

POLITECNICO DI TORINO

Faculty of Engineering

Master of Science in Mechatronic Engineering

Master Thesis

Graph Neural Networks for Topology Recognition of AMS Integrated Circuits



Advisor

prof. Daniele J. Pagliari

Candidate:

Aurelio Teliti

Company tutors
STMicroelectronics

Dr. Dalibor Barri

October 2023

Scienza è il distinguere quello che si sa da quello che non si sa.
Galileo Galilei

Summary

Due to the complex nature of the constraints and objectives involved, developing Analog and Mixed Signal (AMS) Integrated Circuits is still mainly a manual process, and existing Electronic Design Automation (EDA) tools are still limited in their capabilities or restricted to a few specific steps. However, recent developments in AI (Artificial Intelligence) have opened up novel prospects for automating AMS design procedures. Specifically, identifying circuit topologies within a netlist is a crucial aspect of AMS EDA, as certain circuit structures necessitate specific constraints, for example, in terms of their placement on the layout (e.g., symmetry, matching, etc.). Traditionally, topology recognition has relied on subgraph isomorphism algorithms like VF2. Unfortunately, these methods suffer from long execution times when applied to large netlists. The objective of this thesis is to create an AI-driven pipeline for topology recognition capable of achieving both high accuracy and a reduction in computational time compared to the previous methods, like VF2.

The proposed pipeline is composed of three phases. An initial pre-processing refines and prepares the dataset, enhancing the quality and relevance of the input data. In order to do so, this process removes all information not relevant to topology recognition. The central part of the pipeline consists of a Graph Neural Network (GNN). The GNN receives as input a bipartite graph that comprises two distinct sets of nodes. One set encompasses the various devices within the circuit, such as transistors, while the other encapsulates the network connections, or nets, that link these devices together. The bipartite graph's ability to maintain the relationships between devices provides the GNN with a comprehensive view of the circuit. The GNN then learns the patterns within the circuit that correspond to different topologies. Lastly, in the third stage, post-processing is executed, in which the outcomes generated by the GNN undergo further refinement using the VF2 algorithm. This combined approach leverages the strengths of both the GNN and VF2, resulting in a topology recognition process that is both robust and accurate.

To assess the effectiveness of this pipeline, the thesis undertakes a comparative analysis among three scenarios: utilizing the GNN in isolation, combining the GNN with VF2, and employing VF2 as a standalone tool. This comprehensive evaluation offers valuable insights into the capabilities of the entire pipeline. The obtained results, conducted on a graph with roughly 700,000 nodes, demonstrate the excellent performance of the solution compared to the VF2 algorithm. The accuracy is the same for both solutions, 0.9999. However, precision has improved from 0.9641 (VF2 algorithm) to 0.9662 (GNN + VF2), while recall has slightly decreased from 0.9860 (VF2 algorithm) to 0.9808 (GNN + VF2). The most significant results were achieved in terms of computational time, which decreased from an average of 10 hours and 54 minutes to an average of 1 minute and 57 seconds.

Contents

List of Figures	VII
List of Tables	X
1 Introduction	1
2 Background	4
2.1 Analog Mixed-Signal Circuits	4
2.1.1 Definition of Topologies	5
2.2 Graph Neural Networks	7
2.2.1 What is a Graph?	7
2.2.2 Machine Learning and Deep Learning	8
2.2.3 Graphs in Machine Learning	10
2.2.4 Inductive and Transductive Learning	13
2.2.5 Main GNN Architectures	15
2.3 A Graph representation for Circuits	21
2.4 VF2 Algorithm	22
2.5 The AMBEATion Project	24
3 Related Work	27
3.1 Comparative Analysis of Topology Recognition Techniques	28
4 Materials and Methods	31
4.1 Extend the Graph Representation for Circuits with Bulk Connectivity Information	31
4.2 Pre-Processing Techniques	32
4.3 Training	33
4.3.1 Resources	34
4.3.2 Architectures Exploration	34
4.3.3 Features Selection	36

4.3.4	Addressing Class Imbalance	38
4.4	VF2 as Post-Processing Technique	39
5	Results	41
5.1	Metrics	41
5.1.1	Confusion Matrix	41
5.1.2	Accuracy, Precision and Recall	42
5.2	VF2 Results	43
5.3	GNNs Results	45
5.3.1	Results without Pre-Processing	45
5.3.2	Results with Pre-Processing	49
5.4	GNN and VF2 Results	53
6	Conclusion and Future Works	57
A	Acronyms	59
	References	60

List of Figures

2.1	Mixed-signal integrated circuit: The metal areas on the right-hand side are capacitors, on top of which are large output transistors; the left-hand side is occupied by the digital logic. [8]	5
2.2	An example of current mirrors	6
2.3	An example of differential pair	6
2.4	Comparison between undirect edge (on the left) and direct edge (on the right). [13]	8
2.5	Illustration of a MLP: The input layer is depicted by blue dots, the hidden layer is denoted by black dots, and the output layer is by green dots.	9
2.6	Adjacency matrices that represent the same graph. [13]	11
2.7	Node embedding update. [13]	13
2.8	An end-to-end prediction task with a GNN model. [13]	13
2.9	Node classification in transductive settings. At the training time, the learning algorithm has access to all the nodes and edges, including those nodes for which labels are to be predicted (denoted by question marks). [19]	14
2.10	Node classification in inductive settings. Once learned, the model can be applied to new unseen nodes (denoted by question marks). There may or may not exist edges between such new nodes and the nodes used for training. [19]	14
2.11	Multi-layer Graph Convolutional Network (GCN) [21]	15
2.12	Visual illustration of the GraphSAGE Sample and Aggregate approach. [22]	17

2.13	Left: The attention mechanism ($W\vec{h}_i, W\vec{h}_j$) employed by our model, parameterized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a Leaky ReLU activation. Right: An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain \vec{h}'_1	19
2.14	Illustration depicting the operational mechanism of RGCN. [24]	21
2.15	(a) An NMOS current mirror primitive, CM-N(2), with two transistors. (b) Its representation as a bipartite graph. [4] . . .	22
2.16	(a) A differential OTA (for simplicity, body connections are not shown). (b) Its bipartite graph, showing the subgraph that can be recognized as a current mirror (for clarity, edge labels are not shown). [4]	22
2.17	An instance of subgraph matching. The graph on the left (pattern graph), is present four times within the graph on the right (target graph)	23
3.1	Chip Design Flow. [13]	27
4.1	Cross-sectional view of a field-effect transistor, showing source, gate, drain and body (or bulk) terminals [31]	32
4.2	Left: The original circuit containing resistors and capacitors, detected as a non-transistor device. Right: After applying the pre-processing technique, the resistor's terminations are short-circuited.	34
4.3	Comprehensive Graph Neural Network Architectures which include a pre-processing layer, a message passing layer and a post-processing layer	36
4.4	Features from the dataset provided by STMicroelectronics . .	37
5.1	Confusion matrix for binary classification [39]	42
5.2	Results of the topology recognition of CMs using the VF2 algorithm	44
5.3	Results of the topology recognition of CMs for GCN model without pre-processing	46
5.4	Results of the topology recognition of CMs for GraphSAGE model without pre-processing	47

5.5	Results of the topology recognition of CMs for GAT model without pre-processing	48
5.6	Results of the topology recognition of CMs for RGCN model without pre-processing	49
5.7	Results of the topology recognition of CMs for GCN model with pre-processing	50
5.8	Results of the topology recognition of CMs for GraphSAGE model with pre-processing	51
5.9	Results of the topology recognition of CMs for GAT model with pre-processing	52
5.10	Results of the topology recognition of CMs for RGCN model with pre-processing	54
5.11	Comprehensive Workflow Pipeline, the complete sequence of data preprocessing, RGCN-based classification, and post-processing steps in the project’s workflow	55
5.12	On the left , the confusion matrix without post-processing, and on the right , the confusion matrix with post-processing .	55

List of Tables

4.1	Class Distribution of the Dataset	38
5.1	Multiple test of VF2 Algorithm Performance on a Dual-core, 8 GB RAM Setup	44
5.2	Accuracy, Precision, and Recall results for topology recognition of CMs using VF2 algorithm	45
5.3	Accuracy, Precision, and Recall results for topology recognition of CMs for GCN model without pre-processing	46
5.4	Accuracy, Precision, and Recall results for topology recognition of CMs for GraphSAGE model without pre-processing	47
5.5	Accuracy, Precision, and Recall results for topology recognition of CMs for GAT model without pre-processing	48
5.6	Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model without pre-processing	49
5.7	Accuracy, Precision, and Recall results for topology recognition of CMs for GCN model with pre-processing	50
5.8	Accuracy, Precision, and Recall results for topology recognition of CMs for GraphSAGE model with pre-processing	51
5.9	Accuracy, Precision, and Recall results for topology recognition of CMs for GAT model with pre-processing	53
5.10	Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model with pre-processing	53
5.11	Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model with pre-processing and post-processing	56
5.12	Comparison of multiple test of only RGCN and RGCN + VF2 time performance on a Dual-core, 8 GB RAM Setup	56

Chapter 1

Introduction

Driven by the rising demands of mobile communication, consumer electronics, and automotive and medical applications, analog and mixed-signal (AMS) integrated circuits (ICs) are projected to grow at an 11% annual rate [1]. Analog circuit design and verification processes are time-consuming tasks and highly dependent on human experience [2, 3]. From the generation of the topology of the circuit to device-sizing, placement, and routing, the whole design process is subject to many constraints. The latter has to be obeyed to guarantee the functionality and robustness of the circuit, which is, e.g., influenced by process variations, changing operating conditions, and parasitics. On the other hand digital design flows are highly automated, making the analog part a huge bottleneck during the design of modern systems on chip (SoCs) containing AMS modules [3]. Hence, several efforts in the related research community are focusing on automating, as much as possible, the most critical/essential tasks related to analog circuit design, verification, and layout generation. In the meantime, the recent developments in the area of machine learning (ML) do provide powerful instruments and models that can be used to achieve this challenging goal. Given the particular structure of circuits, a particularly suited category of ML models are the so-called Graph Neural Networks (GNN).

GNNs learn to propagate information across the nodes of a graph, leveraging both local and global dependencies. This ability to capture relational information allows GNNs to make informed predictions or classifications based not only on the features of individual nodes but also on the context provided by their neighbors. For instance, in a recommendation system using a GNN, the GNN can capture the relationships between users and items in a recommendation graph. This enables it to make personalized recommendations by

considering how users are connected to various items, unlike a simple feedforward neural network, which treats each user or item independently without considering their connections in the graph, which are essential for capturing the underlying structure and patterns in graph data. In recent years, GNNs have been used to build models able to classify either analog circuits [4] and digital circuits [5]. In both cases, as in this work, these circuits were converted from their equivalent text representations, or what is called “netlists”, to a universal graph representation suitable for GNNs.

The primary objective of this thesis is to employ a GNN to tackle the task of recognizing topologies within a netlist. Traditional methods for topology recognition often rely on manual inspection or rule-based algorithms, which can be time-consuming. On the other hand, since GNNs have the ability to capture complex relationships and patterns within graph-structured data, they are particularly promising for automating this process. Recognizing circuits topologies is a fundamental step to automate AMS IC placement, since different topologies require different types of constraints during placement, such as symmetry and matching between devices [6]. In this context, a "topology" refers to arrangement or configuration of electronic components (such as resistors, capacitors, inductors, transistors, etc.). These group of components are interconnected to perform a specific function. For instance, examples are "current mirrors" and "differential pairs." The topology of a current mirror defines how transistors are connected to replicate a reference current, while the topology of a differential pair outlines how two transistors are arranged to amplify the difference between two input voltages. In the context of AMS IC placement, recognizing these diverse topologies becomes crucial because each topology may have specific requirements for optimal performance, and automating their placement is an important step during the physical design and manufacturing of these circuits. By identifying the topology of a given circuit, appropriate placement strategies can be applied, ensuring that the resulting layout meets the required performance criteria and design specifications.

In this thesis, a bipartite graph is used to represent the connectivity structure of circuits. A bipartite graph consists of two sets of nodes: one set represents the nets between devices, while the other set represents the devices (e.g., transistors, resistors). By using this representation, the relationships between networks and devices can be fully captured. After converting netlists to graphs, different types of GNN are designed to learn and analyze the structure and patterns of the graph. In order to allow for the extraction of as much information as possible from both the nodes and the edges of the

graph, we associate features to both nodes and edges of the bipartite graph.

The outcomes of the GNN are then compared with those of a more traditional algorithm that relies on graph matching as a primary feature for topology detection. However, as explained before, this approach has notable drawbacks. The traditional algorithm depends on a library, which involves matching a circuit to pre-established templates, necessitating an exhaustive enumeration of potential topologies within a database. Furthermore, the reliance on searching through all potential vertex pairings between the two graphs results in a substantial computational cost issue, particularly when dealing with large graphs, which in turn translates into several hours of execution time on modern server-class processor. The use of GNNs in this work aims to address both of these issues, achieving a topology recognition process that is more flexible and efficient.

The thesis is organized as follows: Chapter 2 explains the background and necessary theory to understand the project. What is a current mirror, how a circuit is translated in a graph, what is a GNN, and how the problem is addressed. Chapter 3 gives a brief overview of the GNN approaches for topology recognition that have been considered in this work, including a comprehensive analysis of the existing works that propose different or similar solutions. Chapter 4, on the other hand, introduces all the problems and the approaches taken to address these issues in order to achieve the final results. Chapter 5 presents all the results obtained during the work. Finally, Chapter 6 addresses some final thoughts and considerations on the results and, most importantly, on the overall project, as well as some comments on possible future works.

This thesis has been carried out as part of the Horizon 2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange (RISE) project AMBEAT-ion (Analog/Mixed Signal Back End Design Automation based on Machine Learning and Artificial Intelligence Techniques). The aim of the projects is to reduce the handmade flows by actively working on developing better physical design and verification methods in particular for AMS integrated circuit placement.

Chapter 2

Background

This chapter covers all the theoretical aspects and background that are necessary to fully understand the project in all its components. The chapter is divided into three different sections. The first section (2.1) defines AMS integrated circuits and topologies. The second section (2.2) gives an overview of the background that is required to fully understand GNNs. The third section (2.5) gives an overview of how this work is integrated in the "AMBEATion Project".

2.1 Analog Mixed-Signal Circuits

A mixed-signal integrated circuit (Fig. 2.1) is an integrated circuit that has both analog devices and digital devices on a single semiconductor die [7]. Designing and manufacturing AMS ICs presents a greater challenge when compared to or digital-only integrated circuits. This complexity arises because they involve both traditional active elements, such as transistors, and high-performance passive elements like coils, capacitors, and resistors, all integrated into the same chip. This intricate fusion of active and passive components enables mixed-signal ICs to bridge the gap between continuous analog signals and discrete digital information processing. As a result, these circuits find a diverse range of applications in modern electronics, facilitating the seamless interaction between analog and digital domains. From precision measurement instruments to communication systems and sensor networks, the application of analog mixed-signal technologies brings forth versatile solutions that exploit the strengths of both analog and digital.

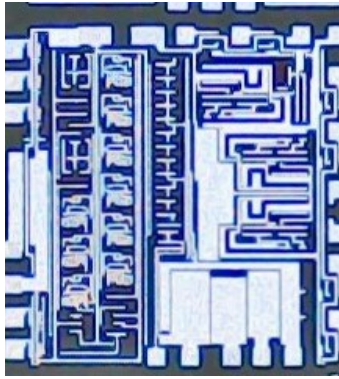


Figure 2.1: Mixed-signal integrated circuit: The metal areas on the right-hand side are capacitors, on top of which are large output transistors; the left-hand side is occupied by the digital logic. [8]

2.1.1 Definition of Topologies

Different topologies can be found within analog parts in AMS integrated circuits, and being able to recognize them without human intervention is very important to automate the whole IC design process. This becomes particularly important during the placement phase, especially for analog circuits. Analog circuits require symmetric positioning to avoid issues such as imbalanced signal paths, cross-talk, and undesirable interference. Achieving a balanced and symmetrical layout is essential for preserving signal integrity and minimizing undesirable coupling effects between components. Automatic topology detection and placement optimization mechanisms play a crucial role in ensuring that analog circuits function optimally and meet performance requirements.

In this work, the AMS netlists considered included a variety of distinct topologies:

- **Current Mirrors (CMs):** Current mirrors (Fig. 2.2) are a fundamental building block in analog circuit designs. They facilitate the replication of a reference current by mirroring it onto another branch. This topology finds widespread use in biasing, amplification, and regulation circuits.
- **Differential Pairs (DPs):** Differential pairs (Fig. 2.3) are arrangements of two transistors with their sources connected and their gates receiving complementary signals. This topology is fundamental for differential signal processing, used in amplifiers, comparators, and balanced circuits.

- **DPx (Differential Pairs with Additional Transistors):** DPx represents an extension of the basic differential pair by introducing additional transistors for improved performance. These added transistors can enhance common-mode rejection, increase linearity, or provide other desirable characteristics.
- **Cascode and Long-Tailed Pair (CascLV):** The cascode topology involves connecting the collector of a transistor to the base of another, enhancing performance by improving bandwidth and output impedance. Long-tailed pairs are differential pairs where one transistor has a tail current source, improving common-mode rejection. This topology is commonly used in differential amplifiers.

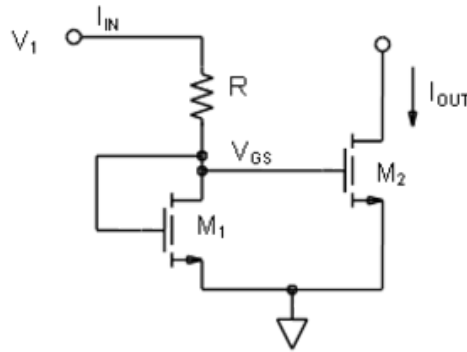


Figure 2.2: An example of current mirrors

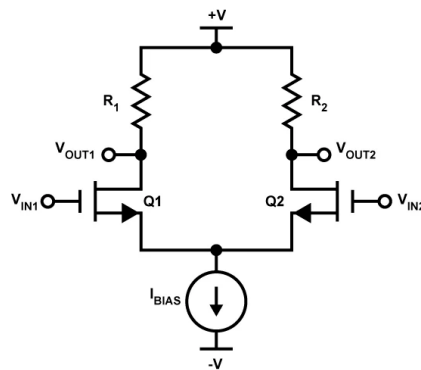


Figure 2.3: An example of differential pair

Each of these topologies plays a distinct role in analog IC (integrated

circuits) design, offering specific advantages and addressing particular challenges. Understanding and accurately identifying these topologies within the dataset are pivotal steps in effectively analyzing and designing analog circuits.

2.2 Graph Neural Networks

Graphs are all around us; real-world objects are often defined in terms of their connections to other things. For instance, social networks like Facebook and Twitter can be described as large graphs using the relationships between individuals. Similarly, transportation systems, such as subway networks, can be represented using graphs to illustrate the connections between stations and routes. Researchers have developed neural networks that operate on graph data, called Graph Neural Networks (GNNs), for over a decade [9]. Algorithms that process graphs to identify paths, assign values to nodes, and more, have been a fundamental part of AI and computer science since its birth. For example, Dijkstra’s algorithm is used to find the shortest path in networks, and the PageRank algorithm is employed by search engines like Google to rank web pages based on their link structures. Moreover, the recent developments in deep learning, driven by the increased computational power, the availability of large-scale labeled graph datasets, and advances in optimization techniques for training and inference, have also led to an increase in the capabilities and expressive power of GNNs. Consequently, they are increasingly being utilized across a spectrum of applications and domains. Practical applications are: antibacterial discovery [10], fake news detection [11] and recommendation systems [12].

This sub-section provides an introduction to Graph Neural Networks. First, a look is given at graphs with some common examples. The second part explains how graphs can be used in the context of machine learning. In the third part, the differences between inductive and transductive learning are explained. Lastly, the architectures used in this work are introduced.

2.2.1 What is a Graph?

A graph is a set of objects, called nodes (or vertices), and a set of connections between pairs of vertices, called edges. Graphs are employed in various real-world scenarios because they can represent and model different relationships within a given context. These relationships are typically defined by

connections or interactions between individual elements or entities within a system. Graphs, with their nodes (representing elements) and edges (representing connections), offer a framework for visually and computationally expressing how these elements relate to one another. For example, in a social network, individuals (nodes) are connected by friendships or interactions (edges). In a transportation network, locations or stations (nodes) are linked by routes or roads (edges). The nature of these relationships can vary, including friendships, distances, dependencies, similarities, and more. Graphs provide a flexible and intuitive means to capture and analyze these relationships, making them useful for addressing problems where understanding and leveraging these relationships is crucial. Here are defined some key terms associated with graphs:

- **Node:** is a representation of an entity within the graph. Nodes can represent people, products, or any other kind of entity related to a particular problem.
- **Edge:** is a connection between two nodes, indicating a relationship between them. Edges can be directed or undirected (as shown in Figure 2.4) and may have associated attributes or weights as additional information



Figure 2.4: Comparison between undirect edge (on the left) and direct edge (on the right). [13]

2.2.2 Machine Learning and Deep Learning

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behaviour. Machine learning involves systems that learn from training data to perform specific tasks [14]. Deep learning, on the other hand, refers to the use of layered artificial neural networks to extract increasingly higher-level features from data. A neural network is a computational model inspired by the structure and function of the human brain. It consists of interconnected nodes,

called neurons, that work together to process and transmit information. Each neuron receives input signals, applies weights to them, and produces an output signal based on a specific activation function. These computations are performed in parallel across the network, allowing for complex information processing.

In the 1950s Frank Rosenblatt expanded on this model and developed an algorithm, known as the Perceptron [15], that could learn the weights to generate an output. This marked an important advancement in neural network development. In this context, it is important to introduce the Multilayer Perceptron (refer to Figure 2.5), or MLP, a type of neural network composed of multiple layers of processing units, including an input layer, one or more hidden layers, and an output layer. Each processing unit, or neuron, in the MLP is connected to all neurons in the adjacent layers. The information flows through the network from the input layer (blue dots), through the hidden layers (black dots), and finally to the output layer (green dots). MLPs are particularly effective in solving complex problems that require non-linear relationships between input and output. Through training, an MLP can learn to adjust the weights of its connections to produce the desired output for a given set of inputs. This ability to learn from training data is achieved through a process known as backpropagation, where the network adjusts its weights based on the error between the predicted output and the true output.

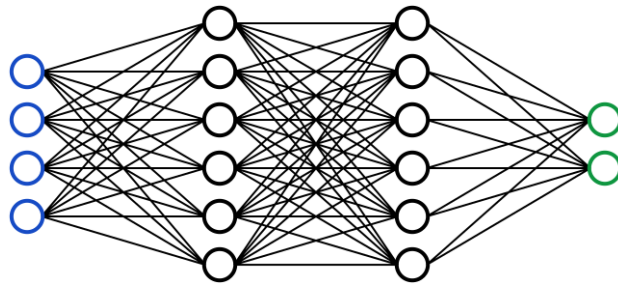


Figure 2.5: Illustration of a MLP: The input layer is depicted by blue dots, the hidden layer is denoted by black dots, and the output layer is by green dots.

It's important to note that there are different types of learning paradigms in machine learning and deep learning, such for instance Supervised Learning. Supervised learning algorithms require a training set of example inputs and their corresponding desired outputs, which the algorithm uses to learn a

model of the mapping from inputs to outputs. Once the model is learned, the algorithm can generate outputs for new inputs. Unsupervised learning algorithms do not require a labeled training set, but instead, learn a model of the input data by detecting patterns in it. Unsupervised learning algorithms can be used to discover structure in data or to cluster data into groups. Lastly, self-supervised learning algorithms also work with unlabeled input data; the desired outputs are not provided. Instead, the algorithm learns a model of the input data by learning, for example, to reconstruct the input after passing through multiple transformations, or to restore a missing part of an input (a word removed from a sentence, or a patch from an image, etc). Other forms of learning, such as Reinforcement Learning also exist, but they are out of the scope of this work.

2.2.3 Graphs in Machine Learning

There are three general types of prediction tasks on graphs: graph-level, node-level, and edge-level. In a graph-level task, a single property for an entire graph or a sub-graph is predicted. For instance, in social network analysis, the prediction might concern whether a social community (a sub-graph of the entire network) is more likely to adopt a new trend or not. In a node-level task, a prediction is made regarding some property for each node in a graph. As an example, in a recommendation system, the likelihood of a user engaging with specific content, such as movies or products, is predicted for each user in a social network. In the case of an edge-level task, the goal is to predict the property or presence of edges in a graph. In a biological network, predictions might revolve around whether a protein-protein interaction exists between pairs of proteins, thereby determining the edges in the network. The first challenge in addressing these prediction tasks is how to represent a graph in a format that is compatible with neural networks. Neural networks typically require input data in the form of tensors, which are multidimensional arrays or vectors with a fixed number of dimensions (N-dimensions). Therefore, an important initial step in solving graph-based prediction problems is to find a way to convert the complex structure of a graph into a format that can be used as a tensor and subsequently fed into a neural network.

One solution can be to use the graph adjacency matrix (Fig. 2.6). However, this representation has drawbacks: the number of nodes in a graph can be on the order of millions and the number of edges for a node can be highly variable, which could lead to a very sparse adjacency matrix. Another problem is that there are many adjacency matrices that encode the same problem

and there is no guarantee that these different matrices would produce the same results in a deep neural network. In other words, this solution is not permutation invariant.

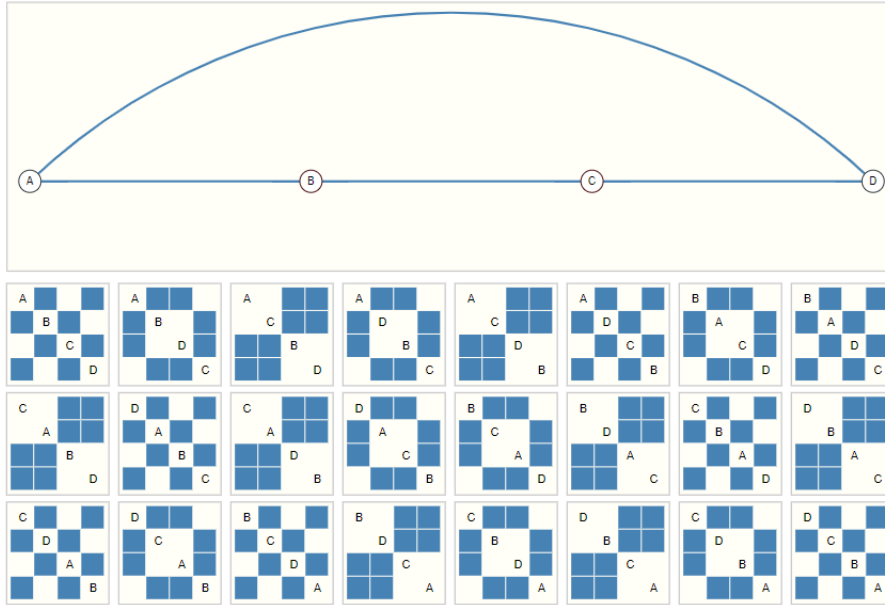


Figure 2.6: Adjacency matrices that represent the same graph. [13]

To address those issues one way of representing sparse matrices is as adjacency lists. This solution provides a more efficient way to represent sparse graphs by only storing information about existing connections, leading to a more memory-efficient way to represent to same structure. Additionally, the permutation invariance problem is mitigated, making this solution more suitable and digestible when using neural networks.

Given these elements, it is now possible to describe a simple GNN architecture. A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global context) that preserves graph symmetries (permutation invariance). Modern GNNs architectures are based on the "message passing neural network" framework [16] and the Graph Nets architecture schematics introduced in [17], in which GNNs adopt a "graph-in, graph-out" architecture meaning that these model types accept a graph as input and progressively transform embeddings *, without changing the connectivity of the input graph. The simplest Graph Neural Network (GNN) architecture

*In the context of Graph Neural Networks (GNNs), an embedding refers to a vector representation of nodes, edges, or entire graphs in a graph data structure. These embeddings

can be understood as a series of iterative message-passing steps, where each node in the graph aggregates information from its neighbors to update its own representation using an aggregation function. This process is repeated for multiple iterations to capture increasingly refined features and relationships within the graph.

Here’s a breakdown of the key stages within a Graph Neural Network (GNN):

- **Initialization:** Each node in the graph is assigned an initial embedding, often corresponding to its own features or attributes.
- **Message Passing:** each node aggregates information from its neighbors. This information typically includes the embeddings of neighboring nodes and the embedding of the edges connecting the node with its neighbors. The aggregated information can be obtained using different functions (e.g. sum, max, etc.).
- **Updating Node Embeddings:** Once nodes have gathered information from their neighbors (Fig. 2.7), they update their own embeddings by incorporating the aggregated information. This update involves combining the node’s current embedding with the aggregated information using an MLP. This step allows nodes to refine their representations based on local neighborhood information.

GNN layers can be stacked to create deeper architectures. Stacking multiple GNN layers allows the network to capture more complex and abstract features by aggregating information from increasing distances in the graph. The output embeddings of one GNN layer serve as the input embeddings for the next layer. The embeddings generated by GNNs encapsulate both local and global information from the graph:

- **Local Information:** In the early layers, each node’s embedding is influenced primarily by its immediate neighbors. This allows nodes to capture local structural patterns and relationships within their neighborhoods.
- **Global Information:** As the layers progress, the updated embeddings start to capture more global information. Information from distant nodes gradually propagates through the network, enabling nodes to learn about broader graph-wide properties and structures.

capture the essential features and characteristics of graph elements in a vector space

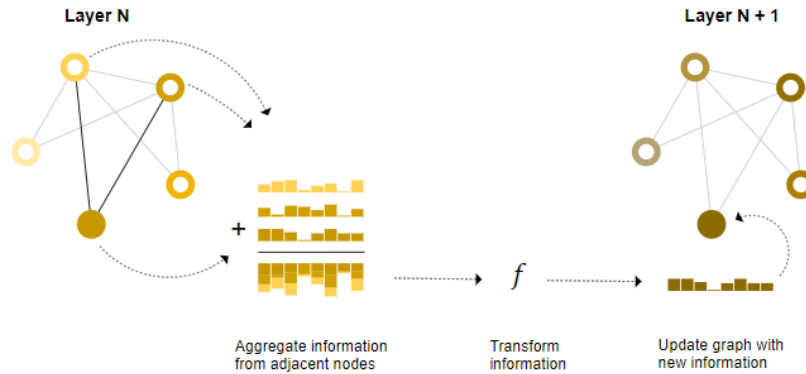


Figure 2.7: Node embedding update. [13]

After iteratively repeating the initial three steps mentioned earlier (initialization, message passing, and updating), the resulting final node embeddings become ready for utilization in a variety of downstream tasks, as shown in Figure 2.8. These applications include node classification, where the refined embeddings serve as inputs for a MLP model to make informed predictions.

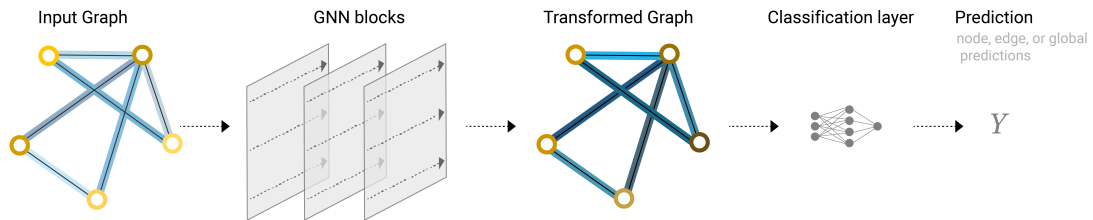


Figure 2.8: An end-to-end prediction task with a GNN model. [13]

2.2.4 Inductive and Transductive Learning

There exist two different paradigms for learning on graphs, transductive and inductive.

Inductive learning [18, 19] involves training a model on a specific graph's nodes and edges (Fig. 2.10). The model then predicts outcomes for new nodes and edges, even those not included in the initial graph. This approach allows the model to generalize its knowledge to unseen data, making it useful for handling novel instances.

Transductive learning [18, 19] maintains the same node and edge sets during both training and prediction (Fig. 2.9). During training, the algorithm

accesses all nodes and edges, including those not requiring predictions. However, to prevent the model from using information from the labels of test nodes during training, a label mask is applied, ensuring that the test node labels are not utilized for training purposes. Transductive learning doesn't extend to new graphs and is mainly employed for predicting within the existing dataset without a focus on generalization.

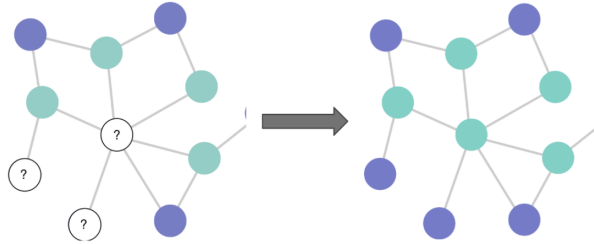


Figure 2.9: Node classification in transductive settings. At the training time, the learning algorithm has access to all the nodes and edges, including those nodes for which labels are to be predicted (denoted by question marks). [19]

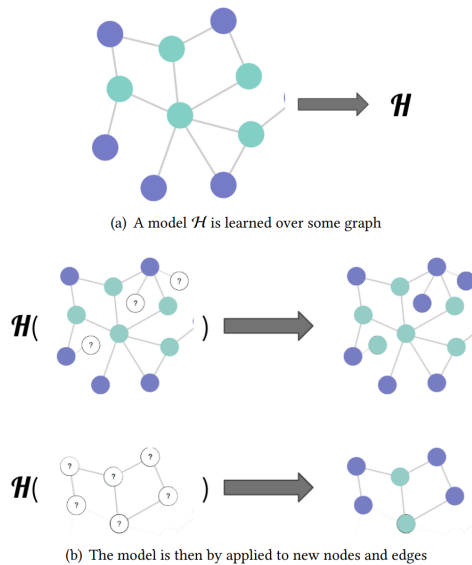


Figure 2.10: Node classification in inductive settings. Once learned, the model can be applied to new unseen nodes (denoted by question marks). There may or may not exist edges between such new nodes and the nodes used for training. [19]

2.2.5 Main GNN Architectures

This chapter provides a comprehensive overview of the diverse architectures of Graph Neural Networks (GNNs) utilized throughout the course of this work. These different architectures present various strategies for aggregating and exploiting data from neighboring nodes or the node itself. However, they all share the same fundamental stages: initialization, message passing, and updating node embeddings.

GCN: Graph Convolutional Networks

The Graph Convolutional Network [20] extends the concept of convolution from grid data to graph data. It achieves this by constructing graph convolutions through the stacking of multiple convolutional layers, with each layer followed by a point-wise non-linearity function, as shown in Figure 2.11. In GCN, the filter parameters are shared across all locations within the graph. This approach also incorporates symmetric-normalized aggregation, where information from neighboring nodes is combined in a symmetrically weighted manner, preserving graph structure, and self-loop updates, which involve considering a node’s own features in the aggregation process. In equation 2.1, is presented the update rule that encapsulates the essence of the GCN architecture. This equation captures the key transformation applied within the network, providing a mathematical representation of its fundamental operation.

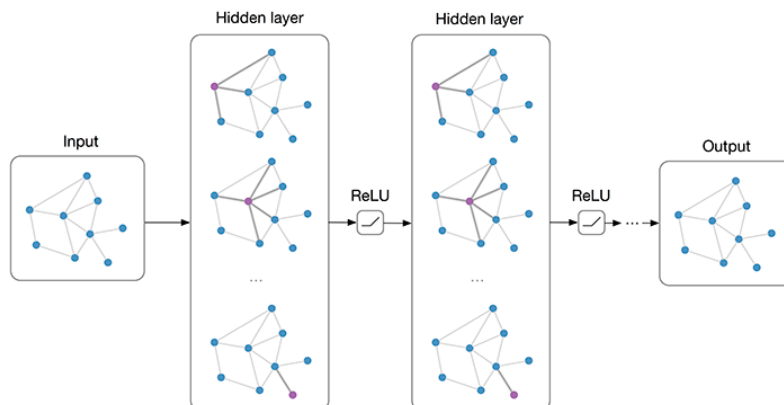


Figure 2.11: Multi-layer Graph Convolutional Network (GCN) [21]

$$h_v^{(k)} = \sigma \left(W^{(k)} \cdot \frac{\sum_{u \in N(v)} h_u^{(k-1)}}{|N(v)|} + B^{(k)} \cdot h_v^{(k-1)} \right) \quad (2.1)$$

- $h_v^{(k)}$: This represents the embedding or feature vector associated with node v at layer k and embodies the node’s acquired representation at the present layer.
- σ : This is the activation function, often a non-linear function like the sigmoid or ReLU, applied element-wise to the expression inside the parentheses. It introduces non-linearity into the model.
- $W^{(k)}$: This is the weight matrix for layer k . It is a learned parameter in the GCN and is used to transform the aggregated information from neighboring nodes.
- $\sum_{u \in N(v)} h_u^{(k-1)}$: This is the sum of the feature vectors of all neighboring nodes u of node v at the previous layer $k - 1$. It aggregates information from neighboring nodes.
- $|N(v)|$: This represents the number of neighbors that node v has in the graph. It normalizes the aggregated information by the degree of node v .
- $B^{(k)}$: This is a bias term specific to layer k . It’s another learned parameter in the GCN and is added to the weighted sum.
- $h_v^{(k-1)}$: This is the feature vector of node v at the previous layer $k - 1$. It represents the node’s learned representation from the previous layer.

GraphSAGE: Graph Sample and Aggregated

GraphSAGE [22] operates on the principles of Sampling and Aggregating, where it determines the sampling strategy for selecting a subset of neighbors and subsequently aggregates the embedding information from these neighbors to update its own embedding. In GraphSAGE, a subset of neighboring nodes is sampled at different depth layers, and then an aggregator function combines the embeddings of these neighbors, as illustrated in Figure 2.12. During each iteration, nodes collect information from their local neighbors. Different aggregator functions capture information from neighbors at varying depths or hop distances from the focal node. Consequently, nodes accumulate

progressively more information from increasingly distant parts of the graph. In equation 2.2 is shown the update rule of GraphSAGE.

$$h_v^{(k)} = \sigma \left(W^{(k)} \cdot \text{CONCAT}(h_v^{(k-1)}, \text{AGGR}_{u \in N(v)}(\{h_u^{(k-1)}\})) \right) \quad (2.2)$$

- $\text{CONCAT}(\mathbf{h}_v^{(k-1)}, \text{AGGR}_{u \in N(v)}(\{\mathbf{h}_u^{(k-1)}\}))$: This part of the equation involves concatenation.
- $\text{AGGR}_{u \in N(v)}(\{\mathbf{h}_u^{(k-1)}\})$: This is an aggregation operation that involves collecting feature vectors $\mathbf{h}_u^{(k-1)}$ from neighboring nodes u in the set $N(v)$ (the neighbors of node v). The aggregation operation can vary but typically involves some form of combining these neighboring feature vectors, for example, by taking their average or sum. So, AGGR is a more general form of the summation present in the GCN architecture.
- The other symbols have the same meaning as in Section 2.2.5

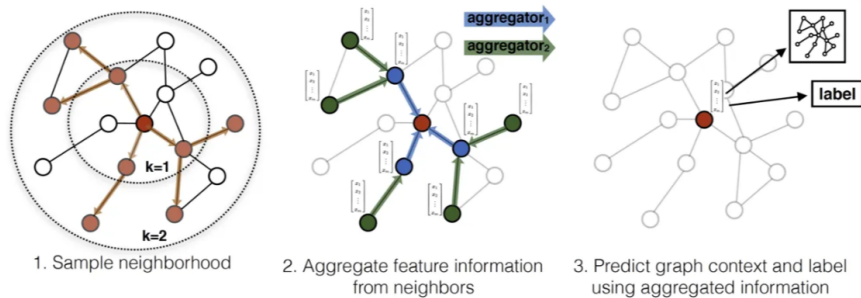


Figure 2.12: Visual illustration of the GraphSAGE Sample and Aggregate approach. [22]

GAT: Graph Attention Network

The Attention Mechanism is a technique used to emphasize or give varying degrees of importance to different parts of input data when making decisions. It is particularly useful for handling variable-sized inputs and focusing on the most relevant information. In the context of natural language processing and computer vision, attention mechanisms have been widely applied to tasks such as machine translation, text summarization, image captioning,

and more. In the context of Graph Attention Network (GAT) [23], this concept is extended to graph-structured data. The GAT employs masked self-attention layers to achieve its objectives. A self-attention mechanism, at its core, enables each node in the graph to compute its hidden representation by selectively attending to its neighboring nodes. This means that when a node updates its representation, it considers the relevance of information from neighboring nodes, assigning different weights to each neighbor based on their significance. The GAT takes this a step further by incorporating multi-head attention (as shown in figure 2.13), a technique that enhances the stability of the attention mechanism during the learning process. In the GAT, attention operations within a particular layer are performed multiple times, with each operation having its own set of parameters. The outputs from these attention operations are then combined, either by concatenation or averaging, to produce a refined representation for each node. This approach allows GAT to implicitly assign varying levels of importance to different nodes within a neighborhood, ensuring that the model captures the nuanced relationships and dependencies in the graph data. In 2.3 is shown the equation of the update rule of GAT.

$$h_v^{(k)} = \sigma \left(W^{(k)} \cdot \left[\sum_{u \in N(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)} \right] \right) \quad (2.3)$$

- $\alpha_{vu}^{(k-1)}$: This represents the attention weight associated with node u when node v is at layer $k - 1$. It signifies how much attention or importance node v gives to node u when aggregating information. $\alpha_{vv}^{(k-1)}$ represents the self-attention weight for node v at layer $k - 1$. It signifies how much attention node v gives to its own previous representation when aggregating information.
- The other symbols have the same meaning as in Section 2.2.5

Where the attention weights $\alpha^{(k)}$ are generated by an attention mechanism $A^{(k)}$, normalized such that the sum over all neighbors of each node v is 1, as shown in equation 2.4:

$$\alpha_{vu}^{(k)} = \frac{A^{(k)}(h_u^{(k)}, h_v^{(k)})}{\sum_{w \in N(v)} A^{(k)}(h_w^{(k)}, h_v^{(k)})} \quad (2.4)$$

- $A^{(k)}(h_u^{(k)}, h_v^{(k)})$: This is the output of an attention mechanism $A^{(k)}$, a mathematical function, that computes the attention weight between the

feature vectors $h_u^{(k)}$ and $h_v^{(k)}$ of nodes u and v at layer k . This mechanism learns to assign different attention scores based on the similarity or relevance of these feature vectors.

- The other symbols have the same meaning as in Section 2.2.5

The denominator of the equation $\sum_{w \in N(v)} A^{(k)}(h_w^{(k)}, h_v^{(k)})$ is the sum of attention scores computed by the attention mechanism $A^{(k)}$ between the feature vector $h_v^{(k)}$ of node v at layer k and the feature vectors $h_w^{(k)}$ of all neighboring nodes w in the neighborhood ($N(v)$) of node v at the same layer k . It represents the total attention given by node v to its neighbors.

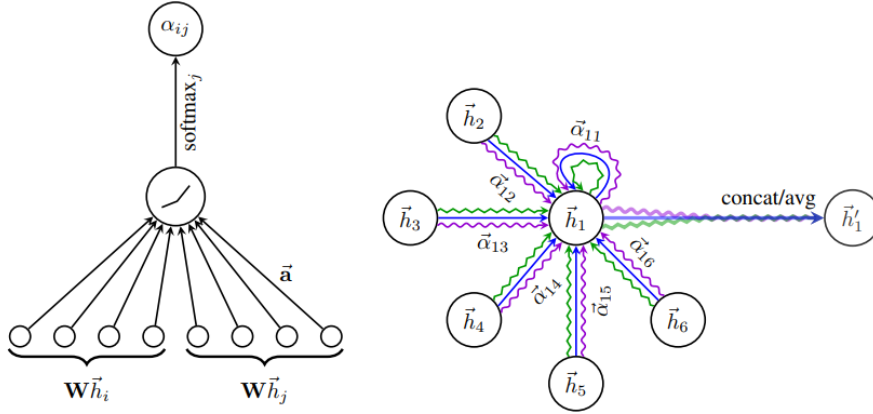


Figure 2.13: **Left:** The attention mechanism $(W\vec{h}_i, W\vec{h}_j)$ employed by our model, parameterized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a Leaky ReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain \vec{h}'_1 .

RGCN: Relational Graph Convolutional Network

RGCN [24] is an extension of GCN, that distinguishes itself by generalizing from the focus on only neighborhood operations in the original work [20] to accommodating relational data between neighbors. For instance, consider a social network where nodes represent users, and edges represent different

types of relationships, such as "friends," "family," or "colleagues." In the context of GCN, the network might primarily focus on learning from the immediate connections or neighbors of a user to make predictions. However, RGCN takes it a step further by considering the specific types of relationships (e.g., "friendship" vs. "colleague") between the users. This extension allows RGCN to capture more nuanced and context-aware information. For example, when making recommendations, RGCN can better understand that a user might have different preferences for recommendations from friends compared to recommendations from colleagues, leading to more personalized and accurate suggestions. Expanding upon the general message-passing framework of GCN, RGCN modifies the equation to take in account multi-relational edges, as illustrated in Equation 2.5. Figure 2.14 provides an illustrative example of R-GCN in action.

$$h_v^{(k)} = \sigma \left(\sum_{r \in R} \sum_{j \in N_v^{(r)}} \frac{1}{c_{v,r}} W_r^{(k-1)} h_j^{(k-1)} + W_0^{(k-1)} h_v^{(k-1)} \right) \quad (2.5)$$

- $\sum_{r \in R}$: This is a summation over all relation types (r) in the set R . It means we're considering each type of relation in the summation.
- $\sum_{j \in N_v^{(r)}}$: This is a summation over all nodes j that are in the neighborhood ($N_v^{(r)}$) of node v with respect to relation r . It means we're considering each neighboring node with a specific relation r .
- $\frac{1}{c_{v,r}}$: This represents a weight factor for the relation r specific to node v . It normalizes the contribution of each neighboring node based on the relation type and node v .
- $W_r^{(k-1)}$: This is the weight matrix specific to relation r at layer $k - 1$. It is a learned parameter in the model. It is used to linearly transform the feature vectors of neighboring nodes with relation r .
- $W_0^{(k-1)}$: This is an additional weight matrix at layer $k - 1$ that is not relation-specific. It is also a learned parameter in the model and is used to linearly transform the feature vector of node v from the previous layer.
- The other symbols have the same meaning as in Section 2.2.5

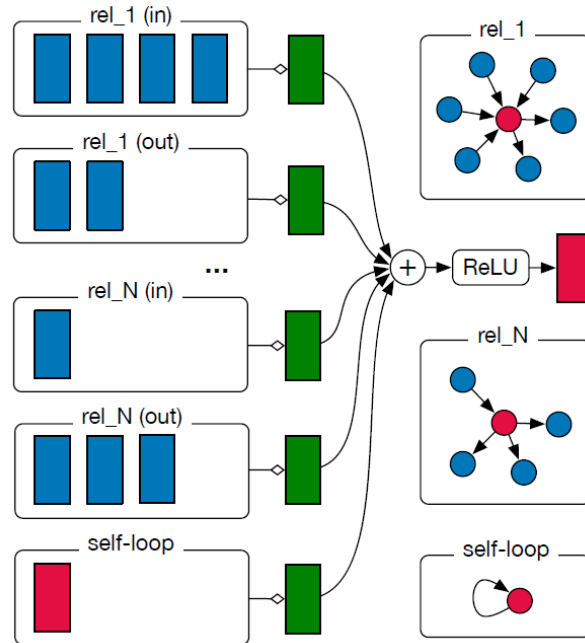


Figure 2.14: Illustration depicting the operational mechanism of RGCN. [24]

2.3 A Graph representation for Circuits

A fundamental aspect of the work involved transforming the data, initially in a digital circuit format, into a usable form. As demonstrated in [4, 25], an ideal representation of a circuit in the field of Graph Neural Networks (GNNs) is that of a bipartite undirected graph. In this type of graph, nodes are divided into two distinct types: one containing the devices (transistors, resistors, capacitors, diodes), and the other containing the nets, which represent the connections between these devices.

In [4], which serves as a key reference for this thesis, the graph representation is structured as follows: each edge connected to a transistor is assigned a three-bit label, l_g , l_d and l_s . Here, l_g takes a value of 1 if the edge from the transistor vertex connects to the net vertex through its gate, and 0 otherwise. Similarly, l_s and l_d take a value of 1 if the transistor connects to the net through its source, drain, or bulk, respectively, and 0 otherwise. This labeling scheme, shown in Figures 2.15 and 2.16, provides a characterization of the relationships between transistors and nets, enhancing the depth of information available for the GNN. In Chapter 4.1, this representation is expanded to incorporate additional information.

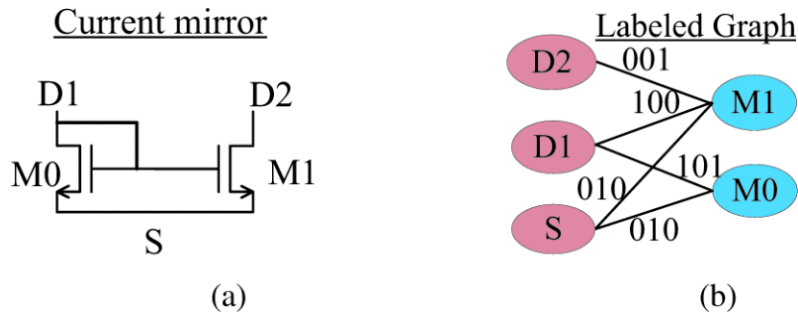


Figure 2.15: (a) An NMOS current mirror primitive, CM-N(2), with two transistors. (b) Its representation as a bipartite graph. [4]

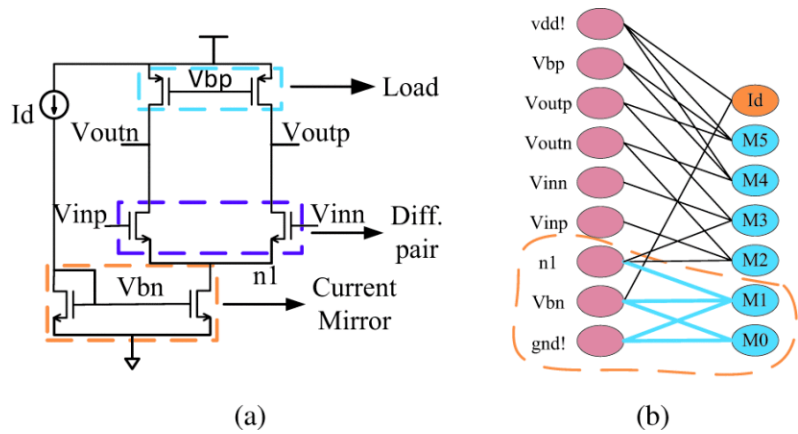


Figure 2.16: (a) A differential OTA (for simplicity, body connections are not shown). (b) Its bipartite graph, showing the subgraph that can be recognized as a current mirror (for clarity, edge labels are not shown). [4]

2.4 VF2 Algorithm

The VF2 algorithm is a deterministic algorithm used for graph matching. It is utilized in the context of the AMBEATion project (Chapter 2.5) for topology recognition, where it aims to find an isomorphism mapping between two graphs (as shown in Figure 2.17): G_t (the target graph) and G_p (the pattern graph or primitive template). The "primitive template" or "pattern graph" refers to the graph representation of the netlist of a specific functional topology or pattern that we want to recognize within the target circuit. On the other hand, the "target graph" refers to the graph representation of the netlist of the circuit for which we want to perform topology recognition. In both cases, the graph represents the structure and connections of the elements

(devices) and nets (connections between elements) within the circuit.

To create the graph representations of the target netlist and the primitive templates, we start with the conversion of the netlists from CDL, a standard netlist format used in the AMS design industry, to JSON format. This conversion process preserves the necessary information required for topology recognition. The target netlist, as well as the template netlists, are represented as graphs. Specifically, an undirected bipartite graph is used (see Chapter 2.3). In these graphs, the vertices are divided into two sets: one set represents the elements or devices (such as resistors, transistors, etc.), and the other set represents the nets or connections (such as VSS, VDD, etc.). The edges in the graph represent the connections between individual terminals of the elements and the nets.

The algorithm works by creating states that represent potential mappings between vertices in the two graphs. Each state identifies candidate pairs of vertices that could be included in the mapping. The algorithm then determines whether to add these candidate pairs to the mapping and move to the next state. The mapping process continues recursively until a complete and valid mapping is obtained. In the case of topology recognition, this means achieving a matching result that accurately recognizes the desired functional topologies.

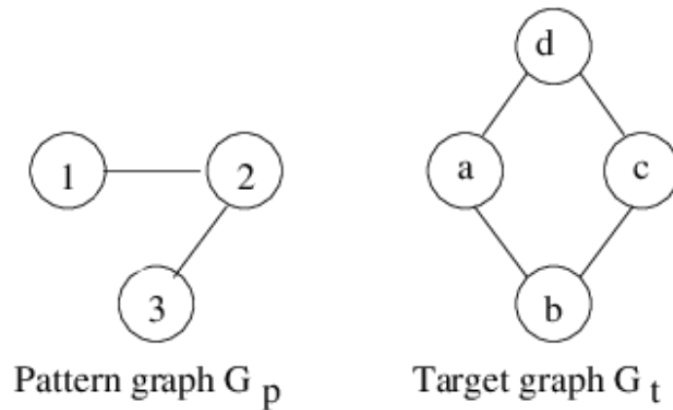


Figure 2.17: An instance of subgraph matching. The graph on the left (pattern graph), is present four times within the graph on the right (target graph)

At the end, a post-processing step is applied with the goal of filtering and refining the matched results obtained from the VF2 algorithm. The main tasks performed are the following:

- Support CM/CascLV with multiple branches: In the context of recognizing CM or CascLV with multiple branches, the algorithm is designed to identify individual instances of CM/CascLV within the target netlist. If multiple candidates for CM/CascLV are recognized within a single CM/CascLV with multiple branches, the algorithm merges them into a single recognized instance. This ensures that the recognition accurately captures the presence of CM/CascLV structures with multiple branches in the circuit.
- Support length/width constraints of instances: In the recognition of certain circuit elements (such as DP), it is important to consider the length and width constraints of the instances. The algorithm takes into account specified length and width ranges for the DP candidates. If any DP candidates fall outside of these specified ranges, they are removed from consideration. Additionally, to ensure consistency, the algorithm also verifies that the length of the devices within the recognized instances of DP remains the same.
- Support recognition of parallel DP pairs: In some cases, differential pairs may appear in parallel structures within the circuit. The recognition algorithm is designed to recognize these parallel DP and merge them into a single recognized entity. By identifying parallel DP candidates and grouping them together, the algorithm accurately captures the presence of parallel DP in the circuit topology.

The issue of scalability in the current graph matching-algorithm used is the main concern. As the size and complexity of the target netlist graph grow, the computational resources and the search space for matching templates increase exponentially, which is characteristic of NP problem complexity, thereby impacting the algorithm’s scalability and performance.

2.5 The AMBEATion Project

The EU-funded AMBEATion project [26] adopts a holistic approach, leveraging artificial intelligence techniques to enhance the design automation of AMS IC. These efforts aim to boost designer efficiency in mixed-signal physical designs, reducing the time to market actually needed. During this thesis, the primary objective was to enhance topology recognition for analog circuits, considering that analog design processes are notably less automated compared to their digital counterparts.

In this section is provide an overview of the essential components of the AMS design flow developed within the AMBEATion Project and their respective functionalities. This serves to explain the core elements that constitute the project’s framework, offering a comprehensive understanding of its inner workings. Namely, the AMBEATion flow is composed of the following modules, each realized as a set of *Python* scripts:

- **cdl2json:** Its primary role is to process the input netlist. CDL2JSON transforms the original netlist into a JSON file format, which is suitable for the subsequent data manipulation and analysis tasks.
- **Topology Recognition:** The objective of this step is to identify functional topologies within the schematics, including current mirrors and differential pairs. This identification process is essential for accurately specifying constraints in the layout placement phase. It also represents the central focus of this thesis.
- **Digital Area Estimation:** The AMBEATion Project incorporates the Digital Area Estimation component into its workflow to assess whether a digital block within an analog-mixed-signal (AMS) design can be accommodated effectively within the designated layout area. This component is activated subsequent to the identification of digital topologies in the workflow through Topology Recognition. The resulting output from the Digital Area Estimation component is intended for utilization by the top-level group placer (in forthcoming scriptware releases) to inform placement decisions for the digital logic elements.
- **PCell Analysis:** This script has the capability to reconstruct the polygons comprising the device’s layers, extracting the correspondence between length and width in the schematic and layout representations. Additionally, it can incorporate technology-specific constraints, such as spacing requirements, into the design.
- **Device Placers:** An AMS group auto-placer, which leverages constraints derived from topology recognition, individual device layouts generated through PCell analysis, and additional relative positioning of the devices within each group facilitated by a Level 1 placer. These inputs, along with digital placeability information, collectively contribute to the generation of the ultimate Level 2 layout for the circuit.

- **Flatten:** Flattening refers to the transformation of hierarchical modules or subcircuits into a single-level representation, simplifying tasks that necessitate a flat circuit structure.
- **Plotter:** Utilized for generating visual outputs, such as graphical representations of circuit layouts.

A more comprehensive examination of the current topology recognition tool utilized in the project's present state has been provided in Section 2.4.

Chapter 3

Related Work

In the field of chip design, there has been a gradual integration of multiple software tools aimed at efficiently and reliably performing tasks such as synthesis, simulation, testing, and verification of electronic designs. This set of tools is collectively known as Electronic Design Automation (EDA), streamlining the step-by-step and time-consuming chip design process depicted in Figure 3.1. At the end of the EDA flow the quality of the design, with regard to power, performance, and area (PPA), can be assessed. Often, adjustments are required in intermediate steps, leading to multiple iterations of the design process.

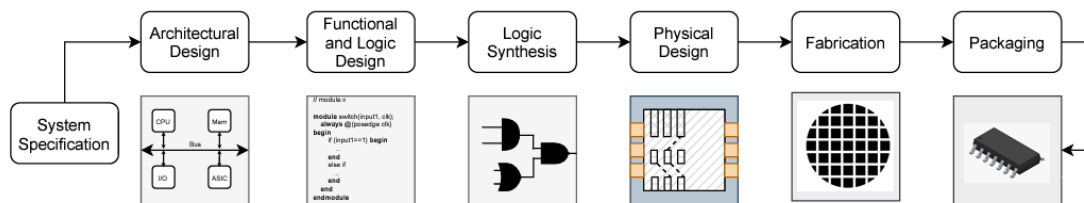


Figure 3.1: Chip Design Flow. [13]

In recent years, the ever-increasing complexity driven by Moore’s Law has necessitated greater efforts in designing and validating a broader range of chips, given that chip capacity has been doubling approximately every two years. EDA tools have adapted to address these new challenges by offering automated solutions tailored for different design constraints. EDA tools frequently encounter NP-complete problems, which can be more efficiently tackled using machine learning (ML) techniques. Consequently, ML has been seamlessly integrated into EDA, particularly in tasks such as logic synthesis,

layout constraint extraction, placement, routing, testing, and verification.

In the reference [27], four primary areas of ML application within EDA have been identified. First, ML is employed to predict optimal parameter configurations for traditional methods. Second, ML models can learn the characteristics of models and their performance to forecast the behavior of unseen designs without the resource-intensive synthesis step. Additionally, ML can aid in exploring the design space while optimizing PPA. Related to this work, in [4] researchers have applied GNNs to the preliminary step of analog topology recognition, which is followed by placement and routing. Similarly, Graph based Reinforcement Learning (RL) is employed in [28] to explore the design space, learn policies, and execute transformations to achieve optimal floor-planing.

One crucial factor facilitating the use of ML in EDA is the copious amount of data generated by EDA tools during the design process. To leverage ML effectively with this data, it necessitates preprocessing and labeling. Existing solutions often represent this data in a 2D Euclidean space, enabling the utilization of ML methods like convolutional neural networks (CNNs). However, the recent development of GNN has gained considerable attention due to their efficacy in handling data inherently structured as graphs. In the context of EDA, circuits, intermediate RTL, netlists, and layouts are most naturally represented as graphs. Over the past years, several studies have recognized this opportunity and have incorporated GNNs to address various EDA challenges [29]. This section provides a comprehensive review of recent studies employing for topology recognition.

3.1 Comparative Analysis of Topology Recognition Techniques

In MAGICAL [30], topology recognition diverges from ML-based approaches seen in this work. Instead, MAGICAL maintains a reliance on a graph abstraction of the circuit, while GANA [4], ALIGN [6], and the presented work incorporate ML techniques. The topology recognition in MAGICAL is grounded in graph analysis of the circuit netlist. The process begins with parsing the netlist file and abstracting the circuit into a graph representation. Transistor pairs that form certain structural patterns are detected as seed symmetric device pairs that serve as starting points for topology recognition. They are selected from a pattern library based on interconnected transistor pins' sources. Instead of using computationally expensive graph isomorphism

algorithms, the method checks connection relationships and device attributes between pairs of devices for matching. For example, when recognizing differential pairs, it iterates through pairs of transistors connected to the same net via source pins and verifies gate pins connect to different nets while matching device attributes. Graph traversal from these seed pairs expands recognized constraints and reduces ambiguity. Symmetric transistor patterns visited during traversal form symmetric groups, sharing a common symmetry axis during placement. Nets connecting symmetric devices are recognized as symmetric nets. Additional symmetry constraints, including self-symmetric devices and transistor pairs in bias circuits, are detected. To ensure feasibility during placement, each device is allowed at most one symmetry constraint.

Like MAGICAL, ALIGN also represents the netlist as a graph and subsequently identifies features within the graph across various hierarchical levels. ALIGN does not solely depend on netlist hierarchy; instead, it automatically identifies and annotates hierarchies, even when hierarchical blocks are absent. While graph-based methods excel at recognizing fixed structures through subgraph isomorphism operations, analog design presents a challenge due to the multitude of variations in implementing circuit functionalities. For instance, differential pairs can be implemented in various ways, including purely transistor-level structures and configurations that combine transistors with amplifier building blocks. At higher design hierarchy levels, operational amplifiers can be constructed in different ways, with sub-blocks containing transistor groups recognized as differential pairs, current mirrors, differential loads, or OTA blocks. Recognizing these structures by enumerating graph patterns is manageable at lower design hierarchy levels, but it becomes impractical at higher levels due to the sheer number of permutations. Expert human designers rely on their experience to intuitively identify these patterns when examining schematics. In contrast, ALIGN addresses this challenge by leveraging ML methods that recognize standard structures based on their distinctive characteristics.

Both GANA and the approach presented in this work aim to replicate human flexibility in recognizing these structures using Graph Neural Networks through different approaches. GANA’s approach consists of four key phases:

- **Netlist Flattening:** In the preprocessing stage, the input netlist is flattened to bypass designer-specific hierarchies. This approach ensures independence from individual designer preferences, allowing for consistent integration of design constraints into recognized blocks. For example, bias networks and operational transconductance amplifiers (OTAs) may

logically belong to different hierarchies in the netlist, but they are often combined in layout optimization. Preprocessing also identifies netlist features that impact performance but not functionality.

- **GCN-based Recognition:** A graph representation of the flattened netlist is created, and sub-blocks are identified using a Graph Convolutional Network (GCN)-based approach. This stage annotates netlist nodes as part of specific sub-blocks, even when vertices may belong to multiple sub-blocks due to design variations.
- **Primitive Annotation:** Within each sub-block, lower-level primitives are recursively identified. At the lowest level, primitives are detected using an exact graph isomorphism approach, as their element-level structures remain invariant across circuits.
- **Post-processing:** After primitive annotation, post-processing is employed to determine which primitives are integral to a specific unit and which are auxiliary. This step ensures recognition accuracy, even when dealing with variations in sub-blocks.

While GANA primarily focuses on utilizing GNNs to recognize specific structures like Operational Amplifiers (Op-Amps), the approach outlined in this work distinguishes itself by addressing the recognition of a single topologies within the netlist graph, like CMs.

Chapter 4

Materials and Methods

This chapter aims to provide an overview of the main contribution of this work. The first subchapter (4.1) will introduce an extension in the representation of circuits as graphs presented in Chapter 2.3. The second subchapter (4.2) will present the pre-processing technique used to improve the accuracy of the results. The third subchapter (4.3) will delve into the training phase: how imbalanced classes are handled and how the data are prepared for different architectures. In the last subchapter (4.4) is explained how the VF2 algorithm is used in conjunction with the developed GNN model, in order to produce more robust and accurate result.

4.1 Extend the Graph Representation for Circuits with Bulk Connectivity Information

As mentioned in the previous chapter 2.3, the procedure of preparing circuits to be used as input for a GNN needs the conversion of those to a graph format that is appropriate for a GNN. In previous researches, such as [4], a bipartite structure was used. The graph was partitioned into two sets of nodes: one set representing the devices, such as resistors and transistors, and another set representing the connections between those devices (called nets).

Within this bipartite structure, each device has its features. Also the edges in the graph are associated with their specific features. Previously, three edge features, l_g , l_d , and l_s , were used to identify whether the connection originated from the gate, drain, or source of a transistor. However, in this work, the number of edge features was extended by introducing two new features called l_b and l_{dd} . These additional features served the purpose of indicating whether

a connection was linked to the bulk (or body) of a transistor or if it simply represented a connection between two devices that are not transistors, for example a net representing the connection between a resistor and diode will be labeled with l_{dd} .

Adding the bulk to the edge features is important because the gate potential is measured relative to the substrate (or bulk) potential, and the channel conductive is formed in the substrate material. Therefore the transistor behavior strongly depends on the substrate body potential, which in turn depends on the bulk connection, making this feature important to optimally describe the transistor behaviour. Figure 4.1 shows a MOSFET transistor with its four terminals (source, gate, drain and bulk).

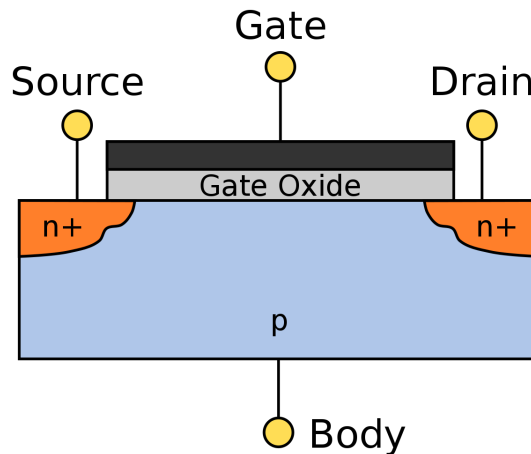


Figure 4.1: Cross-sectional view of a field-effect transistor, showing source, gate, drain and body (or bulk) terminals [31]

In addition to the discussed aspects, it is important to note that the pre-processing technique, discussed in chapter 4.2, involves removing all non-transistor devices. Consequently, in this context, only the edge features related to transistor terminals become relevant, as devices such as resistors and capacitors have been filtered from the graph.

4.2 Pre-Processing Techniques

In the field of Machine Learning (ML), pre-processing refers to the set of actions taken to prepare and clean raw data before feeding them into a model

for training [32]. This often involves tasks such as removing irrelevant or duplicate information and scaling or normalizing features to make them more comparable. The goal is to enhance the quality of data, improving the performance and/or accuracy of the model.

In the context of this work, a dataset containing various devices was provided by STMicroelectronics. The aim of this work was to recognize the different topologies present within the circuit. These topologies (discussed in 2.1.1) are essentially different arrangements of transistors, making other devices (e.g. resistors) not useful for the purpose of identifying these layouts. Hence, a pre-processing step was introduced to preserve the circuit layout while at the same time removing all unnecessary devices for the recognition of topologies.

This process can be described as follows: In the first phase, all devices that are not transistors are identified. In the second phase, these non-transistor devices are removed. In the third phase, to ensure that the topology remains unchanged after the removal of these devices, the nets connected to these devices are merged. This results in the original circuit as a simple short-circuit of all the terminations of the removed devices, as shown in Figure 4.2. By employing this technique, the model’s accuracy was increased, as will be discussed in chapter 5. Additionally, this technique led to performance improvements, as removing devices from the circuit reduced the size of the graph provided as input to the GNN. This reduction in size decreased the computational cost required both during the training phase and during inference.

4.3 Training

One of the most important phases of the work is the training of the GNN model. In this section, four main points are addressed. The first subsection introduces the frameworks, computational resources, and all the tools utilized during the training process. The second section delves into a more detailed analysis of the employed end-to-end architectures. The third subsection explains which features were used and which features were available for the nodes. The fourth subsection explain how data were managed in order to mitigate the impacts of significant class imbalances.

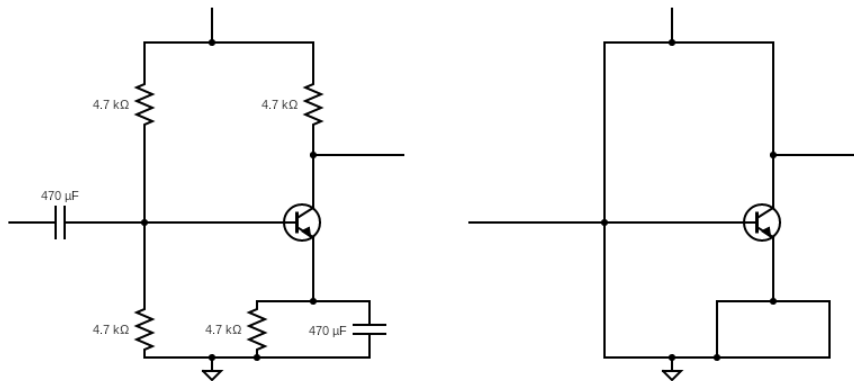


Figure 4.2: **Left:** The original circuit containing resistors and capacitors, detected as a non-transistor device. **Right:** After applying the pre-processing technique, the resistor’s terminations are short-circuited.

4.3.1 Resources

Throughout this work, several frameworks for the development of neural networks have been employed. First, a significant contribution has come from *PyTorch* [33]. *PyTorch* is a widely used open-source deep learning library that support the creation and training of neural networks. Building upon the foundation of *PyTorch*, another framework has been employed: *PyTorchGeometric* [34]. This framework is an extension library specifically developed for handling graph-structured data, enabling deep learning on graphs.

For graph analysis and manipulation, the framework *NetworkX* [35] has primarily been utilized. *NetworkX* provides tools for the creation, manipulation, and study of complex graphs.

On the computational front, the primary resource for model training has been a NVIDIA T4 GPU with 16GB of VRAM, which was made available through Google Colab. This GPU helped to accelerate the training process by significantly reducing the time required for model training.

4.3.2 Architectures Exploration

In this work, as discussed in chapter 2.2.5, various GNNs architectures have been employed. These architectures can be categorized into two groups: one group includes architectures capable of utilizing edge feature, learning by

incorporating them. The representative architecture in this category is the RGCN. On the other hand, the remaining three architectures, namely GCN, GraphSAGE, and GAT, do not leverage these features, they update neural networks weights only using relationships between nodes and nodes features. They are unable to learn incorporating the edge features of the connection between various devices.

GNN layers are typically not employed in isolation (refer to Figure 2.8). Instead, they are incorporated within a larger architecture. As shown in Figure 4.3, where a more detailed end-to-end architecture is illustrated, GNNs are commonly stacked to enable nodes to gather information from more distant nodes. For instance, when two GNNs are stacked, they can update node features within a range of 2 hops from a given node. For classification tasks, it is essential to employ a MLP layer at the output of the network, as GNNs only update node features. The architecture of Figure 4.3 can be divided into three components: the pre-processing layer, the message passing layer, and the post-processing layer.

The pre-processing layer is designed to handle certain data transformations or feature engineering tasks before the information is passed to the message passing layer. However, for this work, the pre-processing layer was not used.

Moving on, there is a set of GNN layers, referred to as message passing layers. As described in chapter 2.2.3, GNNs are stacked one after another. Increasing the number of GNN layers enhances the network's ability to extract information from nodes that are progressively farther away. This is because, at each layer, nodes incorporate information from their neighbors, enabling the extraction of increasingly global information. However, it may appear that adding more layers can improve results, but a common issue in GNNs known as "over-smoothing" sets a limit on the number of layers that can be used [36]. Over-smoothing in GNNs happen when increasing the number of GNN layers, the learned node representations become too similar from one another. The information from neighboring nodes is repeatedly aggregated and smoothed out to the point where nodes in the graph lose their distinct characteristics. The optimal number of layers typically derives from careful considerations and testing. In this case, it was observed that the topologies are primarily local and not global. Therefore, it is not interesting to use many layers, as this would introduce additional unnecessary information into the model, leading to poorer performance. After careful consideration, it was determined that the optimal number of message passing layers is two. This aligns with the nature of the problem, as the nodes in within the same topology are typically 2 hops apart: one hop to reach the net of the bipartite

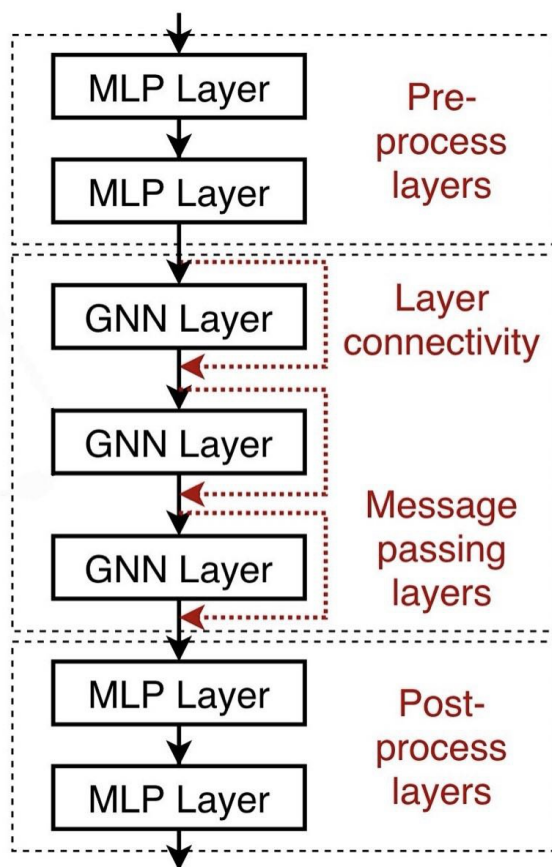


Figure 4.3: Comprehensive Graph Neural Network Architectures which include a pre-processing layer, a message passing layer and a post-processing layer

structure and another hop to reach the other device within the topology.

Finally, the last layer is the post-processing layer, which is essential because after the GNN block, the features of various nodes are simply updated. These features must be classified, and a MLP layer is used for that purpose. The MLP takes the individual node features as input and provides the classifications for all nodes. It is important to note that, as presented in Chapter 4.4, another post-processing layer, that is not based on neural networks, has been added after this final layer.

4.3.3 Features Selection

The main goal of this work, as already presented, is to recognize the various topologies within AMS ICs. The ability to recognize these structures

is largely linked to how the various devices are interconnected. Recognizing these structures through information related to individual devices (node features) and to the connection between devices (edge features). In Figure 4.4, are shown the available features for a PFET transistor which include: instance, device type, w (width), l (length), nfing (number of fingers), parallel, number, and nets.

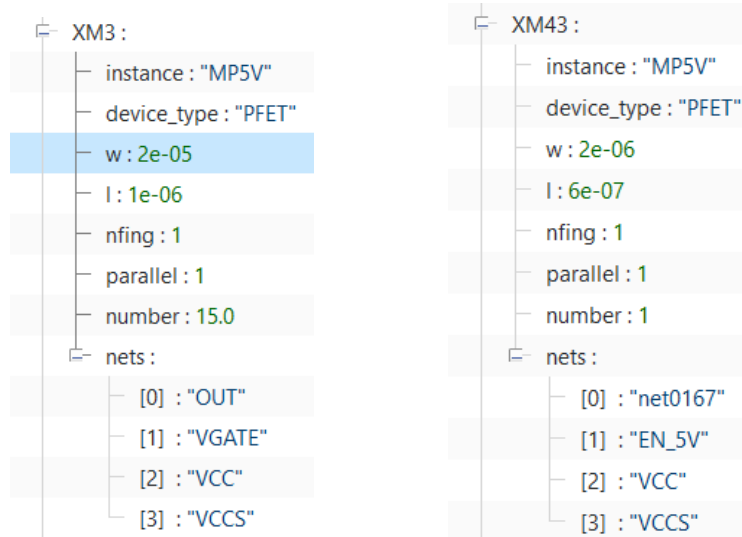


Figure 4.4: Features from the dataset provided by STMicroelectronics

The instance refers to a specific realization of an object, and it is not useful for making correct predictions as it comes from how devices are generated inside the design tool. On the other hand, the device type, such as PFET, NFET, Transistor, etc., is extremely important in recognizing a topology. Width (w) and length (l) represent the physical dimensions of the device. Although these features were introduced in the model, they did not improve its capabilities in recognizing different topologies. In addition, "nfing" pertains to the parameter defining the number of fingers of a MOS (Metal Oxide Semiconductor) device, influencing its behavior and characteristics. "Parallel" describes the configuration where devices share common electrical nodes (such as drain, gate, source and bulk) in a circuit to collectively increase current capacity. "Number" is a term used to specify the quantity of parallel devices in a schematic, simplifying their definition without requiring explicit graphical representations of connections. In the beginning, also these features were introduced in the model, but they were subsequently removed as they did not contribute to the model's capabilities. The nets feature simply represents all the nets to which the device is connected. Among these

various features, the only one explicitly utilized, as node feature, was the device type, but also the nets were used for the construction of the graph. It is important to note that despite the limited number of features used, much of the information derives from the graph itself. GNN models, in addition to the classical features, also use information related to neighboring devices, allowing for the extraction of additional information.

Another important set of features that has been used is related to edges. Looking at Figure 4.4, it is possible to see that the nets are represented as a list. In this list, the order is extremely important because the nets are connected in this order to the following terminations: gate, drain, source, and bulk. This way, it is possible to uniquely determine which net is connected to which termination. These information are then translated into features as explained in chapter 4.1. It is important to note that these features cannot be used with all different architectures. In fact, among the architectures presented in chapter 2.2.5, only the RGCN implements learning using edge features.

4.3.4 Addressing Class Imbalance

Class imbalance refers to a situation in a classification problem where the distribution of data among different classes is significantly different [37]. One or more classes have large or small number of instances compared to other classes. This problem poses challenges for deep learning algorithms because they tend to be biased towards the majority class. The model may achieve high accuracy by simply predicting the majority class most of the time, while performing poorly on the minority class(es) of interest. This is especially problematic when the minority class contains important or rare instances that need to be identified correctly.

Table 4.1: Class Distribution of the Dataset

Class	Count	Percentage (%)
net	366,736	48.96
others	394,525	50.6
CM	1,759	0.23
DP	384	0.05
DPx	20	0.00
CascLV	1,242	0.16

The issue described exists in the dataset used for this work. As can be seen in Table 4.1, which lists the different percentages of all the topologies presented in chapter 2.1.1, it is evident that all four of them are present in less than one percent of the total number of nodes. The total number of nodes is composed mostly of "others" and "nets," where they respectively represent devices that are not part of a topology and the physical connection between devices. To address this problem in this study, the undersampling technique was employed. Undersampling involves reducing the number of instances in the majority class to achieve a more balanced representation of classes in order to avoid producing a biased model.

One notable factor is the significant presence of nets within the graph. Nets are necessary as they allow representing the circuit in the form of a graph while preserving its topological properties. This means that a particular structure identifiable in the original circuit can also be identified in its bipartite graph form. However, the large number of nets has a negative effect as it further reduces the actual number of topologies present in the circuit compared to the total number of nodes. To overcome this issue during the training phase, a typical technique used in anomaly detection called one-class classification was employed.

One-class classification [38] is a machine learning technique that focuses on modeling and identifying instances of a single class, known as the target class or positive class, in this work, the target class or positive class is represented by a specific topology. This work mainly focused mainly on CMs. Unlike traditional classification problems where multiple classes are considered, one-class classification aims to distinguish instances of interest from the outliers. The main objective of one-class classification is to build a model that captures the characteristics and patterns of the target class, allowing it to differentiate between instances belonging to that class and those that do not.

4.4 VF2 as Post-Processing Technique

An important solution for the purposes of this study is the introduction of a post-processing technique following the evaluation by the GNN. This phase is crucial since it necessitates obtaining the most accurate final result possible to potentially replace the current solutions, which, although time-consuming, remain very accurate, being based on rule-based algorithms. In this work, the choice was made to utilize the algorithm presented in chapter 2.4. This algorithm employs a graph matching technique, and its primary drawback

is its computational and time demands. In this study, this graph matching algorithm is incorporated after the GNN's results. This is done to attempt to eliminate potential false positives. Indeed, the algorithm is executed solely on nodes classified as positive. As a result, the number of nodes on which the algorithm performs graph matching is significantly reduced, greatly reducing the required processing time.

Chapter 5

Results

The following chapter presents the results obtained during this work. In the first section (5.1), the metrics that were used to evaluate the obtained results are presented. In the second section (5.2), the results of the VF2 algorithm presented in chapter 2.4 are reported, allowing for a comparison of the current time and computational resources required for topology recognition within a circuit. Subsequently, in the third section (5.3), the results of the various architectures (GCN, GraphSAGE, GAT and RGCN) are presented. This section is divided into two further subsections: one addressing the results without the pre-processing technique (presented in chapter 4.2), and the second discussing the results with the addition of this method. In the fourth section (5.4), the results of the architecture that yielded the best outcomes, in combination with the VF2 algorithm, are presented. These results are then ultimately compared with the results obtained using only the VF2 algorithm.

5.1 Metrics

To evaluate and compare the different results, various metrics were taken into consideration. Specifically, the following metrics were utilized: a confusion matrix that includes the concepts of false positives, false negatives, true positives, and true negatives; accuracy, precision, and recall.

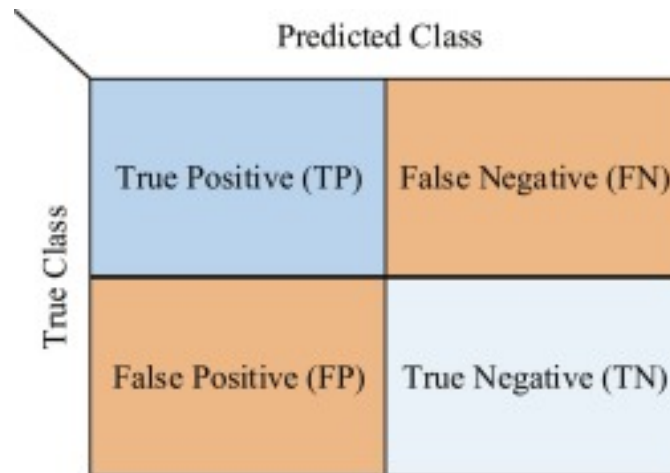
5.1.1 Confusion Matrix

The confusion matrix is a representation used to summarize the performance of a classification task. Relying on accuracy, which is generally used, is not sufficient to provide a comprehensive description of the obtained results,

especially in the case of a significant class imbalance, as is the case in this work. This is because accuracy tends to be high when the algorithm classifies all nodes as the majority class. The confusion matrix gives insight into how the model is learning and what the primary types of errors might be. To define the confusion matrix, the following terms need to be presented:

- **True Positives (TP):** These are the cases where the model correctly predicts the positive class.
- **False Positives (FP):** These are the cases where the model incorrectly predicts the positive class when it's the negative class.
- **True Negatives (TN):** These are the cases where the model correctly predicts the negative class.
- **False Negatives (FN):** These are the cases where the model incorrectly predicts the negative class when it's a positive class.

In Figure 5.1, an example of a confusion matrix is shown.



The figure shows a 2x2 confusion matrix. The vertical axis is labeled 'True Class' and the horizontal axis is labeled 'Predicted Class'. The four quadrants are: Top-Left (blue) is True Positive (TP), Top-Right (orange) is False Negative (FN), Bottom-Left (orange) is False Positive (FP), and Bottom-Right (light blue) is True Negative (TN).

	Predicted Class	
True Class	True Positive (TP)	False Negative (FN)
	False Positive (FP)	True Negative (TN)

Figure 5.1: Confusion matrix for binary classification [39]

5.1.2 Accuracy, Precision and Recall

Accuracy measures how many of the total predictions made by a classification model are correct. It calculates the ratio of correct predictions (both true positives and true negatives) to the total number of predictions. It's a measure of overall correctness. In 5.1 is shown the mathematical formulation.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (5.1)$$

Precision focuses on the accuracy of positive predictions. It tells us the ratio of correctly predicted positive instances to the total number of positive predictions. Precision is especially important when the cost of false positives is high. The mathematical formulation is shown in 5.2.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (5.2)$$

Recall measures the ability of a model to correctly identify all relevant instances, specifically the ratio of correctly predicted positive instances to all actual positive instances. In 5.3 is shown the mathematical formulation of recall.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5.3)$$

5.2 VF2 Results

The current implementation of the VF2 algorithm, presented in chapter 2.4, handles the recognition of various analog and digital topologies. Among these topologies are current mirrors, differential pairs, NOR and NAND gates, and others. It is possible to define a list of different topologies that need to be recognized. During this work, we focused on recognizing CMs, which are one of the most common topologies within the dataset used for this work. In Table 5.1, it is possible to observe some tests that were conducted using VF2 for the recognition of CMs. The problem highlighted in chapter 2.4 becomes evident: the inefficiency when dealing with very large netlists. In this case, it is important to note that the algorithm only searches for CMs, which simplifies the graph matching compared to the case where all topologies are searched.

Therefore, the execution time would be even higher when considering more types of topologies.

Table 5.1: Multiple test of VF2 Algorithm Performance on a Dual-core, 8 GB RAM Setup

Test Case	Execution Time
Test 1	10h 25min
Test 2	11h 10min
Test 3	10h 48min
Test 4	10h 58min

As for the results, the confusion matrix summarizing the outcomes is shown in Figure 5.2 and the values of the main metrics used for evaluating the model’s performance in Table 5.2.

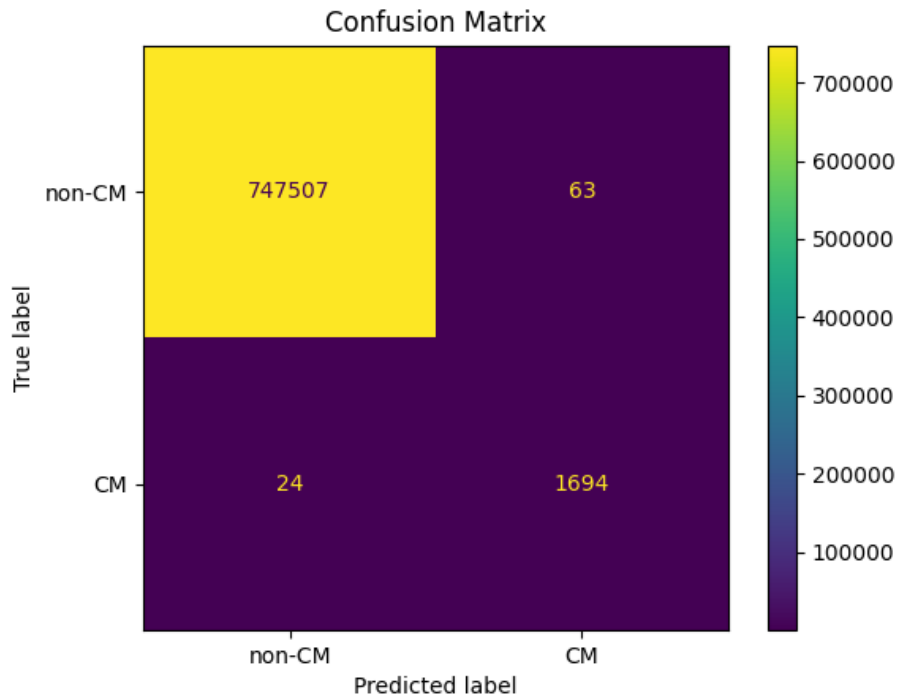


Figure 5.2: Results of the topology recognition of CMs using the VF2 algorithm

Table 5.2: Accuracy, Precision, and Recall results for topology recognition of CMs using VF2 algorithm

Metric	Value
Accuracy	$\frac{1694+747507}{1694+747507+24+63} \approx 0.9999$
Precision	$\frac{1694}{1694+63} \approx 0.9641$
Recall	$\frac{1694}{1694+24} \approx 0.9860$

5.3 GNNs Results

This section is divided into two further sections: the first one deals with the results without any pre-processing technique, while the second one addresses the results using the pre-processing technique presented in chapter 4.2. It is important to note that, as explained in chapter 4.3.2, all the models in this section utilize two stacked GNN layers followed by an MLP layer, which is essential for the classification process.

5.3.1 Results without Pre-Processing

GCN

GCN is one of the simplest models employed. The main results are shown in Figure 5.3, where the confusion matrix is presented. In Table 5.3, the results of the key metrics used to evaluate performance are shown, revealing that the precision is notably low. This lower precision can be attributed to the model’s inherent challenge of accurately detecting these specific structures within the graph, which inherently demands a higher level of precision in classification. Additionally, it’s worth considering that the limited number of available circuits for training may also contribute to this precision challenge.

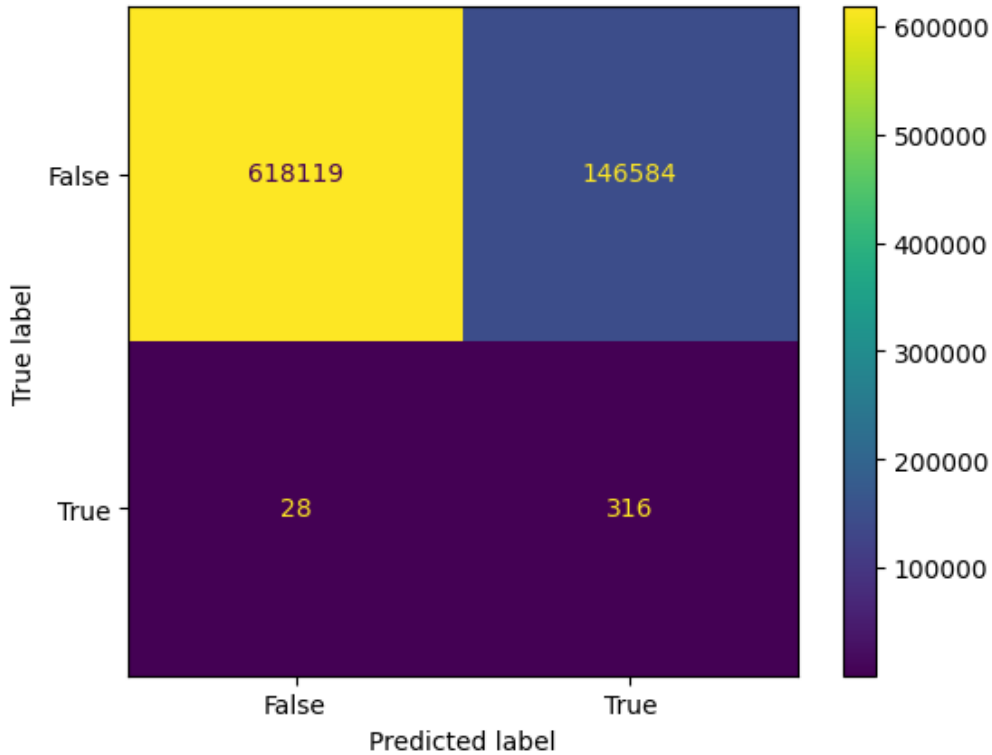


Figure 5.3: Results of the topology recognition of CMs for GCN model without pre-processing

Table 5.3: Accuracy, Precision, and Recall results for topology recognition of CMs for GCN model without pre-processing

Metric	Value
Accuracy	$\frac{316+618119}{316+618119+28+146584} \approx 0.8084$
Precision	$\frac{316}{316+146584} \approx 0.0022$
Recall	$\frac{316}{316+28} \approx 0.9186$

GraphSAGE

GraphSAGE manages to improve the accuracy obtained from GCN; however, there still exists an excessive number of false positives. The results are summarized in figure 5.4 and table 5.4.

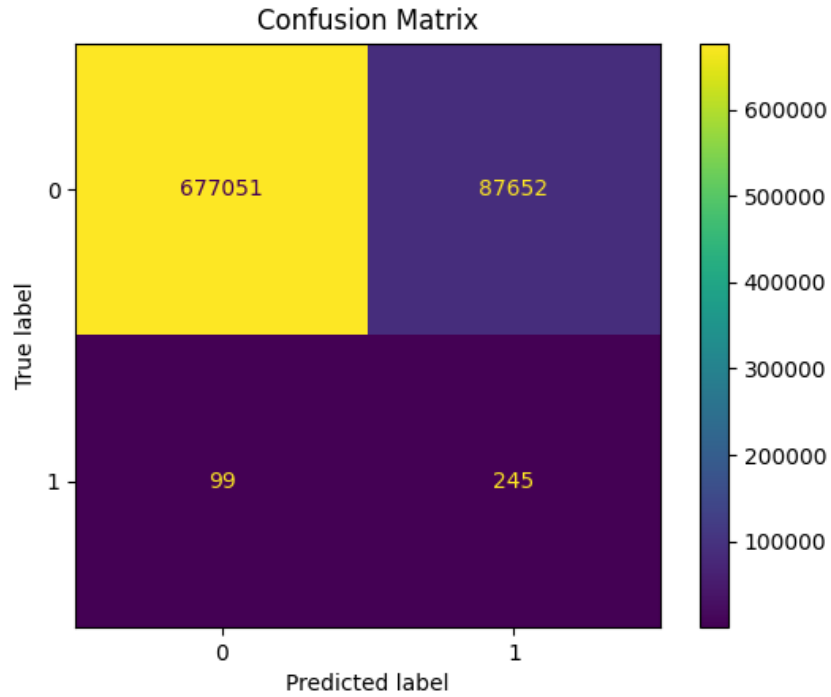


Figure 5.4: Results of the topology recognition of CMs for GraphSAGE model without pre-processing

Table 5.4: Accuracy, Precision, and Recall results for topology recognition of CMs for GraphSAGE model without pre-processing

Metric	Value
Accuracy	$\frac{245+677051}{245+677051+99+87652} \approx 0.8853$
Precision	$\frac{245}{245+87652} \approx 0.0028$
Recall	$\frac{245}{245+99} \approx 0.7122$

GAT

GAT is the best model when pre-processing is not applied. The results are displayed in figure 5.5 and table 5.5.

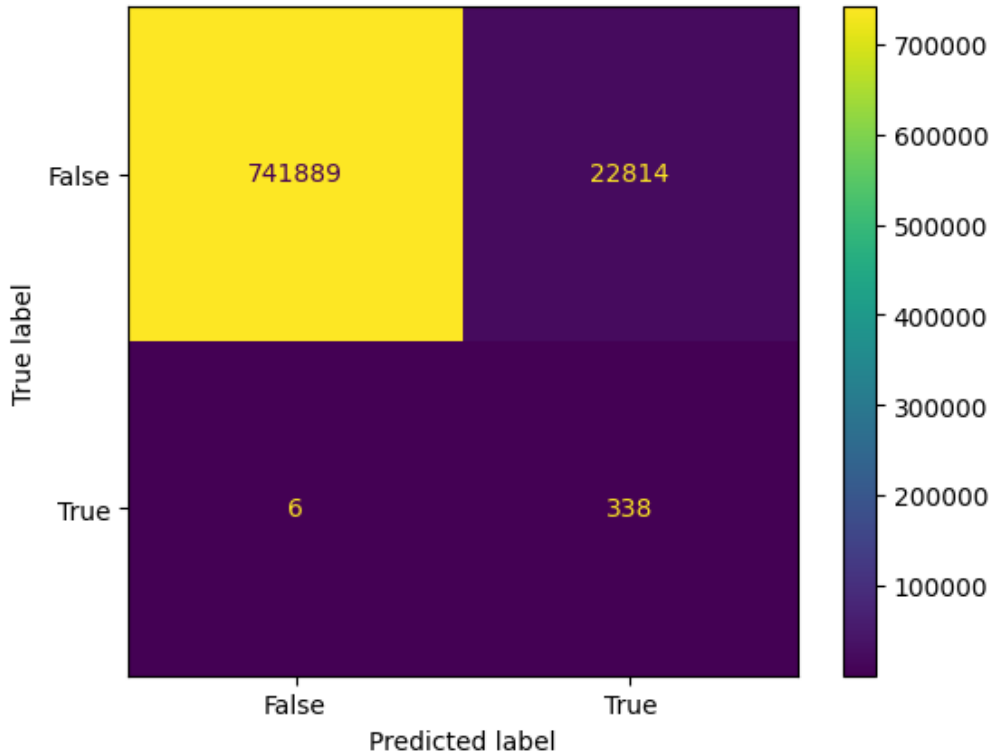


Figure 5.5: Results of the topology recognition of CMs for GAT model without pre-processing

Table 5.5: Accuracy, Precision, and Recall results for topology recognition of CMs for GAT model without pre-processing

Metric	Value
Accuracy	$\frac{338+741889}{338+741889+6+22814} \approx 0.9702$
Precision	$\frac{338}{338+22814} \approx 0.0146$
Recall	$\frac{338}{338+6} \approx 0.9826$

RGCN

RGCN is the only model that incorporates edge features during the learning phase, thus introducing additional information compared to the three previous models. The results are presented in figure 5.6 and table 5.6.

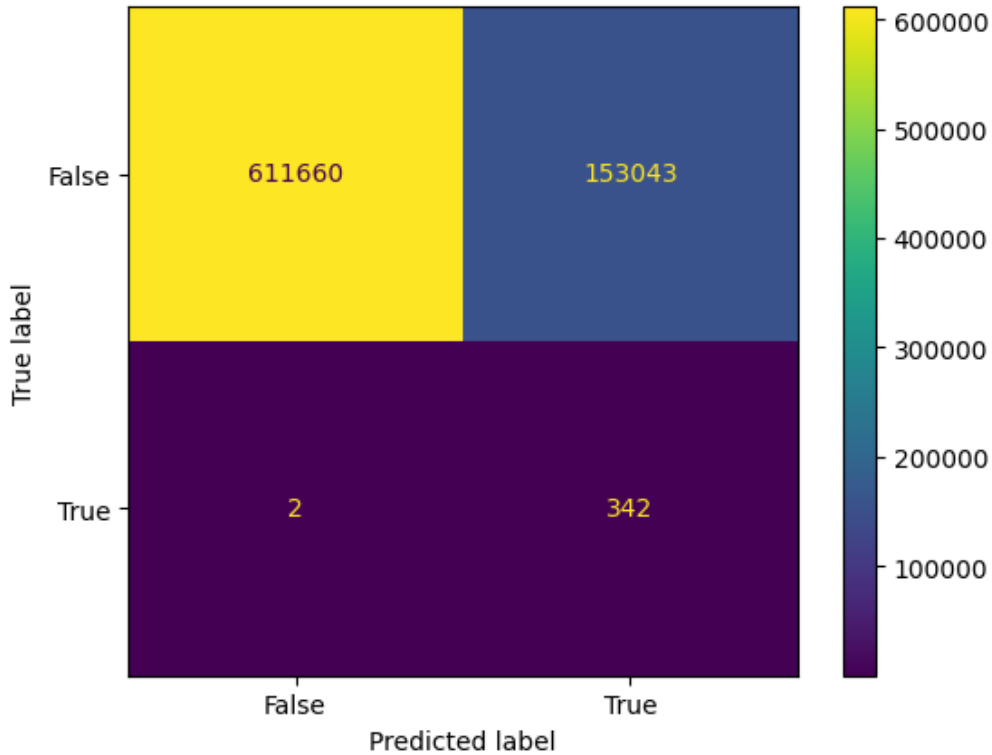


Figure 5.6: Results of the topology recognition of CMs for RGCN model without pre-processing

Table 5.6: Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model without pre-processing

Metric	Value
Accuracy	$\frac{342+611660}{342+611660+2+153043} \approx 0.7999$
Precision	$\frac{342}{342+153043} \approx 0.0022$
Recall	$\frac{342}{342+2} \approx 0.9942$

5.3.2 Results with Pre-Processing

GCN

The use of pre-processing enhances the classification performance of GCN, as shown in figure 5.7 and table 5.7. However, it's important to note that

the performance improvement, while noticeable, is not substantial. This suggests that the model may face intrinsic challenges in learning the complex topological structures within the graph.

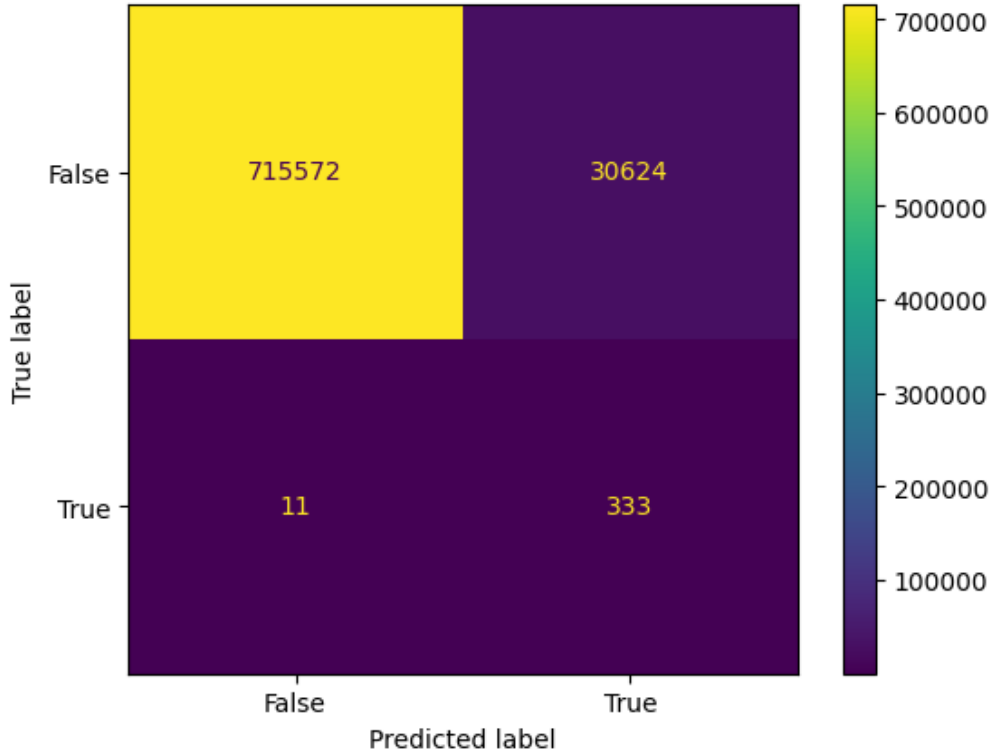


Figure 5.7: Results of the topology recognition of CMs for GCN model with pre-processing

Table 5.7: Accuracy, Precision, and Recall results for topology recognition of CMs for GCN model with pre-processing

Metric	Value
Accuracy	$\frac{333+715572}{333+715572+11+30624} \approx 0.9590$
Precision	$\frac{333}{333+30624} \approx 0.0108$
Recall	$\frac{333}{333+11} \approx 0.9680$

GraphSAGE

GraphSAGE also improves its performance compared to the implementation without pre-processing. The results are presented in figure 5.8 and table 5.8.

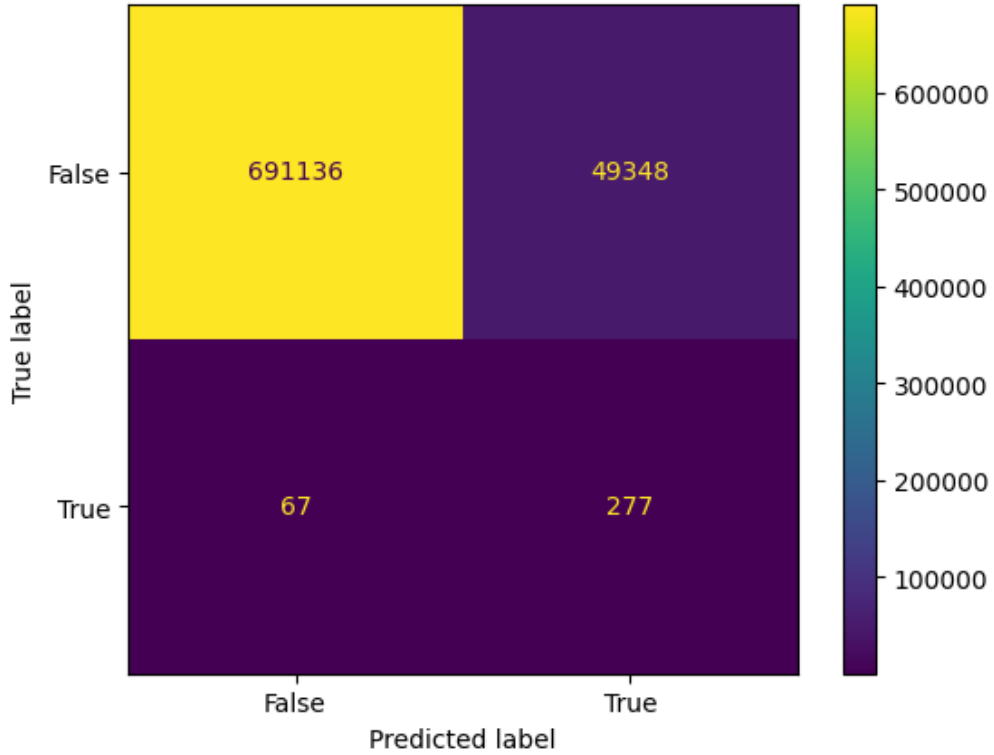


Figure 5.8: Results of the topology recognition of CMs for GraphSAGE model with pre-processing

Table 5.8: Accuracy, Precision, and Recall results for topology recognition of CMs for GraphSAGE model with pre-processing

Metric	Value
Accuracy	$\frac{277+691136}{277+691136+67+49348} \approx 0.9333$
Precision	$\frac{277}{277+49348} \approx 0.0056$
Recall	$\frac{277}{277+67} \approx 0.8052$

GAT

Compared to the other models, when utilizing the proposed pre-processing technique, the GAT model experiences a decrease in its performance. The results are displayed in figure 5.9 and table 5.9. One potential explanation for this performance decrease could be that the inherent attention mechanism in the GAT model, which weighs the importance of neighboring nodes differently, may interact in a complex way with the pre-processing step, leading to suboptimal results.

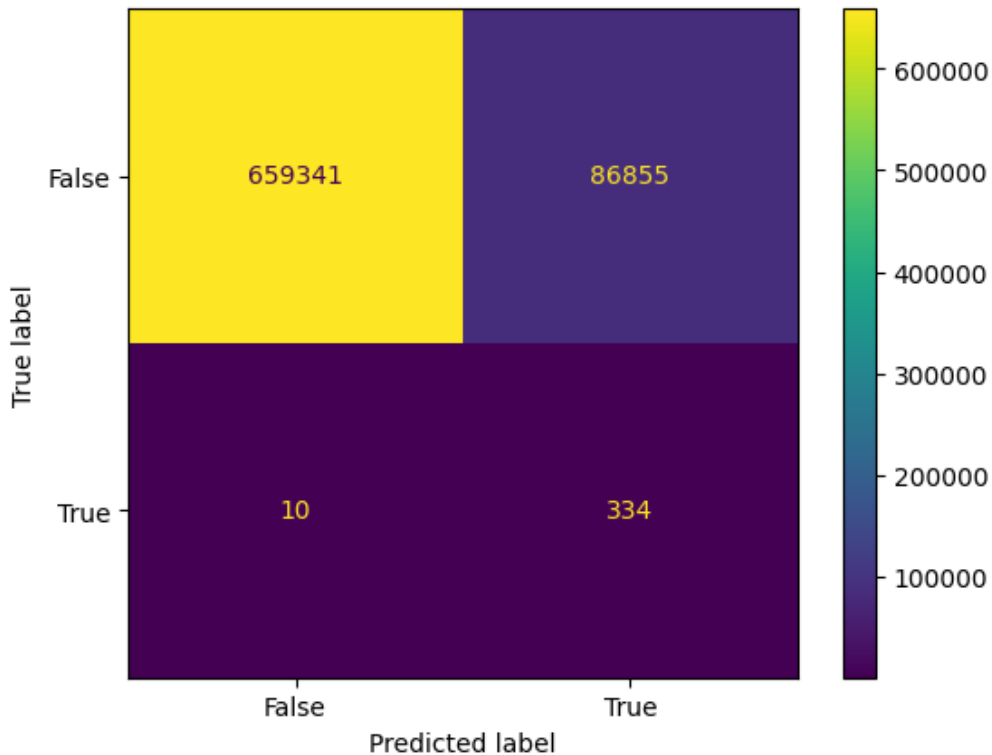


Figure 5.9: Results of the topology recognition of CMs for GAT model with pre-processing

RGCN

When combining the pre-processing technique with the RGCN model, a significant improvement in performance is achieved, making this model the best among all those tested. Results of this model are shown in figure 5.10 and table 5.10. For all the considered models, the recall is quite high, indicating

Table 5.9: Accuracy, Precision, and Recall results for topology recognition of CMs for GAT model with pre-processing

Metric	Value
Accuracy	$\frac{334+659341}{334+659341+10+86855} \approx 0.8836$
Precision	$\frac{334}{334+86855} \approx 0.0038$
Recall	$\frac{334}{334+10} \approx 0.9709$

that all the circuit modules are effectively recognized. However, even with pre-processing, the precision remains very low, meaning that many of the recognitions are false positives.

It’s worth noting, though, that the number of false positives (approximately 4,500 for RGCN) is orders of magnitude lower than the total number of nodes in the graph (over 740,000). From this initial phase of experiments, it can be concluded that by ‘filtering’ out the false positives generated by the network, it would be possible to achieve high recognition accuracy, even in terms of recall, with significantly reduced computational time compared to the VF2 algorithm. This was the motivation for using VF2 as post-processing on the network outputs, which led to substantial improvements in the results, as demonstrated in the next section.

Table 5.10: Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model with pre-processing

Metric	Value
Accuracy	$\frac{343+741674}{343+741674+1+4522} \approx 0.9939$
Precision	$\frac{343}{343+4522} \approx 0.0705$
Recall	$\frac{343}{343+1} \approx 0.9971$

5.4 GNN and VF2 Results

As explained in chapter 4.4, an essential part of this work was the introduction of a post-processing technique capable of primarily reducing issues

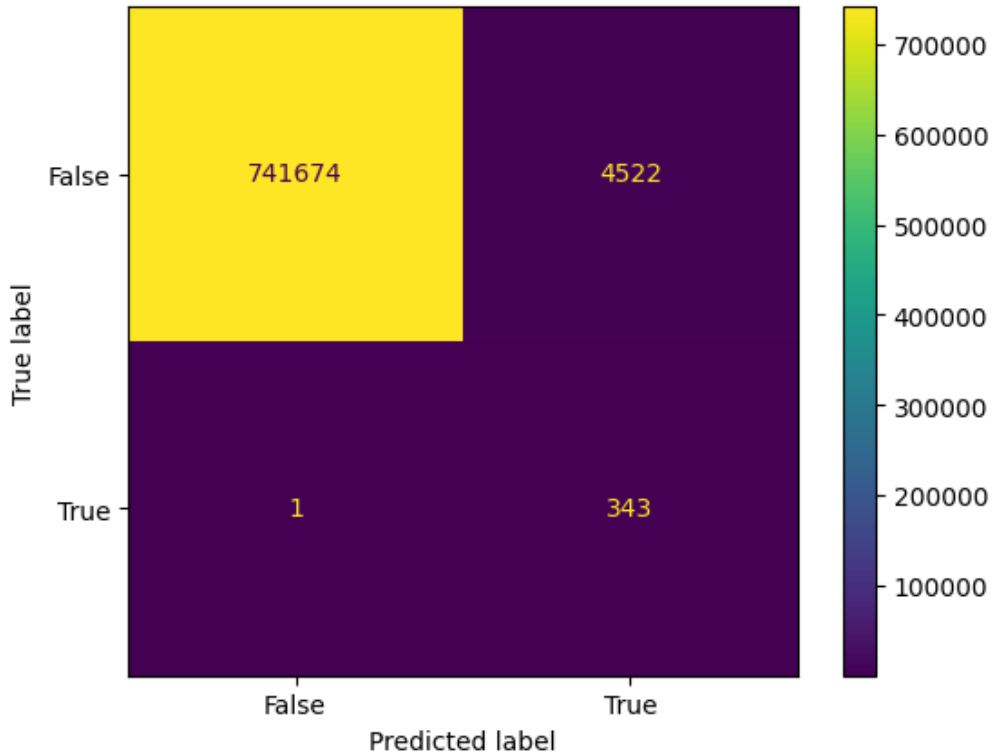


Figure 5.10: Results of the topology recognition of CMs for RGCN model with pre-processing

related to a high number of false positives. The entire workflow of this process can be observed in figure 5.11. Initially, the data is pre-processed by removing unnecessary devices for topology recognition purposes. Subsequently, this pre-processed data is fed into the two RGCN layers, which generate embeddings for each node and then classify them using an MLP layer. Following this classification, the nodes identified as positive undergo post-processing. This final phase significantly reduces the number of false positives.

In figure 5.12, both confusion matrices are visible, comparing the pipeline that solely includes the use of GNNs with the pipeline that combines GNNs with VF2. Table 5.11, on the other hand, presents a comparison of results related to key metrics used to assess model performance, which consistently favor the use of the post-processing technique.

Another critical aspect is summarized in Table 5.12, where the time required to execute both pipelines is compared. As discussed in Chapter 2.4, the VF2 algorithm faces substantial challenges when dealing with netlists containing numerous devices, leading to significantly longer execution times

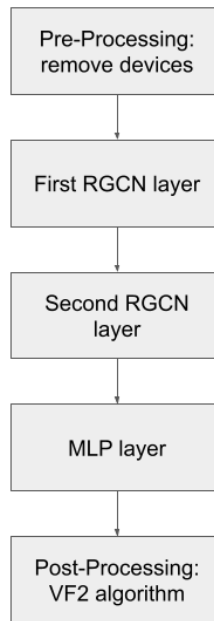


Figure 5.11: Comprehensive Workflow Pipeline, the complete sequence of data preprocessing, RGCN-based classification, and post-processing steps in the project’s workflow

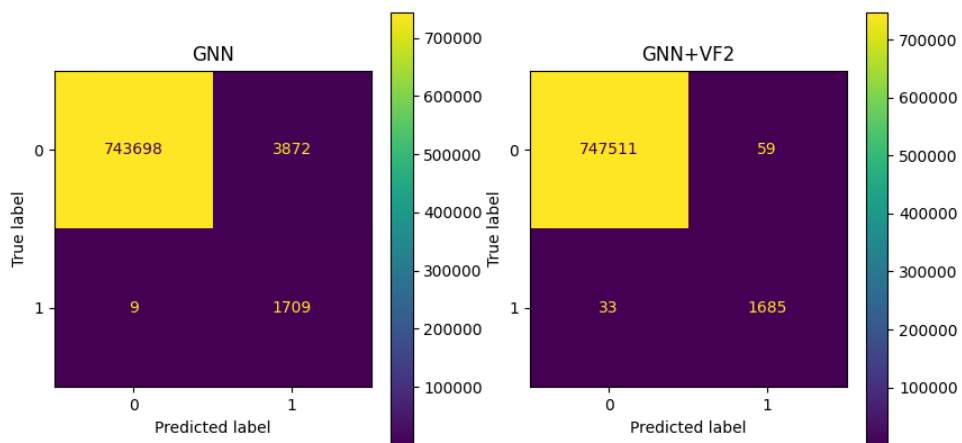


Figure 5.12: **On the left**, the confusion matrix without post-processing, and **on the right**, the confusion matrix with post-processing

Table 5.11: Accuracy, Precision, and Recall results for topology recognition of CMs for RGCN model with pre-processing and post-processing

Metric	RGCN	RGCN + VF2
Accuracy	0.9948	0.9999
Precision	0.3062	0.9662
Recall	0.9948	0.9808

when used in isolation compared to the combination with a GNN. This discrepancy arises from the fact that GNNs do not rely on a resource-intensive algorithm like graph matching, which forms the foundation of VF2. Moreover, since the majority of nodes are classified by the GNN, the set of nodes that undergo VF2 processing is substantially smaller compared to the entire graph, reducing the computational resources required by the graph matching algorithm. In the scenario presented in this work, the estimated execution times are around 2 minutes, resulting in a speed up of approximately 330x with respect to the purely rule-based approach.

Table 5.12: Comparison of multiple test of only RGCN and RGCN + VF2 time performance on a Dual-core, 8 GB RAM Setup

Test Case	VF2	RGCN + VF2
Test 1	10h 25min	1min 54s
Test 2	11h 10min	2min 12s
Test 3	10h 48min	1min 59s
Test 4	10h 58min	2min 03s

Chapter 6

Conclusion and Future Works

This work was motivated by the growing demand for integrated circuits and the need to reduce physical design times for AMS ICs. The available dataset for this thesis consisted of a single large AMS netlist, highlighting the potential for better results with a larger dataset. In general, datasets in this field are often limited in availability due to the substantial time and expertise required for their design and development, making them less likely to be shared by companies.

The primary objective of this thesis was to explore the application of GNNs in the realm of AMS ICs physical design. It aimed to assess whether these techniques could enhance existing methods in terms of time efficiency and computational resource utilization while ensuring a high level of accuracy and robustness in the results, which are essential requisites in this field.

The initial phase of the study focused on understanding the capabilities of GNNs, excluding any pre- or post-processing techniques. Results from this phase revealed a significant issue: a high number of false positives. To address this challenge, the introduction of a post-processing technique became necessary. The VF2 algorithm, which leverages a graph-matching algorithm, was chosen to substantially reduce the number of false positives, improving the overall pipeline's performance. Subsequently, extensive research was conducted on pre-processing to enhance the model's performance. This research led to the decision to eliminate all devices not involved in various topologies, removing any irrelevant information for the model's recognition purposes.

Initially, the primary focus was on recognizing current mirrors, with other

circuit topologies not being considered. The idea is that this work can be expanded by introducing a single model capable of recognizing various topologies, as recognizing current mirrors alone is insufficient to replace the current algorithms used for such tasks. An initial multiclassification model test yielded the result that all nodes were predicted as "others", highlighting the challenge of addressing class imbalance among the different classes.

Nevertheless, this work demonstrates the potential of Graph Neural Networks in the field of AMS ICs physical design, since the results of this work in current mirror recognition closely align with those of the VF2 algorithm while significantly reducing the time required for topology recognition.

Appendix A

Acronyms

AMS	Analog Mixed Signal
CM	Current Mirror
CascLV	Cascode and Long-Tailed Pair
DP	Differential Pair
DPx	Differential Pairs with Additional Transistors
EDA	Electronic Design Automation
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GraphSAGE	Graph Sample and Aggregated
ICs	Integrated Circuits
ML	Machine Learning
MLP	Multilayer Perceptron
RGCN	Relational Graph Convolutional Network
SoC	System on Chip

Bibliography

1. *The McClean Report* (IC Insights, 2022).
2. Mina, R., Jabbour, C. & Sakr, G. E. A Review of Machine Learning Techniques in Analog Integrated Circuit Design Automation. *Electronics* **11**, 435. ISSN: 2079-9292. <https://www.mdpi.com/2079-9292/11/3/435> (2023) (Jan. 31, 2022).
3. Eick, M., Strasser, M., Lu, K., Schlichtmann, U. & Graeb, H. E. Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **30**, 180–193. ISSN: 0278-0070, 1937-4151. <http://ieeexplore.ieee.org/document/5689366/> (2023) (Feb. 2011).
4. Kunal, K. *et al.* *GANNA: Graph Convolutional Network Based Automated Netlist Annotation for Analog Circuits* in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE) 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, Grenoble, France, Mar. 2020), 55–60. ISBN: 978-3-9819263-4-7. <https://ieeexplore.ieee.org/document/9116329/> (2023).
5. Hong, X., Lin, T., Shi, Y. & Gwee, B. H. *ASIC Circuit Netlist Recognition Using Graph Neural Network* in *2021 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA) 2021 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)* (IEEE, Singapore, Singapore, Sept. 15, 2021), 1–5. ISBN: 978-1-66543-988-6. <https://ieeexplore.ieee.org/document/9617311/> (2023).
6. Kunal, K. *et al.* *ALIGN: Open-Source Analog Layout Automation from the Ground Up* Pages: 4. 1 p. (June 2, 2019).

7. Mohanty, S. P. *Nanoelectronic mixed-signal system design* OCLC: ocn894333799. 788 pp. ISBN: 978-0-07-182571-9 (McGraw-Hill Education, New York, 2015).
8. in. *Wikipedia* Page Version ID: 1174239606 (Sept. 7, 2023). https://en.wikipedia.org/w/index.php?title=Mixed-signal_integrated_circuit&oldid=1174239606 (2023).
9. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* **20**. Conference Name: IEEE Transactions on Neural Networks, 61–80. ISSN: 1941-0093 (Jan. 2009).
10. Stokes, J. M. *et al.* A Deep Learning Approach to Antibiotic Discovery. *Cell* **180**, 688–702.e13. ISSN: 1097-4172 (Feb. 20, 2020).
11. Monti, F., Frasca, F., Eynard, D., Mannion, D. & Bronstein, M. M. *Fake News Detection on Social Media using Geometric Deep Learning* Feb. 10, 2019. arXiv: [1902.06673\[cs,stat\]](https://arxiv.org/abs/1902.06673). <http://arxiv.org/abs/1902.06673> (2023).
12. Eksombatchai, C. *et al.* *Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time* Nov. 20, 2017. arXiv: [1711.07601\[cs\]](https://arxiv.org/abs/1711.07601). <http://arxiv.org/abs/1711.07601> (2023).
13. Sanchez-Lengeling, B., Reif, E., Pearce, A. & Wiltschko, A. B. A Gentle Introduction to Graph Neural Networks. *Distill* **6**, e33. ISSN: 2476-0757. <https://distill.pub/2021/gnn-intro> (2023) (Sept. 2, 2021).
14. Janiesch, C., Zschech, P. & Heinrich, K. Machine learning and deep learning. *Electronic Markets* **31**, 685–695. ISSN: 1019-6781, 1422-8890. arXiv: [2104.05314\[cs\]](https://arxiv.org/abs/2104.05314). <http://arxiv.org/abs/2104.05314> (2023) (Sept. 2021).
15. Rosenblatt, F. The perceptron - A perceiving and recognizing automaton (1957).
16. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. *Neural Message Passing for Quantum Chemistry* June 12, 2017. arXiv: [1704.01212\[cs\]](https://arxiv.org/abs/1704.01212). <http://arxiv.org/abs/1704.01212> (2023).
17. Battaglia, P. W. *et al.* *Relational inductive biases, deep learning, and graph networks* Oct. 17, 2018. arXiv: [1806.01261\[cs,stat\]](https://arxiv.org/abs/1806.01261). <http://arxiv.org/abs/1806.01261> (2023).

18. Lachaud, G., Conde-Cespedes, P. & Trocan, M. *Comparison between Inductive and Transductive Learning in a Real Citation Network using Graph Neural Networks* in *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) 2022* IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (Nov. 2022), 534–540.
19. Mishra, P., Piktus, A., Goossen, G. & Silvestri, F. *Node Masking: Making Graph Neural Networks Generalize and Scale Better* May 16, 2021. arXiv: [2001.07524\[cs, stat\]](https://arxiv.org/abs/2001.07524). <http://arxiv.org/abs/2001.07524> (2023).
20. Kipf, T. N. & Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks* Feb. 22, 2017. arXiv: [1609.02907\[cs, stat\]](https://arxiv.org/abs/1609.02907). <http://arxiv.org/abs/1609.02907> (2023).
21. *How powerful are Graph Convolutional Networks?* <http://tkipf.github.io/graph-convolutional-networks/> (2023).
22. Hamilton, W. L., Ying, R. & Leskovec, J. *Inductive Representation Learning on Large Graphs* Sept. 10, 2018. arXiv: [1706.02216\[cs, stat\]](https://arxiv.org/abs/1706.02216). <http://arxiv.org/abs/1706.02216> (2023).
23. Veličković, P. *et al.* *Graph Attention Networks* Feb. 4, 2018. arXiv: [1710.10903\[cs, stat\]](https://arxiv.org/abs/1710.10903). <http://arxiv.org/abs/1710.10903> (2023).
24. Schlichtkrull, M. *et al.* *Modeling Relational Data with Graph Convolutional Networks* Oct. 26, 2017. arXiv: [1703.06103\[cs, stat\]](https://arxiv.org/abs/1703.06103). <http://arxiv.org/abs/1703.06103> (2023).
25. Ohlrich, M., Ebeling, C., Ginting, E. & Sather, L. *SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm* in *30th ACM/IEEE Design Automation Conference* 30th ACM/IEEE Design Automation Conference. ISSN: 0738-100X (June 1993), 31–37.
26. *Analog/Mixed Signal Back End Design Automation based on Machine Learning and Artificial Intelligence Techniques | AMBEATion | Project | Fact sheet | H2020 | CORDIS | European Commission* <https://cordis.europa.eu/project/id/101007730> (2023).
27. Huang, G. *et al.* *Machine Learning for Electronic Design Automation: A Survey*. *ACM Transactions on Design Automation of Electronic Systems* **26**, 40:1–40:46. ISSN: 1084-4309. <https://doi.org/10.1145/3451179> (2023) (June 5, 2021).

28. Mirhoseini, A. *et al.* A graph placement methodology for fast chip design. *Nature* **594**. Number: 7862 Publisher: Nature Publishing Group, 207–212. ISSN: 1476-4687. <https://www.nature.com/articles/s41586-021-03544-w> (2023) (June 2021).
29. Lopera, D. S. *et al.* A Survey of Graph Neural Networks for Electronic Design Automation in 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD) 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD) (Aug. 2021), 1–6. <https://ieeexplore.ieee.org/document/9531070> (2023).
30. *MAGICAL: Toward Fully Automated Analog IC Layout Leveraging Human and Machine Intelligence: Invited Paper* <https://ieeexplore.ieee.org/document/8942060> (2023).
31. in. *Wikipedia* Page Version ID: 1171802316 (Aug. 23, 2023). https://en.wikipedia.org/w/index.php?title=Field-effect_transistor&oldid=1171802316 (2023).
32. Kotsiantis, S. B., Kanellopoulos, D. & Pintelas, P. E. Data Preprocessing for Supervised Learning. *International Journal of Computer and Information Engineering* **1**, 4104–4109. <https://publications.waset.org/14136/data-preprocessing-for-supervised-learning> (2023) (Dec. 28, 2007).
33. *Pytorch* GitHub. <https://github.com/pytorch> (2023).
34. *Pytorch Geometric* <https://github.com/pyg-team> (2023).
35. *NetworkX* GitHub. <https://github.com/networkx> (2023).
36. Cai, C. & Wang, Y. *A Note on Over-Smoothing for Graph Neural Networks* June 23, 2020. arXiv: 2006.13318[cs, stat]. <http://arxiv.org/abs/2006.13318> (2023).
37. López, V., Fernández, A., García, S., Palade, V. & Herrera, F. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences* **250**, 113–141. ISSN: 0020-0255. <https://www.sciencedirect.com/science/article/pii/S0020025513005124> (2023) (Nov. 20, 2013).
38. Perera, P., Oza, P. & Patel, V. M. *One-Class Classification: A Survey* Jan. 8, 2021. arXiv: 2101.03064[cs]. <http://arxiv.org/abs/2101.03064> (2023).

BIBLIOGRAPHY

39. Demir, F. in *Artificial Intelligence-Based Brain-Computer Interface* (eds Bajaj, V. & Sinha, G. R.) 317–351 (Academic Press, Jan. 1, 2022). ISBN: 978-0-323-91197-9. <https://www.sciencedirect.com/science/article/pii/B9780323911979000138> (2023).
40. *MAGICAL: Toward Fully Automated Analog IC Layout Leveraging Human and Machine Intelligence: Invited Paper* <https://ieeexplore.ieee.org/document/8942060/> (2023).
41. Lopera, D. S. *et al.* *A Survey of Graph Neural Networks for Electronic Design Automation in 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)* 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD) (IEEE, Raleigh, NC, USA, Aug. 30, 2021), 1–6. ISBN: 978-1-66543-166-8. <https://ieeexplore.ieee.org/document/9531070/> (2023).