices in previous ranks. By forcing the first rank to have $I$ vertices, the last to have $O_{max}$, and all others to have $(N-O)/R$, this guarantees the circuit will have the correct number of inputs and a minimum number of outputs.

It must be discovered if ANNs trained using circuits generated in this manner will yield useful ANNs, which will be explored in future sections. This method can generate infinite combinations of circuits, but it is not known if the circuits will have difficult-to-test structures which are found in industrial circuits. It is possible that if ANNs are trained on the randomly-generated circuits the ANN quality will be unacceptably low, but this will be discovered by comparing an ANN trained with

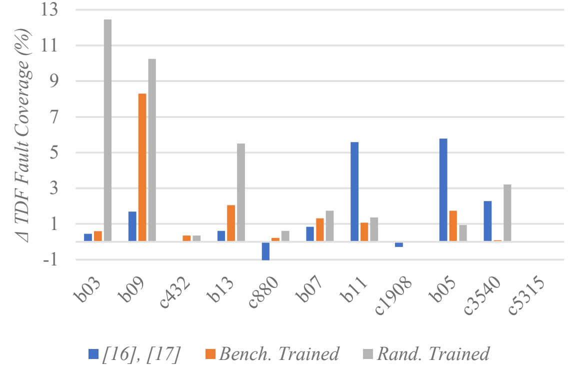



*Figure 4: This plot of results from Table 2 illustrates the relative changes in TDF coverage for three TPI methods: heuristic, benchmark-trained, and randomly-trained.*

"real" circuits (i.e., industrial benchmarks) against an ANN trained with random circuits of similar characteristics (similar numbers of inputs, outputs, circuit levels, and nodes). This is performed in Section V.

## V. EXPERIMENTS & RESULTS

### A. Experimental setup

All experiments are performed on industry-representative workstations: these perform fault simulation and TPI using software written explicitly for this study. The workstations use Intel i7-8700 processors and possess 8 GBs of RAM, and all software was written in C++ and compiled using the MSVC++14.15 compiler with maximum optimization parameters. Original software was used in lieu of industry tools to make a fair comparison of the proposed ANN against a method from literature: only code which analyzes the "quality" of a TP is changed between the method proposed method and the comparison method, thereby minimizing other sources of CPU time and fault coverage differences.

The comparison TPI method is a modified version of [16]. [16] iteratively adds the "best" TP (with no favoritism between control and observe TPs), which the authors believe is representative of TPI implemented in industry, although industrial tools will have additional computation-time optimizations. Conveniently, this article's ANN can directly replace the "quality" measuring method in [16], which eliminates other sources of fault coverage and CPU time differences. However, the method in [16] does not attempt to insert TPs to increase delay fault coverage. To correct this, the "quality" calculation to judge the detection probability of a stuck-at fault (the controllability of a line, multiplied by the observability of the line) is directly replaced by the calculation proposed in [17] which attempts to model the probability a transition will occur on a line and be observed at a circuit output (the controllability of a line, multiplied by the inverse of this controllability, multiplied by the observability of the line). This TPI method will attempts to increase delay fault coverage while using the same information as the proposed ANN. Therefore, the only difference in run-time and fault coverage quality will be in judging the "quality" of a given TP.

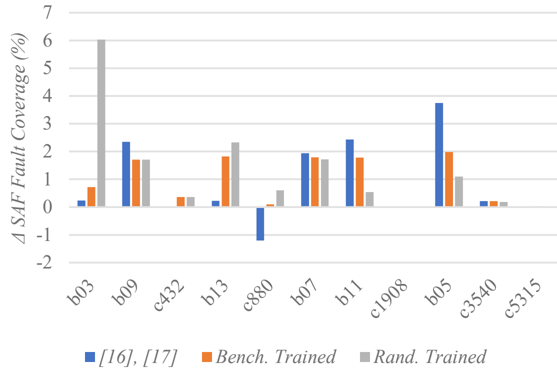



*Figure 5: This plot of results from Table 2 illustrates the relative changes in SAF coverage for three TPI methods: heuristic, benchmark-trained, and randomly-trained.*

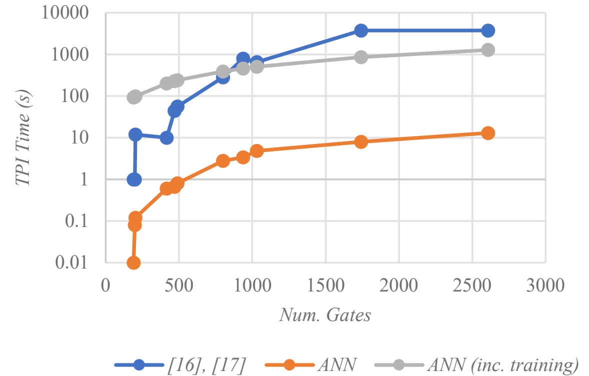



*Figure 6: This plot of results from Table 2 illustrates CPU TPI time for three: heuristic and ANN-based TPI (with and without training time).*

This study uses post-synthesis logic netlists of the ISCAS'85 [21] and the ITC'99 [22] benchmarks. A randomly selected subset, given in Table 1, is used solely for training: this table provides the number of sub-circuits (i.e., TP locations) extracted for training (the number of extracted sub-circuits is proportional to the circuit size).

It is presumed circuits are tested in a full-scan environment with scan chains loaded from a 31-bit long PRPG and delay tests are performed using "launch-of-shift", hence "inputs" and "outputs" in Tables 1 and 2 include flip-flops outputs and inputs, respectively. When TPs are present, control points are enabled by a common "TP enable" pin which is active for half of all vectors applied. Although other methods to selectively enable sub-sets of TPs exist in literature [23] such architectures are not the scope of this study.

After many trials of ANN training, the final ANN configuration chosen was 1 hidden neuron layer of 128 neurons, with each hidden neuron using a sigmoid [24] activation function. Sub-circuit sizes are 2 gates forward and backwards of a TP location (i.e., $B = 2$ and $F = 2$ from Figure 2).

When performing TPI and fault simulation, the number of vectors applied (and used for TPI calculation) is individualized for each benchmark and is given in the column labeled "Vec." This value is calculated based on projections given in [25]: fault simulation with random vectors is performed on TP-free circuits until 63.2% stuck-at fault coverage is obtained, and this number of random vectors is then used to project the number of random vectors needed to obtain 95% stuck-at fault coverage: this number or ten thousand is used, whichever is lesser. This represents an industrial environment where either **1)** TPs are intended to increase stuck-at fault coverage as much as possible after 95% stuck-at fault coverage is achieved, or **2)** 95% stuck-at fault coverage is not obtainable in ten thousand vectors and TPs are inserted to assist in obtaining 95% stuck-at fault coverage.

### B. *Delay fault coverage of a benchmark-trained ANN.*

The first experiment examines the delay fault coverage obtained when TPs are inserted using a conventional TPI method (described in Section V.A) compared to using this article's ANN, and the results of this experiment are given in Table 2. These benchmarks were not used for ANN training to prevent giving an advantage to the ANN. TPs are inserted using the ANN until either **1)** no TP is calculated to increase fault coverage, **2)** 99% fault coverage is predicted to be obtained, or **3)** the number of TPs is greater than 1% of all nodes. Afterwards, traditional TPI is performed until the same number of TPs are inserted, with the number TPs given under the heading "TPs".

Fault coverages are given in Table 2 and changes in fault coverage are plotted in Figures 4 and 5, which shows notable trends. First, inserting TPs using the proposed method ("Bench. Trained") never decreases TDF or SAF coverage, while the conventional method ("[16], [17]") can (see c880 and c1908). Additionally, this article's ANN-based method achieves comparable TDF and stuck-at fault coverage compared to the heuristic method. This trend will motivate future studies.

### C. *Time to perform TPI*

An additional result extracted from the previous experiment is the time required to perform TPI. This is given in Table 2 under the heading "TPI Time (s)" for the two TPI methods, which shows this article's ANN substantially decreases TPI time, and this remains true when training data generation and ANN training is considered. These results are transposed to Figure 6, which plots TPI CPU time relative to the number of logic gates in each circuit. When the training data generation time (3,544 seconds) and ANN training time (864 seconds, which includes exploring multiple ANN configurations) is distributed among benchmarks by circuit size (i.e., more time is added to circuits with more logic gates), ANN-based TPI shows unfavorable time for small circuits, but for large (i.e., circuits where TPI is most needed), the CPU time is substantially faster.

### D. *Random benchmark training*

The final experiment performs TPI using an ANN, except this ANN is trained using randomly generated circuits. The random circuits have the same number of inputs and nodes, and a comparable number of outputs and levels as the benchmarks used to train the ANN in the first experiment, as given in Table 1. These random circuits may contain marginally more outputs and circuit levels, but this value is always within 1% of the original benchmarks. The same number of sub-circuits are also extracted for training. TPI is performed using the same limits as the first experiment and the same number of TPs are inserted.

TABLE 2
EXPERIMENT RESULTS

| | | | | | | Fault Coverage, TDF (%) | | | | Fault Coverage, SAF (%) | | | | TPI Time (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bench. | In. | Out. | Gates | Vec. | TPs | No TPs | [16], [17] | Bench. Trained | Rand. Trained | No TPs | [16], [17] | Bench. Trained | Rand. Trained | [16], [17] | ANN | ANN (inc. training) |
| b03 | 34 | 34 | 190 | 216 | 1 | 78.31 | 78.76 | 78.91 | 90.77 | 92.14 | 92.38 | 92.86 | 98.17 | 1.00 | 0.01 | 92.20 |
| b09 | 29 | 29 | 198 | 216 | 1 | 66.00 | 67.70 | 74.31 | 76.25 | 86.85 | 89.20 | 88.56 | 88.56 | 1.00 | 0.08 | 96.15 |
| c432 | 36 | 7 | 203 | 5832 | 2 | 95.49 | 95.49 | 95.83 | 95.83 | 98.71 | 98.71 | 99.08 | 99.08 | 12.00 | 0.12 | 98.61 |
| b13 | 63 | 63 | 415 | 512 | 6 | 81.19 | 81.80 | 83.24 | 86.70 | 94.11 | 94.34 | 95.93 | 96.44 | 10.00 | 0.60 | 201.96 |
| c880 | 60 | 26 | 469 | 1331 | 4 | 93.35 | 92.32 | 93.58 | 93.97 | 98.29 | 97.08 | 98.39 | 98.89 | 44.00 | 0.66 | 228.22 |
| b07 | 50 | 57 | 490 | 512 | 4 | 77.10 | 77.94 | 78.42 | 78.84 | 85.89 | 87.83 | 87.68 | 87.61 | 56.00 | 0.80 | 238.55 |
| b11 | 38 | 37 | 801 | 10000 | 8 | 86.52 | 92.10 | 87.60 | 87.89 | 94.15 | 96.58 | 95.93 | 94.69 | 283.00 | 2.80 | 391.44 |
| c1908 | 33 | 25 | 938 | 10000 | 9 | 98.85 | 98.56 | 98.85 | 98.84 | 99.66 | 99.66 | 99.66 | 99.66 | 794.00 | 3.36 | 458.47 |
| b05 | 35 | 70 | 1032 | 10000 | 10 | 70.08 | 75.86 | 71.82 | 71.02 | 76.23 | 79.98 | 78.21 | 77.33 | 649.00 | 4.80 | 505.52 |
| c3540 | 50 | 22 | 1741 | 10000 | 12 | 89.77 | 92.06 | 89.85 | 92.98 | 95.39 | 95.61 | 95.61 | 95.58 | 3746.00 | 8.00 | 852.73 |
| c5315 | 178 | 123 | 2608 | 6859 | 13 | 97.50 | 97.52 | 97.54 | 97.52 | 98.97 | 98.95 | 99.00 | 98.97 | 3746.00 | 13.00 | 1278.39 |

Randomly-trained ANN results from Table 2 are plotted in Figures 4 and 5 ("Rand. Trained"), which show several trends. First, the impact on TDF coverage compared to the benchmark-trained ANN is considerably higher, except for one (b05). Second, stuck-at fault coverage is decreased compared to the benchmark-trained ANN for several benchmarks, but it still obtains a consistent increase in SAF coverage compared to heuristic-based TPI.

## VI. DISCUSSIONS & FUTURE DIRECTIONS

Results show the ANN obtains results comparable to conventional heuristics, but also obtains results in significantly less time. Given a challenge of TPI (and other EDA problems) is managing and sharing computational resources, reducing TPI time without reducing TP quality is beneficial to circuit designers. Future studies will focus on **1)** finding features and training methods which further increase TP quality and **2)** inserting TPs under additional constraints, especially power and delay [26]. Also, when LBIST is applied at-speed, it is possible scanned-in vectors can excite false timing paths [27], which can create false failures. The authors will explore if ANNs can select test points which prevent such false failures.

Using ad hoc random circuits showed great potential. The ability to increase TDF coverage as opposed to stuck-at fault coverage is most interesting, as it implies the correlation of stuck-at behavior to designed circuit functions is stronger than equivalent transition behavior. The authors are interested in studying different random circuit generation methods which better model functional logic circuits and finding the impact these circuits have on ANN straining.

## VII. REFERENCES


[1] I. Koren and Z. Koren, "Defect Tolerance in VLSI Circuits: Techniques and Yield Analysis," *Proc. IEEE*, vol. 86, no. 9, pp. 1819–1838, Sep. 1998.

[2] S. Roy, B. Stiene, S. Millican, and V. Agrawal, "Improved Random Pattern Delay Fault Coverage Using Inversion Test Points," in *Proc. IEEE 28th North Atlantic Test Workshop (NATW)*, Essex, VT, 2019.

[3] B. Krishnamurthy, "A Dynamic Programming Approach to the Test Point Insertion Problem," in *Proc. 24th ACM/IEEE Design Automation Conference*, New York, NY, USA, 1987, pp. 695–705.

[4] S. O. Haykin, *Neural Networks and Learning Machines*, 3rd ed. New York: Pearson, 2008.

[5] M. Chern *et al.*, "Improving Scan Chain Diagnostic Accuracy Using Multi-stage Artificial Neural Networks," in *Proc. 24th Asia and South Pacific Design Automation Conference*, New York, NY, USA, 2019, pp. 341–346.

[6] L. R. Gómez and H. Wunderlich, "A Neural-Network-Based Fault Classifier," in *Proc. 25th IEEE Asian Test Symposium (ATS)*, 2016, pp. 144–149.

[7] Y. Sun and S. Millican, "Test Point Insertion Using Artificial Neural Networks," in *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Miami, FL, 2019.

[8] Y. Ma *et al.*, "High Performance Graph Convolutional Networks with Applications in Testability Analysis," in *Proc. 56th Annual Design Automation Conference (DAC)*, New York, NY, USA, 2019, pp. 18:1–18:6.

[9] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*. New York, NY, USA: Wiley-Interscience, 1987.

[10] R. David, "Signature Analysis for Multiple-Output Circuits," *IEEE Trans. Comput.*, vol. C–35, no. 9, pp. 830–837, Sep. 1986.

[11] P. H. Bardell and W. H. McAnney, "Self-Testing of Multichip Logic Modules," in *Proc. International Test Conference (ITC)*, 1982, pp. 200–204.

[12] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-polynomial Linear Feedback Shift Registers," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 223–233, Feb. 1995.

[13] S. Ghosh, E. MacDonald, S. Basu, and N. A. Touba, "Low-power Weighted Pseudo-random BIST Using Special Scan Cells," in *Proc. 14th ACM Great Lakes Symposium on VLSI (GLSVLSI)*, New York, NY, USA, 2004, pp. 86–91.

[14] P. Nigh and A. Gattiker, "Test Method Evaluation Experiments and Data," in *Proc. International Test Conference (ITC)*, 2000, pp. 454–463.

[15] N. A. Touba and E. J. McCluskey, "Test Point Insertion Based on Path Tracing," in *Proc. VLSI Test Symposium*, 1996, pp. 2–8.

[16] H.-C. Tsai, K.-T. Cheng, C.-J. Lin, and S. Bhawmik, "A Hybrid Algorithm for Test Point Selection for Scan-based BIST," in *Proc. 34th Design Automation Conference (DAC)*, 1997, pp. 478–483.

[17] S. Ghosh, S. Bhunia, A. Raychowdhury, and K. Roy, "A Novel Delay Fault Testing Methodology Using Low-Overhead Built-In Delay Sensor," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2934–2943, Dec. 2006.

[18] I. Pomeranz and S. M. Reddy, "Design-for-testability for Path Delay Faults in Large Combinational Circuits Using Test Points," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 4, pp. 333–343, Apr. 1998.

[19] F. Brglez, "On Testability Analysis of Combinational Networks," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 1984, vol. 1.

[20] J. Mahmod, S. Millican, U. Guin, and V. Agrawal, "Delay Fault Testing: Present and Future," in *Proc. IEEE VLSI Test Symposium (VTS)*, Monterey, CA, 2019.

[21] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN," in *Proc. IEEE International Symposium on Circuits and Systems*, 1985.

[22] S. Davidson, "ITC'99 Benchmark Circuits - Preliminary Results," in *Proc. IEEE International Test Conference*, 1999, pp. 1125–1125.

[23] N. Tamarapalli and J. Rajski, "Constructive Multi-phase Test Point Insertion for Scan-based BIST," in *Proc. IEEE International Test Conference*, 1996, pp. 649–658.

[24] A. Menon, K. Mehrotra, C. K. Mohan, and S. Ranka, "Characterization of a Class of Sigmoid Functions with Applications to Neural Networks," *Neural Netw.*, vol. 9, no. 5, pp. 819–835, Jul. 1996.

[25] T. W. Williams, "Test Length in a Self-Testing Environment," *IEEE Des. Test Comput.*, vol. 2, no. 2, pp. 59–63, Apr. 1985.

[26] M. Nakao, S. Kobayashi, K. Hatayama, K. Iijima, and S. Terada, "Low Overhead Test Point Insertion for Scan-based BIST," in *Proc. IEEE International Test Conference*, 1999, pp. 348–357.

[27] F. P. Higgins and R. Srinivasan, "BSM2: next generation boundary-scan master," in *Proc. 18th IEEE VLSI Test Symposium (VTS)*, Montreal, Quebec, Canada, 2000, pp. 67–72.