# High Performance Graph Convolutional Networks with Applications in Testability Analysis

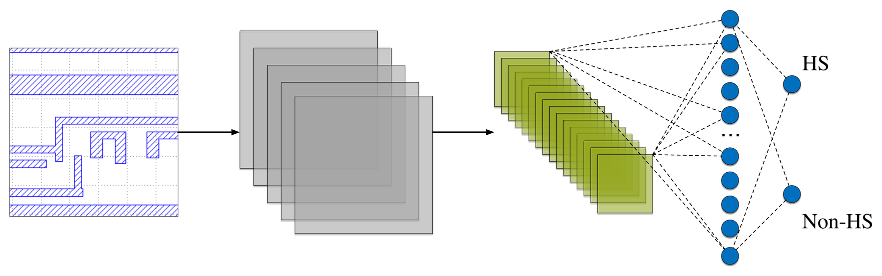

Yuzhe Ma[1], Haoxing Ren[2], Brucek Khailany [2], Harbinder Sikka[2], Lijuan Luo[2], Karthikeyan Natarajan[2], Bei Yu[1]

[1]The Chinese University of Hong Kong
[2]NVIDIA

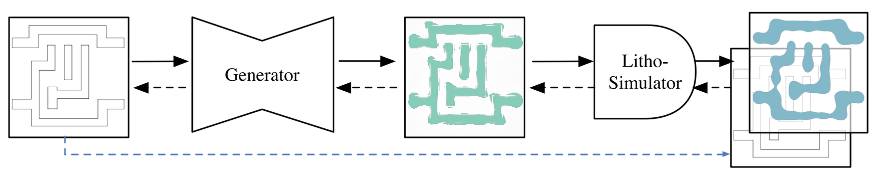

## Background

### Learning for EDA

- Lithography hotspot detection [Yang et.al TCAD'2018]




- Mask optimization [Yang et.al DAC'2018]

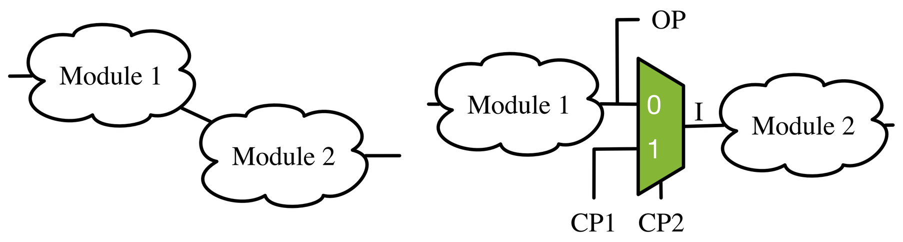



### More Considerations

- Existing attempts still rely on regular format of data, like images;
- Netlists and layouts are naturally represented as graphs;
- Few DL solutions for graph-based problems in EDA.

### Test Points Insertion

- Fig. (a): Original circuit. Module 1 is unobservable. Module 2 is uncontrollable;
- Fig. (b): Insert test points to the circuit;
- (CP1, CP2) = (0, 1) → line I = 0; (CP1, CP2) = (1, 1) → line I = 1;
- CP2 = 0 → normal operation mode.



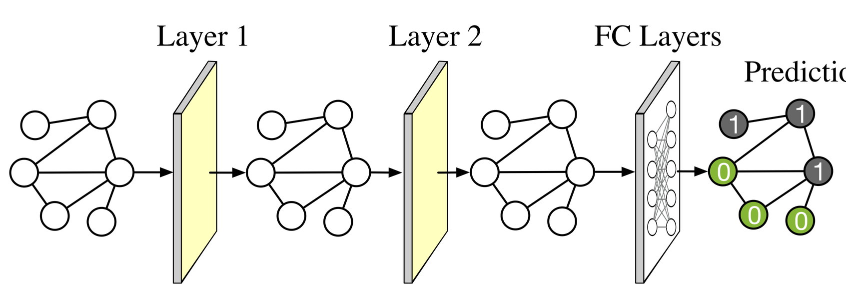

(a) (b)

## Problem Overview

- Given a netlist, identify where to insert test points, such that:
  - Maximize fault coverage;
  - Minimize the number of test points and test patterns.
  - Focus on observation points insertion in this project.
- It is a binary classification problem from the perspective of DL model;
- A classifier can be trained from the historical data;
- Need to handle graph-structured data;
- Strong scalability is required for realistic designs.

## Node Classification

### Fundamental framework;




- Represent a netlist as a directed graph. Each node represents a gate.
- Initial node attributes: SCOAP values [Goldstein et. al DAC'1980].
- Compute node embeddings first, then perform classification.

### Node embedding: two-step operation

- Neighborhood feature aggregation: weighted sum of the neighborhood features.

$$g_d^{(v)} = e_{d-1}^{(v)} + w_{pr} \times \sum_{u \in \mathrm{PR}(v)} e_{d-1}^{(u)} + w_{su} \times \sum_{u \in \mathrm{SU}(v)} e_{d-1}^{(u)}$$

- Projection: a non-linear transformation to a higher dimension.

$$e_d = \sigma(g_d \cdot W_d)$$

### Classification

- A series of fully-connected layers

## Node Embedding Computation

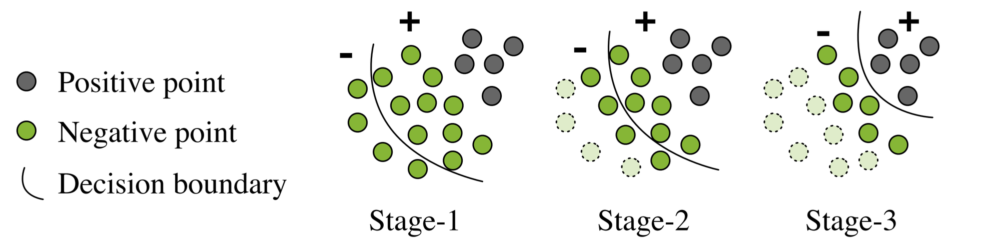
**Require:** Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node attributes $\{x^{(v)} : \forall v \in \mathcal{V}\}$; Search depth $D$; non-linear activation function $\sigma(\cdot)$; Weight matrices $W_d$ of encoders $E_d, d = 1, ..., D$;
**Ensure:** Embedding of for each node $e_D^{(v)}, \forall v \in \mathcal{V}$.

```
e_0^(v) ← x^(v), ∀v ∈ V;
for d = 1, ..., D do
    for all v ∈ V do
        Compute g_d^(v);
        e_d^(v) ← σ(W_d · g_d^(v));
    end for
end for
```

## Multi-stage Classification

### Imbalance issue

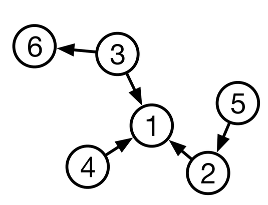
- High imbalance ratio: much more negative nodes than positive nodes in a design;
- Poor performance: bias towards majority class;
- Solution: multi-stage classification.
  - Impose a large weight on positive points.
  - Only filter out negative points with high confidence in each stage.




## Efficient Inference and Training

### Inference

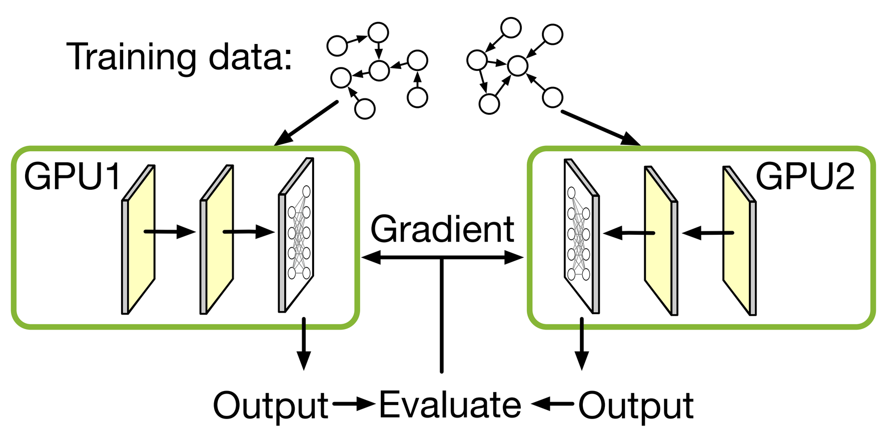
- Neighborhood overlap leads to duplicated computation → poor scalability.
- Fact: adjacency matrix is highly sparse! It can be stored using compressed format.
- Transform weighted summation to matrix multiplication.



$$G_d = A \cdot E_{d-1} = \begin{array}{c|c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1\\2\\3\\4\\5\\6 \end{array} & \begin{bmatrix} 1 & w_1 & w_1 & w_1 & 0 & 0 \\ w_2 & 1 & 0 & 0 & w_1 & 0 \\ w_2 & 0 & 1 & 0 & 0 & w_2 \\ w_2 & 0 & 0 & 1 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & w_1 & 0 & 0 & 1 \end{bmatrix} \end{array} \times \begin{bmatrix} e_{d-1}^{(1)} \\ e_{d-1}^{(2)} \\ e_{d-1}^{(3)} \\ e_{d-1}^{(4)} \\ e_{d-1}^{(5)} \\ e_{d-1}^{(6)} \end{bmatrix}$$

### Training

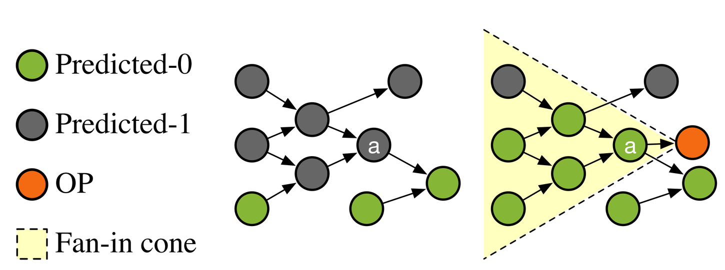
- Adjacency matrix cannot be split as conventional way.
- A variant of conventional data-parallel scheme.
  - Each GPU process one graph instead of one "chunk";
  - Gather all to calculate the gradient.




## Test Points Insertion Flow

### OP Impact

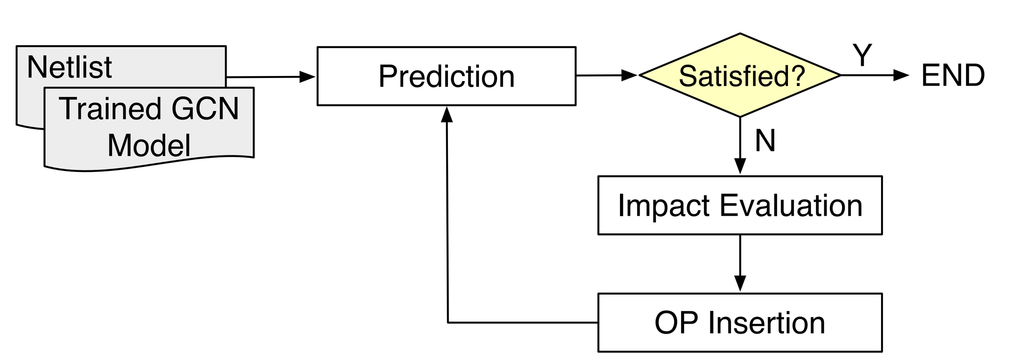
- Not every difficult-to-observe node has the same impact for improving the observability;
- Select the observation point locations with largest impact to minimize the total count;
- Impact: The positive prediction reduction in a local neighborhood after inserting an observation point.
  - E.g., the impact of node a in the figure is 4.




(c) (d)

### Iterative OPs Insertion Flow

- Iterative prediction and OPs insertion.
- Once an OP is inserted, the netlist would be modified and node attributes would be re-calculated.
- Sparse representation enables incremental update on adjacency matrix.
- Exit condition: no positive predictions left.




## Experimental Results

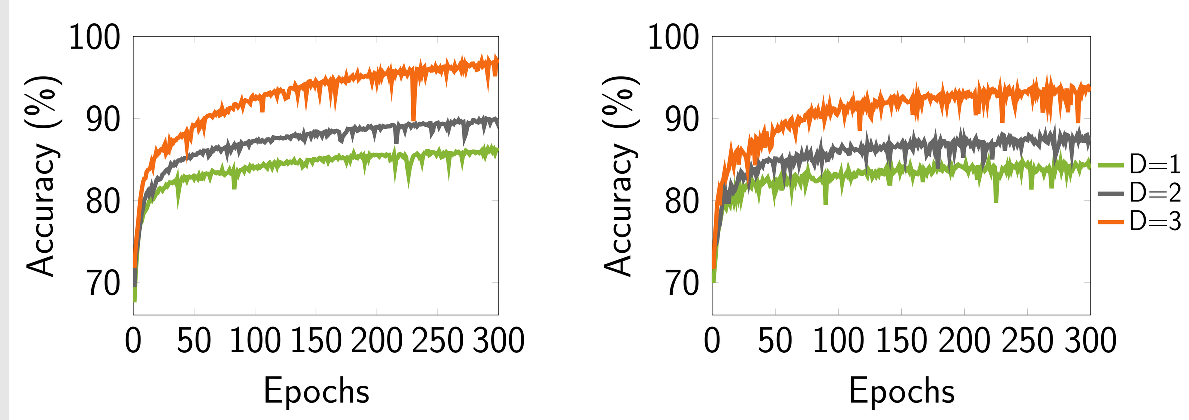
### Benchmarks

- 4 Industrial designs under 12nm technology node.

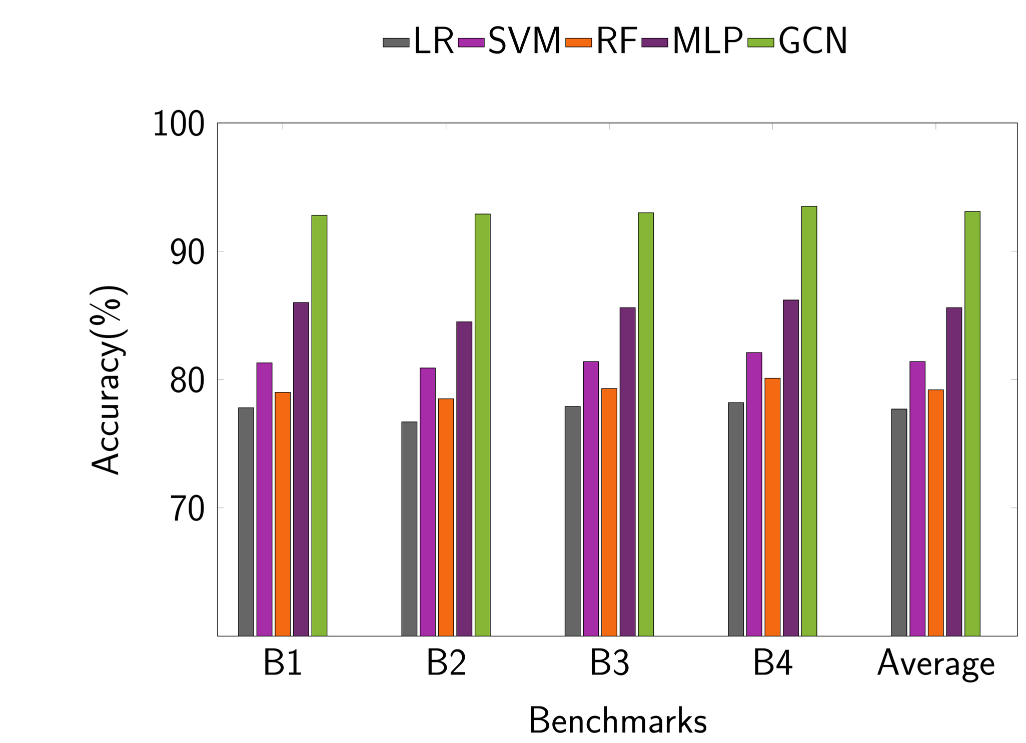
| Design | #Nodes | #Edges | #POS | #NEG |
|---|---|---|---|---|
| B1 | 1384264 | 2102622 | 8894 | 1375370 |
| B2 | 1456453 | 2182639 | 9755 | 1446698 |
| B3 | 1416382 | 2137364 | 9043 | 1407338 |
| B4 | 1397586 | 2124516 | 8978 | 1388608 |

### Classification Results

- Comparison among different search depths.

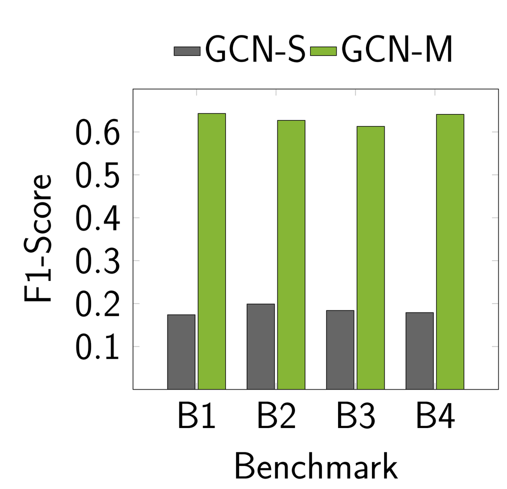


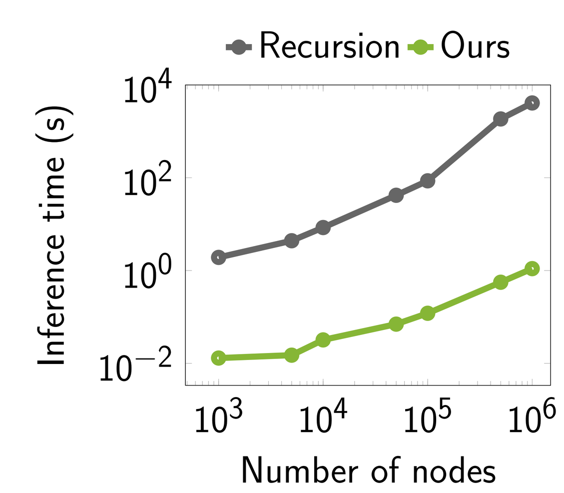

- Baselines: classical learning models with feature engineering applied in industry;
- Single GCN outperforms other classical learning algorithms on balanced datasets.




### Multi-stage Classification

- Classification: Single-stage GCN vs. Multi-stage GCN
  - Significant improvement of classification performance on real designs;
- Scalability: Recursive computation vs. Matrix multiplication
  - $10^3$X speedup on inference time for a design with > 1 million cells.




### Testability Results Comparison

- Baseline: Conduct OPs insertion with a commercial industrial tool;
- Without loss on fault coverage, our flow achieves 11% reduction on test points inserted and 6% reduction on test pattern count.