NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

AI Techniques in IC Image and Netlist Analysis for Hardware Assurance

Bah-Hwee Gwee
ebhgwee@ntu.edu.sg

Nanyang Technological University, Singapore
27 Mar 2024

Outline

- Introduction
- IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification
- Conclusions

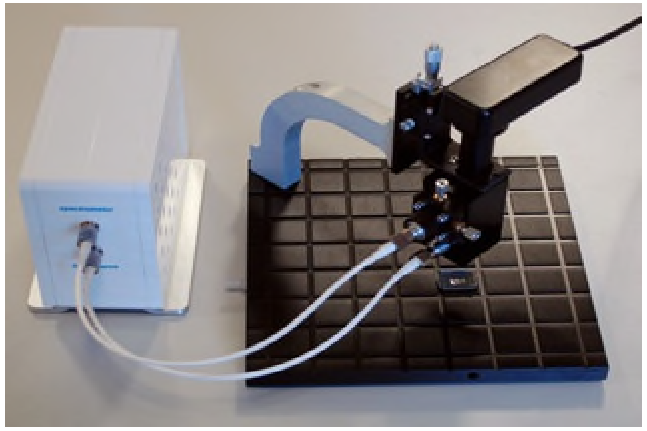
Outline

Introduction – IC Circuit Extraction

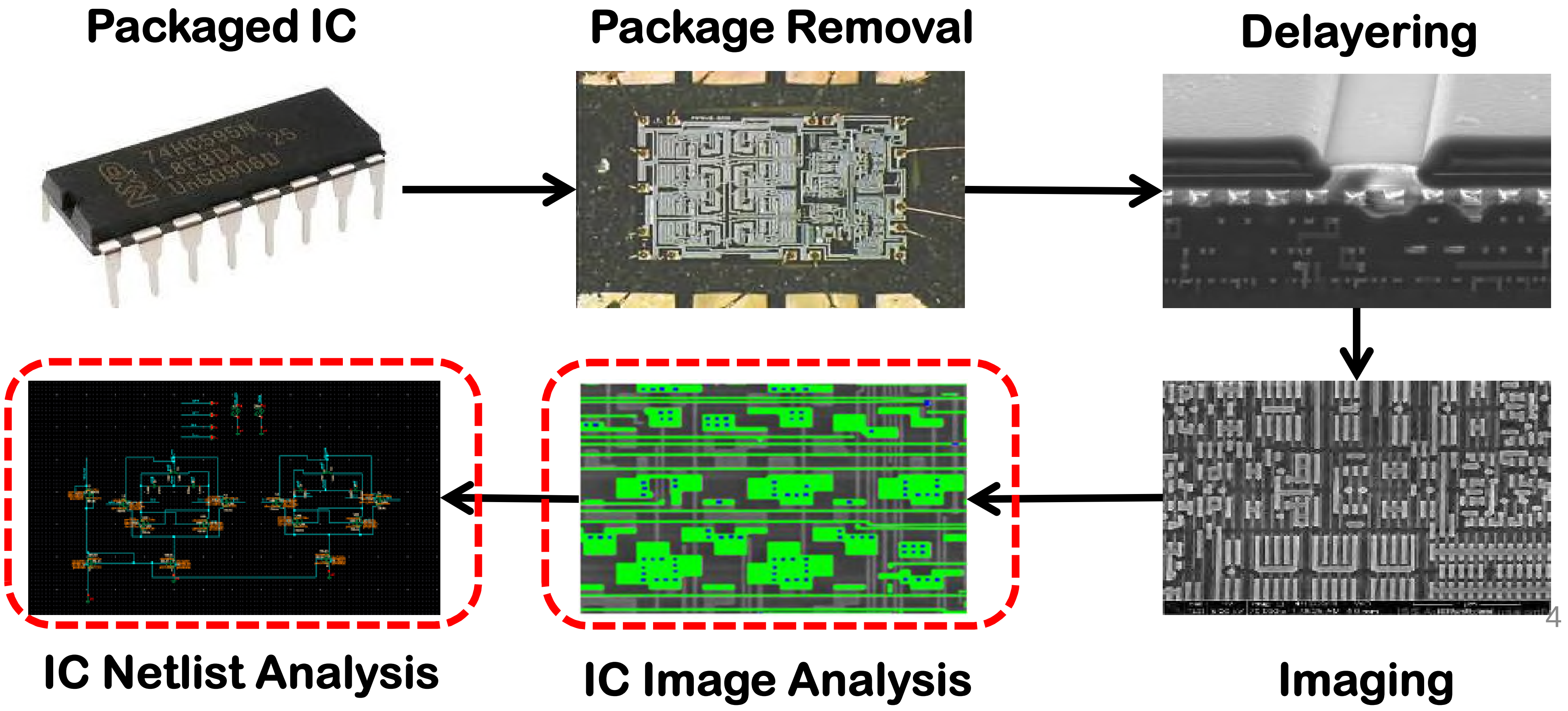
- The objectives of IC Circuit Extraction:
  - Intellectual Property (IP) infringement investigation;
  - Detection of malicious hardware, e.g. hardware Trojans;
  - Hardware failure analysis

IP Infringement

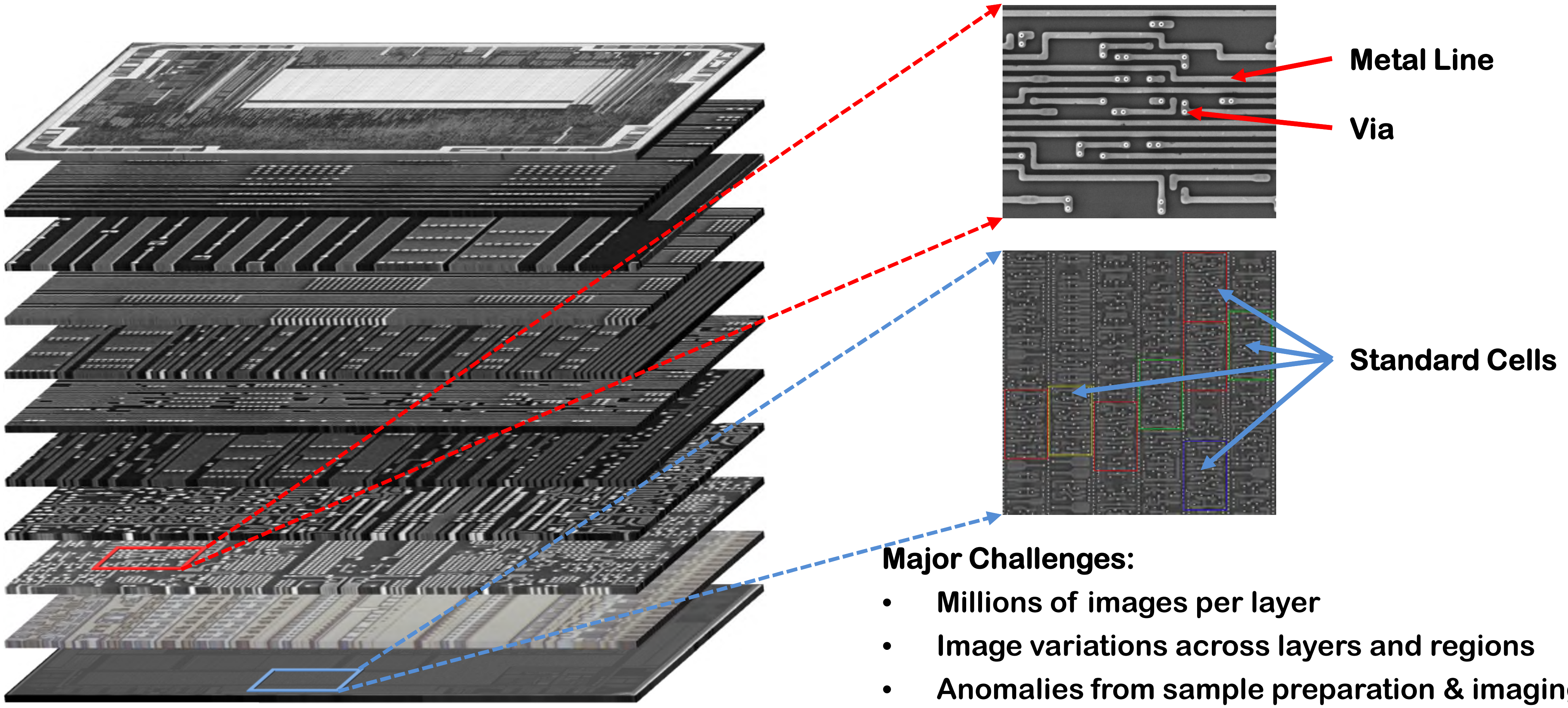
Hardware Trojans


Failure Analysis

Introduction – IC Circuit Extraction



Introduction – Delayered IC Images



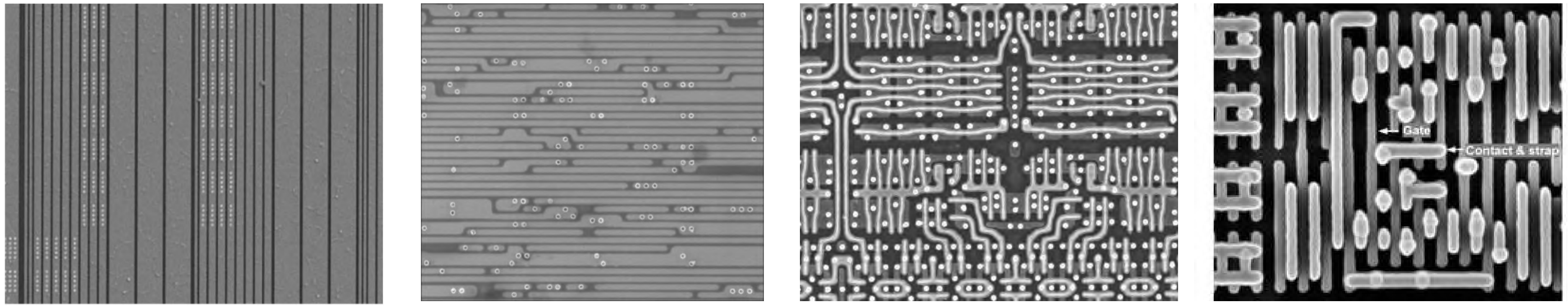
Major Challenges:

- Millions of images per layer
- Image variations across layers and regions
- Anomalies from sample preparation & imaging
- Small feature size / narrow gap between features

Outline

IC Image Analysis – Tasks

- Tasks for IC image analysis
  - Image stitching (and results evaluation)
  - Feature extraction (segmentation & object detection)
  - Image stacking (and netlist generation)



Sample delayered IC images taken by SEM (Scanning Electron Microscopy) from different chips, different layers, and with different imaging settings.

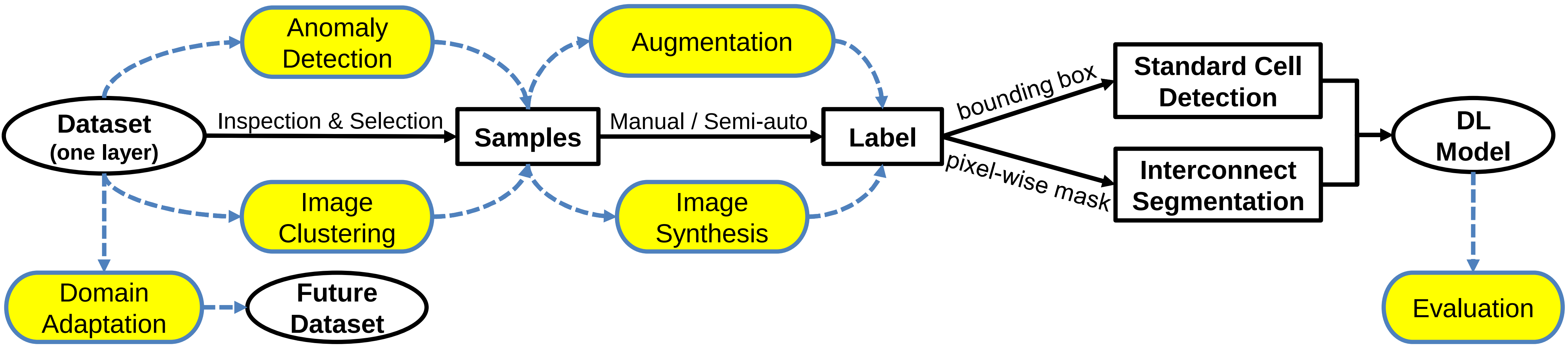
IC Image Analysis – Goals

- Techniques that are less human-dependent
  - The human involvement should be limited.
- Techniques that are not data-hungry
  - Less data labelling is desired.
- Techniques that are adaptable
  - Continuous data labelling for different chips, layers and imaging settings, is not acceptable.

Deep Learning (DL) in IC Image Analysis

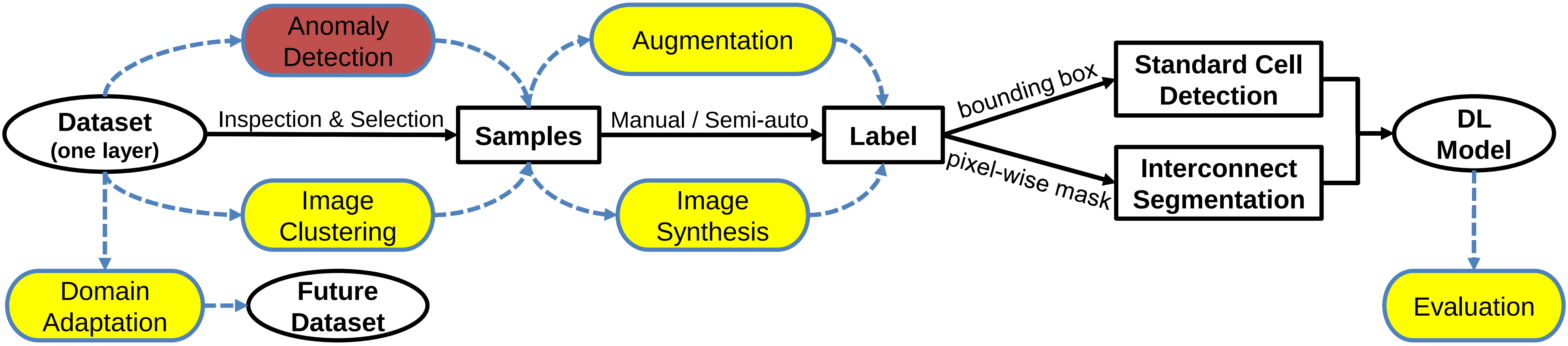
- Advantages of Deep Learning (DL)-based Methods
  - ❖ Fast due to inherent leverage on parallel processing hardware (e.g. CNNs on GPUs)
  - ❖ Learning extraction rules from large amount of data ensures robust extraction against image noise
  - ❖ End-to-end training and inference allows realization of complex tasks

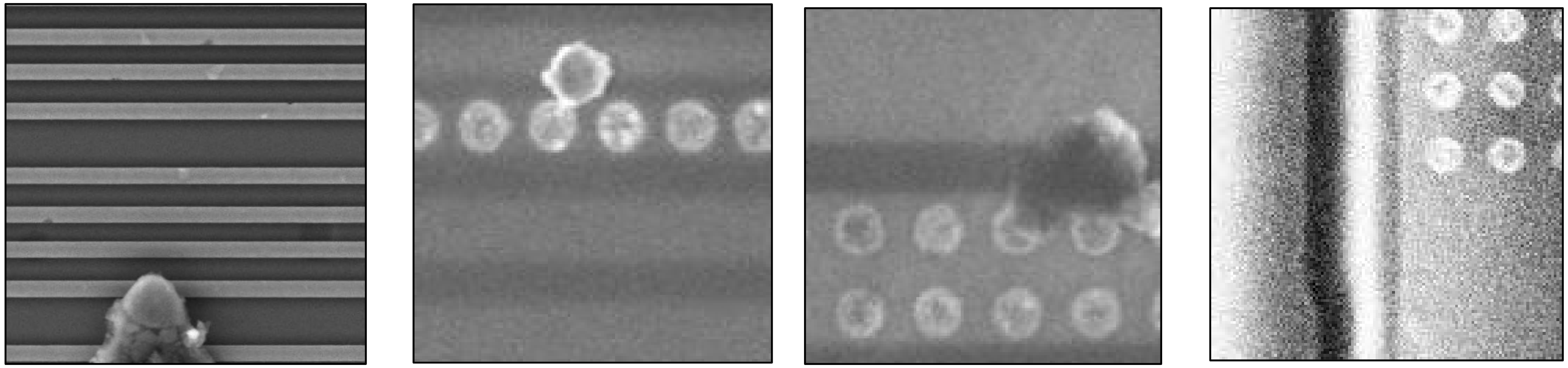
Data Analysis and Preparation in DL-Based IC Image Analysis



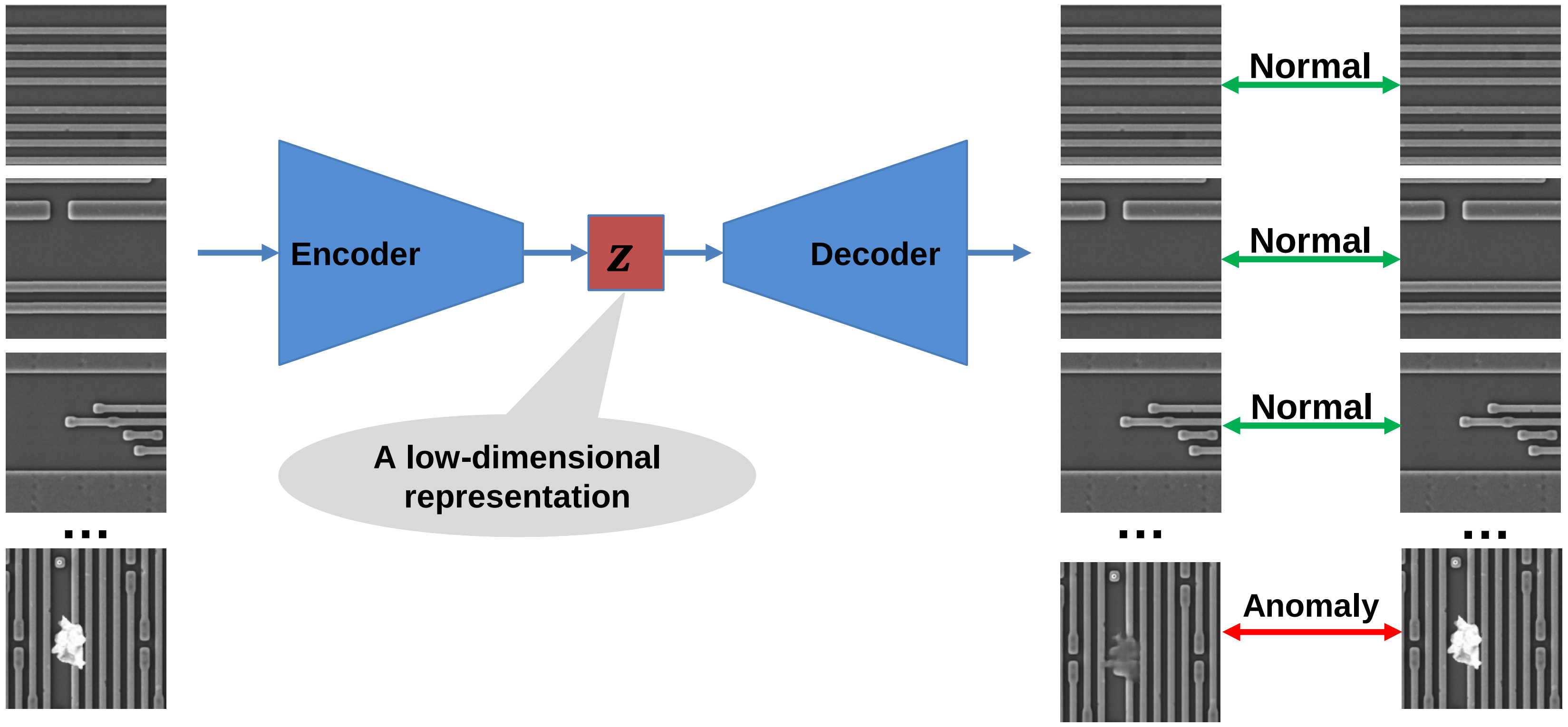
Outline

Anomalies in Image Data

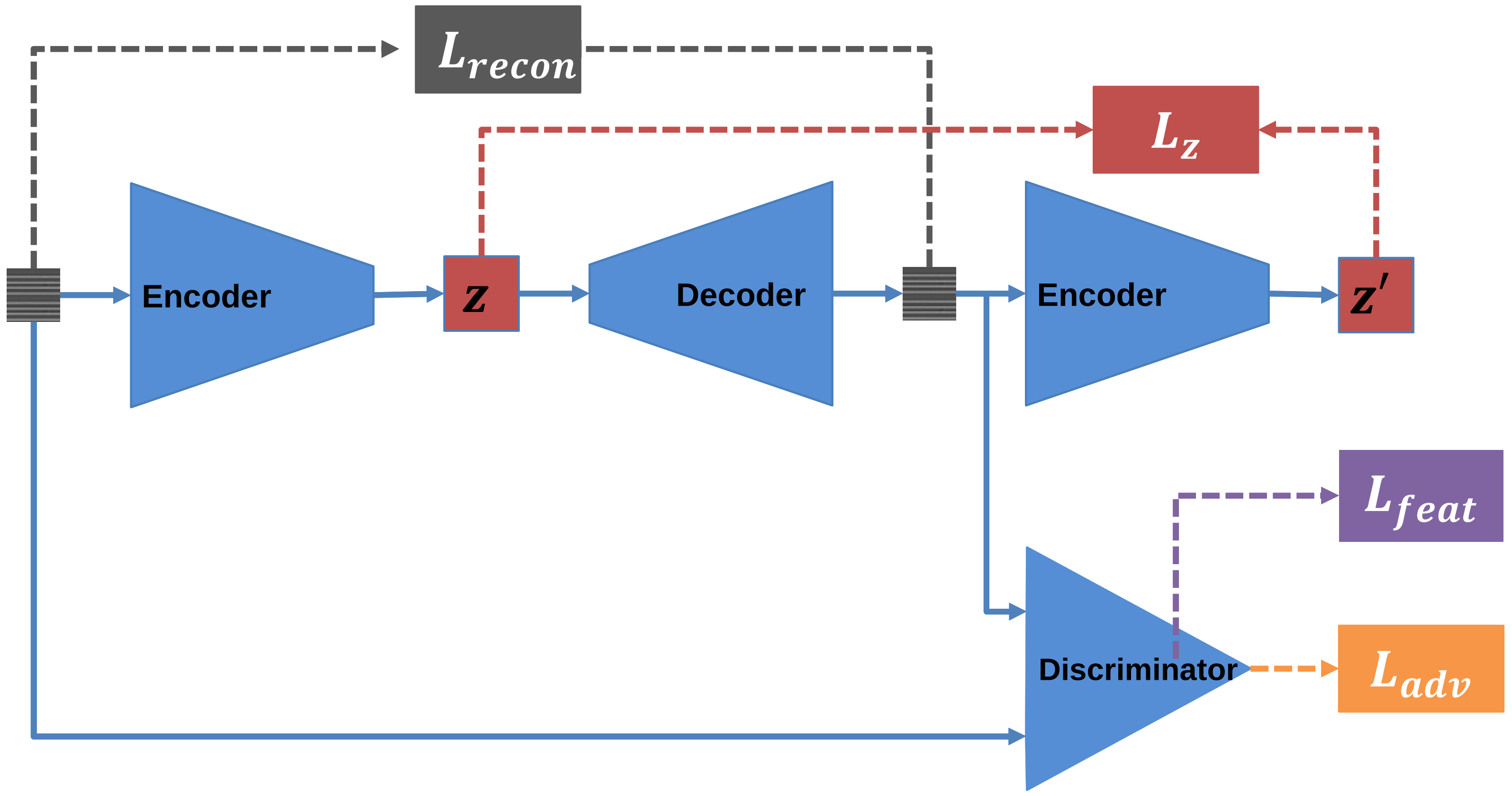




Self-supervised Anomaly Detection with GAN (Generative Adversarial Networks)



Self-supervised Anomaly Detection with GAN



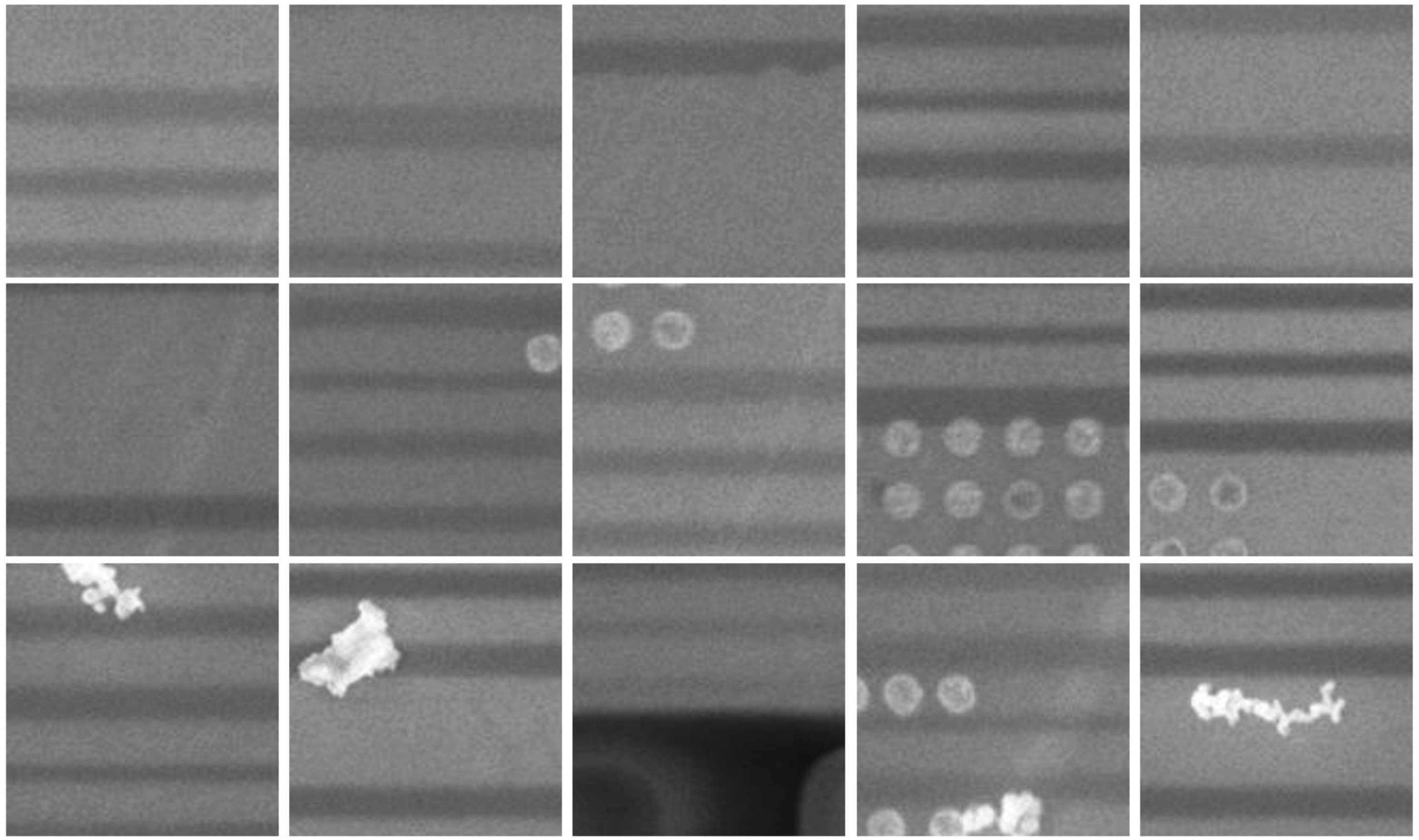
Training Stage:

- Alternating training between Encoder/Decoder (Generator) and Discriminator
- Generator is optimized with weighted sum of 4 loss terms
- Discriminator is optimized with adversarial loss

Testing Stage:

- Reconstruction loss, z-loss, and feature loss are computed for patches of input images
- 3 loss values are normalized as anomaly scores to determine anomalous images.

Self-supervised Anomaly Detection: Score Ranking

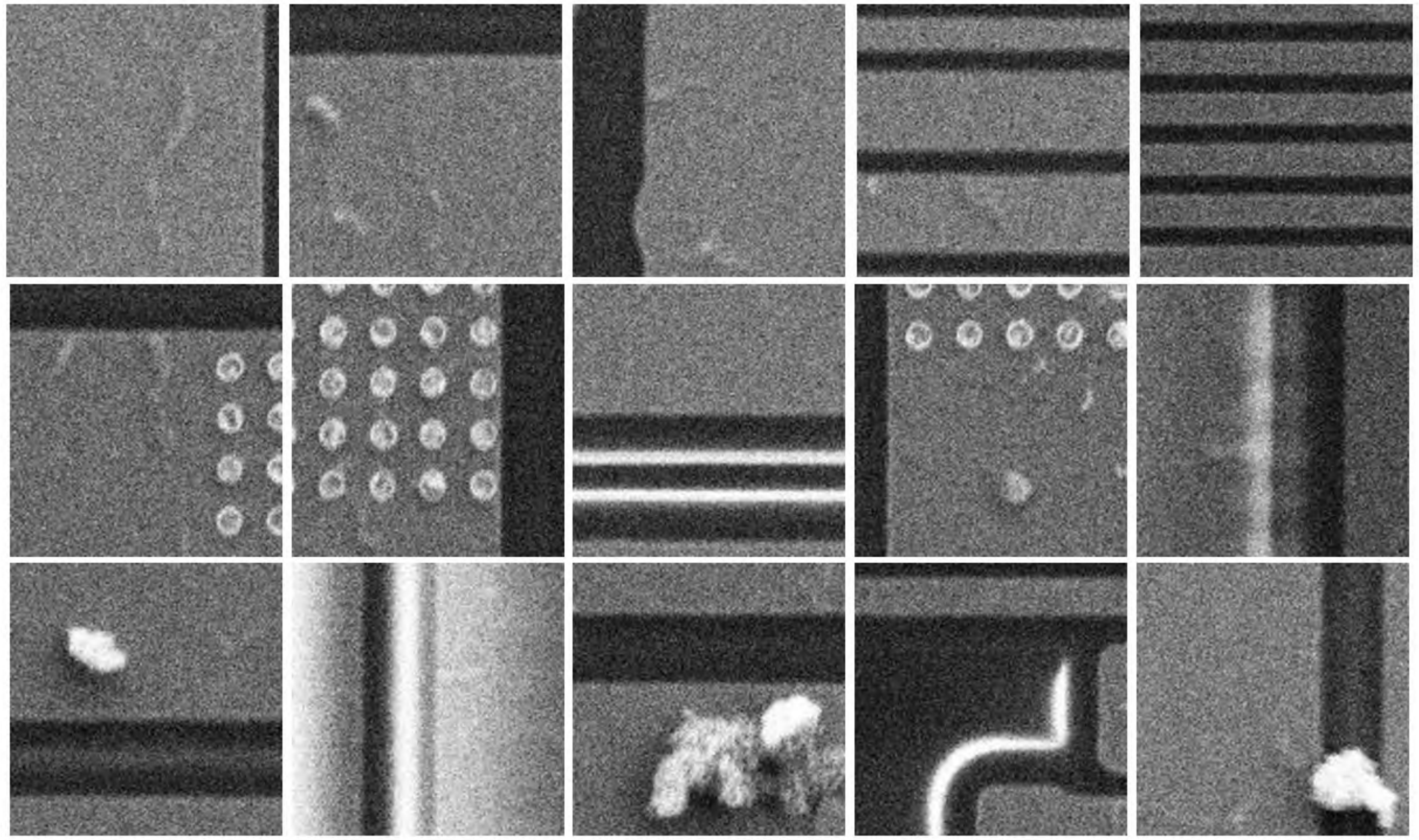


Low anomaly score

Medium anomaly score

High anomaly score

Self-supervised Anomaly Detection: Score Ranking

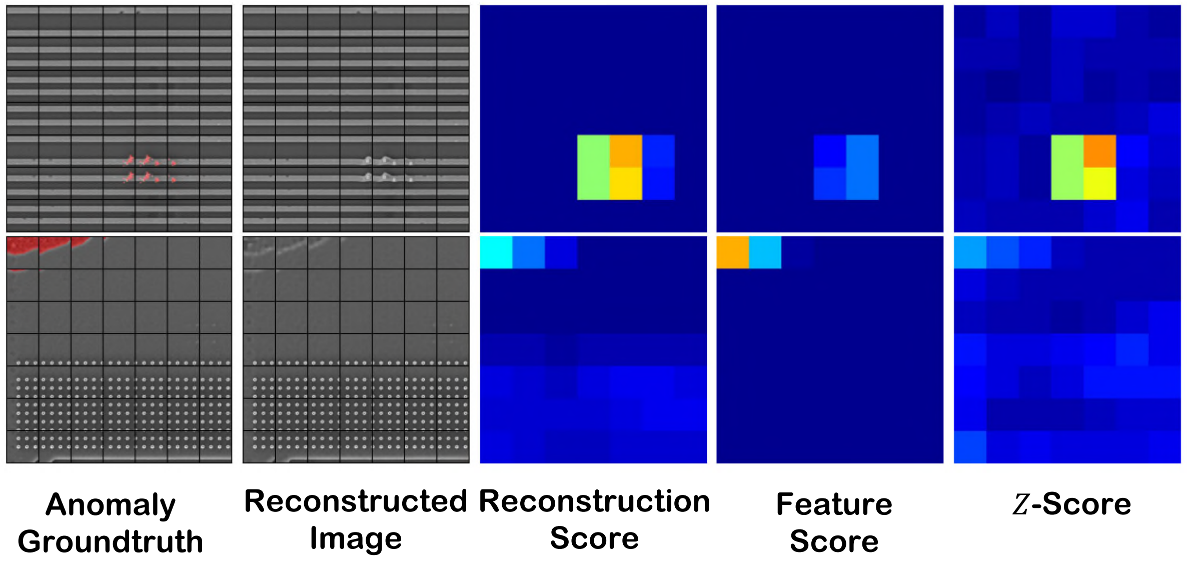


Low anomaly score

Medium anomaly score

High anomaly score

Self-supervised Anomaly Detection: Accuracy



Anomalous regions are highlighted by high loss values (anomaly score).

Method	AUC	F1	TPR	FPR
ResNet f-anoGAN [17]	0.5623	0.3013	0.1896	0.0063
ConvNet f-anoGAN	0.5638	0.3165	0.1896	0.0008
GANomaly [18]	0.9334	0.7464	0.6724	0.0118
Ours (IAD only)	0.9728	0.8348	0.7845	0.0086

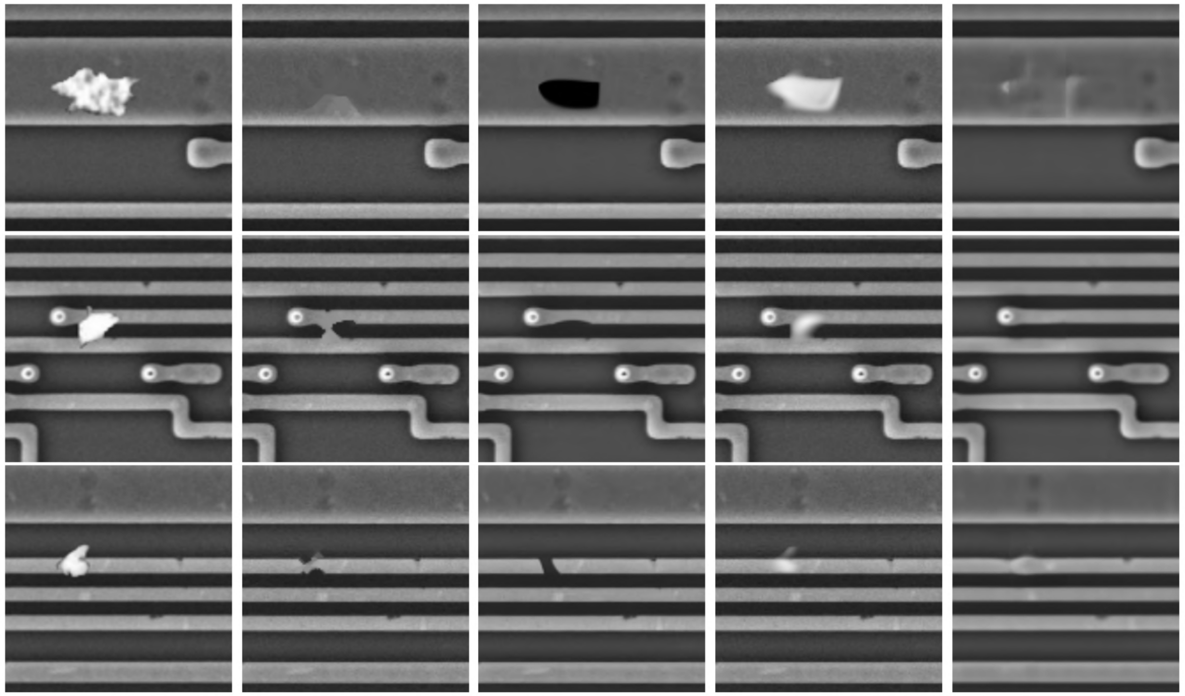
Better performance than reported methods, without supervision/data labeling.

[1] L. Huang, D. Cheng, X. Yang, T. Lin, Y. Shi, K. Yang, B.-H. Gwee, and B. Wen, "Joint Anomaly Detection and Inpainting for Microscopy Images via Deep Self-supervised Learning," *IEEE International Conference on Image Processing (ICIP)*, 2021.

Joint Task on Anomaly Detection & Inpainting

Concurrent anomaly detection and inpainting

By adding pairs of corrupted and corresponding clean images into training



Model	PSNR	SSIM
IDBP [26]	31.3390	0.9575
IRCNN [6]	31.2245	0.9586
TSLRA [5]	30.4554	0.9563
Ours (IAD + Inpainting)	**34.4798**	**0.9627**

Good performance on image inpainting

Method	AUC	F1	TPR	FPR
ResNet f-anoGAN [17]	0.5623	0.3013	0.1896	0.0063
ConvNet f-anoGAN	0.5638	0.3165	0.1896	0.0008
GANomaly [18]	0.9334	0.7464	0.6724	0.0118
Ours (IAD only)	0.9728	0.8348	0.7845	0.0086
Ours(IAD+Inpainting)	**0.9927**	**0.9123**	0.8966	0.0063

Further improve image anomaly detection

[1] L. Huang, D. Cheng, X. Yang, T. Lin, Y. Shi, K. Yang, B.-H. Gwee, and B. Wen, "Joint Anomaly Detection and Inpainting for Microscopy Images via Deep Self-supervised Learning," *IEEE International Conference on Image Processing (ICIP)*, 2021.

Outline

DL-Based IC Image Analysis Flow

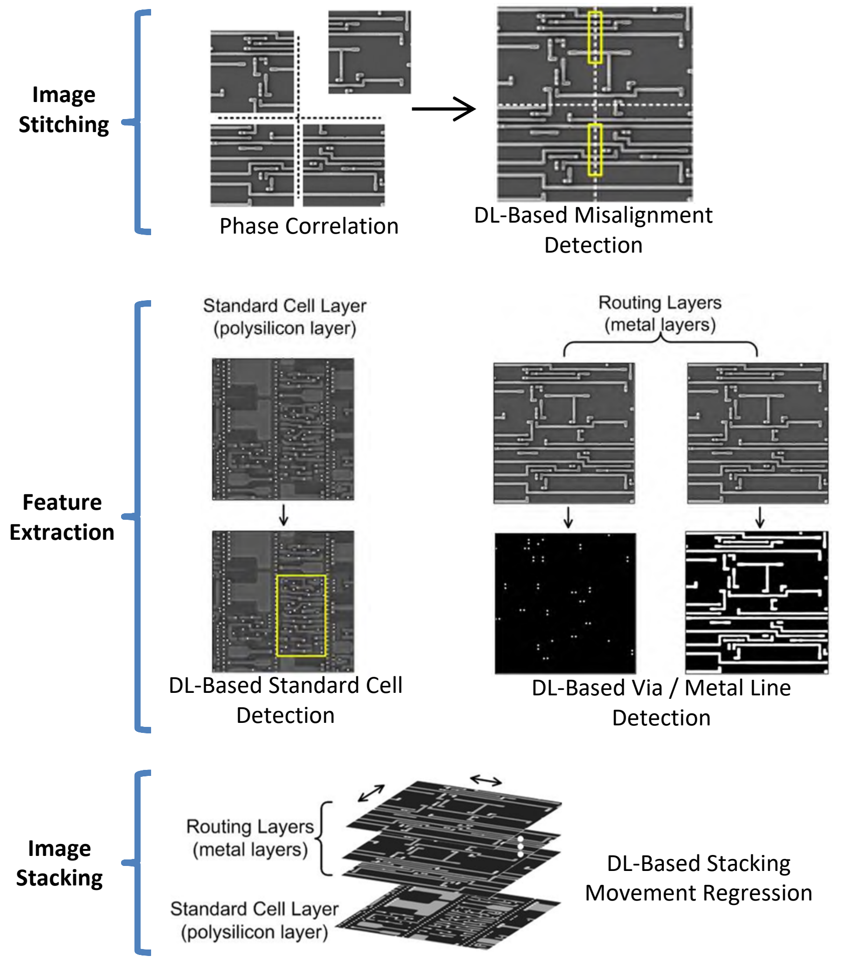


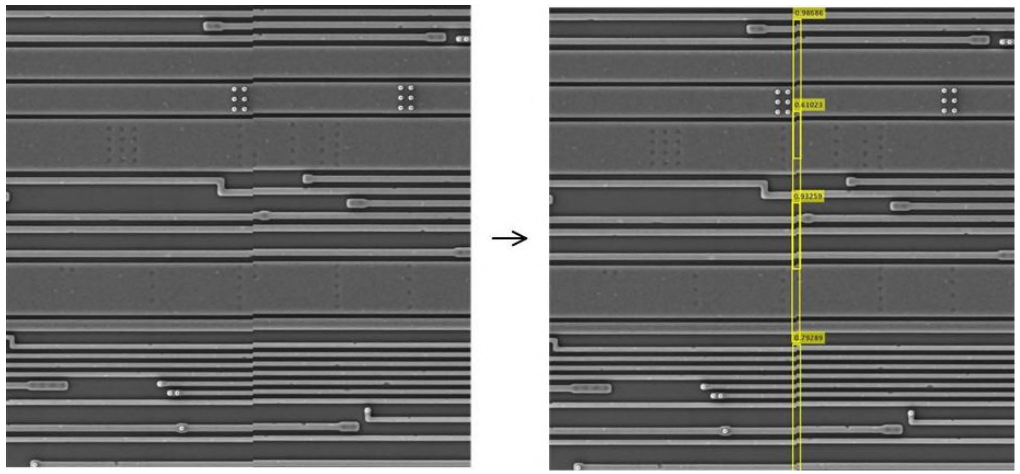
Image Stitching
Phase Correlation
DL-Based Misalignment Detection
Feature Extraction
Standard Cell Layer (polysilicon layer)
Routing Layers (metal layers)
DL-Based Standard Cell Detection
DL-Based Via / Metal Line Detection
Image Stacking
Routing Layers (metal layers)
Standard Cell Layer (polysilicon layer)
DL-Based Stacking Movement Regression

- **Image Stitching Stage:**
  - ❖ **Stitch SEM images using phase correlation**
  - ❖ **Use reported DL object detection model to check stitching results and detect misalignment**
  - ❖ **Use a fully automated method to prepare synthetic training data for detection**

- **Feature Extraction Stage:**
  - ❖ **Use reported DL object detection model to detect standard cells**
  - ❖ **Use reported DL semantic segmentation model to segment contacts, vias and metal lines**
  - ❖ **Use a fully automated method to prepare synthetic training data for detection and a semi-automated method to prepare training data for segmentation**

- **Image Stacking Stage:**
  - ❖ **Use custom DL regression model to estimate stacking movement**
  - ❖ **Use a fully automated method to prepare synthetic training data for regression**

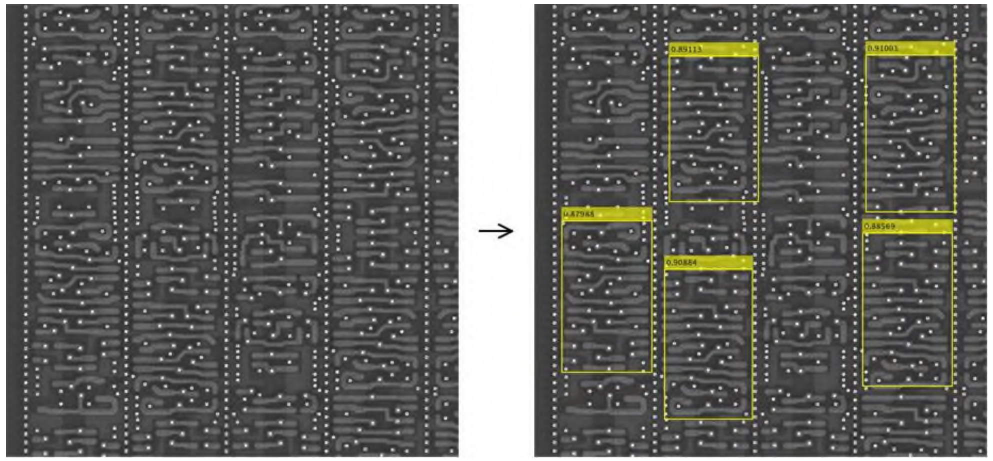
Experiment Results: Misalignment Detection



Sample DL-based Misalignment Detection Result

- **DL-based Misalignment Detection**
  - ❖ Stitching misalignments > ~2 pixels were correctly identified by our DL object detection model with high accuracy
  - ❖ Fast processing speed: ~0.5 seconds to process an image of size 1,600×1,600 pixels (on GPU)

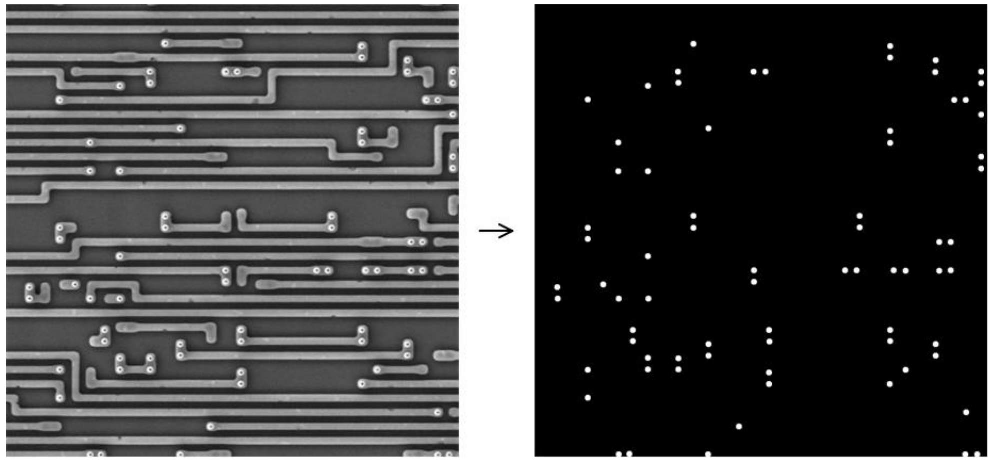
Experiment Results: Standard Cell Detection



Sample DL-based Standard Cell Detection Result

- DL-based Standard Cell Detection
  - ❖ Multiple instances of standard cells were correctly identified by our DL object detection model with high accuracy
  - ❖ Fast processing speed: ~3 seconds to process an image of size 10,000×10,000 pixels (on GPU)

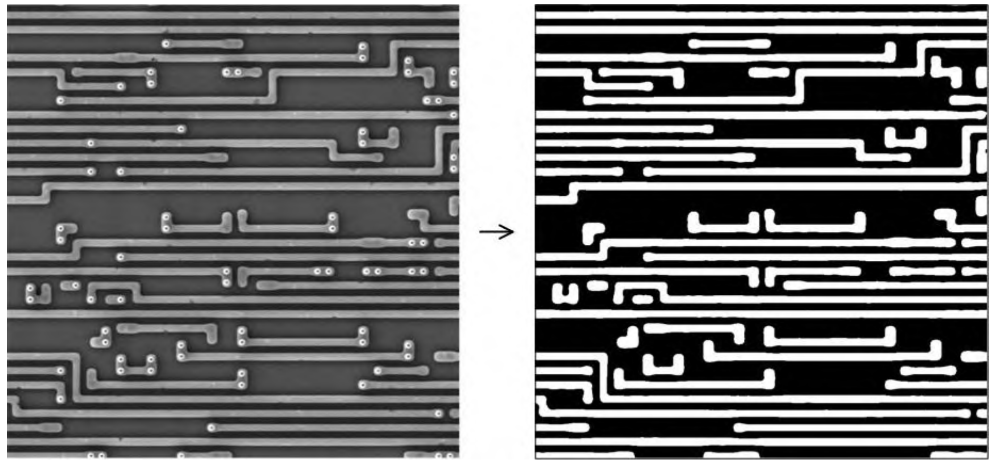
Experiment Results: Via/Contact Segmentation



Sample DL-based Via Segmentation Result

- **DL-based Via/Contact Segmentation**
  - ❖ Vias and contacts were correctly segmented by our DL semantic segmentation model; models achieved high pixel accuracy >97%
  - ❖ Fast processing speed: ~0.4 seconds to process an image of size 1,024×1,024 pixels (on GPU)

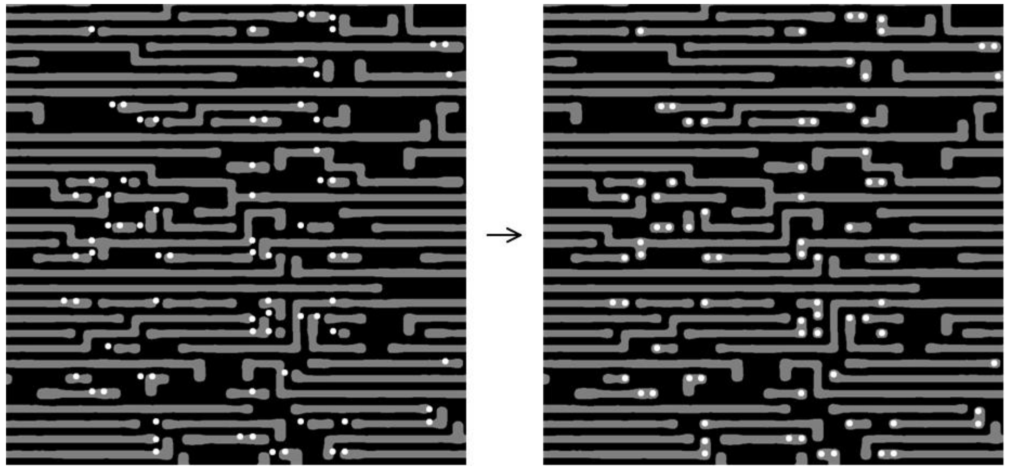
Experiment Results: Metal Line Segmentation



Sample DL-based Metal Line Segmentation Result

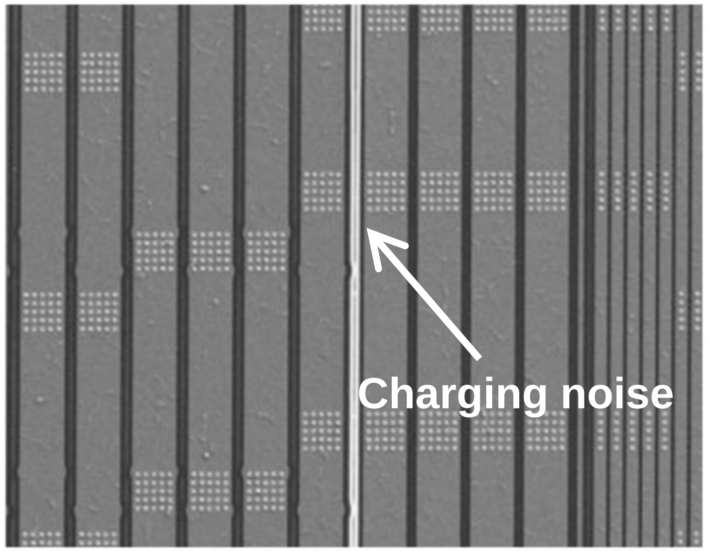
- **DL-based Metal Line Segmentation**
  - ❖ **Metal lines were correctly segmented by our DL semantic segmentation model; models achieved high pixel accuracy >97%**
  - ❖ **Fast processing speed: ~0.4 seconds to process an image of size 1,024×1,024 pixels (on GPU)**

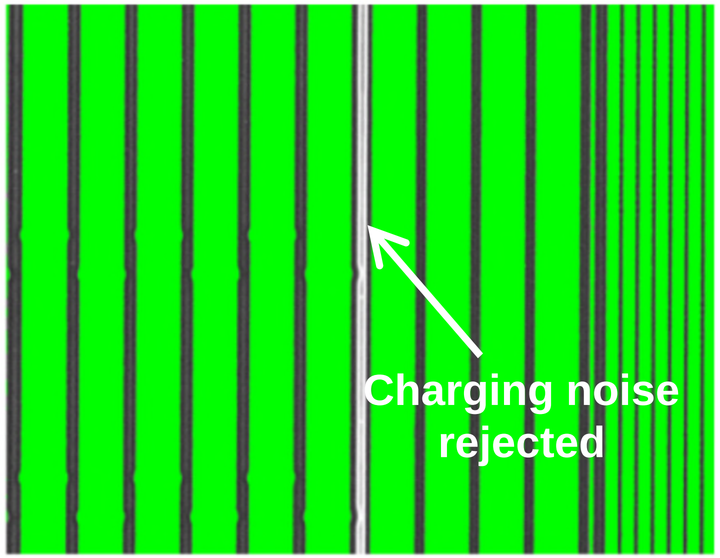
Experiment Results: Image Stacking


Sample DL-based Image Stacking Result

- **DL-based Image Stacking**
  - Stacking movements were correctly estimated by our DL regression model and vias from the lower layer were correctly aligned to metal lines from the upper layer; can move up to 50 pixels in both directions
  - Fast processing speed: ~0.8 seconds to process an image of size 980×980 pixels (on GPU)

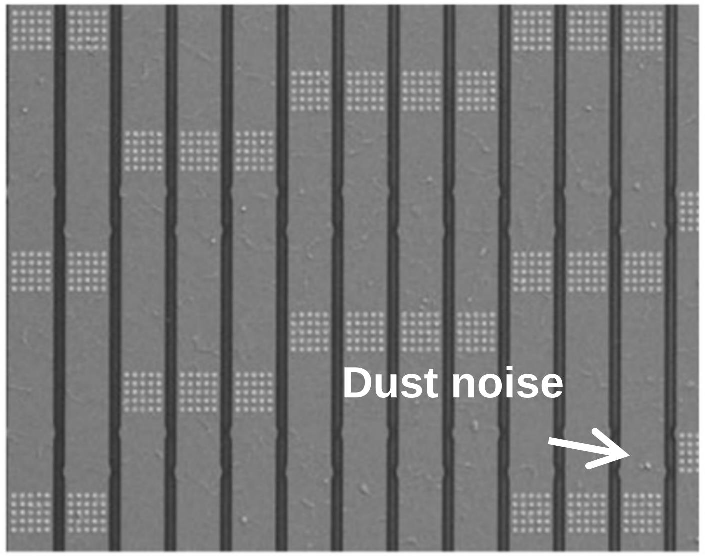
Experiment Results: Robust Extraction against Image Noises

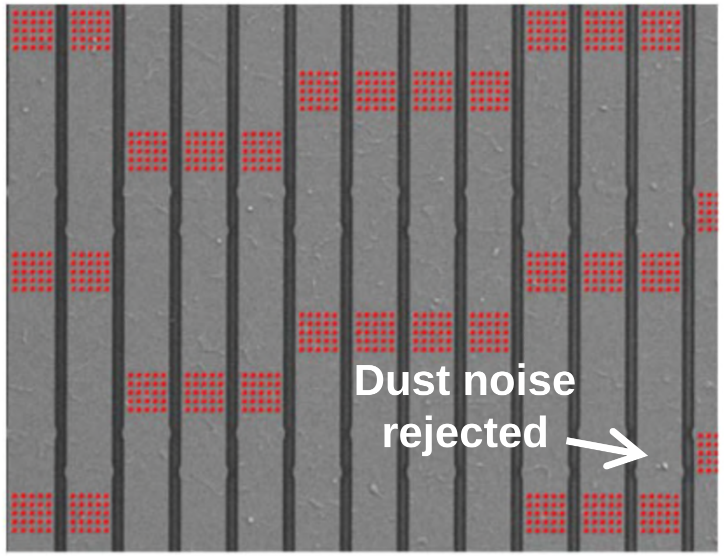




❖ **Metal line segmentation - Charging Noise was correctly rejected by our DL model**

Experiment Results: Robust Extraction against Image Noises





❖ Via segmentation result - Dust Noise was correctly rejected by our DL model

Experiment Results: Comparison of DL-based Method with Classical Image Processing Techniques

Comparison Results of Via Annotation Errors Using DL Model and Image Processing Techniques (CHT = Circular Hough Transform)

Method / Errors/image	using DL model	using image processing technique	
		CHT Sensitivity=0.85	CHT Sensitivity=0.9
FP	0.47	6.47	14.21
FN	0.02	11.36	2.57

❖ Achieved much lower False Positive (FP) and False Negative (FN) Errors

[2] T. Lin, Y.-Q. Shi, N. Shu, D.-R. Cheng, X.-N. Hong, J.-S. Song, and B.-H. Gwee, "Deep Learning-Based Image Analysis Framework for Hardware Assurance of Digital Integrated Circuits," *Microelectronics Reliability*, 2021.

Experiment Results: Comparison of DL-based Method with Classical Image Processing Techniques

Comparison Results of Metal Line Annotation Errors Using DL Model and Image Processing Techniques

Method / Errors/image	using DL model	using image processing technique	
		Median filtering	Median filtering
		Neighbourhood size=12	Neighbourhood size=15
Short-circuit errors	0.83	4.51	2.57
Open-circuit errors	0.26	0.26	0.28

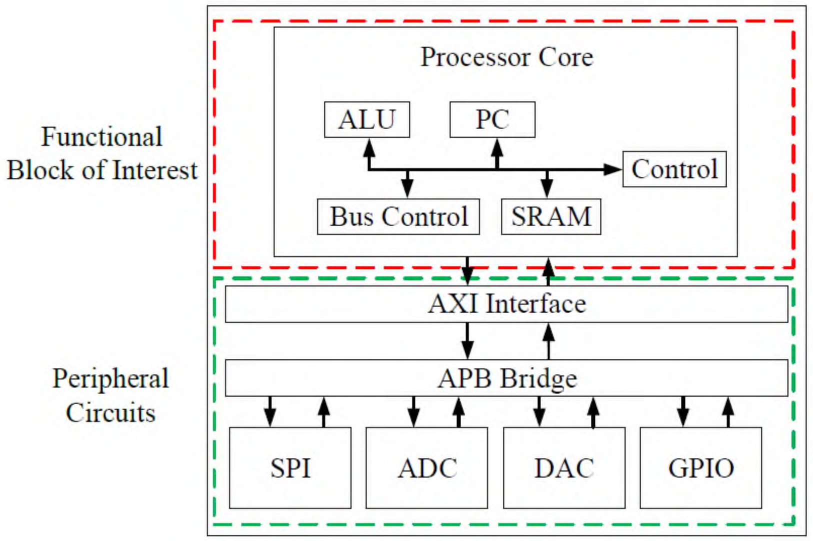
❖ Achieved much lower Short-Circuit and Open-Circuit Errors

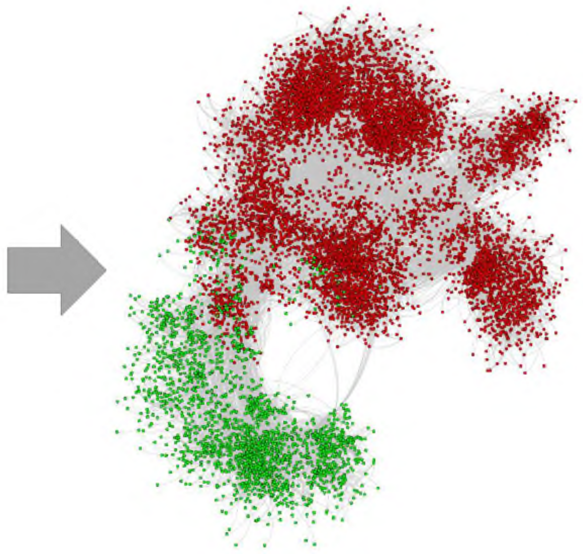
[2] T. Lin, Y.-Q. Shi, N. Shu, D.-R. Cheng, X.-N. Hong, J.-S. Song, and B.-H. Gwee, "Deep Learning-Based Image Analysis Framework for Hardware Assurance of Digital Integrated Circuits," *Microelectronics Reliability*, 2021.

Outline

IC Netlist Analysis – Tasks

- **Modern SoC netlists consist of many functional blocks and sub-circuits:**
  - Difficult to analyse as a whole.
  - Not all functional blocks or sub-circuits are of interest.
- **A 'divide-and-conquer' approach is usually adopted, which consists of:**
  - **Netlist Partition**: to partition a large circuit netlist into smaller sub-circuits.
  - **Netlist Identification**: to identify the functionality of a sub-circuit.





'Divide-and-conquer' approach

Outline

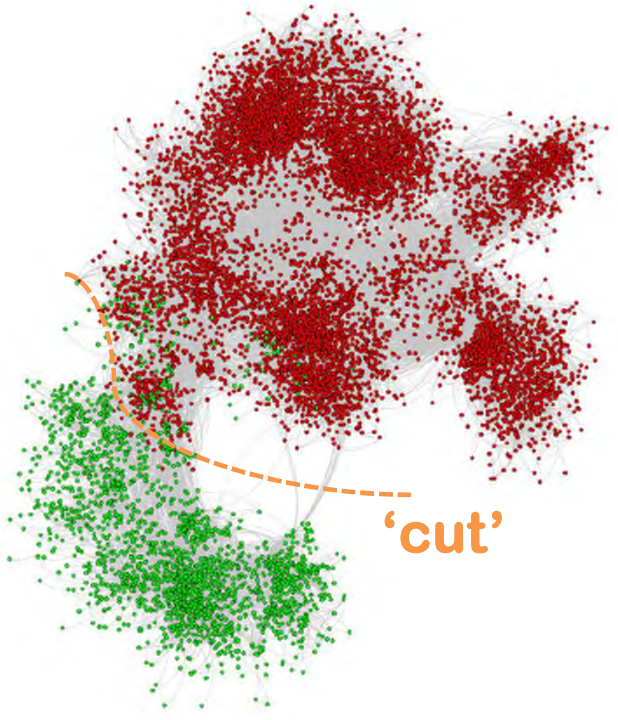
- Introduction
- IC Image Analysis
  - Self-Supervised Anomaly Detection
  - DL-Based Image Analysis Flow
- IC Netlist Analysis
  - Netlist Partition
  - Netlist Identification
- Conclusions

Netlist Partition: The Problem

- **To solve the 'Normalized-cut' (N-cut) graph partition/clustering problem:**
  - Observation: sub-circuits have more connections within than in-between.
  - To 'cut' as little connections as possible yet to have meaningful size for each partition.

$$n\text{-}cut = \frac{1}{k}\sum_{i=1}^{k}\frac{link\left(V_i, V_i \backslash V\right)}{link(V_i, V)}$$

- **Existing methods and issues:**
  - N-cut problem is NP-hard and its solution is usually approximated.
  - Existing methods either do not optimize for n-cut directly such as spectral clustering or may stuck at local minima such as methods based on iterative search algorithms.
  - Further, existing methods only leverage on connectivity but not node features.

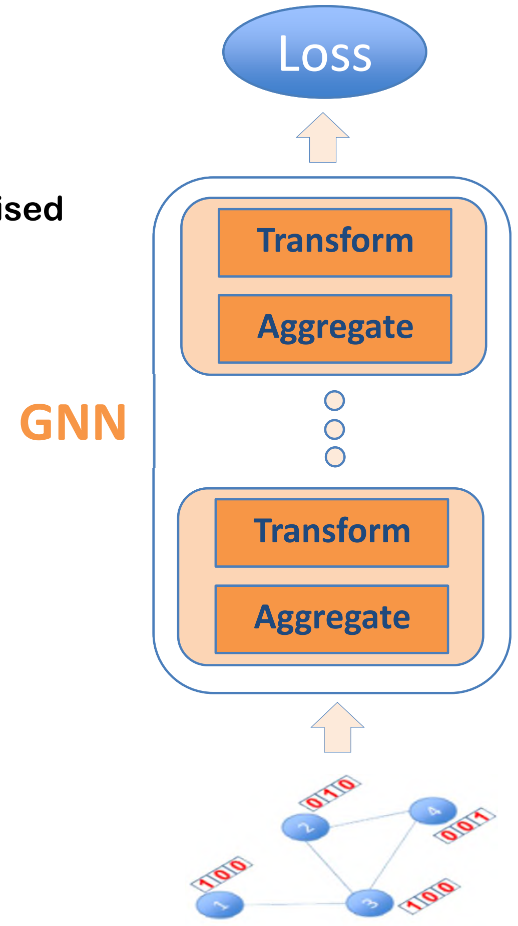


Graph Neural Network (GNN) for Netlist Partition

- **Advantages of GNN:**
  - GNN leverages on both **connectivity** and **node features**.
  - Can optimize for an objective function (e.g. N-cut) **directly** as a loss function (unsupervised setting).

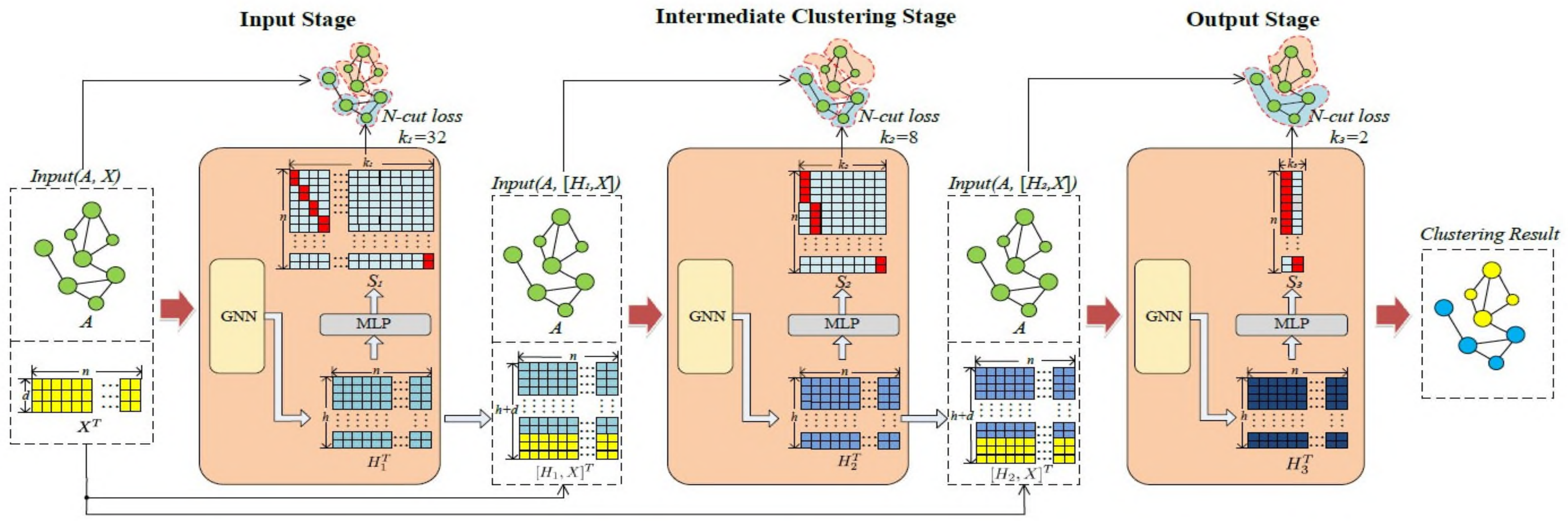
- **Challenges of GNN:**
  - GNN is inherently **local** and deep architecture is difficult.
  - Need to find a meaningful node feature for the intended task.

- **We propose a novel GNN for netlist partition named 'GraphClusNet':**
  - A novel **hierarchical architecture** which finds clusters from local to global.
  - An **n-cut-based loss function** to optimize for the objective function directly.
  - A **location-based node feature** which suits the partition task and avoids local minima.



Proposed Architecture

- **Multi-stage hierarchical architecture:**
  - Intuition: sub-circuits group hierarchically into larger circuits.
  - Optimize for ‘n-cut’ loss at each stage.
  - Final stage can perform either bipartition or multiway partition.



Architecture of GraphClusNet

Proposed Loss Function

- **'N-cut' based loss function:**
  - The numerator computes the intra-cluster connections of each cluster.
  - The denominator computes the total connections of each cluster.
  - Effectively searches for clusters that have more connections within and less connections in-between.

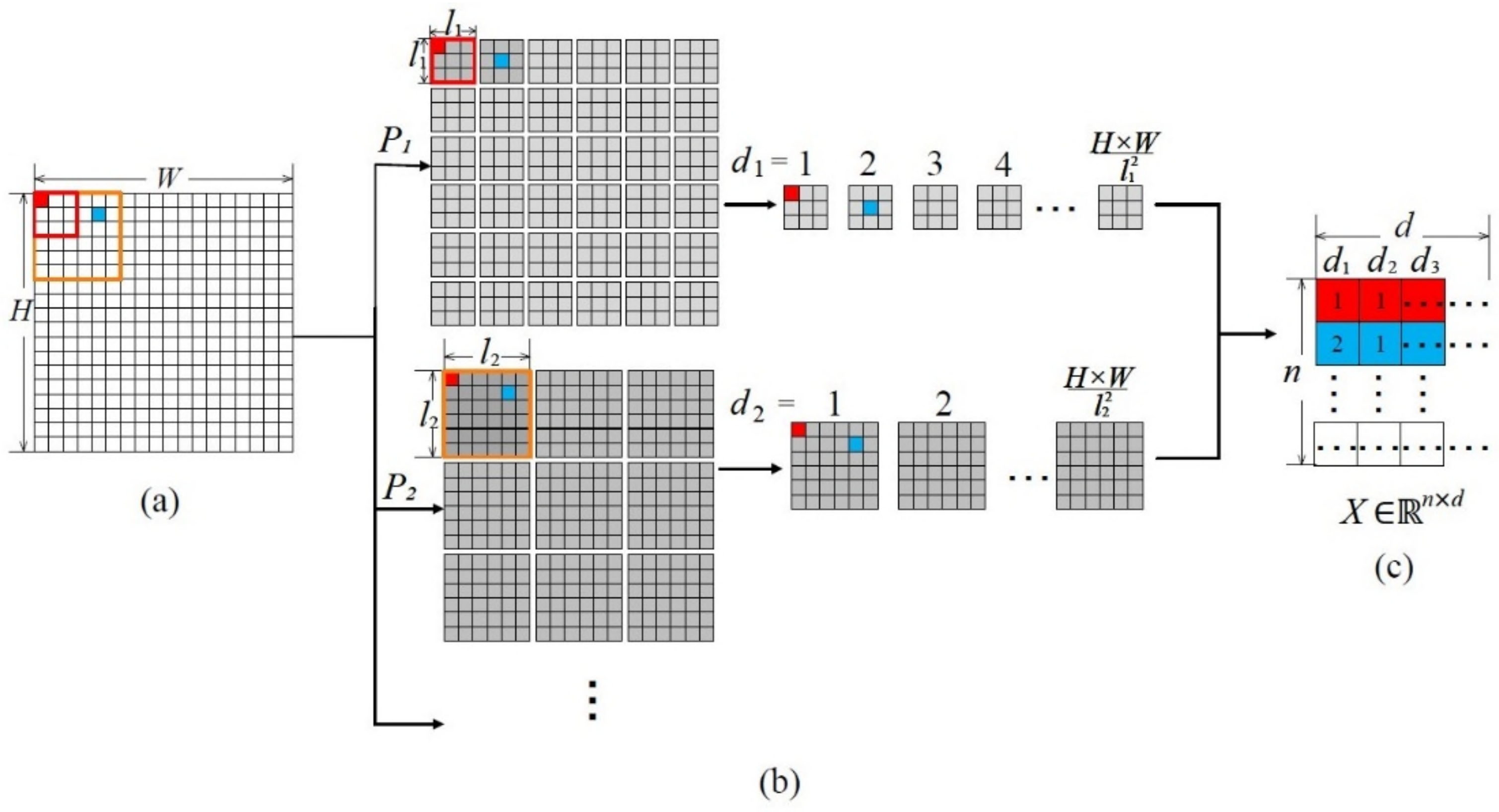
$$\mathcal{L}_{ncut} = 1 - \frac{\mathrm{Diag}(S^T A S)}{\mathrm{Diag}(S^T D S)}$$

where *A* is the adjacency matrix, *D* is the degree matrix, and *S* is the cluster assignment matrix.

- **Allows direct optimization of the N-cut objective function.**

Proposed Node Feature

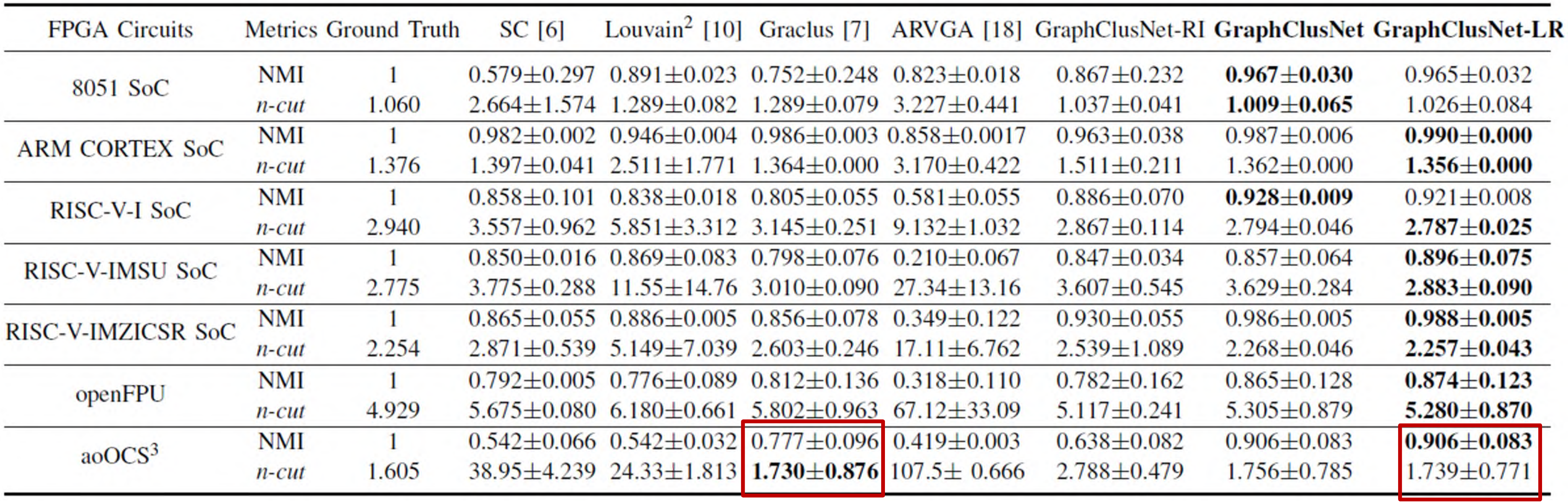
- **Location-based node feature:**



- Intuition: logic gates from the same sub-circuit tend to locate close to each other on the floorplan.
- Divide floorplan into squares of different sizes.
- Assign node feature to nodes based on their location number at each square size.
- Effectively provides a node feature where nodes close to each other have more similar entries.

Partition Results: Bipartition on SoC Netlists

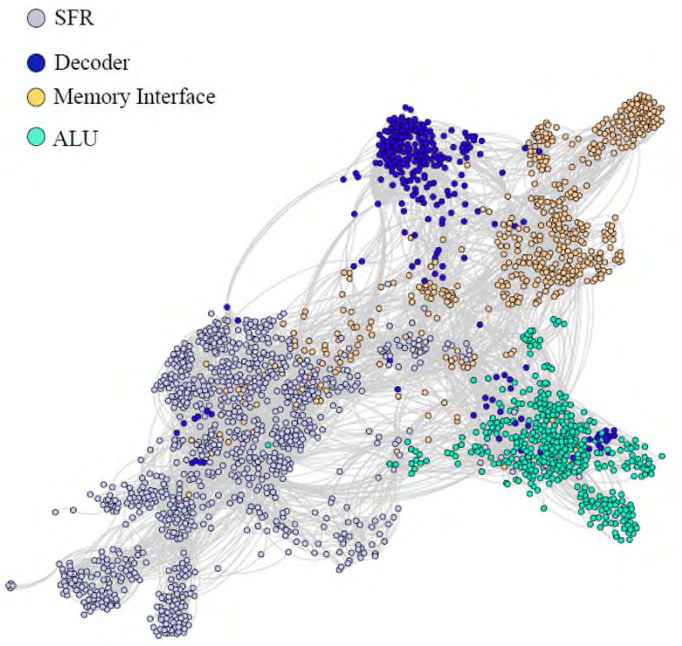
- **Performed bipartition on real FPGA SoC circuit netlists:**
  - To extract major functional block from a netlist.
  - Our proposed GraphClusNet achieved highest NMI and usually lowest n-cut among competing methods.
  - It avoided local minima and can obtain more meaningful partitions.



FPGA Circuits	Metrics	Ground Truth	SC [6]	Louvain^2 [10]	Graclus [7]	ARVGA [18]	GraphClusNet-RI	**GraphClusNet**	**GraphClusNet-LR**
8051 SoC	NMI	1	0.579±0.297	0.891±0.023	0.752±0.248	0.823±0.018	0.867±0.232	**0.967±0.030**	0.965±0.032
	n-cut	1.060	2.664±1.574	1.289±0.082	1.289±0.079	3.227±0.441	1.037±0.041	**1.009±0.065**	1.026±0.084
ARM CORTEX SoC	NMI	1	0.982±0.002	0.946±0.004	0.986±0.003	0.858±0.0017	0.963±0.038	0.987±0.006	**0.990±0.000**
	n-cut	1.376	1.397±0.041	2.511±1.771	1.364±0.000	3.170±0.422	1.511±0.211	1.362±0.000	**1.356±0.000**
RISC-V-I SoC	NMI	1	0.858±0.101	0.838±0.018	0.805±0.055	0.581±0.055	0.886±0.070	**0.928±0.009**	0.921±0.008
	n-cut	2.940	3.557±0.962	5.851±3.312	3.145±0.251	9.132±1.032	2.867±0.114	2.794±0.046	**2.787±0.025**
RISC-V-IMSU SoC	NMI	1	0.850±0.016	0.869±0.083	0.798±0.076	0.210±0.067	0.847±0.034	0.857±0.064	**0.896±0.075**
	n-cut	2.775	3.775±0.288	11.55±14.76	3.010±0.090	27.34±13.16	3.607±0.545	3.629±0.284	**2.883±0.090**
RISC-V-IMZICSR SoC	NMI	1	0.865±0.055	0.886±0.005	0.856±0.078	0.349±0.122	0.930±0.055	0.986±0.005	**0.988±0.005**
	n-cut	2.254	2.871±0.539	5.149±7.039	2.603±0.246	17.11±6.762	2.539±1.089	2.268±0.046	**2.257±0.043**
openFPU	NMI	1	0.792±0.005	0.776±0.089	0.812±0.136	0.318±0.110	0.782±0.162	0.865±0.128	**0.874±0.123**
	n-cut	4.929	5.675±0.080	6.180±0.661	5.802±0.963	67.12±33.09	5.117±0.241	5.305±0.879	**5.280±0.870**
aoOCS^3	NMI	1	0.542±0.066	0.542±0.032	0.777±0.096	0.419±0.003	0.638±0.082	0.906±0.083	**0.906±0.083**
	n-cut	1.605	38.95±4.239	24.33±1.813	**1.730±0.876**	107.5± 0.666	2.788±0.479	1.756±0.785	1.739±0.771

Partition Results: Multiway Partition

- **Performed multiway partition on 8051 microcontroller core netlist:**
  - To extract multiple functional blocks from a netlist.
  - Our proposed GraphClusNet achieved **highest NMI** and **F1-score** among competing methods.



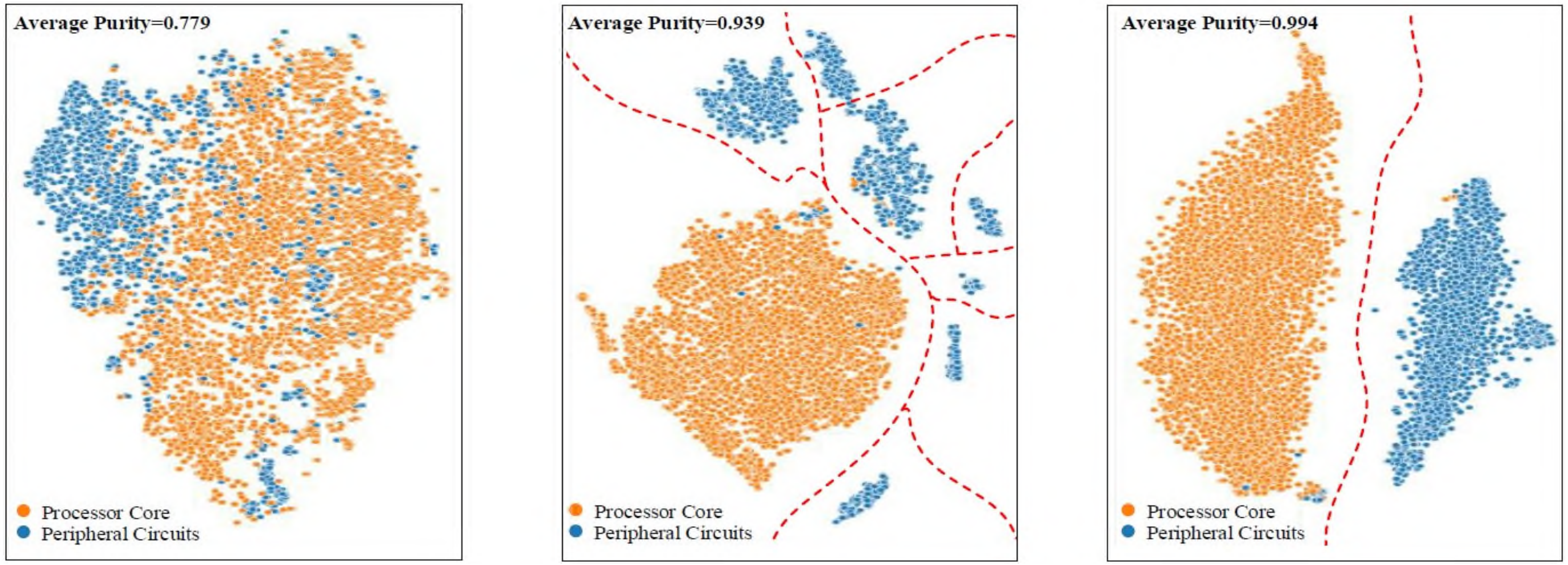
8051 Core Circuit

Functional Blocks/IC	No. Nodes	Metrics	SC [6]	Louvain [10]	Graclus [7]	ARVGA	**GraphClusNet**
ALU	456	F1-score	0.8896±0.0387	0.9251±0.0294	0.9270±0.0074	0.6661±0.0109	**0.9339±0.0322**
SFR	1027	F1-score	0.8968±0.0263	0.7658±0.1074	0.8869±0.1013	0.7216±0.0110	**0.9431±0.0048**
Memory Interface	494	F1-score	0.5805±0.0986	0.5489±0.1242	0.7125±0.1324	0.5767±0.0188	**0.8060±0.0661**
Decoder	252	F1-score	0.6738±0.1434	0.6426±0.0810	0.8221±0.1580	0.5928±0.0070	**0.9260±0.0103**
8051 Core	2229	NMI	0.5966±0.0574	0.5621±0.0329	0.6574±0.0683	0.4742±0.0070	**0.7176±0.0429**

[3] X. Hong, T. Lin, Y. Shi and B. H. Gwee, "GraphClusNet: A Hierarchical Graph Neural Network for Recovered Circuit Netlist Partitioning," *IEEE Transactions on Artificial Intelligence*, 2022.

Visualization of Partition Results

- **We visualized node embeddings after each stage of GNN:**
  - **Local clusters were merged into higher level clusters.**
  - **Cluster purity also improved at higher levels.**



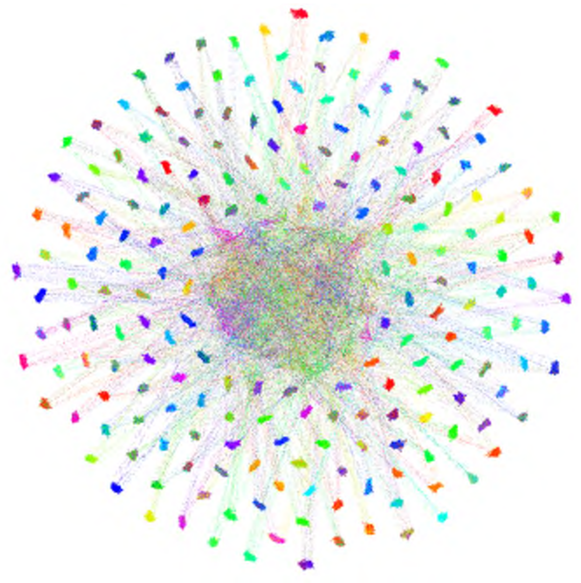
t-SNE Visualization of Node Embeddings after Each Stage of GNN (8051 SoC)

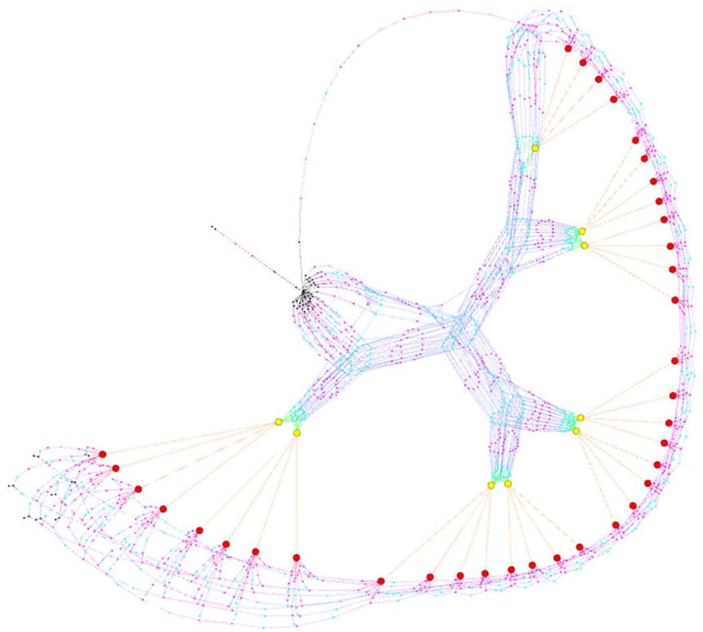
[3] X. Hong, T. Lin, Y. Shi and B. H. Gwee, "GraphClusNet: A Hierarchical Graph Neural Network for Recovered Circuit Netlist Partitioning," *IEEE Transactions on Artificial Intelligence*, 2022.

Outline

Netlist Identification: The Problem

- **To identify the functionality of a flattened netlist:**
  - Used to be done manually with expert knowledge.
  - Observation: different circuit graphs have distinctive structures and gate compositions.
  - Netlist identification problem may thus be formulated as a graph classification problem using machine-learning methods.


Encryption Circuit


Filter Circuit

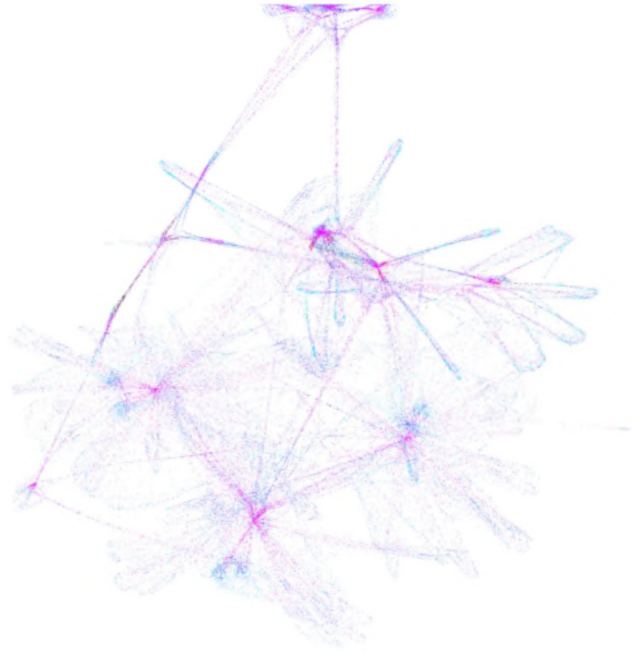
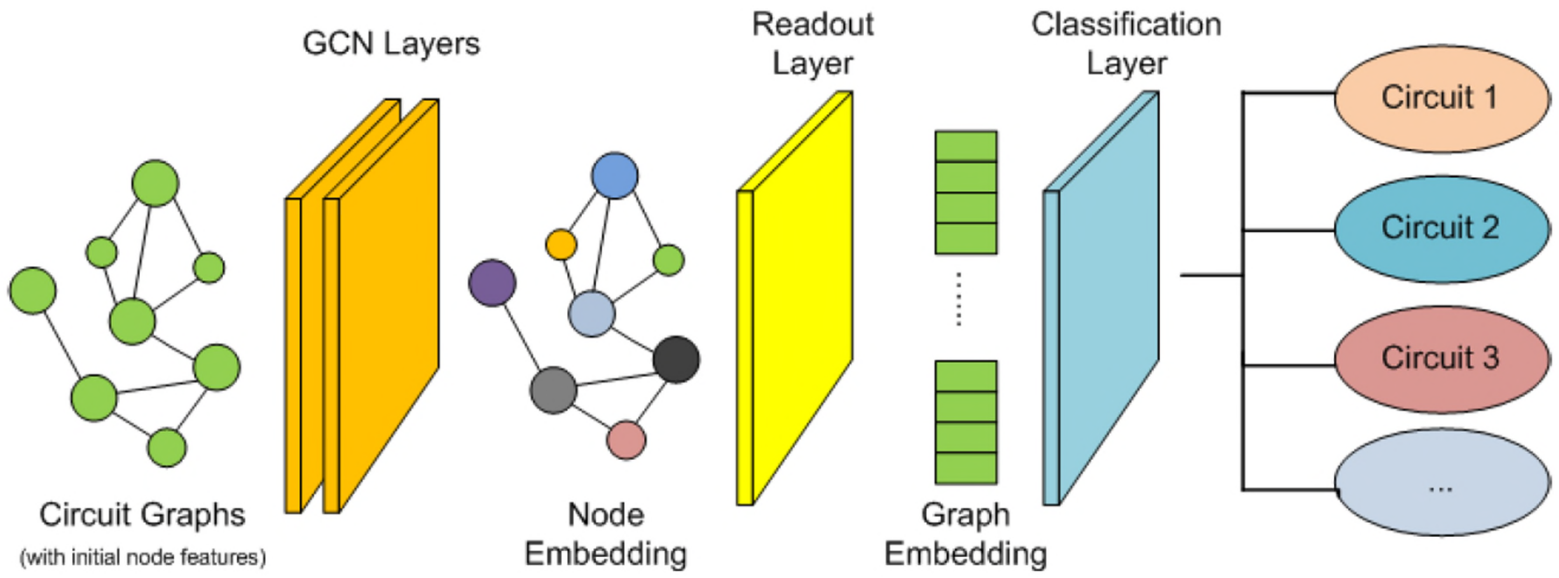


Image Processing Circuit

GNN for Netlist Identification

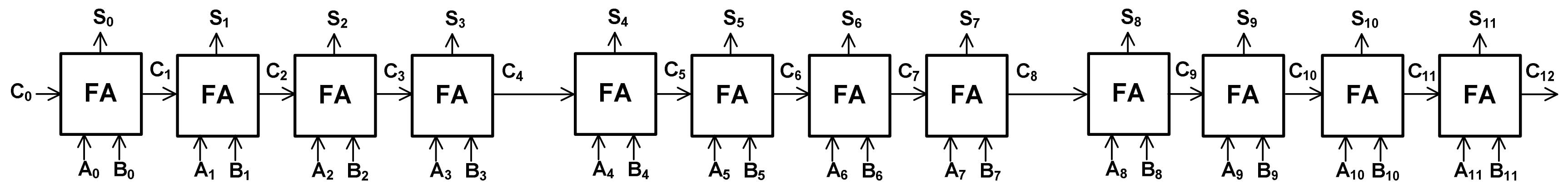
- **Train a GNN to classify unknown netlists into known classes:**
  - Input is a circuit graph with **gate type** as node feature and output is a class label indicating the type of circuit.
  - Our GNN consists of two layers of Graph Convolutional Network (GCN).



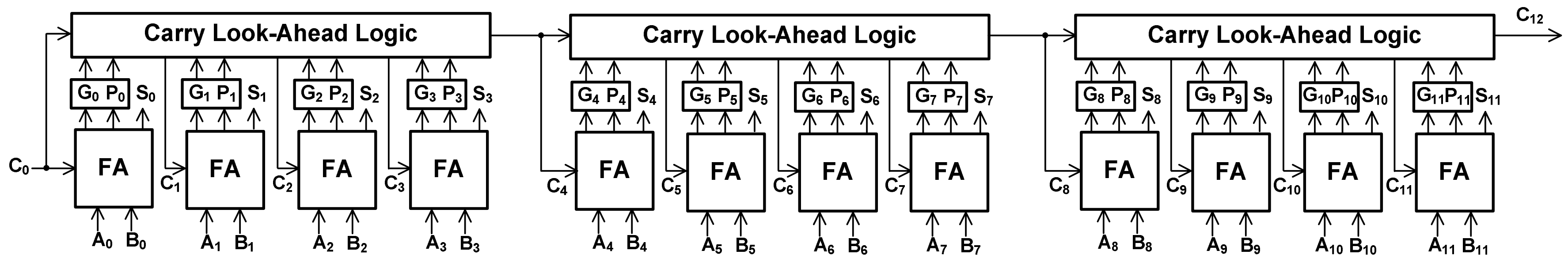
Our Proposed GNN for Netlist Identification

Case Study: Adder Circuit Classification

- **Classify four types of adder circuits:**
  - Four adder **structures**: Ripple Carry Adder (RCA), Carry Look-Ahead Adder (CLA), Carry Select Adder (CSLA), and Carry Skip Adder (CSKA).



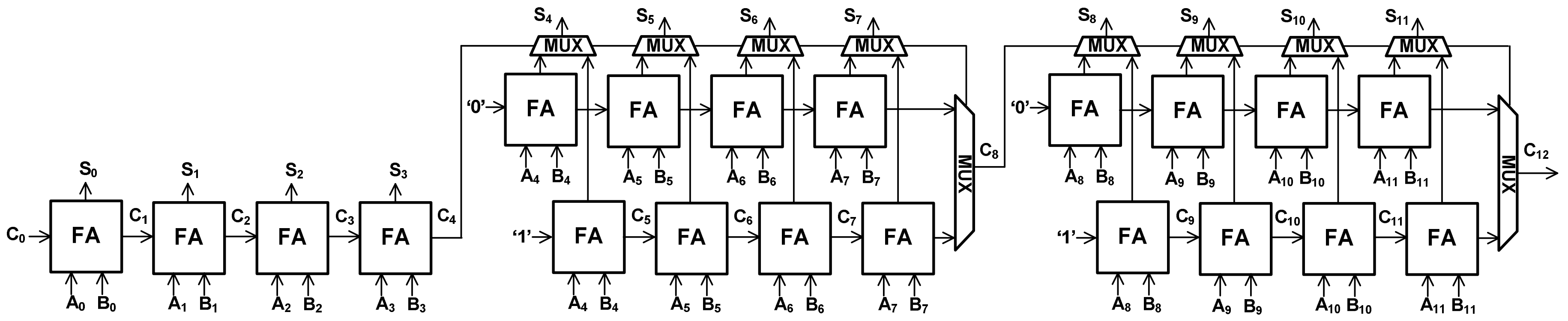
12-bit Ripple Carry Adder (RCA)



12-bit Carry Look-Ahead Adder (CLA)

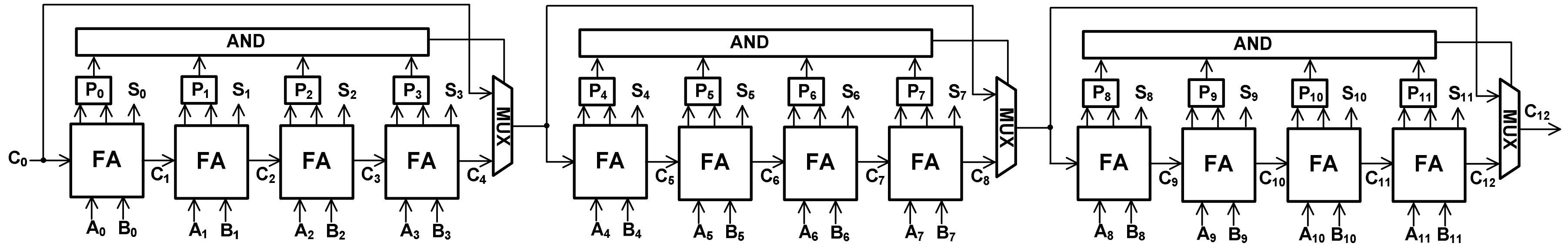
FA = Full Adder
G = Generate
P = Propagate

Case Study: Adder Circuit Classification



FA
MUX
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12
A0 B0 A1 B1 A2 B2 A3 B3 A4 B4 A5 B5 A6 B6 A7 B7 A8 B8 A9 B9 A10 B10 A11 B11
'0'
'1'

12-bit Carry Select Adder (CSLA)



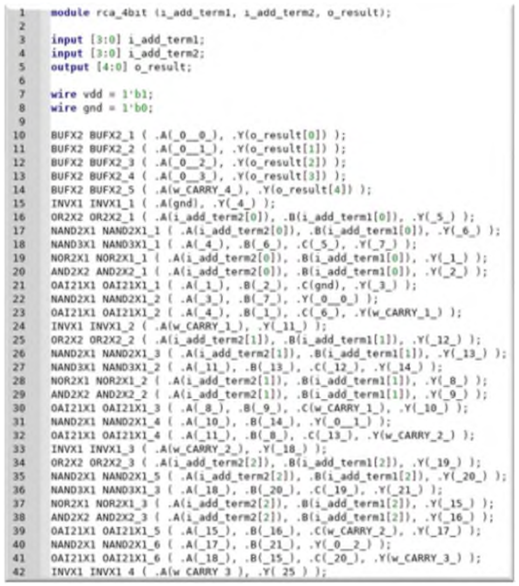
AND
FA
MUX
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12
A0 B0 A1 B1 A2 B2 A3 B3 A4 B4 A5 B5 A6 B6 A7 B7 A8 B8 A9 B9 A10 B10 A11 B11

12-bit Carry Skip Adder (CSKA)

FA = Full Adder
P = Propagate
AND = AND Gate
MUX = Multiplexer

Data Preparation

- **Synthesized circuit netlists of varying bit-widths for training and testing data:**
  - Synthesized 4 types of adder circuits from 5-bit to 64-bit resulting a total of 240 netlists.
  - Used 40 netlists for training GNN and remaining 200 netlists for testing.
  - Used one-hot encoded gate type as node features.



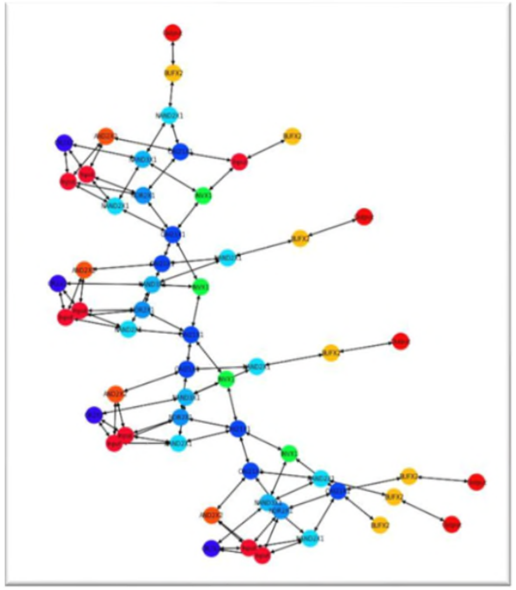
```
module rca_4bit (i_add_term1, i_add_term2, o_result);

input [3:0] i_add_term1;
input [3:0] i_add_term2;
output [4:0] o_result;

wire vdd = 1'b1;
wire gnd = 1'b0;

BUFX2 BUFX2_1 ( .A(_0__0_), .Y(o_result[0]) );
BUFX2 BUFX2_2 ( .A(_0__1_), .Y(o_result[1]) );
BUFX2 BUFX2_3 ( .A(_0__2_), .Y(o_result[2]) );
BUFX2 BUFX2_4 ( .A(_0__3_), .Y(o_result[3]) );
BUFX2 BUFX2_5 ( .A(w_CARRY_4_), .Y(o_result[4]) );
INVX1 INVX1_1 ( .A(gnd), .Y(_4_) );
OR2X2 OR2X2_1 ( .A(i_add_term2[0]), .B(i_add_term1[0]), .Y(_5_) );
NAND2X1 NAND2X1_1 ( .A(i_add_term2[0]), .B(i_add_term1[0]), .Y(_6_) );
NAND3X1 NAND3X1_1 ( .A(_4_), .B(_6_), .C(_5_), .Y(_7_) );
NOR2X1 NOR2X1_1 ( .A(i_add_term2[0]), .B(i_add_term1[0]), .Y(_1_) );
AND2X2 AND2X2_1 ( .A(i_add_term2[0]), .B(i_add_term1[0]), .Y(_2_) );
OAI21X1 OAI21X1_1 ( .A(_1_), .B(_2_), .C(gnd), .Y(_3_) );
NAND2X1 NAND2X1_2 ( .A(_3_), .B(_7_), .Y(_0__0_) );
OAI21X1 OAI21X1_2 ( .A(_4_), .B(_1_), .C(_6_), .Y(w_CARRY_1_) );
INVX1 INVX1_2 ( .A(w_CARRY_1_), .Y(_11_) );
OR2X2 OR2X2_2 ( .A(i_add_term2[1]), .B(i_add_term1[1]), .Y(_12_) );
NAND2X1 NAND2X1_3 ( .A(i_add_term2[1]), .B(i_add_term1[1]), .Y(_13_) );
NAND3X1 NAND3X1_2 ( .A(_11_), .B(_13_), .C(_12_), .Y(_14_) );
NOR2X1 NOR2X1_2 ( .A(i_add_term2[1]), .B(i_add_term1[1]), .Y(_8_) );
AND2X2 AND2X2_2 ( .A(i_add_term2[1]), .B(i_add_term1[1]), .Y(_9_) );
OAI21X1 OAI21X1_3 ( .A(_8_), .B(_9_), .C(w_CARRY_1_), .Y(_10_) );
NAND2X1 NAND2X1_4 ( .A(_10_), .B(_14_), .Y(_0__1_) );
OAI21X1 OAI21X1_4 ( .A(_11_), .B(_8_), .C(_13_), .Y(w_CARRY_2_) );
INVX1 INVX1_3 ( .A(w_CARRY_2_), .Y(_18_) );
OR2X2 OR2X2_3 ( .A(i_add_term2[2]), .B(i_add_term1[2]), .Y(_19_) );
NAND2X1 NAND2X1_5 ( .A(i_add_term2[2]), .B(i_add_term1[2]), .Y(_20_) );
NAND3X1 NAND3X1_3 ( .A(_18_), .B(_20_), .C(_19_), .Y(_21_) );
NOR2X1 NOR2X1_3 ( .A(i_add_term2[2]), .B(i_add_term1[2]), .Y(_15_) );
AND2X2 AND2X2_3 ( .A(i_add_term2[2]), .B(i_add_term1[2]), .Y(_16_) );
OAI21X1 OAI21X1_5 ( .A(_15_), .B(_16_), .C(w_CARRY_2_), .Y(_17_) );
NAND2X1 NAND2X1_6 ( .A(_17_), .B(_21_), .Y(_0__2_) );
OAI21X1 OAI21X1_6 ( .A(_18_), .B(_15_), .C(_20_), .Y(w_CARRY_3_) );
INVX1 INVX1_4 ( .A(w_CARRY_3_), .Y(_25_) );
```

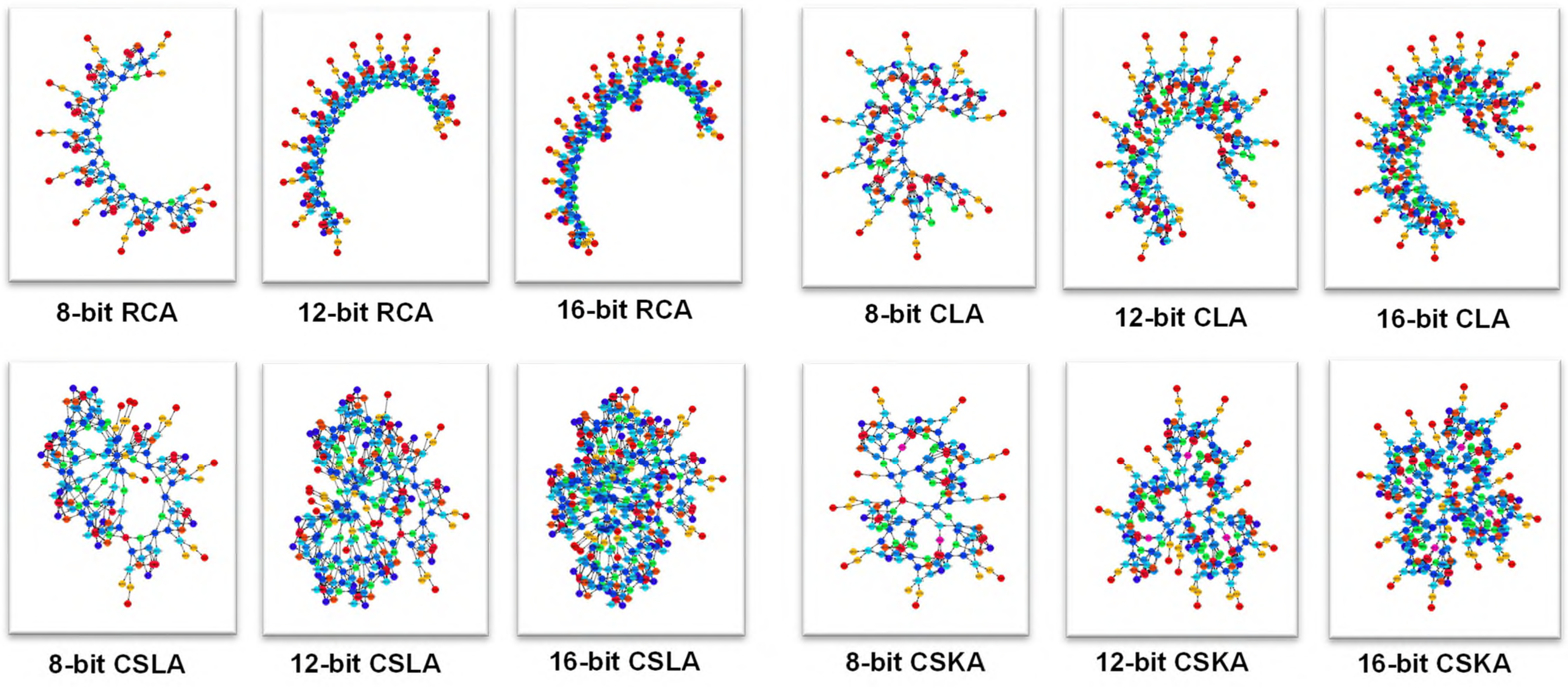
Netlist of A 4-bit RCA



Circuit Graph of 4-bit RCA

- Different node colours represent different gate types

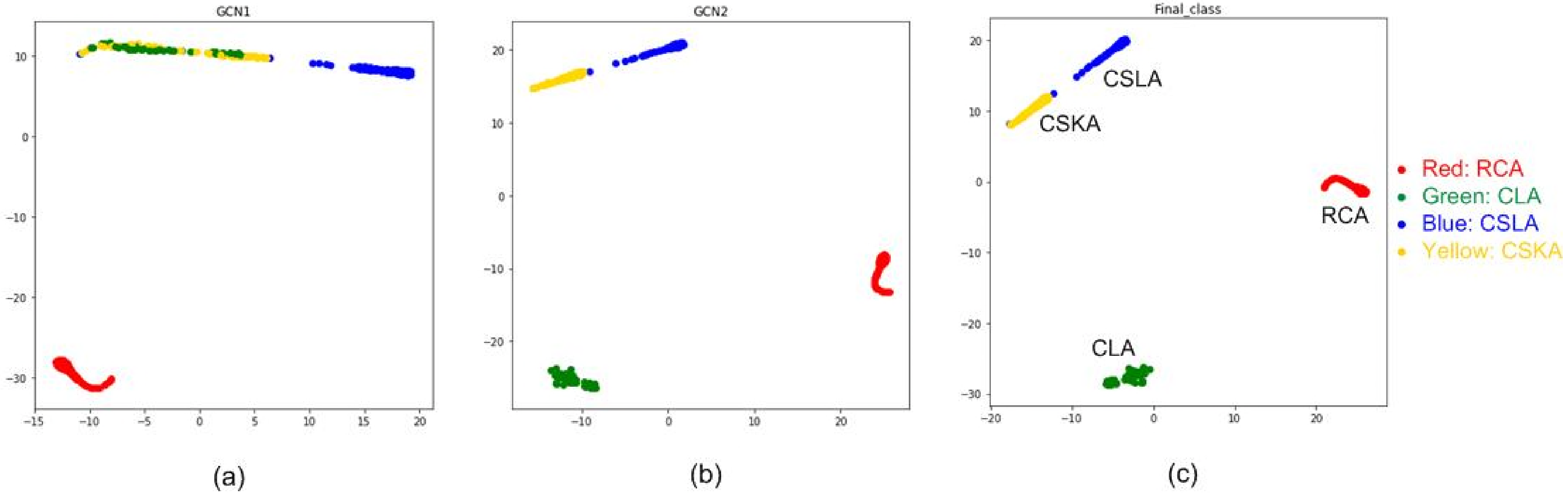
Graph Visualization



Graph Visualization of Adder Circuit Netlists

Netlist Classification Results

- **Our GNN achieved high classification accuracy on unseen test data:**
  - **GNN achieved classification accuracy of 99% on unseen test data.**
  - **Graph embeddings of different class netlists grow separated after each layer of GNN demonstrating its discriminating power.**



t-SNE Visualization of Adder Circuit Graph Embeddings after Each Layer of GNN

[4] X. Hong, T. Lin, Y. Shi and B. H. Gwee, "ASIC Circuit Netlist Recognition Using Graph Neural Network," in *Proc. 2021 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, 2021.

Conclusions

- Delayered IC image analysis is one of the most reliable approach to chip integrity and functionality assurance.
- With the problem scale and limited data, we aim to develop less data-hungry and adaptable deep learning algorithms for automatic IC image and netlist analysis.
- A self-supervised GAN-based network has been presented for concurrent IC image anomaly detection and inpainting.
- A deep learning-based framework for IC image analysis has been presented. Deep learning models can be effectively applied to retrieve the standard cells and interconnects in IC images.
- GNN has demonstrated some unique advantages over conventional machine-learning methods for netlist analysis including its ability to process graph connectivity together with node features.
- Novel GNN architectures for netlist partition and netlist identification have been presented.

Thank You !

Questions?