2015 IEEE 33rd VLSI Test Symposium (VTS)

Improving Accuracy of On-chip Diagnosis via Incremental Learning

Xuanle Ren, Mitchell Martin and R. D. (Shawn) Blanton
Department of Electrical and Computer Engineering, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA, USA
{xuanler, mitchel1, blanton}@ece.cmu.edu

***Abstract*—On-chip test/diagnosis is proposed to be an effective method to ensure the lifetime reliability of integrated systems. In order to manage the complexity of such an approach, an integrated system is partitioned into multiple modules where each module can be periodically tested, diagnosed and repaired if necessary. The limitation of on-chip memory and computing capability, coupled with the inherent uncertainty in diagnosis, causes the occurrence of misdiagnoses. To address this challenge, a novel incremental-learning algorithm, namely dynamic k-nearest-neighbor (DKNN), is developed to improve the accuracy of on-chip diagnosis. Different from the conventional KNN, DKNN employs online diagnosis data to update the learned classifier so that the classifier can keep evolving as new diagnosis data becomes available. Incorporating online diagnosis data enables tracking of the fault distribution and thus improves diagnostic accuracy. Experiments using various benchmark circuits (e.g., the cache controller from the OpenSPARC T2 processor design) demonstrate that diagnostic accuracy can be more than doubled.**

***Key words*—On-chip diagnosis, k-nearest-neighbor, machine learning, diagnostic accuracy, lifetime reliability**

I. Introduction

Ensuring the lifetime reliability of integrated systems has become a central concern. A robust system should be able to continue acceptable operations over its intended lifetime even in the presence of failures [1]. Although manufacturing tests are performed to help ensure reliability, a chip may still degrade and even fail in the field due to various locations of failure. Early-life failure, also called infant mortality, is caused by the defects that are not exposed during manufacturing tests. However, electrical and thermal stress during in-field use will eventually degrade the defect to a significant failure in functionality [2], [3]. Wear-out, also called aging, manifesting as progressive performance degradation, is induced by various mechanisms, e.g., negative-bias temperature instability (NBTI) and hot-carrier injection (HCI) [4].

Various methods have been proposed to detect and avoid failures. First, forward error control (FEC) method uses error correction codes (ECC) to detect and correct data faults during transmission by adding data redundancy to the packets [5]–[8]. However, FEC does not target permanent failures, e.g, early-life and wear-out [9]. On the other hand, most error detection/correction methods incur significant power, performance and area penalties [10]. Second, failure-prediction schemes provide an early warning of circuit aging before errors appear. Specifically, an aging sensor periodically checks the slack of a critical path in order to avoid the occurrence of delay faults. This approach requires an aging sensor and a stability checker for each flip-flop which however incurs a large area overhead [11]. Third, on-chip self-test schemes test the system periodically for failure detection [10], [12]–[14]. To reduce the overhead, the DFT already on chip for manufacturing test is reused to perform the in-field testing based on vectors stored in off-chip flash memory.

To enable failure localization, diagnosis is performed when on-chip self-test detects a failure. There are generally two diagnostic approaches, namely, effect-cause and cause-effect [15]. In effect-cause, a complete model of the design, its test-vector set, and the test response from the failing circuit are analyzed by diagnostic software to identify possible fault locations. However, it requires both significant memory and runtime. Compared to effect-cause, cause-effect is more feasible. However, it too has challenges since it requires the generation, storage and usage of a fault dictionary that contains the test response for every fault of interest. The size of a full fault dictionary is measured in terabytes for modern designs [15]. In [14], an on-chip diagnosis scheme that performs diagnosis and repair however at the module level (i.e., sub-core/sub-uncore) results in a much more compact dictionary. Instead of storing the simulation response of each fault per test, the compact fault dictionary only stores a single bit per test that indicates the pass/fail status of a subset of faults that enable module-level diagnosis. Such an approach significantly reduces the size of a fault dictionary at the cost of diagnostic resolution (i.e., one module is assumed to be faulty, but multiple modules are diagnosed as fault candidates). After the faulty module is located, self-repair is performed to replace or bypass it [16]. However, if the diagnostic resolution is non-ideal and no further analysis is done to narrow down the fault location, all fault candidates have to be repaired, resulting in inefficient use of on-chip resources.

In this paper, a novel incremental-learning algorithm that we call dynamic k-nearest-neighbor (DKNN) is proposed to improve the accuracy of on-chip, module-level diagnosis. Here, accuracy is defined as the probability that the identified module is the one with the failure, assuming that a single module is faulty. It is assumed that on-chip testing is performed with a test clock that has a higher frequency than the system clock because it allows failure sources that slowly degrade system timing to be tracked over time. For example, delay

978-1-4799-7597-6/15/$31.00 ©2015 IEEE

IEEE computer society

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.

degradation due to NBTI [4] can be monitored, enabling system adjustments (e.g., task scheduling, sleep scheduling, etc.) that mitigate adverse effects on system lifetime. Another consequence of using a faster clock for test, means failures will be much more frequent, thus creating sufficient data for learning a model for improving diagnostic accuracy. Different from the conventional KNN, DKNN employs online data to update the learned classifier, enabling the classifier to evolve as new data becomes available. Consequently, DKNN is able to track the fault distribution, especially when the fault distribution is non-stationary[1]. In [17]–[19], pattern recognition methods are used to make diagnostic decisions in analog circuits. In particular, possible circuit defects are identified through inductive fault analysis. Then a set of classifiers, trained offline using data from fault simulation, is used to map each defect to a score according to its likelihood of occurrence. In [20], an incremental KNN algorithm that also revises the composition of data set by exploiting a "correct-error" teacher is proposed. In their approach, each instance in the data set is associated with a dynamically-evolving weight, that requires additional overhead if implemented on chip. Moreover, the memory required increases as more data is collected, making it difficult to determine the required amount of memory a priori. Compared to the incremental KNN in [20], DKNN is more hardware-friendly because it maintains a fixed-size data set, and only requires little additional logic for data replacement.

Finally, the effectiveness of DKNN is validated using two benchmark circuits, L2B (the L2 cache bank controller of the OpenSPARC T2 processor [21]) and c7552 (an ISCAS benchmark circuit [22]), details of which will be elaborated upon in Section IV.

This work has two main contributions:

- DKNN copes with non-stationary distributed data by updating the classifier incrementally using online data.
- DKNN can be implemented on chip for improving diagnostic accuracy, using little additional logic and a fixed-size data set.

The rest of this paper is organized as follows. Section II provides details of the DKNN algorithm and demonstrates how it improves the accuracy of on-chip diagnosis. Section III presents the on-chip implementation of DKNN while Section IV evaluates the performance using the benchmark circuits and the UCI repository databases [23]. Section V draws conclusions.

II. Methodology

The flow for improving the accuracy of on-chip diagnosis is depicted in Figure 1. On-chip test/diagnosis generates a coarse diagnosis result which is then refined, if necessary, by the DKNN classifier.

A. Module-level Diagnosis

For the purpose of on-chip test/diagnosis, the core/uncore to be tested is partitioned into a set of interconnected modules,

[1] If the probability that a fault is located in a specific module changes over time, then the fault distribution is non-stationary; otherwise, it is stationary.

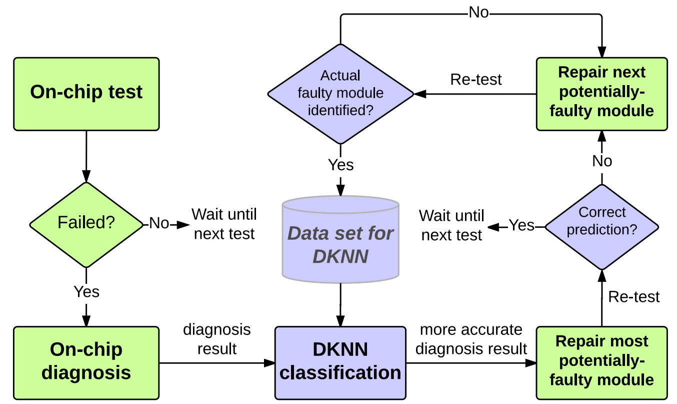

Figure 1: An on-chip diagnosis result is fed into the on-chip DKNN classifier. The module predicted by the DKNN classifier is repaired. If the predicted label proved to be incorrect, the DKNN data set is updated.

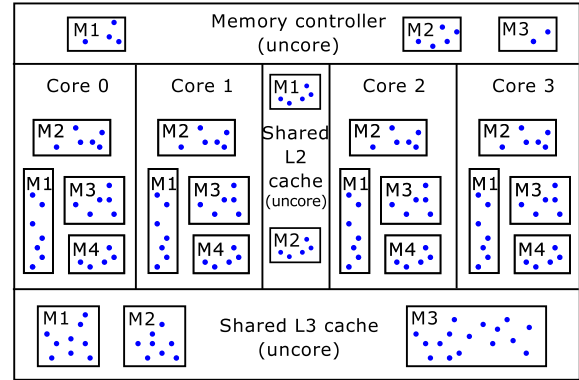

Figure 2: A processor with four cores and several uncores is shown hierarchically. Each core/uncore is partitioned into multiple modules, each of which is assumed to be independently repairable. Within each module lies a number of faults (shown as blue dots) that allow faulty modules to be distinguished via testing [14].

each of which is independently repairable [14]. Figure 2 shows a fictional multicore processor that contains four cores and several uncores. Uncores are defined as system components that are neither processor cores nor co-processors (e.g., memory controller, interrupt handlers, etc.) [13]. In Figure 2, each core/uncore is partitioned into multiple modules, and particular faults identified within each module capture those early-life failure and wear-out locations that allow faulty modules to be distinguished through diagnosis.

Periodic on-chip test/diagnosis starts with testing a core/uncore using a method such as the CASP (Concurrent Autonomous chip self-test using Stored test Patterns) technique [12]. If the core/uncore is faulty, then diagnosis is initiated to locate the faulty module. Specifically, diagnosis generates a list of potentially-responsible delay faults on a per module basis [14]. Further, counting the number of potentially-responsible

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.

delay faults for each module provides an indication of the likelihood that the corresponding module is indeed the location of failure. Predicting the faulty module to be the one with the maximum fault-count is called "select-max".

Table 1 shows three diagnosis results for L2B through simulation. Each simulation first injects a delay fault (which mimics early-life failure or wear-out) to a specific gate, and then runs diagnosis that results in a fault-count associated with each module. The first diagnosis result has ideal resolution, i.e., only one module has non-zero fault-count. Thus, M_9 is most likely to be faulty. However, most diagnosis results do not have ideal resolution. The second diagnosis result has three modules associated with non-zero fault-counts, either of which may be the faulty one. In this case, M_5 is deemed to be faulty using the select-max strategy. The third diagnosis result has nine modules associated with non-zero fault-counts. The select-max strategy would deem M_7 as the faulty module but the actual faulty module is M_9.

The examples from Table 1 demonstrate that select-max is not always correct. Based on the simulation results (described in Section IV), the diagnostic accuracy of select-max is 71% for L2B, and 20% for c7552. Here, a diagnostic outcome is deemed accurate if the module with the maximum fault-count is indeed the location of the failure. If select-max is incorrect (i.e., the wrong repaired), the benefit of on-chip test/diagnosis is hindered. Thus, improving diagnostic accuracy is a critical goal of on-chip test/diagnosis.

B. Dynamic k-nearest-neighbor

k-nearest-neighbor (KNN) is an instance-based non-parametric machine-learning algorithm used for classification. Specifically, an unlabeled instance[2] is classified based on the class of its k nearest neighbors [24]. KNN classification depends on the local-similarity nature of data, i.e., two instances belonging to the same class are supposed to be close to each other in the hyperspace that captures the data. This assumption is also applicable to module-level diagnosis. Figure 3 provides an example illustrating that failures in different modules may result in different diagnosis results based on how the modules are interconnected. For example, Figure 3(a) shows two modules of a core/uncore, namely M_1 and M_2, that are connected to each other. The failure effects stemming from M_1 can propagate to M_2, but the converse is obviously not true. Thus, as shown in Figure 3(b), a failure in M_1 (red dots) may cause both M_1 and M_2 to report non-zero fault-counts, but a failure in M_2 (blue dots) cannot cause M_1 to report a non-zero fault-count. In this case, it is easy to find the classification boundary in the two-dimensional space between the faults residing in M_1 and M_2, which indicates that the conventional KNN is capable of identifying the similarities among faults residing in the same module.

DKNN is an incremental-learning algorithm [25]–[27] (Figure 4). An incremental classifier can evolve using new instances without having to re-process past instances. It requires less data to build the initial classifier and consumes less computational resources. In the DKNN algorithm, a new data instance (i.e., a diagnosis result) is fed into the classifier, resulting in a predicted label. After the actual label is known, both the predicted label and the actual label are fed into the replacement block for comparison. If the predicted label is correct, the data set used by DKNN is left intact. Otherwise, the nearest neighbor that is responsible for predicting the wrong label is replaced with the instance. In other words, the nearest neighbor whose label is the same as the instance is replaced in case of wrong classification. By employing replacement, the size of the data set is constrained to fit the on-chip resources allocated for learning.

The replacement of data instances gradually revises the composition of the data set so that it is more reflective of the instances now being generated by the chip. For example, if the distribution of the data changes, then the DKNN may make a wrong prediction, but then it gradually adapts the distribution of the DKNN data set by incorporating new data. In addition, the replacement can refine the data set by removing noisy data that has an adverse effect on classification. Specifically, if a noisy data causes a wrong label, it is very likely to be replaced. Moreover, although data replacement may cause the classifier to overfit the faults appearing in a specific period of time, the overfitting can be mitigated by dynamically updating the data set, which is also an advantage of DKNN over the conventional KNN.

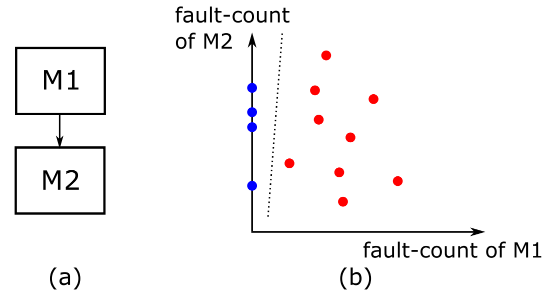

Figure 3: (a) Two modules M_1 and M_2 are connected in an core/uncore as shown. (b) A failure in M_1 may cause both M_1 and M_2 to report non-zero fault-counts (red dots), while a failure in M_2 cannot cause a non-zero fault-count for M_1 as illustrated by the blue dots that lie along the y-axis.

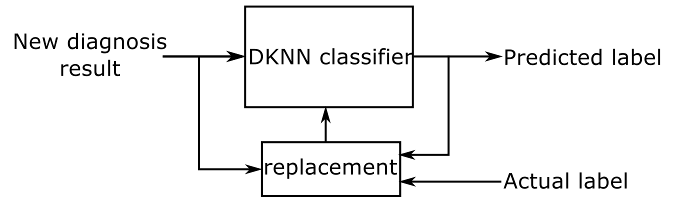

Figure 4: The incremental learning scheme of DKNN. If the predicted label is different from the actual label, then the DKNN classifier is updated.

[2] A data instance is considered to be "labeled" if its class is known, otherwise it is "unlabeled".

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.

Faulty gate	Faulty module	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
U6525_temp0_Y	M_9	0	0	0	0	0	0	0	0	2	0
U7423_temp1_Y	M_5	0	0	3	0	15	0	4	0	0	0
n11747	M_9	77	21	32	34	63	0	83	4	57	11

Table 1: L2B, partitioned into ten modules, is diagnosed at the module level through simulation. The first step of simulation is to inject a delay fault into a specific gate (the first column). The second step is to run on-chip diagnosis that results in a fault-count associated with each module (the third column). The second column is the module that contains the faulty gate.

C. DKNN for Improving the Accuracy of On-chip Diagnosis

Algorithm 1 describes the use of DKNN for improving the accuracy of on-chip diagnosis. The DKNN data set is initialized using fault simulation data, i.e., injecting faults into the gate-level netlist and then generating a fault-count for each module using simulation tools.

A periodic test/diagnosis starts with testing a core/uncore. Only if the test fails, then diagnosis is applied to identify which module of the core/uncore is most likely the reason for the observed test failure. The diagnosis result is reported as a n-dimensional vector of integers, where n is the number of modules. For each diagnosis result, a check for ideal resolution (i.e., only one fault-count is non-zero) is performed. If the resolution is ideal, then prediction via DKNN is obviously not needed. If the resolution is non-ideal, then DKNN is performed to predict which of the n modules is the faulty one. Specifically, the module is labeled using the outcome of a majority vote provided by its k nearest neighbors, and then data replacement is performed if the predicted label is incorrect. It is noted that only "relevant data" is considered when the classifier is searching for nearest neighbors. For example, in Table 1, the second diagnosis result has three modules (i.e., M_3, M_5 and M_7) associated with non-zero fault-counts, and therefore only data instances whose labels are 3, 5, or 7 are considered relevant. In addition, if a tie occurs during majority vote (i.e., two or more modules contribute the same number of neighbors), then either one module is selected as the label.

When a module is predicted to be faulty, it will be repaired and the system will be re-tested to determine if it was the actual faulty module. If the system passes, then the prediction is assumed to be correct; otherwise, the second-potentially faulty module will be repaired. Test, diagnosis, and repair continue until the actual faulty module is identified. Finally, after the actual faulty module is identified, the data replacement is performed.

Algorithm 1 DKNN for improving the accuracy of on-chip diagnosis.

```
1:  partition a core/uncore into n modules, i.e., M_1,M_2,...,M_n
2:  initialize the data set S using simulation data
3:  for each periodic test/diagnosis do
4:      run on-chip test
5:      if test passes then
6:          wait until next test
7:      else
8:          run on-chip diagnosis
9:          collect fault-counts, i.e., v = (c_1, c_2, ..., c_n)
10:         if v has only one non-zero fault-count then
11:             l_pred (i.e., predicted module) ← i, s.t. c_i > 0
12:         else
13:             find k nearest neighbors of v from S, i.e.,
                (v_1, l_1), ..., (v_k, l_k)
14:             l_pred ←majority vote of l_1, ..., l_k
15:                 (if a tie occurs, select either one)
16:             repair module l_pred and run on-chip test
17:             l_actl (i.e., actual faulty module) ← l_pred
18:             while test fails do
19:                 repair next potentially-faulty module l_next
20:                 run on-chip test
21:                 l_actl ← l_next
22:             if l_pred ≠ l_actl then
23:                 for i ← 1 to k do
24:                     if l_i = l_pred then
25:                         replace (v_i, l_i) with (v, l_actl) in S
26:                         break
```

III. Architecture

Figure 5 shows the block diagram for improving the accuracy of on-chip diagnosis. DKNN takes as input the diagnosis result from the on-chip diagnosis scheme described in [14], performs classification using the labeled data that is stored in the on-chip buffer, and updates the data set if the classification is later deemed to be incorrect. The architecture of the KNN classifier is shown in Figure 6. The DKNN classifier hardware employs an IP-core design proposed in [28]. The fully pipelined KNN architecture shown in Figure 6 has $m+k$ stages, where m is the number of features, and k is the number of nearest neighbors. The white nodes compute the distances, the dark gray nodes find the labels of the k nearest neighbors, and the light gray nodes find the majority of the KNN labels to be used for prediction. Assuming that the size of the data set is n, the architecture can classify a new data instance in $n+m+k$ clock cycles.

DKNN is described using Verilog and synthesized using Synopsys Design Compiler [29]. The area, critical path, and power are 45,305μm^2, 5.3ns, and 700μW, respectively, for the following parameter values: k =5, the number of features is 16, and the size of data set is 200. In addition, the latency of predicting the faulty module ($\sim$1.3μs@166MHz) is much smaller than that of on-chip test, diagnosis and repair [13], [14], [16].

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.

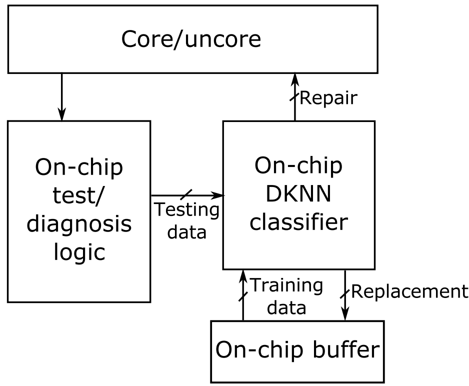

Figure 5: A DKNN classifier is used to improve accuracy of on-chip diagnosis. The block diagram shows the communication of data between various blocks within the system.

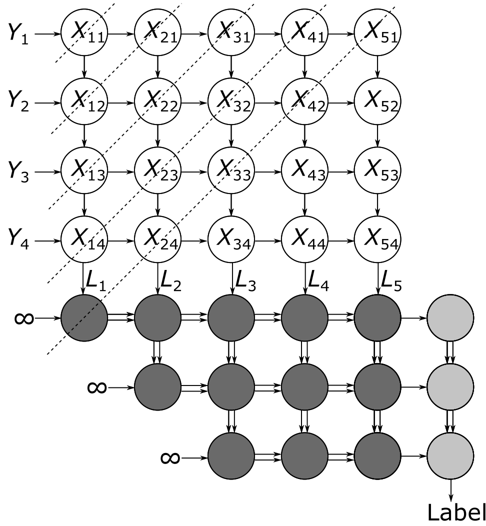

Figure 6: A pipelined architecture for KNN from [28]. The architecture classifies a data instance in $n+m+k$ clock cycles, where n is the size of the data set, m is the number of features, and k is the number of nearest neighbors. In this case, n =5, m =4 and k =3.

IV. Experiment

To validate the effectiveness of DKNN, two benchmark circuits, namely L2B [21] and c7552 [22], are used for simulation. L2B is the L2 cache bank controller of the OpenSPARC T2 processor, which is partitioned into 10 modules, and c7552 is one of the ISCAS benchmark circuits, which is partitioned into 12 modules.

The benchmark circuits, with delay faults injected into them, are tested/diagnosed, and then the faulty module is predicted using DKNN. The conventional KNN and select-max are also performed to the benchmark circuits for comparison. Table 2 shows the accuracy which is defined as the probability that the predicted module is actually faulty under the assumption that a single module is faulty. The result demonstrates that DKNN

Bench-mark circuits	No. of injected faults	Fault distribution	Select-max	Conven-tional KNN	DKNN
L2B	997	stationary	71%	79%	83%
		non-stationary	71%	51%	82%
c7552	2,694	stationary	21%	44%	46%
		non-stationary	21%	38%	46%

Table 2: The diagnostic accuracies for select-max, the conventional KNN, and DKNN are compared using the benchmark circuits L2B and c7552.

performs better than both the conventional KNN and select-max. Compared to select-max, DKNN improves the accuracy from 71% to 83% for L2B, and from 21% to 46% for c7552. After all injected faults are classified by DKNN, the amount of data replacement is 165 for L2B, and 1,212 for c7552, representing 16.5% and 45.0% of the initial data sets for the two benchmark circuits, respectively.

To simulate non-stationary fault distribution, we assume that the DKNN data set is initially constructed without any faults from M_9 and M_{10}, but the DKNN classifier has to deal with the faults from M_9 and M_{10} later in time. The result demonstrates that the DKNN is able to maintain high accuracy because it can dynamically incorporate new diagnosis data from M_9 and M_{10}. However, the conventional KNN performs even worse than select-max because it cannot predict any fault from M_9 and M_{10} (Table 2).

Figure 7(a) shows the overall diagnostic accuracy of DKNN versus the size of the DKNN data set. The accuracy is improved as the size of the DKNN data set increases. However, the accuracy saturates if the size is larger than 200. Figure 7(b) shows the overall diagnostic accuracy of DKNN versus k (i.e., the number of nearest neighbors). It shows that the highest overall accuracy is achieved when k =5. Thus, 200 is selected as the size of the DKNN data set, and 5 is selected as the value of k for KNN classification.

Finally, the performance of DKNN, as a generic machine-learning algorithm, is evaluated. Specifically, the accuracy of DKNN is compared with the conventional KNN and the incremental KNN algorithm proposed in [20], using seven data sets from the UCI repository databases [23]. These data sets are commonly used for verifying the performance of classifiers. The classification results demonstrate that DKNN performs slightly better than both the conventional KNN and the incremental KNN in [20] (Figure 8).

V. Conclusion

In this paper, we have developed a dynamic k-nearest-neighbor (DKNN) algorithm for improving the accuracy of on-chip, module-level diagnosis. Different from the conventional KNN, DKNN described here employs online data to update the learned classifier dynamically, so that the learned classifier can evolve as new data becomes available. Specifically, if the classification for a data instance is proved to be wrong, the nearest neighbor that is responsible for predicting the wrong label is replaced by the data instance itself. When used to

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.

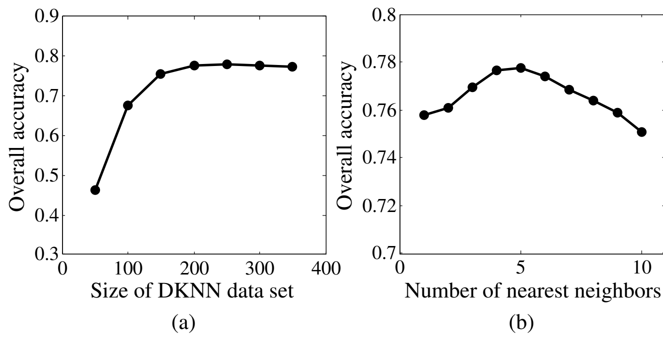

Figure 7: (a) The diagnostic accuracy of DKNN is improved as the size of the DKNN data set increases given k =5. (b) The diagnostic accuracy of the DKNN varies for different values of k (i.e., the number of nearest neighbors) given the size of the DKNN data set is 200.

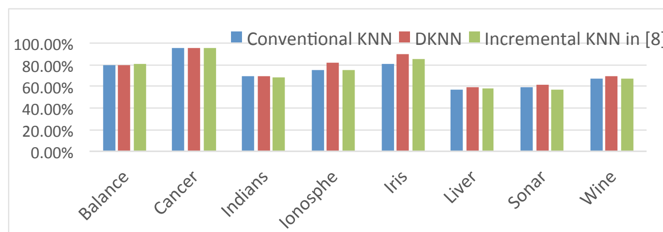

Figure 8: The performance of the conventional KNN, DKNN, and the incremental KNN in [20] are compared using seven randomly selected datasets from the UCI repository databases. 10-fold cross-validation is used, and size of data set is 200.

improve the accuracy of on-chip diagnosis, the experiments demonstrate that DKNN significantly improves the accuracy for benchmark circuits, L2B and c7552. DKNN can achieve acceptable accuracy with limited memory, or in case of non-stationary fault distribution, which makes it suitable for on-chip use. Finally, experiments using the UCI repository [23] show that DKNN performs as well or better than conventional KNN and another incremental KNN algorithm [20].

REFERENCES

[1] S. Borkar, "Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation," in *Micro*, 2005.

[2] T. Mak, "Infant Mortality-The Lesser Known Reliability Issue," in *International On-Line Testing Symposium*, 2007.

[3] Y. M. Kim, Y. Kameda, H. Kim, M. Mizuno, and S. Mitra, "Low-cost Gate-oxide Early-life Failure Detection in Robust Systems," in *Symposium on VLSI Circuits*, 2010.

[4] N. C. Laurenciu, Y. Wang, and S. D. Cotofana, "A Direct Measurement Scheme of Amalgamated Aging Effects with Novel On-chip Sensor," in *International Conference on Very Large Scale Integration*, 2013.

[5] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Built-in Soft-error Resilience for Robust System Design," in *Integrated Circuit Design and Technology*, 2005.

[6] M. Zhang, S. Mitra, S. Member, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential Element Design with Built-in Soft Error Resilience," *Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 12, pp. 1368–1378, 2006.

[7] S. Shamshiri, A. Rani, K. Cheng, and S. Barbara, "End-to-End Error Correction and Online Diagnosis for On-Chip Networks," in *International Test Conference*, 2011.

[8] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera, "Architectures for Online Error Detection and Recovery in Multicore Processors," in *Design, Automation and Test in Europe*, 2011.

[9] A. Ghofrani, R. Parikh, S. Shamshiri, A. Deorio, K. Cheng, and V. Bertacco, "Comprehensive Online Defect Diagnosis in On-Chip Networks," in *VLSI Test Symposium*, 2012.

[10] Y. Li, Y. M. Kim, E. Mintarno, D. Gardner, and S. Mitra, "Overcoming Early-life Failure and Aging for Robust Systems," in *Design and Test of Computers*, 2009.

[11] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit Failure Prediction and Its Application to Transistor Aging," in *VLSI Test Symposium*, 2007.

[12] Y. Li, S. Makar, and S. Mitra, "CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns," in *Design, Automation and Test in Europe*, 2008.

[13] Y. Li, D. S. Gardner, O. Mutlu, and S. Mitra, "Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips," in *VLSI Test Symposium*, 2010.

[14] M. Beckler and S. Blanton, "On-chip Diagnosis for Early-life and Wear-out Failures," in *International Test Conference*, 2012.

[15] D. Lazarevski, "VLSI Fault Diagnosis Problems and Decisions," in *ARSA-Advanced Research in Scientific Areas*, 2012.

[16] Y. Li, E. Cheng, S. Makar, and S. Mitra, "Self-repair of Uncore Components in Robust System-on-Chips: An OpenSPARC T2 Case Study," in *International Test Conference*, 2013.

[17] K. Huang, H.-G. Stratigopoulos, S. Mir, C. Hora, Y. Xing, and B. Kruseman, "Diagnosis of Local Spot Defects in Analog Circuits," *Instrumentation and Measurement, IEEE Transactions on*, vol. 61, no. 10, pp. 2701–2712, 2012.

[18] C. Yang, S. Tian, Z. Liu, J. Huang, and F. Chen, "Fault Modeling on Complex Plane and Tolerance Handling Methods for Analog Circuits," *Instrumentation and Measurement, IEEE Transactions on*, vol. 62, no. 10, pp. 2730–2738, 2013.

[19] M. A. El-Gamal, A.-K. S. Hassan, and A. A. Ibrahim, "Analog Fault Diagnosis Using Conic Optimization and Ellipsoidal Classifiers," *Journal of Electronic Testing*, vol. 30, no. 4, pp. 443–455, 2014.

[20] K. Forster, S. Monteleone, A. Calatroni, D. Roggen, and G. Troster, "Incremental k-NN Classifier Exploiting Correct-Error Teacher for Activity Recognition," in *Machine Learning and Applications*, 2010.

[21] "OpenSPARC T2," http://www.oracle.com/technetwork/systems /opensparc/opensparc-t2-page-1446157.html, Oracle.

[22] "ISCAS Benchmarks," http://web.eecs.umich.edu/ jhayes/iscas.restore/.

[23] "UC Irvine Machine Learning Repository," http://archive.ics.uci.edu/ml/, University of California, Irvine.

[24] L. Kozma, "K Nearest Neighbors Algorithm (kNN)," Tech. Rep., 2008.

[25] R. Polikar, L. Udpa, S. Member, S. S. Udpa, and V. Honavar, "Learn++ : An Incremental Learning Algorithm for Supervised Neural Networks," *Systems, Man, and Cybernetics*, vol. 31, no. 4, pp. 497–508, 2001.

[26] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning," in *Advances in Neural Information Processing Systems*, 2001.

[27] R. Elwell, R. Polikar, and S. Member, "Incremental Learning of Concept Drift in Nonstationary Environments," *Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[28] E. S. Manolakos and I. Stamoulias, "IP-cores Design for the kNN Classifier," in *International Symposium on Circuits and Systems*, 2010.

[29] "Design Compiler," http://www.synopsys.com/Tools/Implementation/ RTLSynthesis/DesignCompiler/Pages/default.aspx, Synopsys.

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 31,2021 at 04:40:49 UTC from IEEE Xplore. Restrictions apply.