Measurement of Dataset Quality and Computational Quality of Graph Neural Networks on Analog Integrated Circuit Recognition System

Arif Abdul Mannan[1], Koichi Tanno[2]
Faculty of Engineering, University of Miyazaki, Miyazaki, Japan[1]
Department of Electrical Engineering, Brawijaya University, Malang, Indonesia[1, 2]

Abstract*—To meet the needs of AI design in analog IC design automation and especially in big data applications, the application of sensitive and precise Graph Neural Network (GNN) recognition will become increasingly necessary. This sensitive GNN recognition is needed to accommodate the highly stringent trade-offs on analog IC design requirements, combined with the increasingly large learning data requirements. Furthermore, more precise GNN recognition will be a fundamental requirement for a more sensitive system to accommodate noise in floating point (FP) calculations, namely inaccuracies and imprecision in IEEE 754 FP calculations. In this study, by refining a previously proposed method that uses the output vector representation (OVR) of the untrained Graph Neural Network (GNN), and by exploiting the numerical reproducibility error in FP calculations, a method for measuring the sensitivity and precision of GNNs for use in analog IC design recognition has been proposed. The results of the sensitivity and precision measurements of the proposed method show a complex combination of effects between dataset quality (the presence or absence of data duplication), the sensitivity of the GNN to distinguish each feature in the dataset, the complexity of the calculations that occur in the GNN, and the level of quality of the FP calculations performed by the processing unit related to the non-ideal FP calculations. With certain GNN configurations, the proposed method also succeeded in measuring the difference in FP calculation quality of Central Processing Unit (CPU) and Graphics Processing Unit (GPU), where, for big data applications, from the tests carried out, the maximum amount of data that can be distinguished by the CPU is 25 to 100 times more than with the GPU. Because it only uses untrained GNN OVR and does not involve any training process in obtaining results, the proposed method is still unable to obtain a correlation with the final value of GNN performance after training is complete. The measurement method using GNN OVR involving the learning process is future work.

***Keywords*—Big data; Graph Neural Network; Artificial Intelligence; calculation quality measurement; analog circuit design**

I. INTRODUCTION

Artificial intelligence (AI) is highly needed in the world of Integrated Circuit (IC) design. However, the application of AI in analog IC design is still very limited. The main reason for the slow implementation of AI in analog IC design is the very stringent trade-offs inherent in analog IC design and the high dependence on the knowledge of engineers. Small parameter changes in an analog IC design can cause significant changes in the behavior and output of the analog IC being designed [1], [2], [3], [4], [5]. Therefore, to design an AI system for analog IC design applications, a neural network (NN) system capable of distinguishing even the smallest changes in analog circuit design parameters is required [6], [7]. In other words, in big data applications, AI applied to analog IC design must have a high level of sensitivity [8], [9], [10], [11].

The application of NN, particularly graph neural networks (GNNs), to analog IC design has been discussed previously [1], [8], [12]. The application of GNNs such as Graph Convolutional Network (GCN) [13], Graph Isomorphism Network (GIN) [14], Graph Attention Network V2 (GAT) [15], and GraphSage (GSG) [16] has also been studied. All these GNNs utilize floating-point (FP) calculations. As we know, FP calculations are inherently inaccurate and imprecise due to the internal processes within the Central Processing Unit (CPU) or Graphics Processing Unit (GPU). As long as FP calculations in CPUs/GPUs still use IEEE 754 [17], these inaccuracies and imprecisions are inevitable [18], [19], [20], [21], [22], [23]. Therefore, all GNNs applied to analog IC design must not only be able to distinguish very small parameter changes (high sensitivity) but also possess high precision, allowing them to tolerate the inaccuracies and imprecisions inherent in the floating-point calculation system within the CPU/GPU.

The ability to measure the sensitivity and precision of a neural network (NN) [24] used in analog IC design would be extremely useful for simplifying the AI design process, if feasible. The application of GNN output calculations to detect data duplication in a dataset for analog IC design, to measure the sensitivity of NN inputs, has been successfully implemented using a floating-point digit rounding system [8]. However, this proposed implementation in [8] has a weakness: determining the exact value for the floating-point digit rounding in data duplication detection remains uncertain. This proposed implementation only uses exact values based on best-case trials, leaving the challenge of measuring the precision of floating-point calculations in NNs unresolved.

In this study, a new method for determining the exact value of floating-point digit rounding in measuring the level of data duplication in datasets, specifically for the application of GNNs in analog IC design, is proposed. This new idea is based on exploiting the inaccuracy and imprecision of floating-point calculations within the CPU/GPU itself. Therefore, in addition to obtaining the ability to measure the level of duplication and/or measure the level of GNN's ability to distinguish each data in the dataset (GNNs sensitivity) more precisely, with the technique proposed in this study, the ability to measure the level of precision of the CPU/GPU (or other NN processors, including the possibility of FPGA [25]) in processing GNN calculations can also be obtained. Therefore, the use of 32-bit, 64-bit, or even the possibility of using 16-bit FP in [26] can be determined. Although this measurement method is intended for analog IC design, the use of this proposed method can be applied directly to other applications, whether using GNN or other artificial NN (ANN), as long as they have the same OVR structure.

This study consists of the following sections: The method for obtaining calculation quality measurements from GNNs is described in Section II. Simulations and discussions are presented in Section III, while conclusions and future work are presented in Section IV.

II. GNN Calculation Quality Measurement

In this study, the measurement of GNN calculation quality is obtained by optimizing and developing a previous method that also uses an untrained GNN output vector representation (OVR) in [8]. The use of untrained GNN OVR is very useful because it is expected that the process of measuring GNN performance (sensitivity and precision) can be done even before the training process occurs. Before discussing the method of measuring the calculation quality of GNNs, graph datasets and GNN output will first be explained.

For a graph-based dataset, consider a dataset that contains $\mathcal{V}$ number of graph information as $\mathcal{G}(\mathcal{V},\mathcal{E})$, with additional information that can also be added, such as the input features $\{x_v, \forall_v \in \mathcal{V}\}$, have variate number of vertex (node v) and edge (ε) array length on every graph. The GNN OVR z_v is obtained from every neighborhood vector $h_v^{(l)}$ for all $v \in \mathcal{V}$ as shown in Eq. (1):

$$z_v = h_v^{(L)} \quad ,\forall v \in \mathcal{V} \tag{1}$$

As explained in the previously proposed method, in this study, the same GNN OVR z_v structure, as shown in Eq. (2), will be used as the basis for the calculation. This results in variations in $\mathcal{V}$ and $\mathcal{E}$ being negligible and resulting in data with a fixed C dimensional output (C is the number of classes in the dataset) being generated.

$$z_v = \left(z_{v_1}, z_{v_2}, z_{v_3}, \cdots z_{v_C}\right), \forall v \in \mathcal{V} \tag{2}$$

Based on the output value of GNN OVR z_v, $prpDist$ parameter and $prDiff$ parameter will be obtained [8]. The $prpDist$ is the pseudo distance parameter, calculated simply by using the sum of the absolute value of every axis's scalar value difference between two points, as shown in Eq. (3):

$$prpDist_p = \begin{cases} \sum_{i=1}^{C} \left| z_{r_i} - z_{p_i} \right| & at\ r \neq p \quad r,p \in \mathcal{V} \\ 0 & at\ r = p \quad r,p \in \mathcal{V} \end{cases} \tag{3}$$

With z_r and z_p is vector value of the reference point and point under test in the C-dimension output, respectively. The $prDiff$ is the "difference" parameter, calculating the relative difference between two points of OVR data on the C dimensional space into one scalar number, as shown in Eq. (4):

$$prDiff_p = \sum_{i=1}^{C} (z_{r_i} - z_{p_i}) n_{r_i} \tag{4}$$

With n_{r_i} is obtained from the i-th element of N_r, the normalized reference point array (NRA), consists of fractional number with range from $0 \leq n_{r_i} \leq 1$ and calculated using Eq. (5):

$$N_r = \frac{Z_r - z_{r_{min}}}{z_{r_{max}} - z_{r_{min}}} \tag{5}$$

A. Twice GNN OVR Generation

Techniques for measuring the imperfections of FP calculations using shadow execution [18], [19], and incorporating external factors into the calculation [20] have been proposed previously. In this "calculation quality measurement," the measurement is obtained by calculating the difference between two GNN OVR results performed using the same environment and configuration. Ideally, these two GNN OVR results should be the same, but due to the numerical reproducibility error in FP calculations [21], [22], [23], the two GNN OVR results will differ. Simply put, in this study, the measurement method is carried out by exploiting numerical reproducibility errors in FP calculations. As shown in Algorithm 1, these two (or more) GNN OVR outputs can be easily obtained by using one of the following three methods:

1) Identical input generation: Recognize the untrained GNN twice using the same input dataset. Based on Eq. (1) and Eq. (2), this identical input generation method can be described as Eq. (6):

$$\begin{gathered} z1_v = \left(z1_{v_1}, z1_{v_2}, z1_{v_3}, \cdots z1_{v_C}\right) = h_v^{(L)}, \forall v \in \mathcal{V} \\ z2_v = \left(z2_{v_1}, z2_{v_2}, z2_{v_3}, \cdots z2_{v_C}\right) = h_v^{(L)}, \forall v \in \mathcal{V} \end{gathered} \tag{6}$$

2) Reversed input generation: Recognize the untrained GNN once using the initial input dataset, then perform a second recognition using the initial dataset with its graph sequence reversed. The GNN OVR output from the second recognition is then reversed to match the order of the first recognition results. Based on Eq. (1) and Eq. (2), this reversed input generation method can be described as Eq. (7):

$$\begin{gathered} z1_v = \left(z1_{v_1}, z1_{v_2}, z1_{v_3}, \cdots z1_{v_C}\right) = h_v^{(L)}, \forall v \in \mathcal{V} \\ z_j = \left(z_{j_1}, z_{j_2}, z_{j_3}, \cdots z_{j_C}\right) = h_j^{(L)}, \forall j \in J \end{gathered} \tag{7}$$

With $J = \mathcal{V}[::-1]$ and $z2_v = z_j[::-1]$. The $[::-1]$ is the operation of reverse order of the array data.

3) Randomized input generation: Recognize the untrained GNN once using the initial input dataset, then perform a second recognition using the initial dataset with its graphs sequence randomized. The GNN OVR output from the second recognition is then reordered (in the reverse order of the previous randomization) to match the order of the first recognition results. Based on Eq. (1) and Eq. (2), this randomized input generation method can be described as Eq. (8):

$$z1_v = \left(z1_{v_1}, z1_{v_2}, z1_{v_3}, \cdots z1_{v_c}\right) = h_v^{(L)}, \forall v \in \mathcal{V}$$
$$z_m = \left(z_{m_1}, z_{m_2}, z_{m_3}, \cdots z_{m_c}\right) = h_m^{(L)}, \forall m \in M \qquad (8)$$

With $M = R(\mathcal{V})$ and $z2_v = R^{-1}(z_m)$. The $R()$ is the operation or the process of randomly arranging the array sequence based on a relationship table that has been created randomly before recognition is carried out, and the $R^{-1}()$ is the process of returning an array order from random to normal based on the same table.

Algorithm 1: Two GNN OVR generator

```
PROGRAM OVRMatrixGenIden()
INPUT $h_v$
OUTPUT $z1_v$, $z2_v$
Compute
for $v$ in $\mathcal{V}$ do
    $z1_v \leftarrow$ recog($h_v$)
end for
for $v$ in $\mathcal{V}$ do
    $z2_v \leftarrow$ recog($h_v$)
end for
return $z1_v$, $z2_v$

PROGRAM OVRMatrixGenReversed()
INPUT $h_v$
OUTPUT $z1_v$, $z2_v$
Compute
for $v$ in $\mathcal{V}$ do
    $z1_v \leftarrow$ recog($h_v$)
end for
$h_j \leftarrow h_v$ [::-1]     // $h_j$ is $h_v$ in reverse order
for $j$ in $J$ do
    $temp_j \leftarrow$ recog($h_j$)
end for
$z2_v \leftarrow temp_j$ [::-1]
return $z1_v$, $z2_v$

PROGRAM OVRMatrixGenRandom()
INPUT $h_v$
OUTPUT $z1_v$, $z2_v$
Compute
dataSuffleIndex←RandomList(0, len(h))
for $v$ in $\mathcal{V}$ do
    $z1_v \leftarrow$ recog($h_v$)
end for
for index, data in dataSuffleIndex
    $h_{m_{index}} \leftarrow h_{v_{data}}$ // $h_m$ is $h_v$ in random order
end for
for $m$ in $M$ do
    $temp_m \leftarrow$ recog($h_m$)
end for
for index, data in dataSuffleIndex
    $z2_{data} \leftarrow temp_{index}$
end for
return $z1_v$, $z2_v$
```

To produce the two GNN OVR outputs, only one of the three existing methods is required. In this study, the randomized input generation (with the highest probability of numerical reproducibility error in FP calculations) will be used as a reference for the simulation in Section III.

After the two GNN OVRs are calculated, $prpDist$ and $prDiff$ will be calculated using Eq. (3) and Eq. (4). Each GNN OVR will produce one pair of variables $prpDist$ and $prDiff$. Therefore, the variables $prpDist1$ and $prDiff1$ will be obtained from the OVR $z1_v$, while $prpDist2$ and $prDiff2$ will be obtained from the OVR $z2_v$ calculation.

"**Important!** Starting from $prpDist$ and $prDiff$ calculations onward, using more accurate and precise floating-point calculations is crucial. If the GNN OVR calculation uses 32-bit FP, then calculating $prpDist$ and $prDiff$ requires at least 64-bit FP to obtain a precise measurement of the GNN calculation quality."

B. Calculation Quality Measurement

In this study, two variables will be determined in the GNN calculation quality measurement results: the calculation quality percentage (CQP) and the calculation quality digit (CQD). The CQP is defined as the percentage of data certainty (still able to ascertain its precision) that is still unaffected by floating point inaccuracy and imprecision at each decimal point. Meanwhile, the calculation quality digit is defined as the lowest digit value in a FP calculation result that can still be ascertained for its precision before errors from floating point inaccuracy and imprecision affect it. Examples of GNN calculation quality measurements are presented in TABLE I and Fig. 1. TABLE I shows that up to 7 digits after the decimal point, 100% of the data can be ascertained for its precision. Meanwhile, for up to 8 digits after the decimal point, the data with ascertained precision drops to 83%, indicating that 17% of the data has experienced precision degradation due to non-ideal FP calculations. Therefore, the CQD based on the example of TABLE I is 7.

TABLE I. EXAMPLE OF GNN CALCULATION QUALITY RESULT

Decimal Places	***Calculation Quality Percentage (CQP)***	***Remarks***
0	100%	
1	100%	
2	100%	
3	100%	
4	100%	
5	100%	
6	100%	
7	100%	<= CQD
8	83%	
9	20%	
10	15%	

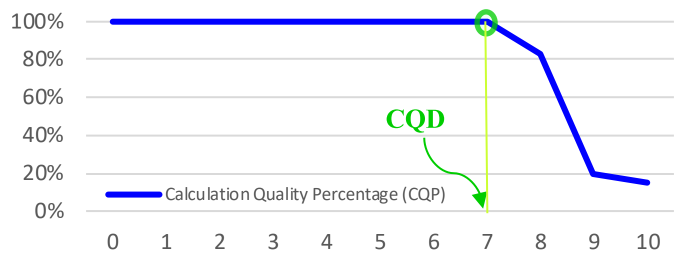

Fig. 1. The example of GNN calculation quality chart.

By using the two pairs of $prpDist$ and $prDiff$ generated in Section II-A, the quality measurement of the GNN calculations can be performed. Assuming that the FP calculations in Section II-A are ideal, then all the results of $prpDist1$ will be the same as $prpDist2$. However, floating point calculations on CPU/GPU are not ideal (floating point inaccuracy and imprecision), so there will be a slight difference between $prpDist1$ and $prpDist2$ as shown in Eq. (9):

$$prpDist1_p \approx prpDist2_p \quad ,\forall p \in \mathcal{V} \qquad (9)$$

If we describe the exact value of $prpDist2$ results, we will obtain Eq. (10). From Eq. (9) and Eq. (10), if applied to prdiff1 and prdiff2, then Eq. (11) will be obtained.

$$\epsilon Dist_p = prpDist1_p - prpDist2_p \quad ,\forall p \in \mathcal{V} \qquad (10)$$

$$\epsilon Diff_p = prDiff1_p - prDiff2_p \quad ,\forall p \in \mathcal{V} \qquad (11)$$

where, $\epsilon Dist$ and $\epsilon Diff$ are the error values resulting from the inaccuracy and imprecision of FP numbers.

By obtaining the $\epsilon Dist$ value, the quality of the GNN calculation will be determined based on the exponent value of the most significant digit in $\epsilon Dist$ compared to the exponent value of the most significant digit in $prpDist1$ or $prpDist2$. The same applies to $\epsilon Diff$. The exponent value of the most significant digit can be obtained using Algorithm 2.

Algorithm 2: Get Exponential Scale

```
PROGRAM GetExpScale()
INPUT $x_v$
OUTPUT $t_v$
Compute
for $v$ in $\mathcal{V}$ do
$t_v \leftarrow$ truncate(log($x_v$))
end for
return $t_v$
```

By obtaining the highest exponent value of $prpDist1$ (one can be selected between $prpDist1$ and $prpDist2$), calculation Eq. (12) can be performed. Therefore, the relative error value $ErDist_p$ of $\epsilon Dist$ based on $prpDist1$ can be obtained from Eq. (13):

$$tDist1_p = GetExpScale(prpDist1_p) \qquad (12)$$

$$ErDist_p = \epsilon Dist_p \odot 10^{tDist1_p} \quad ,\forall p \in \mathcal{V} \qquad (13)$$

The same is true for $prDiff$, so that Eq. (14) and Eq. (15) can be obtained:

$$tDiff1_p = GetExpScale(prDiff1_p) \qquad (14)$$

$$ErDiff_p = \epsilon Diff_p \odot 10^{tDiff1_p} \quad ,\forall p \in \mathcal{V} \qquad (15)$$

Based on the results from Eq. (13) and Eq. (15), the CQP calculation process can be carried out using Algorithm 3. Algorithm 3 is only able to calculate the CQP value at one decimal place. So if a CQP with a decimal point value ranging from 0 to 10 is required (as in the example in TABLE I), then 11 calculations using Algorithm 3 are required.

Algorithm 3: Get Single Quality Percentage

```
PROGRAM GetQualityPercentage()
INPUT $ErDist_p$, $ErDiff_p$, DecimalTest
OUTPUT $Qp$
Compute
for $p$ in $\mathcal{V}$ do
$tempDist \leftarrow ErDist_p * 10^{DecimalTest}$
$resDist_p \leftarrow |tempDist| < 1$
$tempDiff \leftarrow ErDiff_p * 10^{DecimalTest}$
$resDiff_p \leftarrow |tempDiff| < 1$
end for
$resAll$ = logical_and ($resDist_p$, $resDiff_p$)
$resGood$ = nonzero ($resAll$ )
$Qp$ = 100 * len($resGood$) / len($resAll$ )
return $Qp$
```

C. Special Value Decimal Places

With the $\epsilon Dist$ and $\epsilon Diff$ values obtained from Eq. (10) and Eq. (11), and based on the calculations performed in Algorithm 3, the decimal digit values that can be entered do not have to be integers. The digit values entered can be fractional numbers. In this study, the digit values with values $x.3$ and $x.6$ are chosen to increase the resolution of the CQP results. The digit value $x.3$ is chosen because the value of $10^{-0.3}$ has a value close to 0.5 (the exact value used in the actual calculation as shown in Algorithm 3 is $10^{x.30103}$; or the value 10^x multiplied by 2), and the digit value $x.6$ is chosen because $10^{-0.6}$ has a value close to 0.25 (the exact value used in the actual calculation as shown in Algorithm 3 is $10^{x.60206}$; or the value 10^x multiplied by 4). Example of rounding down using 2, 2.3, 2.6 and 3 decimal places after the comma is shown is TABLE II.

TABLE II. Example of Rounding Down using 2, 2.3, 2.6 and 3 Decimal Places

Real Numbers	Rounding (down) n-digits after the decimal point			
	2.0	2.3	2.6	3.0
3.1214	3.1200	3.1200	3.1200	3.1210
3.1231	3.1200	3.1200	3.1225	3.1230
3.1274	3.1200	3.1250	3.1250	3.1270
3.1287	3.1200	3.1250	3.1275	3.1280
3.1330	3.1300	3.1300	3.1325	3.1330

III. Simulation and Discussion

In the simulation in this study, the simulation results of dataset duplication detection from [8] will be included. The results of dataset duplication detection will be named the GNN & Dataset Compatibility (GDC) variable, which consists of: GNN & dataset compatibility map (GDCM) and GNN & dataset compatibility value (GDCV). Similar to CQP, which has its own value at each decimal point, GDCM shows the combined value of GNN sensitivity and dataset quality (dataset uniqueness; the presence or absence of data duplication) at each decimal point. Meanwhile, GDCV is the value of GNN & Dataset Compatibility exactly at the decimal point indicated by CQD.

The main purpose of measuring the calculation quality and compatibility of GNN & Dataset is to help the design of GNN that will be used in analog IC design automation system. In this study, for analog IC design applications, the dataset to be used, extracted using the method in [27], has the features as shown in TABLE III. In this simulation section, this dataset will be referred to as "MOSFET".

TABLE III. MOSFET DATASET FEATURE

Datasets Feature	Quantity
Number of data	2,115
Number of classes	121
Min data per class	1
Max data per class	795
NETLIST length	5,214,505
NETLIST number of line	95,615
Number of MOSFET(s)	74,626
Number of Power Supply(s)	4,234
Number of Resistors(s)	7,804
Number of LC(s)	11

For all simulations measuring the quality of GNN calculations and dataset quality, a personal computer with a 10850K CPU, 64GB RAM, an RTX 4070 Ti GPU, a Windows 10 system, and a Python environment with PyTorch for neural network computations were used. To obtain GNN OVR data, simulations were performed with both the CPU and or GPU in 32-bit FP format. Meanwhile, calculations starting from $prpDist$ and $prDiff$ up to CQP, CQD, GDCM, and GDCV were performed purely on the CPU with a higher FP accuracy configuration of 64-bit FP format.

A. Measurement Results of 1-Node Multi-Edge and Net Connection Abstraction MOSFET

In this simulation section, the quality of GNN calculations and the dataset quality of MOSFET with 1-Node Multi-Edge & Net Connection (1NMC) abstraction in [12] is measured. The measurement results of CQP, CQD, GDCM and GDCV for 2115 data (121 classes) in 1NMC abstraction are shown in TABLE IV and Fig. 2. This simulation was performed using GIN (there is no specific reason for choosing GIN; GCN, GAT, GSG, or other GNNs could also be used), 128-dim input layer dimension, no hidden layer, 128-dim output layer, and GPU calculation as the Processing Unit (PU) to generate GNN OVR. SDD in TABLE IV represents "successfully differentiated data", indicating the number of unique data that are successfully differentiated by GNN, while DIN (Data integrity number) represents the number of data that are not affected by FP calculation noise (inaccuracy and imprecision of FP calculation).

TABLE IV. GIN 128-DIM, NO HIDDEN LAYER, 128-DIM, GPU

Decimal Places	SDD	GDCM	DIN	CQP	Remarks
0.0	23	1.10%	2115	100.00%	
0.3	77	3.60%	2115	100.00%	
0.6	183	8.70%	2115	100.00%	
1.0	404	19.10%	2115	100.00%	
1.3	741	35.00%	2115	100.00%	
1.6	889	42.00%	2115	100.00%	
2.0	1036	49.00%	2115	100.00%	
2.3	1140	53.90%	2115	100.00%	
2.6	1211	57.30%	2115	100.00%	
3.0	1299	61.40%	2115	100.00%	
3.3	1309	61.90%	2115	100.00%	
3.6	1310	61.90%	2115	100.00%	
4.0	1312	62.00%	2115	100.00%	
4.3	1312	62.00%	2115	100.00%	
4.6	1312	62.00%	2115	100.00%	<= CQD
5.0	1316	62.20%	2108	99.70%	
5.3	1334	63.10%	2069	97.80%	
5.6	1376	65.10%	1967	93.00%	
6.0	1516	71.70%	1556	73.60%	
6.3	1665	78.70%	1161	54.90%	
6.6	1806	85.40%	760	35.90%	
7.0	1945	92.00%	215	10.20%	
7.3	2029	95.90%	62	2.90%	
7.6	2074	98.10%	18	0.90%	
8.0	2101	99.30%	6	0.30%	
8.3	2110	99.80%	1	0.00%	
8.6	2114	100.00%	1	0.00%	
9.0	2115	100.00%	1	0.00%	
9.3	2115	100.00%	1	0.00%	
9.6	2115	100.00%	1	0.00%	
10.0	2115	100.00%	1	0.00%	

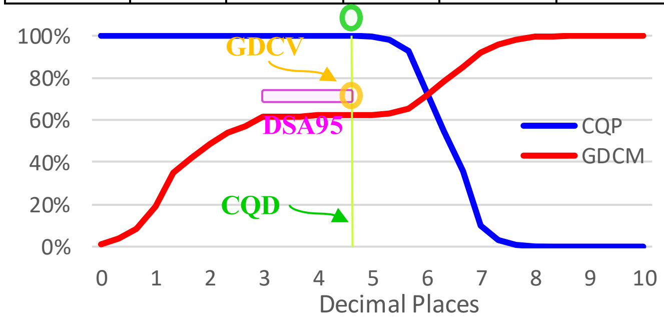

Fig. 2. Dataset and calculation quality measurement for 1NMC abstraction MOSFET using GIN 128-dim, GPU.

From the measurement results outlined in TABLE IV and Fig. 2, the following points can be observed:

1) The CQD value of the 1NMC abstraction MOSFET is 4.3 decimal places. This indicates that the inaccuracy and imprecision of the FP calculation begin to affect the GNN output starting at 4.6 decimal places.

2) The GDCV for the 1NMC abstraction MOSFET is 1312/2115, or 62.0%, at 4.3 decimal places. This indicates that the GNN settings in this measurement failed to distinguish 803 from 2115 data points, even though the GNN calculation was not yet affected by the imperfections of the FP calculation. It can be concluded that the GNN quality of 62.0% is due to either the dataset containing duplicate data or the GNN's insufficient power to distinguish between the data points.

3) If the Data Stability Area of 95 (DSA95) is defined as the area where the GNN can distinguish at least 95% of the data from the GDCV value and has a sensitivity of 100 decimal places, then the DSA95 of a 1NMC abstraction MOSFET is 1.6. This means the GNN can distinguish more than 58.9% of the data (95% from the GDCV of 62.0%) to 1.7 decimal places before the imperfections in the FP calculation begin to degrade the calculation's precision. Basically, the imperfection resulting from the FP calculation can be called noise, producing a random effect on the GNN calculation. In other words, DSA95 can be equated with the Signal to Noise Ratio (SNR) in the Analog IC world, where the noise floor starts from 4.6 decimal places (95% was chosen as a consideration that 95% is included in the stable category in the author's view; if a minimum of 98% is desired, it will be DSA98; the number can be anything, from 0% to 100%).

B. Optimized Dataset Duplication Detection

The GDC quality measurement results obtained can be generated from one or both of the dataset duplication and the GNN's ability to distinguish each data. To be able to obtain a more accurate level of dataset duplication, measurements can be carried out using various methods so that the effect of the GNN's inability to distinguish each data can be minimized. Methods that can be done include increasing or reducing the number of GNN hidden layers, changing the dimensions of the GNN layers, using various types of GNN, and using different PU (CPU / GPU or other NN processors). Of these various methods, the highest GDCV results can be said to be the value closest to the actual dataset duplication.

TABLE V. GIN 128-DIM WITH 0, 1, 2, AND 3-HIDDEN LAYER; GPU

GNN type	n-dim	n-hidden layer	CQD	GDCV	DSA95	PU
GIN	128	0	4.6	62.0%	1.6	GPU
GIN	128	1	4.3	62.0%	1.3	GPU
GIN	128	2	4.3	62.1%	1.3	GPU
GIN	128	3	4.6	62.1%	1.6	GPU

Using the 1NMC abstraction MOSFET dataset, the measurement results of the quality of GNN calculations with several kinds of hidden layers, various kinds of layer dimensions, various types of GNNs, and the use of CPU and GPU processors are shown in TABLE V and Fig. 3, TABLE VI and Fig. 4, TABLE VII and Fig. 5, and TABLE VIII and Fig. 6, respectively.

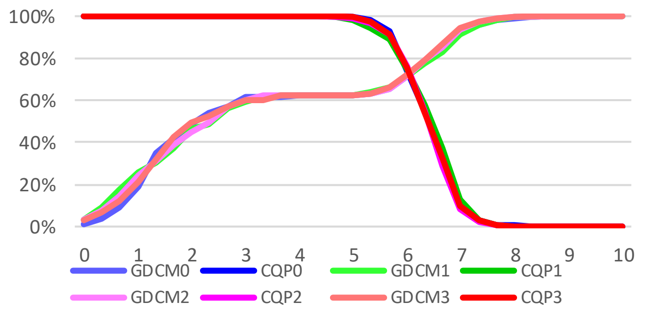

Fig. 3. Dataset and calculation quality measurement for 1NMC abstraction MOSFET using GIN 128-dim, multiple hidden layer, GPU.

TABLE VI. GIN 2, 4, 8, AND 32-DIM WITH 1-HIDDEN LAYER; GPU

GNN type	n-dim	n-hidden layer	CQD	GDCV	DSA95	PU
GIN	2	1	4.6	47.9%	2.3	GPU
GIN	4	1	4.6	62.0%	2.3	GPU
GIN	8	1	4.6	62.0%	2.3	GPU
GIN	32	1	5.0	62.0%	2.0	GPU

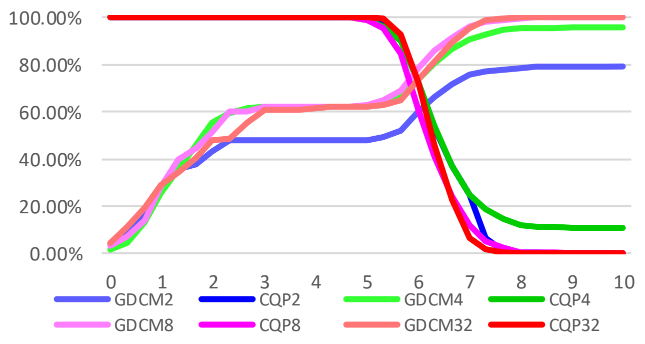

Fig. 4. Dataset and calculation quality measurement for 1NMC abstraction MOSFET using GIN multiple dim, GPU.

TABLE VII. GCN, GIN, GAT, GSG 128-DIM WITH 2-HIDDEN LAYER; GPU

GNN type	n-dim	n-hidden layer	CQD	GDCV	DSA95	PU
GCN	128	2	6.3	61.9%	1.7	GPU
GIN	128	2	4.6	62.1%	1.3	GPU
GSG	128	2	6.3	61.9%	1.7	GPU
GAT	128	2	6.6	62.0%	2.0	GPU

Based on the measurement results as described in TABLE V, TABLE VI, TABLE VII, and TABLE VIII, the highest GDCV value is 1314/2115 (62.1%). This value can be concluded as the closest value to the actual total data duplication (there is still a possibility that all GNN configurations that have been carried out have the same weaknesses and fail to distinguish certain data). Or if measurements are carried out with the aim of designing GNN in analog IC design automation, the GNN configuration that is most sensitive to the 1NMC abstraction MOSFET dataset is the GNN with a configuration that has a GDCV of 62.1%.

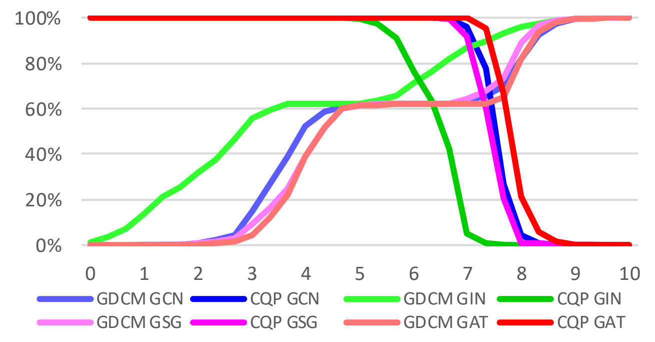

Fig. 5. Dataset and calculation quality measurement for 1NMC abstraction MOSFET using GCN, GIN, GAT and GSG on GPU.

TABLE VIII. GCN, GIN, GAT, GSG 128-DIM WITH 2-HIDDEN LAYER; CPU

GNN type	n-dim	n-hidden layer	CQD	GDCV	DSA95	PU
GCN	128	2	7.3	62.0%	2.3	CPU
GIN	128	2	5.3	62.1%	2.3	CPU
GSG	128	2	6.6	61.9%	2.0	CPU
GAT	128	2	6.6	62.0%	1.6	CPU

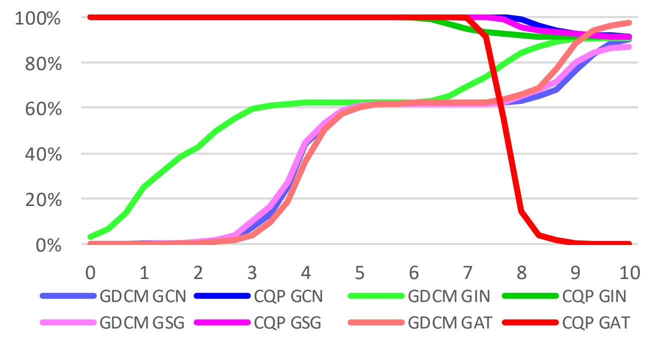

Fig. 6. Dataset and calculation quality measurement for 1NMC abstraction MOSFET using GCN, GIN, GAT and GSG on CPU.

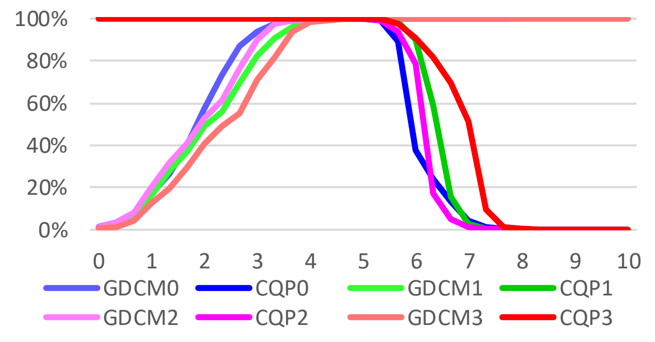

Fig. 7. Dataset and calculation quality measurement for 4NC abstraction MOSFET with multiple hidden layer.

As a comparison of the 1NMC abstraction MOSFET dataset, the measurement process with various GNN configurations was also carried out on the 4-nodes & Net Connection (4NC) abstraction MOSFET dataset in [12] with one of the measurement results shown in Fig. 7. The best GDCV value obtained by this 4NC abstraction MOSFET dataset is 2114/2115 (99.95%). Therefore, compared with the 1NMC abstraction, based on the measurement results obtained, the decision to use the 4NC abstraction is more recommended for use in designing AI for analog IC designs.

C. CPU and GPU 32-Bit FP Performance

The GNN calculation quality measurement in this study can also be used to compare the calculation quality of the processing unit used (such as CPU or GPU) in applying the desired GNN & dataset configuration. The method used in comparing this processing unit calculation quality is by generating one "untrained GNN" (as an initial example, generated on the CPU), and performing recognition (to generate the GNN OVR). Then this "untrained GNN" is transferred to another processing unit (for example, GPU) and performing recognition with the same dataset. The GNN OVR generation process does not have to be only once, in this simulation, each processing unit, four sets of GNN OVR are generated.

The performance comparison of this processing unit calculation quality will be proportional to the difference in CQP and CQD obtained. As an example case, the difference in calculation quality between CPU and GPU for the 1NMC abstraction MOSFET dataset using a 32-dimensional 1-hidden-layer GIN is shown in Fig. 8 and TABLE IX. Both CPU and GPU in this test use 32-bit FP in their calculation process.

TABLE IX. CPU VS. GPU CALCULATION QUALITY MEASUREMENT

GNN type	n-dim	n-hidden layer	CQD	GDCV	DSA95	PU
GIN	32	1	5.3	62.0%	2.0	GPU
GIN	32	1	5.3	62.0%	2.0	GPU
GIN	32	1	5.3	62.1%	2.0	GPU
GIN	32	1	5.0	62.0%	1.7	GPU
GIN	32	1	6.0	62.0%	2.7	CPU
GIN	32	1	6.0	62.0%	2.7	CPU
GIN	32	1	6.0	62.0%	2.7	CPU
GIN	32	1	6.0	62.0%	2.7	CPU

Based on Fig. 8, several results are obtained. First, when the CQP is 100% (without any influence from the FP calculation imperfections, confirmed up to 5.0 decimal places), the 4 sets of GDCMs from the CPU and 4 sets of GDCMs from the GPU have identical results, shows that the results of the dataset quality measurement (GDC) are independent from the PU being used. Second, after the FP calculation imperfections begin to take effect, in the CQP of the GPU calculation results, 4 out of 4 measured sets vary from each other, while in the CQP of the CPU calculation results, 4 out of 4 measured sets are all identical. This shows that FP calculations in CPUs are more repeatable, more capable of producing the same output values when using the same input and calculation formulas compared to GPUs. And third, based on the measurement results shown in Fig. 8 and Table IX, it is observed that the CQD of the GPU is 0.7 to 1.0 decimal places lower than that of the CPU. Since the calculation quality measurement uses two variables $prpDist$ and $prDiff$, for applications in the big data field, the maximum amount of data in the dataset that can be differentiated (utilizing the entire $prpDist$ and $prDiff$ range that is not affected by the imperfections of the FP calculation) by the CPU will be 25 to 100 times greater than that of the GPU.

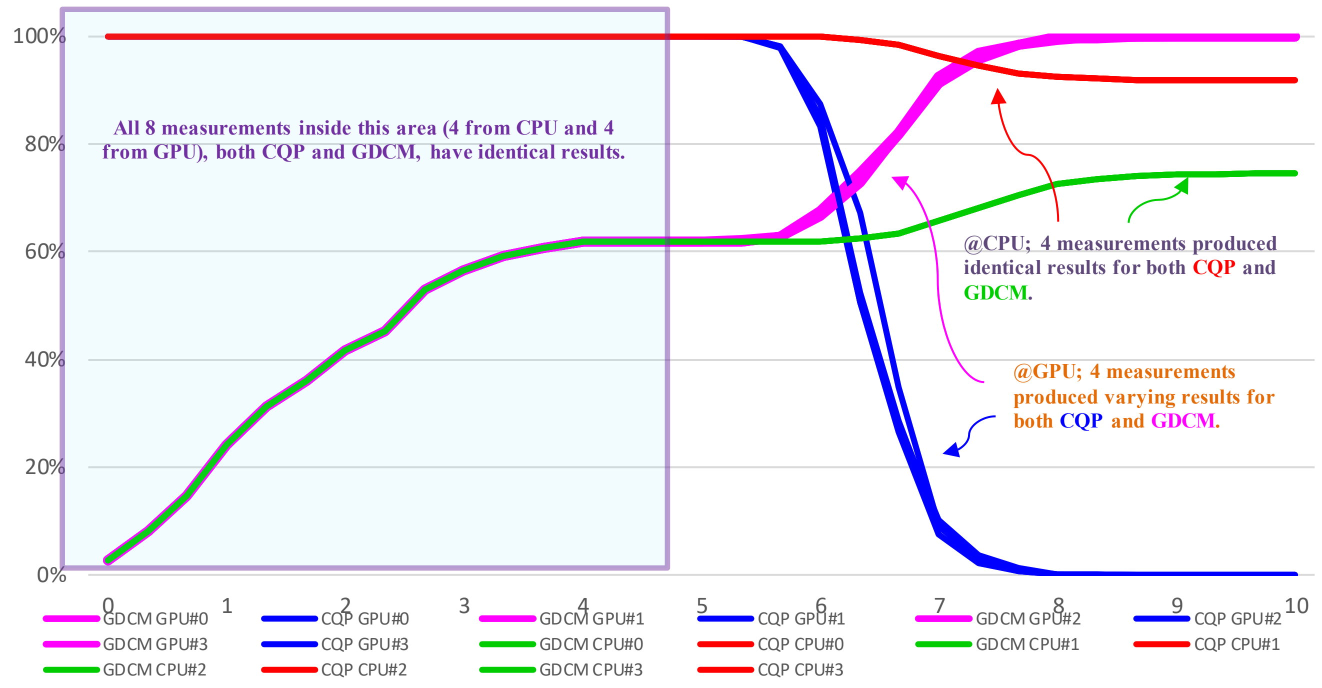

Fig. 8. CPU vs. GPU calculation quality measurement for 1NMC abstraction MOSFET using GIN.

Note: This comparison involves the effects of the programming language used, compiler version, operating system version, device driver version, device microcode version, and other internal processing unit factors. Currently, in this study, these factors cannot be separated and measured individually, so they will be represented in this study as a single variable: the CPU or GPU. If GNN OVR can be generated by applying variations of these factors, then individual measurements for these factors will be possible. Another important thing has also been observed from this comparison of the quality of CPU and GPU calculations: the difference in GNN calculation performance between CPU and GPU is also significantly influenced by GNN & dataset configuration factors, particularly the GNN type (GCN, GIN, GAT, GSG, or other GNN types), as observed in Fig. 5 and Fig. 6. Therefore, the results of this GNN calculation quality measurement can be used as a new opportunity in benchmarking the performance of GNN algorithms.

D. Limitation

The GNN calculation quality measurement in this study has limitations related to the precision of the measurement results. In conducting the measurements, the data used as the basis for calculating CQP, CQD, GDCM, and GDCV, are generated from the recognition of untrained random weight GNN. Therefore, the result of each random weight initialization (the process of creating an untrained GNN) at the beginning of the measurement will produce a small random effect on the measurement results (observed to have very significant random results in certain extreme configurations). The measurement results for 4 times initialization of untrained random weight GNN using the same configuration, namely GIN, 32-dim layer dimension, 1 hidden layer, GPU, are shown in Fig. 9. As well as the example for the measurement results for extreme case conditions using the configuration GIN, 2-dim layer dimension, 1 hidden layer, GPU, are shown in Fig. 10. One example of an extreme configuration that was found to produce very significant random output in this measurement of the quality of GNN calculations is a very low GNN layer dimension configuration.

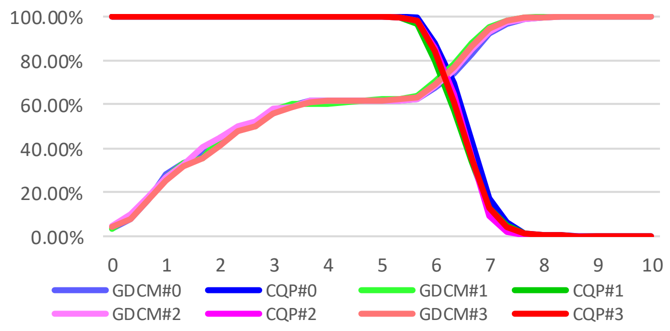

Fig. 9. Dataset and calculation quality measurement for 4N abstraction MOSFET with 32-dim.

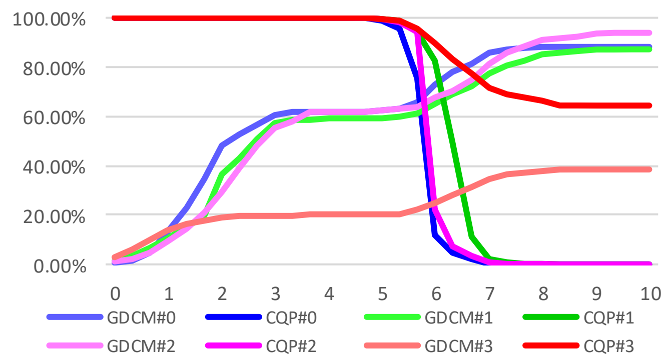

Fig. 10. Dataset and calculation quality measurement for 4N abstraction MOSFET with 2-dim.

IV. Conclusion

In this study, a method for measuring dataset quality and GNN calculation quality has been proposed. The measurement of dataset quality and GNN calculation quality is carried out by utilizing the previously proposed $prpDist$ and $prDiff$ calculations, calculating the $prpDist$ and $prDiff$ variables twice (exploiting the numerical reproducibility error in FP calculations) and comparing the differences between the two results to determine the quality of the GNN calculation. There are five variables generated from this proposed method. These variables are CQP, CQD, GDCM, GDCV, and DSA95. CQP and CQD are the results of measuring the quality of the GNN calculation, influenced by a combination of the quality of the FP calculation of the PU being used and the complexity of the GNN algorithm (GNN type) being applied. GDCM and GDCV are the results of a combination of dataset quality (the presence or absence of data duplication) and the sensitivity of the GNN to be able to distinguish each feature contained in the dataset. Meanwhile, DSA95 is the result of a combination of all factors that influence the previous four variables.

By combining multiple GNN configurations in the measurement, the data duplication value in the dataset can be more accurate than previously proposed methods. By combining certain GNN configurations, a comparison of the calculation capabilities between CPU and GPU can also be obtained. This comparison can be obtained based on measuring the initial location of the FP calculation noise after the decimal point. By using GIN and the 1NMC abstraction, calculations using the CPU are observed to be more precise and repeatable compared to using the GPU, so that the maximum amount of data that can be distinguished by GNN in a dataset for big data applications is 25 to 100 times larger.

Since all measurements were taken before the training process began (all measurements used a GNN with random weights that had not been trained and did not involve the training process and weight changes), and the measurement process did not utilize class information in the dataset, a correlation between the GNN calculation quality measurements and the GNN's performance after the training process (training accuracy and validation accuracy) could not be observed. Studying the behavior of measurement results for extreme configurations that can produce very significant random measurements is also the subject of future work.

Acknowledgment

We would like to express our gratitude to Mrs. Toyama for all the support she provided, including the research environment.

References

[1] J. Lu, et all, “Automatic Op-Amp Generation From Specification to Layout,” in IEEE Transactions on Computer-Aided Design ff Integrated Circuits And Systems, vol. 42, no. 12, December 2023.

[2] Z. Zhao, L. Zhang, “Analog Integrated Circuit Topology Synthesis With Deep Reinforcement Learning,” IEEE Transactions on Computer-Aided Design ff Integrated Circuits And Systems, vol. 41, no. 12, December 2022.

[3] Z. Zhao, J. Luo, J. Liu and L. Zhang, "Signal-Division-Aware Analog Circuit Topology Synthesis Aided by Transfer Learning," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 11, pp. 3481-3490, Nov. 2023.

[4] I. Guven, M. Parlak, D. Lederer and C. De Vleeschouwer, "AI-Driven Integrated Circuit Design: A Survey of Techniques, Challenges, and Opportunities," in IEEE Access, vol. 13, pp. 167364-167389, 2025.

[5] K. Arimura, R. Miyauchi and K. Tanno, “Systematic Offset Voltage Reduction Methods Using Half-Circuit of Input Stage in the Two-Stage CMOS Operational Amplifiers and Comparators,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2025

[6] F. Jiao, S. Montano, C. Ferent, A. Doboli, and S. Doboli, “Analog circuit design knowledge mining: Discovering topological similarities and uncovering design reasoning strategies,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 34, no. 7, pp. 1045–1058, Jul. 2015.

[7] H. Li, X. Liu, F. Jiao, A. Doboli, and S. Doboli, “InnovA: A cognitive architecture for computational innovation through robust divergence and its application for analog circuit design,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 10, pp. 1943–1956, Oct. 2018.

[8] A. A. Mannan, K. Tanno, “Graph Neural Network Output for Dataset Duplication Detection on Analog Integrated Circuit Recognition System,” International Journal of Advanced Computer Science and Applications, vol. 16, no. 5, 2025.

[9] L. Ozyilmaz, T. Yildirim, “Sensitivity analysis for conic section function neural networks,” Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000.

[10] T. Xu, A. L. Cox and S. Rixner, "Design and accuracy trade-offs in Computational Statistics," 2025 IEEE International Symposium on Workload Characterization (IISWC), Irvine, CA, USA, 2025.

[11] Tiong-Hwee Goh, "Semantic extraction using neural network modelling and sensitivity analysis," Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), Nagoya, Japan, 1993.

[12] K. Hakhamaneshi, M. Nassar, M. Phielipp, P. Abbeel and V. Stojanovic, “Pertaining Graph Neural Network for Few-Shot Analog Circuit Modeling and Design,” IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems, Vol. 42, NO. 7, JULY 2023.

[13] T. N. Kipf, M. Welling, “Semi-supervised classification with graph convolutional networks,” 5th International Conference on Learning Representations (ICLR), 2017.

[14] K. Xu, W. hu, J. Leskovec and S. Jegelka, ”How Powerful are Graph Neural Networks?,” International Conference on Learning Representations (ICLR) 2019.

[15] S. Brody, U. Alon and E Yahav, “How Attentive are Graph Attention Networks?,” The Tenth International Conference on Learning Representations (ICLR), 2022.

[16] W. L. Hamilton, R. Ying and J. Leskovec, “Inductive representation learning on large graphs,” 31st Annual Conference on Neural Information Processing Systems, NeurIPS, 2017, pp. 1025–1035.

[17] "IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2019 (Revision of IEEE 754-2008) , pp.1-84, 22 July 2019.

[18] S. Chowdhary, S. Nagarakatte, “Parallel shadow execution to accelerate the debugging of numerical errors,” Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.615–626, 2021.

[19] S. Chowdhary, S. Nagarakatte “Fast shadow execution for debugging numerical errors using error-free transformations,” Proceedings of the ACM on Programming Languages 6 (OOPSLA2), pp. 1845–1872, 2022.

[20] Y. Tan, et all, “Computing Floating-Point Errors by Injecting Perturbations,” arXiv:2507.08467v1 [cs.SE] 11 Jul 2025.

[21] P. Langlois, R. Nheili and C. Denis, "Numerical reproducibility: Feasibility issues," 2015 7th International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2015.

[22] S. Shanmugavelu, et all, “Impacts of floating-point non-associativity on reproducibility for HPC and deep learning applications,” SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, GA, USA, pp. 170-179, 2024.

[23] S. F. J. Apostal, D. Apostal and R. Marsh, "Containers and Reproducibility in Scientific Research," 2018 IEEE International

Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 2018.

[24] H. Al-Rikabi, B. Renczes, "Floating-Point Roundoff Error Analysis in Artificial Neural Networks," 25th IMEKO TC-4 International Symposium on Measurement of Electrical, Brescia, Italy / September 12-14, 2022.

[25] T. Hadjam, A. M. Salah, M. B. Nedjma, M. Abdelmadjid and H. Hamil, "FPGA implementation of a Convolutional Neural Network for image classification," 2022 2nd International Conference on Advanced Electrical Engineering (ICAEE), Constantine, Algeria, 2022.

[26] J. Yun, B. Kang and Z. Fu, "The Hidden Power of Pure 16-bit Floating-Point Neural Networks," arXiv:2301.12809v2 [cs.LG] 3 May 2024.

[27] A. A. Mannan, K. Tanno, "Netlist Feature Extraction for CMOS Analog Circuit Design Warning System," 23rd International Conference on Machine Learning and Cybernetics (ICMLC), pp. 127-132, 2024.

AUTHORS' PROFILE

Arif Abdul Mannan was born in Malang, Indonesia, on January 23, 1988. Currently, he is a PhD student in University of Miyazaki, Japan. He received S.T degrees from the Faculty of Engineering, Brawijaya University in 2010. He also received master's degrees from Double-Degree Program of Faculty of Engineering in Brawijaya University and Miyazaki University for his M.T and M.E degree in 2013. His research interests are circuit analysis on digital circuits (microprocessor design and application, digital CMOS integrated circuit design, FPGA & VHDL), and analog circuit (analog CMOS integrated circuit design including multi-valued logic). He can be contacted at email: arifabdulmannan@ub.ac.id.

Koichi Tanno was born in Miyazaki, Japan, on April 22, 1967. He received B.E. and M.E. degrees from the Faculty of Engineering, University of Miyazaki, Miyazaki, Japan, in 1990 and 1992, respectively, and Ph. D degree from the Graduate School of Science and Technology, Kumamoto University, Kumamoto, Japan, in 1999. From 1992 to 1993, he joined the Microelectronics Products Development Laboratory, Hitachi, Ltd., Yokohama, Japan. He contributed to the research on low-voltage and low-power equalizers for reading channel LSI of hard disk drives. In 1994, he joined the University of Miyazaki, where he is currently a Professor in the Faculty of Engineering, and a Vice-president (Collaborative Research and Community Cooperation). His main research interests are in analog integrated circuit design, nano-mist sprayers, and its application. Dr. Tanno is a senior member of IEEE. He can be contacted at email: tanno@cc.miyazaki-u.ac.jp.