ECE407 - Computer Aided Design for VLSI

Assignment Report - Part B

Test Development using KNN, GNN

Student 1: Lefkios Mavroudi
UCY ID: UC1064650

Student 2: Alexandros Vryonides
UCY ID: UC1066645

Contents

1 Introduction

Part A of this assignment focused on constructing the structural foundation required for machine learning–based testability analysis. In that initial phase, we developed a custom parser for ISCAS'85/89 `.bench` files, built the corresponding directed acyclic graph (DAG), performed topological traversal, and computed all primary input (PI) to primary output (PO) paths. This resulted in the G1 structural graph representation of the circuit, which captures all nodes, edges, and signal dependencies. The graph served as a verified and reliable basis for further analysis.

Building on this foundation, Part B extends the work toward testability prediction using both analytical and machine learning approaches. We compute SCOAP (controllability and observability) measures for every node, derive detectability values, and construct node- level structural features from the circuit graph. These representations form the input to two machine learning models studied in this part of the assignment: a k-Nearest Neighbours (KNN) baseline and a Graph Neural Network (GNN).

The objective of Part B is therefore twofold: (1) to integrate structural testability metrics with learned models, and (2) to evaluate how effectively different learning paradigms can predict node testability, both as a continuous detectability score and as a categorical difficulty level (EASY, MEDIUM, HARD), to obtain a clearer picture of the underlying difficulty levels and a secondary target that helps us validate and interpret the regression results.

This section presents the general, model-agnostic pipeline used for both KNN and GNN. Model-specific details are deferred to Sections 2 and 3.

1.1 Data Processing / Parsing

The first stage is converting a benchmark circuit (such as `c17.bench`) into an analyzable structure.

1. **Netlist Parsing:** The `.bench` file is parsed to extract gate definitions, net connections, and the list of primary inputs (PIs) and primary outputs (POs).
2. **Graph Construction:** We build a directed acyclic graph (DAG) representation:
   - Nodes represent internal nets, gate outputs, and PIs/POs.
   - Directed edges represent signal flow between nets.
3. **Topological Consistency:** A levelized traversal order (from PIs to POs) is established, enabling correct SCOAP computation and path counting.

This graph representation forms the foundation for both SCOAP analysis and machine learning models.

1.2 SCOAP Measures and Detectability

SCOAP provides industry-standard metrics for assessing testability in combinational circuits.

For each node v, we compute:

- **Controllability:**
  - $CC0[v]$: difficulty of forcing a logical 0 at node v,
  - $CC1[v]$: difficulty of forcing a logical 1 at node v.
- **Observability:**
  - $CO[v]$: difficulty of propagating a fault at v to a primary output.

From these, we derive the detectability of stuck-at faults:

$$\begin{aligned} D_{\text{sa0}}[v] &= CC1[v] + CO[v], \\ D_{\text{sa1}}[v] &= CC0[v] + CO[v], \\ D[v] &= \min(D_{\text{sa0}}[v],\, D_{\text{sa1}}[v]). \end{aligned}$$

The scalar detectability score $D[v]$ is the main regression target (for both KNN and GNN) and also forms the basis for discrete difficulty classes.

1.3 Node Feature Extraction and Normalisation

To characterize the structural role of each node in the circuit, we construct a feature vector:

$$x_v = \big[\text{fanin},\, \text{fanout},\, \text{is\_PI},\, \text{is\_PO},\, \text{depth\_from\_PI},\, \text{path\_count},\, CC0[v],\, CC1[v]\big]. \quad (1)$$

These features encode various structural and SCOAP-based properties:

- **fanin:** number of incoming edges to node v,
- **fanout:** number of outgoing edges,
- **is_PI / is_PO:** binary indicators,
- **depth_from_PI:** logical depth obtained via topological traversal,
- **path_count:** number of distinct PI-to-v paths,

- $CC0[v], CC1[v]$**:** SCOAP controllability costs.

The raw feature dimensions differ significantly in scale: for example, `fanin` and `fanout` are small integers, while `path_count` and SCOAP values may grow much larger. To avoid large-magnitude features dominating the Euclidean distance or destabilising training, we apply feature standardisation to the feature matrix:

- For **KNN**, we fit a `StandardScaler` on the *training* nodes only and apply it to both training and test nodes, ensuring a fair evaluation.
- For **GNN**, we fit a second `StandardScaler` on the features of all nodes in the graph before feeding them to the GCN, which helps stabilise graph convolution and training dynamics.

Thus, both models operate on normalised, zero-mean, unit-variance features derived from the same structural and SCOAP information.

1.4 EASY / MEDIUM / HARD Testability Classes

To create interpretable classification labels, we convert the continuous detectability values $\{D[v]\}$ into categorical difficulty levels. While the scalar detectability score captures the exact quantitative effort required to detect a fault at a node, its distribution varies significantly across circuits. As a result, interpreting regression outputs directly is not always intuitive. The qualitative classes therefore provide a circuit-adaptive way to understand and validate the predicted testability levels.

Rather than using fixed global thresholds, we employ circuit-specific percentiles to ensure that the classes reflect the relative difficulty within each benchmark. Specifically, we compute:

- low_thresh: 33rd percentile of detectability,
- high_thresh: 66th percentile of detectability.

These thresholds divide the node set into three balanced groups, preventing class imbalance and enabling effective training for classification models. Each node is then assigned:

- $D[v] \leq \text{low\_thresh} \Rightarrow$ **EASY**,
- $\text{low\_thresh} < D[v] \leq \text{high\_thresh} \Rightarrow$ **MEDIUM**,
- $D[v] > \text{high\_thresh} \Rightarrow$ **HARD**.

These labels serve two purposes: (1) they provide a clear qualitative interpretation of the regression target, and (2) they form an additional supervised learning task that both the KNN and GNN models will attempt to predict. The agreement between regression and classification results offers an important form of cross-validation for our overall testability predictions.

1.5 Dataset Split (Shared by KNN and GNN)

We create a single node-level train/test split that is reused for both regression and classification, and for both KNN and GNN. The split is stratified on the EASY/MEDIUM/HARD labels to preserve the class distribution in both sets.

Concretely, we use `train_test_split` as follows:

```
X_train, X_test, y_reg_train, y_reg_test, y_cls_train, y_cls_test, \
    train_nodes, test_nodes = train_test_split(
        X_array,
        y_reg_arr,
        y_cls_full,
        node_list,
        test_size=0.2,
        random_state=0,
        stratify=y_cls_full
    )
```

Here:

- `X_array` contains the normalised structural features of all nodes,
- `y_reg_arr` stores the numeric detectability $D[v]$,
- `y_cls_full` stores the EASY/MEDIUM/HARD label per node,
- `train_nodes`, `test_nodes` track which node names fall into each split.

For the GNN, these same node sets are converted into boolean masks (`train_mask` and `test_mask`), ensuring a direct and fair comparison between KNN and GNN on exactly the same node subsets.

2 KNN Model

2.1 KNN Algorithm: Concept and Mechanics

The k-Nearest Neighbours (KNN) algorithm is a non-parametric, instance-based learning method. Unlike parametric models, KNN does not learn explicit model parameters; instead, it stores all training samples and makes predictions by comparing new instances directly to observed examples.

Given a test node with feature vector x_{test}, the algorithm:

1. computes its distance to all training nodes,
2. identifies the k closest nodes (the neighbours),
3. produces a prediction based on the labels of those neighbours.

We use Euclidean distance as the similarity measure:

$$d(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}.$$

A key advantage of KNN is its interpretability: predictions are grounded in the structural similarity between circuit nodes, as captured by their feature vectors. KNN also requires minimal hyperparameter tuning, making it a suitable baseline for comparison with more sophisticated models such as the GNN.

2.2 KNN for Regression

For regression, the goal is to predict the numeric detectability value $D[v]$. KNN performs regression using a distance-weighted average of the neighbour labels:

$$\hat{y}(x_{\text{test}}) = \frac{\sum_{i \in N_k(x_{\text{test}})} w_i \, y_i}{\sum_{i \in N_k(x_{\text{test}})} w_i},$$

where:

- $N_k(x_{\text{test}})$ is the set of k nearest neighbours,
- y_i is the true detectability of neighbour i,
- $w_i = 1/d(x_{\text{test}}, x_i)$ is a distance-based weight.

Target transformation. The distribution of detectability values $D[v]$ is positively skewed. To stabilise the regression and reduce the influence of very large values, we predict

$$z[v] = \log(1 + D[v])$$

instead of $D[v]$ directly. After prediction, we invert the transform using $\hat{D}[v] = \exp(\hat{z}[v]) - 1$ and compute all regression metrics in the original D-space.

The input features are the normalised vectors from Section 1.3, restricted to the training/test splits defined in Section 1.5.

2.3 KNN for Classification

For the classification task, each node is assigned to one of three difficulty levels:

$$\text{EASY, MEDIUM, HARD,}$$

obtained from its SCOAP detectability value $D[v]$ using the percentile-based thresholds described in Section 1.4.

We reuse exactly the same normalised feature vectors and the same value of k as in the regression setting. A `KNeighborsClassifier` (with distance-based weighting) is trained on the training nodes and their E/M/H labels. For a test node, the classifier:

1. finds the k closest training nodes in feature space;
2. lets these neighbours *vote* for their class, where closer neighbours have a larger weight and more distant ones contribute less;
3. assigns the test node to the class with the largest total vote.

In this way, nodes that are structurally and testability-wise similar (in the feature space) tend to share the same difficulty label. The resulting E/M/H predictions provide a qualitative view of node testability that complements the numeric detectability regression.

2.4 Choice of k in KNN

The choice of k directly affects prediction stability:

- too small k (e.g., 1) makes predictions noisy and overly sensitive,
- too large k oversmooths predictions and ignores local structure.

Instead of fixing k to a single rule-of-thumb value, we consider a small candidate set that includes both hand-picked odd values and a size-dependent value:

$$k \in \{3, 5, 7, 9, 11, \text{round}(\sqrt{N_{\text{train}}})\},$$

filtered to ensure $1 \leq k \leq N_{\text{train}}$. For each candidate k, we train a `KNeighborsRegressor` on the transformed target $\log(1+D)$ and evaluate the coefficient of determination R^2 on the training set. The value of k with the highest training R^2 is selected:

$$k^\star = \arg\max_k R^2_{\text{train}}(k).$$

The chosen $k^\star$ is then used consistently for both KNN regression and KNN classification, ensuring a single, data-driven neighbourhood size.

2.5 KNN Results and Evaluation

We evaluate the performance of KNN on both regression and classification tasks across the 12 assigned ISCAS benchmark circuits. All metrics are computed on the held-out test nodes.

Regression Metrics

All regression metrics are computed in the original detectability space, after inverting the $\log(1+D)$ transform applied during training. Given the true values D_i and the predicted values $\hat{D}_i$, we compute:

- **MAE (Mean Absolute Error)**:

$$\text{MAE} = \frac{1}{M}\sum_{i=1}^{M} |D_i - \hat{D}_i|$$

- **MSE (Mean Squared Error)**:

$$\text{MSE} = \frac{1}{M}\sum_{i=1}^{M} (D_i - \hat{D}_i)^2$$

- **RMSE (Root MSE)**:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- R^2 **Score**:

$$R^2 = 1 - \frac{\sum_i (D_i - \hat{D}_i)^2}{\sum_i (D_i - \bar{D})^2}.$$

Table 1 summarises the regression metrics for all 12 circuits.

Table 1: KNN regression results per circuit (numeric detectability D).

Circuit	MAE	MSE	RMSE	R^2
c17	0.9018	1.0773	1.0379	-0.6159
c1355	8.0812	217.9894	14.7645	0.9901
c1908	20.0175	1113.8261	33.3740	0.5178
c2670	82.6009	16953.4834	130.2055	0.2320
c3540	28.5338	2506.9385	50.0693	0.6201
c5315	35.3085	3053.6442	55.2598	0.3594
c6288	194.2112	57215.9196	239.1985	0.0385
c7552	87.9587	14135.9534	118.8947	0.3697
s27	2.3491	6.9186	2.6303	-0.3337
s13207	19.8497	4969.2025	70.4926	0.7778
s15850	19.4465	4674.2512	68.3685	0.7052
s38417	8.1999	647.9546	25.4550	0.7998

Classification Metrics

For classification, we focus on overall accuracy and the confusion matrix over the three classes (EASY, MEDIUM, HARD). Accuracy is defined as

$$\text{Accuracy} = \frac{\#\{\text{correct predictions}\}}{\#\{\text{total test samples}\}}.$$

Table 2 reports the KNN classification accuracy for each circuit.

Table 2: KNN classification accuracy per circuit (E/M/H).

Circuit	Chosen k	Accuracy [%]
c17	5	33.33
c1355	23	82.20
c1908	27	68.09
c2670	35	63.52
c3540	37	59.20
c5315	47	69.56
c6288	45	50.00
c7552	55	57.80
s27	5	50.00
s13207	83	73.95
s15850	91	81.03
s38417	139	88.93

To better understand which classes tend to be confused with each other, we inspect the confusion matrices for each circuit. In all matrices below, *rows* correspond to the *true* class and *columns* to the *predicted* class, in the order (EASY, MEDIUM, HARD).

KNN: c17 (Acc 33.33%)

	E	M	H
E	1	0	0
M	0	0	1
H	1	0	0

KNN: c1355 (Acc 82.20%)

	E	M	H
E	45	0	0
M	0	13	21
H	0	0	39

Figure 1: KNN confusion matrices for c17 and c1355.

KNN: c1908 (Acc 68.09%)

	E	M	H
E	54	5	3
M	9	28	29
H	3	11	46

KNN: c2670 (Acc 63.52%)

	E	M	H
E	74	18	10
M	10	60	40
H	5	29	61

Figure 2: KNN confusion matrices for c1908 and c2670.

KNN: c3540 (Acc 59.20%)

	E	M	H
E	79	26	15
M	22	51	37
H	18	24	76

KNN: c5315 (Acc 69.56%)

	E	M	H
E	147	12	13
M	15	103	53
H	19	46	111

Figure 3: KNN confusion matrices for c3540 and c5315.

KNN: c6288 (Acc 50.00%)

	E	M	H
E	89	33	40
M	53	53	56
H	45	18	103

KNN: c7552 (Acc 57.80%)

	E	M	H
E	158	52	40
M	77	105	70
H	31	49	174

Figure 4: KNN confusion matrices for c6288 and c7552.

KNN: s27 (Acc 50.00%)

	E	M	H
E	1	0	1
M	0	0	1
H	0	0	1

KNN: s13207 (Acc 73.95%)

	E	M	H
E	613	65	31
M	108	312	47
H	135	65	355

Figure 5: KNN confusion matrices for s27 and s13207.

KNN: s15850 (Acc 81.03%)

	E	M	H
E	645	58	25
M	105	487	52
H	106	48	551

KNN: s38417 (Acc 88.93%)

	E	M	H
E	1637	175	39
M	43	1273	63
H	61	147	1331

Figure 6: KNN confusion matrices for s15850 and s38417.

Discussion

Overall, the results demonstrate that KNN captures meaningful structural patterns in circuit testability. On several circuits (e.g. c1355, s15850, s38417) the model achieves both high classification accuracy and strong regression R^2, indicating that the chosen structural features and SCOAP-based information are informative. On more challenging benchmarks (e.g. c6288), performance degrades, suggesting that purely local structural features may be insufficient and motivating the use of a graph neural network that can exploit richer topological context.

3 GNN Model

In contrast to KNN, which relies on local feature similarity in Euclidean space, the GNN explicitly leverages the circuit graph structure through message passing. The goal is again to predict the SCOAP-based detectability $D[v]$ for each node, and to derive E/M/H classes from these predictions.

3.1 Graph Representation for GNN

We reuse the same node set and edge list extracted from the `.bench` parser. For compatibility with PyTorch Geometric, we construct:

- an index mapping idx_of : node_name $\mapsto \{0, \ldots, N-1\}$,
- an edge index matrix

$$\text{edge\_index} \in \mathbb{N}^{2\times E},$$

  containing both forward edges $(u \to v)$ and reverse edges $(v \to u)$, so that the GCN effectively operates on an undirected version of the circuit graph.

The node feature matrix X is built from the same structural+SCOAP features as in Section 1.3, followed by standardisation (fitted on all nodes for the GNN).

3.2 Targets and Train/Test Masks

The GNN is trained as a node-level regression model on the detectability scores $D[v]$. As with KNN, we apply a log-transform:

$$y[v] = \log(1 + D[v]),$$

and the GNN outputs an estimate $\hat{y}[v]$. At evaluation time we invert the transform via $\hat{D}[v] = \exp(\hat{y}[v]) - 1$ and compute the same regression metrics (MAE, MSE, RMSE, R^2) in the original D-space.

To ensure strict comparability with KNN, we reuse the exact same node split:

- nodes in `train_nodes` become `train_mask`,
- nodes in `test_nodes` become `test_mask`.

Thus, KNN and GNN are evaluated on the same test nodes, with identical labels.

3.3 GNN Architecture

We employ a simple two-layer Graph Convolutional Network (GCN) with a final linear readout:

- Input: normalised feature matrix $X \in \mathbb{R}^{N \times d}$,
- **Layer 1:** GCNConv($d \to h$), ReLU, dropout,
- **Layer 2:** GCNConv($h \to h$), ReLU, dropout,
- **Output:** linear layer mapping $h \to 1$ (scalar $\hat{y}[v]$).

In code, this corresponds to:

- hidden dimension $h = 64$,
- dropout rate 0.2 after each ReLU.

Formally, one GCN layer performs:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right),$$

where:

- $\tilde{A} = A + I$ is the adjacency matrix with self-loops,
- $\tilde{D}$ is the corresponding degree matrix,
- $H^{(0)} = X$ and $H^{(2)}$ is fed to the final linear layer.

3.4 Training Procedure

The training setup is:

- Optimiser: Adam with learning rate 0.005 and weight decay 10^{-4},
- Loss: Mean Squared Error (MSE) on the log-transformed targets,

$$\mathcal{L} = \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{v \in \mathcal{V}_{\text{train}}} (\hat{y}[v] - y[v])^2,$$

- Number of epochs: 1500.

At each epoch we perform a full forward and backward pass over the graph, but restrict the loss to nodes in `train_mask`. After training, we evaluate on nodes in `test_mask` (in original D-space) using the same regression metrics as for KNN.

3.5 GNN Classification (E/M/H)

To obtain qualitative E/M/H predictions from the GNN, we apply the same percentile-based thresholds used for the ground-truth labels:

1. Predict $\hat{D}[v]$ for all nodes by inverting the log-transform.
2. Apply the circuit-specific low_thresh and high_thresh to $\hat{D}[v]$ to assign a predicted class: EASY, MEDIUM, or HARD.

We then compute classification accuracy and the confusion matrix on the test nodes, allowing a direct comparison with the KNN classifier.

3.6 GNN Results and Evaluation

As with KNN, we report both regression metrics and E/M/H classification behaviour.

Regression Metrics

Table 3 summarises the GNN regression performance on numeric detectability D for each circuit (after inverting the log-transform).

Table 3: GNN regression results per circuit (numeric detectability D).

Circuit	MAE	MSE	RMSE	R^2
c17	0.8257	0.8235	0.9075	-0.2352
c1355	17.4655	647.6320	25.4486	0.9707
c1908	20.1029	969.6974	31.1400	0.5802
c2670	62.6628	9785.5244	98.9218	0.5567
c3540	22.9231	1222.6619	34.9666	0.8147
c5315	28.4267	2089.8276	45.7146	0.5616
c6288	152.4724	36789.3711	191.8056	0.3817
c7552	77.0347	11674.1504	108.0470	0.4795
s27	2.0258	5.1937	2.2790	-0.0012
s13207	22.6227	8292.6602	91.0640	0.6291
s15850	26.0224	5209.9399	72.1799	0.6714
s38417	14.7383	1262.3695	35.5298	0.6099

Classification Metrics

Table 4 reports the GNN classification accuracy for each circuit, using the same E/M/H thresholds as the ground truth.

Table 4: GNN classification accuracy per circuit (E/M/H).

Circuit	Accuracy [%]
c17	66.67
c1355	67.80
c1908	66.49
c2670	64.50
c3540	60.63
c5315	77.46
c6288	48.98
c7552	62.96
s27	75.00
s13207	72.91
s15850	65.72
s38417	81.57

As before, we also examine the confusion matrices to understand which classes are typically confused. The layout mirrors the KNN case: rows are true classes, columns are predicted classes (E, M, H).

GNN: c17 (Acc 66.67%)

	E	M	H
E	1	0	0
M	0	1	0
H	0	1	0

GNN: c1355 (Acc 67.80%)

	E	M	H
E	40	5	0
M	0	34	0
H	0	33	6

Figure 7: GNN confusion matrices for c17 and c1355.

GNN: c1908 (Acc 66.49%)

	E	M	H
E	53	9	0
M	12	41	13
H	2	27	31

GNN: c2670 (Acc 64.50%)

	E	M	H
E	87	15	0
M	8	53	49
H	2	35	58

Figure 8: GNN confusion matrices for c1908 and c2670.

GNN: c3540 (Acc 60.63%)

	E	M	H
E	85	31	4
M	18	75	17
H	6	61	51

GNN: c5315 (Acc 77.46%)

	E	M	H
E	136	35	1
M	5	148	18
H	1	57	118

Figure 9: GNN confusion matrices for c3540 and c5315.

GNN: c6288 (Acc 48.98%)

	E	M	H
E	103	59	0
M	24	137	1
H	9	157	0

GNN: c7552 (Acc 62.96%)

	E	M	H
E	144	106	0
M	13	231	8
H	1	152	101

Figure 10: GNN confusion matrices for c6288 and c7552.

GNN: s27 (Acc 75.00%)

	E	M	H
E	2	0	0
M	0	1	0
H	0	1	0

GNN: s13207 (Acc 72.91%)

	E	M	H
E	512	164	33
M	61	322	84
H	4	123	428

Figure 11: GNN confusion matrices for s27 and s13207.

GNN: s15850 (Acc 65.72%)

	E	M	H
E	378	341	9
M	68	498	78
H	1	215	489

GNN: s38417 (Acc 81.57%)

	E	M	H
E	1426	376	49
M	33	1078	268
H	35	118	1386

Figure 12: GNN confusion matrices for s15850 and s38417.

Discussion

Across most benchmarks, the GNN achieves competitive or superior regression performance compared to KNN, especially on larger circuits (e.g. c3540, c5315, c2670), where exploiting graph structure provides a clear benefit. Classification accuracy also improves for several circuits (c17, c5315, c7552, s27), although on others KNN remains stronger (e.g. s15850, s38417). Overall, the comparison confirms that local structural features already carry significant information, but that graph-aware message passing can further enhance testability prediction on complex designs.

4 Comparison Between KNN and GNN

In this section we directly compare the behaviour of the two models across all circuits, using the regression tables (Tables 1 and 3) and classification tables (Tables 2 and 4), as well as the confusion-matrix figures.

4.1 Regression Performance

Looking at the R^2 values, KNN already provides a strong baseline on many circuits, with particularly high scores on c1355, s13207, s15850 and s38417. The GNN further improves the fit on several mid-to-large benchmarks (e.g. c1908, c2670, c3540, c5315), where the graph structure carries additional information that is not fully captured by local node features alone.

On highly irregular circuits such as c6288, both models struggle, but the GNN still achieves a noticeably higher R^2 than KNN. For very small designs (c17, s27) both approaches exhibit low or even negative R^2, which is expected due to the tiny number of nodes and the coarse granularity of the detectability scale.

Overall, the GNN tends to reduce MSE and increase R^2 on circuits with richer topology, confirming that message passing between neighbouring nodes helps approximate SCOAP-based detectability more accurately.

4.2 Classification Performance

For E/M/H classification, the picture is more mixed. KNN achieves very high accuracy on some large circuits (e.g. s15850, s38417), suggesting that the percentile-based thresholds combined with local structural features already separate the three difficulty levels well. The GNN sometimes lags slightly behind on these benchmarks, but still remains competitive.

On other circuits (notably c17, c5315, c7552 and s27), the GNN improves classification accuracy relative to KNN. In these cases the confusion matrices show that GNN predictions are more concentrated along the diagonal, with fewer severe misclassifications between EASY and HARD. Misclassifications typically occur between neighbouring classes (EASY vs. MEDIUM or MEDIUM vs. HARD), which is acceptable given the ordinal nature of the labels.

4.3 Qualitative Behaviour

The confusion matrices reveal qualitative differences:

- **KNN** sometimes over-relies on the dominant class in the local feature space, leading to blocks of misclassified nodes in MEDIUM or HARD when EASY nodes dominate

a region of feature space.

- **GNN** tends to smooth predictions along the graph, which can correct isolated mislabelled nodes surrounded by correctly classified neighbours, but may also propagate local biases when the training signals are uneven.

In summary, KNN offers an interpretable and surprisingly strong baseline, especially for classification on some circuits, while the GNN generally provides better regression performance and more robust predictions on structurally complex benchmarks. A practical flow might therefore use KNN as a fast baseline and apply the GNN when higher accuracy is needed or when analysing larger circuits where topology plays a more significant role.

5 Division of Work and Main Findings

This section briefly documents the division of work between the two authors and summarises how the KNN and GNN results were compared on the given circuits, highlighting only the most important differences in accuracy.

5.1 Division of Work

The implementation effort for Part B was shared as follows:

- **KNN pipeline (Lefkios Mavroudi).** Lefkios implemented the complete KNN baseline, including: feature normalisation with `StandardScaler`, the regression model on the log-transformed detectability values, the E/M/H classifier, the selection of the optimal k, and the extraction of all KNN metrics and confusion matrices per circuit.
- **GNN pipeline (Alexandros Vryonides).** Alexandros implemented the GNN model in PyTorch Geometric, including: construction of the graph data structure (`edge_index` and node feature matrix), the two-layer GCN architecture, the training loop on the log-transformed targets, and the derivation of E/M/H predictions and corresponding metrics.
- **Shared components.** Both authors contributed to the SCOAP implementation, node feature engineering, generation of the EASY/MEDIUM/HARD labels, and the common train/test split that is reused by both models. The analysis of the tables and confusion matrices was also carried out jointly.

This division ensured that each model was developed and debugged by a different person, while the shared preprocessing guaranteed a fair and direct comparison.

5.2 Accuracy Comparison Methodology

To compare KNN and GNN on the same footing, we adopted the following procedure:

- For each ISCAS circuit, we used the *same* node-level train/test split (Section 2 and Section 3), stratified by the E/M/H labels.
- Both models predicted the log-transformed detectability $\log(1 + D[v])$, which was then inverted back to $D[v]$ before computing MAE, MSE, RMSE and R^2 (Tables 1 and 3).
- E/M/H classification accuracy was measured using the same circuit-specific percentile thresholds for both models (Tables 2 and 4).

- Confusion matrices were inspected to understand which classes (EASY, MEDIUM, HARD) each model tended to confuse on each circuit.

Thus, any differences in performance are due to the learning method (KNN vs. GNN) and not to differences in data, preprocessing, or evaluation protocol.

5.3 Major Differences in Observed Accuracies

Focusing only on the main trends across all circuits, the comparison can be summarised as follows:

- **Regression (numeric detectability).** On medium and large combinational circuits such as c1908, c2670, c3540 and c5315, the GNN generally achieves higher R^2 and lower error than KNN. This indicates that explicitly propagating information along the circuit graph (via message passing) captures dependencies that local feature similarity alone (KNN) cannot fully exploit. On very small circuits (c17, s27) both models are limited by the small number of nodes, and neither clearly dominates.

- **Classification (E/M/H labels).** For the E/M/H task, KNN remains very strong on some large circuits (notably s15850 and s38417), where its local feature-based decision rule already aligns well with the percentile thresholds. On several other circuits (e.g. c17, c5315, c7552 and s27), the GNN provides higher accuracy and confusion matrices that are more concentrated along the diagonal, i.e. fewer drastic EASY$\leftrightarrow$HARD mistakes.

- **Overall behaviour.** In broad terms, KNN offers a fast, interpretable baseline that performs particularly well for the categorical E/M/H labels on some benchmarks, while the GNN usually provides better regression performance and more stable predictions on structurally complex circuits. The two approaches are therefore complementary: KNN is a strong baseline and sanity check, whereas the GNN can be used when higher accuracy on detectability is needed or when analysing larger designs where graph structure is crucial.