Master Thesis

Computational Science

University of Potsdam

# Assessing the Ability of Graph Neural Networks to Predict Power Outages of Cascading Failures In a Power Grid


| | |
|---|---|
| Author: | Tobias Ohlinger |
| E-Mail: | ohlinger@uni-potsdam.de |
| Supervisors: | Prof. Dr. Tobias Scheffer, *Institut für Informatik, Universität Potsdam* |
| | Dr. Mehrnaz Anvari, *Potsdam Institute for Climate Impact Research, Fraunhofer-Institut für Algorithmen und Wissenschaftliches Rechnen SCAI* |



Potsdam Institute for Climate Impact Research


Universität Potsdam

# Abstract

We conducted experiments using three Graph Neural Network (GNN) models, namely the Graph Isomorphism Network with Edge features (GINE), the Topology Adaptive Graph Convolutional Network (TAG), and the Graph Attention Network (GAT). Our goal was to predict power outages in a potentially damaged power grid caused by single line failures during hurricanes. The power grid data was generated using the AC-CFM model applied to the ACTIVSg2000 synthetic Texan power grid, which consists of 2000 nodes. To simulate initial damages, we considered probabilistic scenarios of line failures based on historical wind data from 7 hurricanes or tropical storms that occurred in Texas over the past 22 years.

However, our analysis comparing GNNs with multiple baselines (a model predicting nodal means, a Ridge regression, a Multi-Layer Perceptron (MLP) and a MLP extended with a Node2Vec embedding (Node2Vec+MLP)) revealed that the provided data lacks sufficient structure for GNNs to effectively learn the task. While the Graph Isomorphism Network with Edge features (GINE) showed a significant improvement ($p < 0.05$) over the two best performing baseline (Node2Vec+MLP and MLP), but the improvements fall within the standard deviations. Additionally, the performance improvement over the Node Mean was not significant ($p > 0.05$).

The other two GNNs, TAG and GAT, showed no significant improvement over the baselines, except for Ridge Regression, which performed particularly poorly.

Our assessment, based on common performance measures, demonstrated strikingly similar performance among Node2Vec+MLP, MLP, TAG, and GINE. Furthermore, when calculating the $R^2$ score between model outputs, we obtained values greater than 0.98 for TAG, GINE, and MLP. This indicates that these GNNs failed to effectively utilize the graph-structured data and learned an underlying topology-dependent structure. This observation is supported by the fact that one of the provided node features exhibited a correlation of 0.59 with node labels, providing sufficient information for MLP to learn some connections between input features and node labels, which we assume were similarly learned by the GNNs.



# Zusammenfassung

In dieser Arbeit wurden Experimente mit drei Graph Neural Network (GNN) Modellen durchgeführt, dem Graph Isomorphism Network mit Edge features (GINE), dem Topology Adaptive Graph Convolutional Network (TAG) und dem Graph Attention Network (GAT). Das Ziel war die Vorhersage des Leistungsverlusts an jedem Bus eines Stromnetzes, durch Ausfälle einzelner Leitungen verursacht durch Hurricanes. Dabei konnte schon der Ausgangszustand des Netzes Beschädigungen aufweisen.

Die Stromnetzdaten wurden mithilfe des AC-CFM-Modells und dem ACTIVSg2000 synthetischen Stromnetz von Texas mit 2000 Bussen generiert. Zur Simulation der 'initial damages' wurden probabilistische Szenarien von Leitungsausfällen basierend auf historischen Winddaten von 7 Hurricanes oder tropischen Stürmen, die in den letzten

22 Jahren in Texas aufgetreten sind, berücksichtigt.

In der durchgeführten Analyse wurden GNNs mit mehreren Baselines (Multi-Layer Perceptron (MLP), Ridge Regression, MLP erweitert mit einem Node2Vec-Embedding (Node2Vec+MLP) und einem Modell, das nodale Mittelwerte vorhersagt) verglichen. Dabei zeigte sich, dass die bereitgestellten Daten nicht ausreichend strukturiert waren, um den GNNs die effektive Vorhersage der nodalen Stromausfälle zu ermöglichen.

Während das Graph Isomorphism Network mit Edge features (GINE) eine signifikante Verbesserung ($p < 0,05$) im $R^2$ score gegenüber den beiden am besten abschneidenden Baselines (Node2Vec+MLP und MLP) aufwies, lag die Verbesserung innerhalb der Standardabweichung. Darüber hinaus war die Leistungsverbesserung gegenüber dem Node Mean nicht signifikant ($p > 0,05$).

Die anderen beiden GNNs, TAG und GAT performten schlechter als GINE und zeigten somit keine signifikante Verbesserung gegenüber den Baselines (entweder statistisch oder in Anbetracht der Standardabweichung), mit Ausnahme der Ridge Regression, die besonders schlecht abschnitt.

Die Bewertung basierend auf gängigen Leistungskennzahlen ergab eine erstaunlich ähnliche Performance bei Node2Vec+MLP, MLP, TAG und GINE. Zusätzlich ergaben sich $R^2$-Werte größer als 0,98 für TAG, GINE und MLP, wenn die Modellausgaben miteinander verglichen wurden. Dies deutet darauf hin, dass diese GNNs die graphenstrukturierten Daten nicht effektiv nutzten und eine zugrundeliegende, topologieabhängige Struktur erlernten. Diese Beobachtung wurde durch die Tatsache gestützt, dass eins der bereitgestellten Nodefeatures eine Korrelation von 0,59 mit den Nodelabels aufwies, was ausreichende Informationen für MLP bereitstellte, um einige Verbindungen zwischen Eingabeeigenschaften und Knotenlabels zu erlernen, von denen wir vermuten, dass sie von den GNNs ähnlich erlernt wurden.

# Contents

# Nomenclature

## Latin Letters

| Variable | Meaning |
|---|---|
| $A$ | Adjacency Matrix with Added Self Connections |
| $b$ | Bias |
| $B$ | Susceptance |
| $C$ | Power Capacity of a transmission line |
| $D$ | Degree Matrix |
| $e$ | Euler Number |
| $e_i$ | Edge $i$ in a Graph |
| $e_{ij}$ | Edge connecting Nodes $i$ and $j$ |
| $\mathcal{E}$ | Set of Edges in a Graph |
| $F_{brk}$ | Maximum Force a line can endure |
| $F_{wind,k}$ | Wind Force on a transmission line $k$ |
| $G$ | Conductance |
| $\mathcal{G}$ | Graph |
| $\boldsymbol{h}$ | Hidden Feature Vector |
| $i$ | Imaginary Unit |
| $I$ | Electrical Current |
| $\boldsymbol{I}$ | Identity Matrix |
| $l$ | Loss Function |
| $l_{var}$ | Loss Function weighting loss by variance |
| $l_{non-zero}$ | Loss Function weighting loss contribution of non-zero labels |
| $L$ | Laplacian Matrix |
| $\tilde{L}$ | Normalized Laplacian Matrix |
| $L_1$ | Mean Absolute Error Loss |
| $L_2$ | Mean Squared Error Loss |
| $\mathcal{N}(v_i)$ | Nearest Neighbours of Node $v_i$ |
| $p$ | Probability |
| $P$ | Active Power |
| $r_{brk}$ | Collapse Rate |
| $Q$ | Reactive Power |
| $S$ | Apparent Power |
| $\boldsymbol{S}$ | Sample |

| Variable | Meaning |
|---|---|
| $t$ | Time |
| $U$ | Eigenvector Matrix |
| $v_i$ | Node $i$ in a Graph |
| $V$ | Voltage |
| $\mathcal{V}$ | Set of Nodes in a Graph |
| $\boldsymbol{W}$ | Weight Matrix |
| $\boldsymbol{x}$ | Feature Vector of one Instance |
| $\boldsymbol{y}$ | Label Vector of one Instance |
| $\hat{\boldsymbol{y}}$ | Output Vector of Last Layer |
| $Y$ | Admittance Matrix |

## Greek Letters

| Variable | Meaning |
|---|---|
| $\gamma$ | Time to Failure |
| $\Theta$ | Phase Angle |
| $\Lambda$ | Eigenvalue Matrix |
| $\sigma$ | Standard Deviation |
| $\tau$ | Time interval |
| $\varphi$ | Activation Function |
| $\omega$ | Grid Frequency |

## Abbreviations

| Abbreviation | Meaning |
|---|---|
| AC | Alternating Current |
| AC-CFM | A specific Alternating Current Cascading Failure Model [23] |
| CNN | Convolutional Neural Network |
| DC | Direct Current |
| FP | False Positives |
| FPR | False Positive Rate |
| GAT | Graph Attention Network |
| GCN | Graph Convolutional Network |
| GIN | Graph Isomorphism Network |
| GINE | Graph Isomorphism Network extended to use edge features |
| GNN | Graph Neural Network |
| MLP | Multi Layer Perceptron |
| MSE | Mean Squared Error |
| PR | Precision-Recall |

| Abbreviation | Meaning |
|---|---|
| ROC | Receiver Operating Characteristic |
| SGD | Stochastic Gradient Descent |
| TAG | Topology Graph Adaptive Convolutional Network |
| TN | True Negatives |

# 1 Introduction

Cascading power failures represent a significant problem in power grid operation. Cascading failures can appear as the initial failure of one component requires rerouting of the power. If some components can not satisfy the necessary needs for the rerouted solution they will also fail which again triggers rerouting. Like this failures can 'cascade' through the network. Since power grids are interconnected with various critical infrastructures such as transportation and healthcare systems (e.g., hospitals), power supply failures are capable of triggering chain reactions that affect multiple aspects of societal infrastructure. Consequently, these failures can lead to substantial economic losses, social disruption, and even loss of life. Predicting the occurrence and extent of cascading power failures is crucial for power grid operators to implement appropriate preventive and corrective measures. Recently, Graph Neural Networks (GNNs) have demonstrated promising potential in addressing complex problems related to graph-structured data. This research aims to evaluate the effectiveness of GNNs in predicting power outages resulting from cascading failures in a power grid.

This project's primary goal is to successfully predict power outages for all nodes in a potentially damaged power grid after experiencing single line failures caused by hurricanes. The data used for this analysis is generated using an AC cascading failure model (AC-CFM) [23] applied to the ACTIVSg2000 realistic, synthetic power grid data of the Texan power grid [28]. Additionally, line failures are calculated using a probabilistic approach, where the probability depends on historical wind data from seven hurricanes that occurred in Texas between 2001 and 2020.

Graph neural networks are intuitively well-suited for this task, given the inherent graph properties of the power grid and the positive outcomes observed in existing literature, especially in similar problem settings with smaller power grids.

Consequently, this study will explore the application of three prominent graph convolutional layers: the Graph Attention Network layer [30], the Graph Isomorphism Network layer [32] (expanded to incorporate edge features based on [12]), and the Topology Adaptive Graph Convolutional Networks layer [6]. Due to the lack of well-established benchmarks in this field, the study will compare the performance of these graph convolutional layers against several baseline models. The baselines include a model that always predicts the nodal mean outage, Ridge Regression, a Multi-Layer Perceptron utilizing only nodal data, and a Multi-Layer Perceptron that uses nodal data extended by embeddings generated through the Node2Vec [10] algorithm.

# 2 Literature Review

Several studies have explored the use of machine learning techniques in predicting power system failures. For example, Chun Zhang et al. [33] successfully used a Random Forest with a recursive feature elimination algorithm to predict transient stability in the IEEE 39-Bus network.

In "Predicting basin stability of power grids using graph neural networks" [22], Nauck et al. compared different GNN architectures in their performance when predicting the single node basin stability of homogeneous synthetic networks with 20 or 100 nodes. They showed that GNNs are able to predict the single node basin stability with an accuracy of up to 85%, but their performance depends significantly on the chosen architecture.

In "Guiding cascading failure search with interpretable graph convolutional network" [19], Liu et al. used a GCN trained on offline power grid data to guide the online search for cascading failures in a power grid. The GCN learned the branch vulnerabilities at different operating states of the network to guide the search along with physical estimates on the branch vulnerabilities. Their GCN architecture outperformed the purely physically guided search for cascading failures on two sets, one being an IEEE 24 bus network and the other a network with 730 buses.

Shuvro et al. [26] predicted the load shed and the amount of failed power lines in an IEEE 118 bus network using different machine learning techniques (but not GNNs). They achieved good results with a random forest predictor and a support vector machine.

In "Prediction and mitigation of nonlocal cascading failures using graph neural networks" [13], Jhun et al. introduced a measure for each node based on its importance during cascading failures depending on the impact of a cascade triggered by that node and on the participation of that node in cascades triggered by other nodes. They call this measure the Avelanche Centrality. Their dataset is based on static homogeneous data based on the Motter-Lai-Model [21]. To predict the Avelanche Centraility they use a GNN, which, after training, could be used as a mitigation strategy for the cascading failures. While successful in ranking the impact of the nodes on cascading failures measured in ranking measures, they were not able to achieve a positive $R^2$ score with their setup.

Zhou et al. [34] proposed a Transformer Network to predict the severity of power outages and the number of failed lines after cascading failures by treating a cascade as a sequence. The Transformer Network was up to 50 times faster at predicting power outages and failed lines compared to the initial model used to create the data. All previous papers use DC data. Although DC data is less realistic than AC, DC simulations are still widely used in power grid models since AC models are much more complex and time-consuming. In this work, we focus specifically on AC data as it is closer to the

real-world application.

One successful paper using AC cascading failure data is "Geometric deep learning for online prediction of cascading failures in power grids" [29]. Here, Varbella et al. applied a Transformer Network on a similar task as we do in this report. They trained a deep transformer network on AC cascading failure data to predict whether there will be a power outage or not. The main difference from this report is that the networks used were much smaller ($< 180$ buses), and the dataset only contained data with contingencies starting from the fully functioning grid. Nevertheless, with this data and their setup, they achieved very high accuracies with minimal runtime, enabling an online application in power grids.

Chen et al. proposed a Topological Convolutional Network, which they used to predict the transient stability of historical AC data. Transient stability in a power grid refers to the network maintaining a stable operational state after a disturbance in the grid. Their proposed architecture outperformed various non-GNN benchmarks. The main difference from this work is that they used historical transient stability data, which means that they used graph classification to classify the whole graph as stable or unstable, whereas we predict the load shed.

Kim et al. [14] successfully applied a GCN to AC data to predict the optimal load shedding in the network after line contingency based on the network state and the damaged lines. The data used is very similar to ours as it uses almost the same features. The main differences are that the used networks are much smaller (9-118 buses), and they use different load profiles [1].

As can be seen, there are already many successful applications of GNN techniques to predict and analyze power networks. However, to the best of our knowledge, there is no application yet of GNNs to predict the power outage at each bus after cascading failures based on data obtained by an AC Power Flow model. Additionally, most of the aforementioned papers use DC models (or models that do not use power flow at all), and most of them work on much smaller power grids with the order of 10 or 100 buses, whereas in this work, we will be using data with 2000 buses.

One of the key challenges in predicting cascading failures is the inherent complexity of power grids, which are characterized by a large number of nodes and edges. The interdependence between nodes and the propagation of failures through the grid add another layer of complexity to the problem. GNNs have been shown to be effective in capturing the complex dependencies between nodes and predicting outcomes in graph-structured data.

In this project, we applied common types of GNNs that have been investigated in the context of power grids to predict the power outages at every node after one or multiple line failures in a power grid that can be considered large compared to the power grids used in the aforementioned studies.

[1] A load profile is a specific pattern of generation and consumption in a power grid. Thus different load profiles differ in how much power is generated and consumed at which location.

# 3 Theory

In this chapter, we will describe the theoretical foundations for the applied methodology, i.e., machine learning and Graph Neural Networks (GNNs), as well as the basics of the underlying power grid theory. In the first section (3.1), we will first explain the general concepts of machine learning and Neural Learning before introducing graphs and GNNs. In the second section (3.2), we will describe how AC power flow is modeled in a power grid. This will help in understanding the data used in the machine learning task.

## 3.1 Machine Learning

Machine learning is a part of artificial intelligence that utilizes data to improve performance in a given task [20]. It can solve tasks that are too complex for human instructions, relying on the computer to find and learn structures within the data. This process is generally divided into supervised and unsupervised learning. In supervised learning, the outcome of every training instance is known, whereas in unsupervised learning, it is unknown [2].

In this thesis, Graph Neural Networks (GNNs) are primarily applied in a supervised setting. Therefore, the following sections will focus on the necessary basics for supervised GNNs.

### 3.1.1 Neural Networks

Neural networks are a method used for supervised learning, aiming to approximate a function to predict labels based on input features [9]. The feature vector, denoted as $\boldsymbol{x}$, represents the properties of an instance (e.g., power and voltage at a power grid bus) and contains $N_f$ elements, where $N_f$ is the number of features. On the other hand, the label vector, denoted as $\boldsymbol{y}$, describes the features of the instance that need to be predicted (e.g., power outage and type of a bus) and consists of $N_l$ elements, where $N_l$ is the number of labels.

A neural network consists of layers of so-called neurons. Each neuron takes the feature vector as input and weighs the features using an $N_l \times N_f$ weight matrix $\boldsymbol{W}$, along with a bias $b$, and applies a non-linear activation function $\varphi$. The output $\hat{\boldsymbol{y}}$ is calculated as follows:

$$\hat{\boldsymbol{y}} = \varphi(\boldsymbol{W}\boldsymbol{x} + b),$$

as illustrated in Figure 3.1. This process is known as forward propagation.

The general idea behind training a neuron is to have it examine a training sample $\mathcal{S} = \boldsymbol{x}, \boldsymbol{y}$. After computing the output $\hat{\boldsymbol{y}}$, the next step involves comparing this output

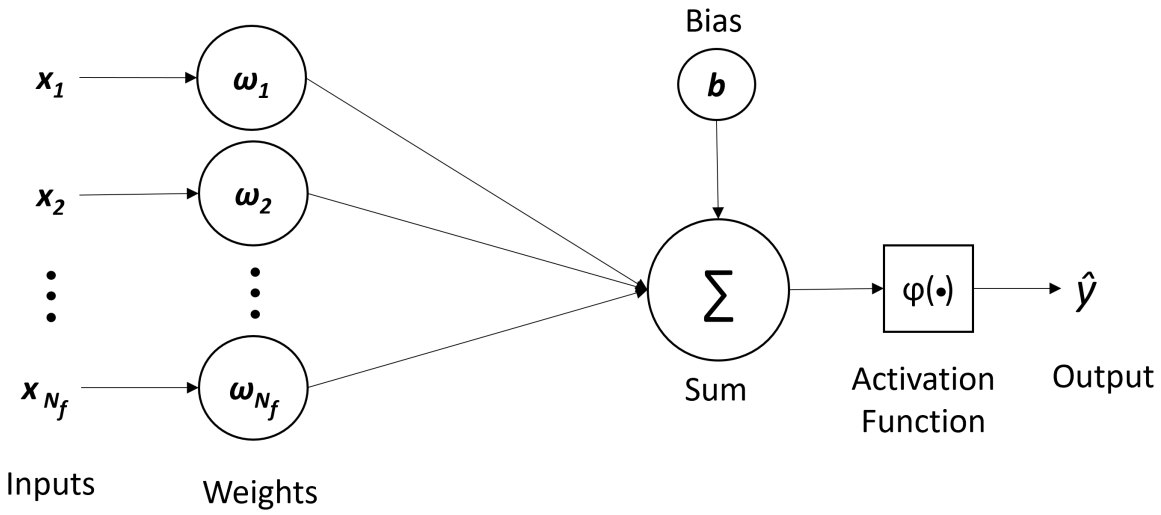



**Figure 3.1:** The figure represents a neuron used in an artificial neural network. The neuron computes the output $\hat{y}$ as a sum of the bias $b$ and the weighted inputs $x_i$ weighed by $w_i$.

to the label and calculating a loss $l(\boldsymbol{y}, \hat{\boldsymbol{y}})$ based on the similarity between the output and the label. Using this loss, the neuron will update its weights and bias. This process is known as backpropagation. By repeatedly performing these two steps, the output should gradually approach the label (see Section 3.1.2).

Since a single neuron lacks expressive power, it is common to create a neural network consisting of one or more layers with numerous artificial neurons that are interconnected (Figure 3.2). The output from one layer serves as the input to the next layer.

In a multilayer neural network, there are three types of layers. First, the input layer, where each neuron has $N_f$ inputs and $N_{f'}$ outputs. Next, the hidden layers, which have as many inputs as the previous layer had outputs. Finally, the output layer, which has $N_l$ outputs.

The purpose of the hidden layers is to learn what are known as hidden features $\boldsymbol{h}^{(l)}$. These hidden features represent a learned embedding of the input data, providing a more structured input for the subsequent layers. These embeddings can even be used to identify important structures within the data by analyzing the output of the hidden layers [9].

### 3.1.2 Training a Neural Network

As mentioned, a neural network is trained by defining a loss function $l(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x}))$, where $\boldsymbol{y}$ represents the labels and $\hat{\boldsymbol{y}}(\boldsymbol{x})$ is the output of the neural network. Common loss functions include the $L_1$ and $L_2$ loss, which are defined as follows:

$$L_1(y, \hat{\boldsymbol{y}}(\boldsymbol{x})) = \frac{1}{N}||\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{x})||_1, \tag{3.1}$$

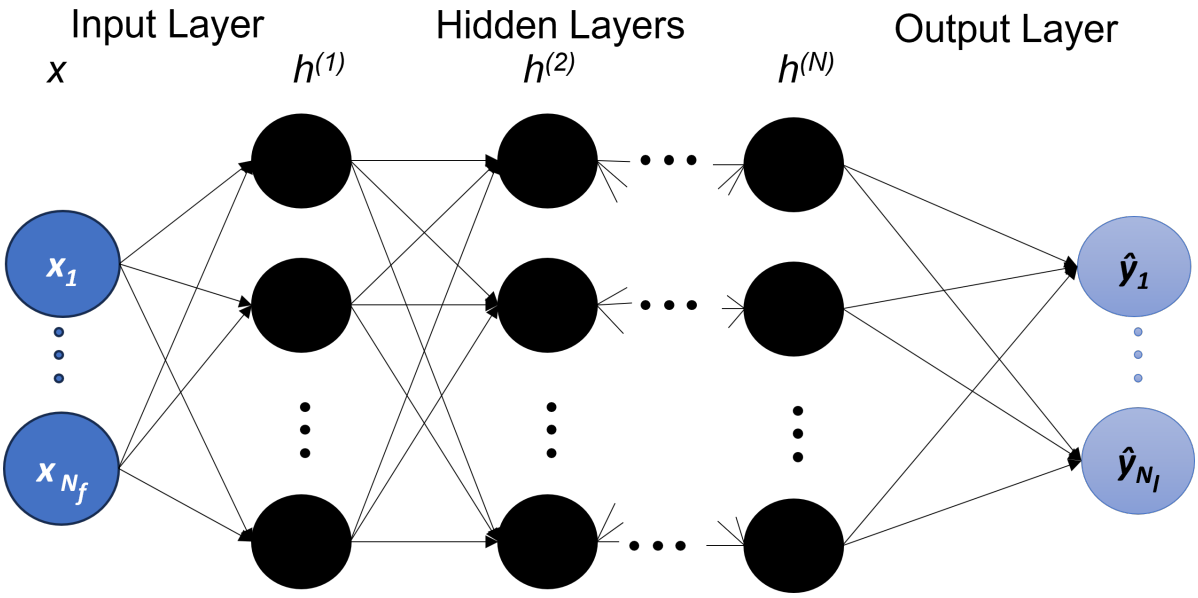



**Figure 3.2:** The figure shows a representation of a fully connected neural network with $N$ hidden layers.

$$L_2(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x})) = \frac{1}{N}||\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{x})||_2. \tag{3.2}$$

The $L_2$ loss is also known as Mean Squared Error (MSE).

To update the weights of the neurons with respect to the loss function, an algorithm called backpropagation is applied. In backpropagation, the derivative of the loss of each weight is calculated using the chain rule. This gradient is then used to update the respective weights. Usually, the gradient is multiplied by a factor to control the amount by which the update impacts the weight. This factor is called the Learning Rate and is a hyperparameter [1] for training the neural network. A larger learning rate can result in faster learning but may also lead to overshooting and unstable learning.

A frequently used form of backpropagation is Stochastic Gradient Descent (SGD), where instead of calculating the gradient of the whole dataset, the gradient is approximated using parts of the data (known as batches) and then backpropagated before processing the next batch. This approach usually allows for a much quicker convergence compared to calculating the full gradient [9].

### 3.1.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a multilayer neural network designed to learn from grid-like data, such as images. Similar to a regular neural network, it consists of fully connected layers, but it also contains convolutional and pooling layers.

[1] A hyperparameter is a configuration setting that is determined before the learning process begins and affects the performance and behavior of the learning algorithm.

A convolutional layer performs convolution operations on the input data using a set of learnable filters or kernels. Each filter extracts specific features or patterns from the input by sliding across the data and computing element-wise multiplications and summations. The output of each filter is called a feature map.

Pooling layers downsample the spatial dimensions of the input by aggregating information from local neighborhoods. The most commonly used pooling operation is max pooling, which selects the maximum value within each neighborhood. Pooling helps to reduce the computational complexity of the network, enhance translation invariance, and improve the network's robustness to variations in the input.

The combination of convolutional layers, pooling layers, and fully connected layers allows the CNN to learn very complex structures within grid-like data [9].

### 3.1.4 Graphs

Before introducing graph neural networks, it is useful to first define what a graph is and the nomenclature used to describe them. A graph consists of a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, where:

$$\begin{aligned} v_i \in v_1, ..., v_n &= \mathcal{V}, \\ e_i \in e_1, ..., e_m &= \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}, \end{aligned} \tag{3.3}$$

where we can also write the edges as $e_{jk}$ which implies that it connects the two nodes $v_j$ and $v_k$. Additionaly the edges of a graph can also be expressed via the $n \times n$ adjacency matrix $A$, where $A_{ij}$ is 1 if $e_{ij}$ exists and 0 otherwise. This representation allows us to describe a power grid (as described in section 3.2.1) by representing each bus in the power grid as a node $v_i$, and each power line as an edge $e_i$ . A graph can be denoted as the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Graph structured data forms the basis for graph neural networks to learn from and operate on.

### 3.1.5 Graph Neural Networks

Graph Neural Networks (GNNs) are neural networks specifically developed to handle graph-structured data. They can be used for various tasks, including node, edge, or graph regression or classification, meaning they can predict properties of individual nodes, edges, or the entire graph.

GNNs are an adaptation of convolutional neural networks (CNNs) to graph-structured data. Instead of regular convolutions that produce grid-like feature maps from grid-like data, GNNs use a neighborhood aggregation scheme to compute a representation vector for each node in the graph. The convolution on node $v_i$ typically aggregates its own node features $\boldsymbol{x}_i$ and the node features of neighboring nodes $\boldsymbol{x}_j$ with $v_j \in \mathcal{N}(v_i)$, where $\mathcal{N}(v_i)$ represents the set of neighboring nodes of $v_i$. In some GNNs, given edge weights $\boldsymbol{e}_{ij}$ are also aggregated when calculating the convolution of the nodes. This process of

information exchange between nodes is referred to as message passing since information is transferred by messages calculated from the node features and passed along the edges connecting the nodes.

A single layer of a GNN consists of $N_{f'}$ artificial neurons, where $N_{f'}$ is called the embedding size, representing the number of hidden features of each node $v_i$ after passing through the layer. The layer takes a graph $\mathcal{G}$ as input, where every node has a number of features $N_f$. The aggregation scheme is applied to calculate the embedding $\boldsymbol{h}_i$ of node $v_i$ at layer $l+1$ as follows:

$$\boldsymbol{h}_i^{(l+1)} = \varphi\left(\boldsymbol{h}_i^{(l)}, \bigoplus_{v_j \in \mathcal{N}(v_i)} \psi(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}, \boldsymbol{e}_{ij})\right),$$

where $\varphi$ is a non-linear activation function, $\psi$ is a learnable message function, and $\boldsymbol{h}_{i/j}^{(l)}$ is the embedding of node $v_i$ or $v_j$ at layer $l$, respectively.

Apart from message passing GNNs, which are also referred to as spatial GNNs, there are also the so-called spectral GNNs.

Spectral GNNs are an alternative approach to process graph-structured data by leveraging the spectral domain. Unlike message passing-based GNNs, spectral GNNs are based on the graph's spectral properties, particularly the eigenvalues and eigenvectors of the graph Laplacian matrix $L$, which is in its normalized form defined as:

$$\tilde{L} = \boldsymbol{I} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U,$$

where $I$ is the identity matrix, $A$ is the adjacency matrix of the graph with added self-connections, $D$ is the degree matrix of $A$, and $U$ and $\Lambda$ are the eigenvector and eigenvalue matrices, respectively.

The graph Laplacian provides crucial information about the graph's structure and connectivity. One of the most prominent spectral GNNs is the Graph Convolutional Network (GCN) [15]. The main idea behind GCN is to learn node representations by utilizing localized spectral filters.

A single layer of GCN can be expressed as follows:

$$\boldsymbol{h}^{(l+1)} = \varphi(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \boldsymbol{h}^{(l)} \boldsymbol{W}),$$

where $h^{(l)}$ represents the node feature matrix at layer $l$, $\boldsymbol{W}$ denotes the learnable weight matrix, and $\varphi$ represents the activation function.

## 3.2 AC Power Flow

To understand the task at hand, particularly the underlying data and the AC power flow simulations in a power grid, we will introduce the necessary theory. In the first section (3.2.1), we will provide a concise overview of the functionality and components of a power grid. Subsequently, in section (3.2.2), we will briefly introduce the basic equations required for simulating a power grid. For a more in-depth explanation, you can refer to [35].

### 3.2.1 Power Grid Modelling

A power grid facilitates the distribution of power generated by power plants, solar panels, and wind turbines to consumers, ranging from small households to large factories.

Due to the efficient operation of different components at various voltage levels, the power grid is separated into the transmission and distribution grid, each with different voltage levels. The distribution grid is the grid supplying the consumers with power from the transmission grid and usually has voltages between 220 V and 220 kV. The transmission grid operates at higher voltages (220 kV 380 kV) and is interconnecting the generators and transmitting the power to areas where no power is generated. In this report we will focus on the transmission grid.

The transmission grid can be visualized as a complex network of buses interconnected by power lines. Each bus can have multiple components, such as generators, loads, and shunts. Generators inject power into the grid, while loads consume power from the network. Shunts, such as capacitors or inductors, serve various purposes, including protecting the grid in case of component malfunctions. Depending on whether a bus predominantly consumes or generates power, it will act as a sink or source for the grid. Power lines interconnect the buses, serving as points of redistribution for incoming and outgoing power.

This complex network can be intuitively represented as a graph, where the nodes $\mathcal{V}$ are the buses, and the edges $\mathcal{E}$ are the transmission lines (as explained in section 3.1.4).

When simulating power grids, the goal is to calculate the voltages and power flows in the network, taking into account the consumption and generation at the nodes. This problem is subject to constraints on the components, such as maximum power transmission capacity and maximum permissible voltages.

Power grids are generally operated in AC (alternating current) because it allows for the use of transformers to easily change voltage levels. This feature enables increased efficiency when transporting power over long distances, as transmission line losses are proportional to the current, which decreases with higher voltages.

While DC power grid simulations are simpler to solve and require less computational power and time, AC simulations are preferred for more realistic results, as utilized in this thesis [8].

### 3.2.2 Network Equations

Compared to a DC circuit, where voltages and currents remain constant, they oscillate sinusoidally over time in an AC circuit. The separate oscillations of voltage $V$ and current $I$ are out of phase. This can be expressed as

$$V(t) = |V|e^{i(\omega t+\Theta_V)}, \tag{3.4}$$

$$I(t) = |I|e^{i(\omega t+\Theta_I)}, \tag{3.5}$$

where $t$ represents time, $|V|$ and $|I|$ are the amplitudes of voltage and current, respectively, $i$ is the imaginary unit, $\omega$ is the grid frequency, and $\Theta_V$ and $\Theta_I$ are the phases

of voltage and current, respectively [8]. The grid frequency is a fixed parameter for the entire grid (e.g., 50 Hz in the European grid [1]) and determines the frequency of voltage and current oscillations. The voltage and current phases define the angle between the voltage and current oscillations. Given the voltage and current, we can calculate the apparent power as

$$S = VI = |V||I|e^{i(2\omega t+\Theta)}, \tag{3.6}$$

where we combined $\Theta = \Theta_V + \Theta_I$.

Usually, this expression of complex power is split into its real and imaginary parts as

$$S = P + iQ, \tag{3.7}$$

where $P = \mathrm{Re}(S)$ is called active power, and $Q = \mathrm{Im}(S)$ is reactive power.

To effectively model the power flow in a power grid, it is necessary to consider the power, voltage, current, and other relevant parameters at all components of the grid. In AC power flow, the relationship between current $I$ and voltage $V$ is described by the admittance $Y$ as $I = YV$ [8]. The admittance is the reciprocal of impedance $Z$, which can be understood as the complex resistance for AC current. To establish this relationship, we utilize the $N_b \times N_b$ admittance matrix $Y_{bus}$ in a system with $N_b$ buses. This matrix defines the admittance between all buses, where $Y_{ij}$ represents the admittance between bus $i$ and bus $j$.The current injections at all buses, denoted as $I_{bus}$, can be related to the voltages $V$ using the equation:

$$I_{bus} = Y_{bus}V. \tag{3.8}$$

To calculate the current injections at the branches, we distinguish between the so-called "to" and "from" buses of a branch, where the "to" bus is the bus where the current is flowing to, and the "from" bus is the bus where the current is flowing from. The currents are then calculated as:

$$I_f = Y_f V, \tag{3.9}$$
$$I_t = Y_t V \tag{3.10}$$

In a system with $N_b$ buses and $N_l$ branches, $Y_t$ and $Y_f$ are $N_b \times N_l$ matrices. The distinction between $Y_{bus}$, $Y_t$, and $Y_f$ lies in the fact that $Y_{bus}$ (as explained earlier) contains the admittance values for the connections between buses, whereas $Y_t$ and $Y_f$ contain the admittances between the branches and the "to" or "from" buses, respectively.

We can derive the nodal admittance matrix $Y_{bus}$ by combining the branch admittance matrices and accounting for the admittances of the shunt elements as follows:

$$Y_{bus} = C_f^T Y_f + C_t^T Y_t + Y_{sh} \tag{3.11}$$

Here, $C_f$ and $C_t$ are $N_l \times N_b$ matrices, where $C_f(ij)$ and $C_t(ik)$ are 1 if branch $i$ connects buses $j$ and $k$, respectively. $Y_{sh}$ is a $N_b \times N_b$ admittance matrix, where $Y_{ij}$ contains the admittance of the shunt element connecting buses $i$ and $j$. If there is no shunt element connecting the buses, $Y_{ij}$ is zero.

Using the expression for the nodal admittance matrix, we can calculate the apparent power at the buses, which matches the injections by the branches as a function of voltage:

$$S_{bus}(V) = \text{Diag}(V)Y_{bus}^{*}V^{*}. \tag{3.12}$$

In addition to the circulating current, we need to model the injections by generators $S_g$ and the consumed power by loads $S_d$. The power injected into the system by the generators can be intuitively expressed as:

$$S_{g,bus} = C_g S_g, \tag{3.13}$$

where $S_g$ is a vector with $N_g$ elements in a network with $N_g$ generators. $C_g$ is an $N_b \times N_g$ matrix containing the connections of the generators to the buses, where element $(i, j)$ of $C_g$ is 1 if generator $j$ is connected to bus $i$.

The loads can be considered properties of the buses. Therefore, $S_d$ simply becomes a vector with $N_b$ elements, where the $i$th element contains the total load at bus $i$. It is important to note that $S_g$ and $S_d$ are independent of $V$ since generators have a fixed voltage that defines the voltage at the connected bus, making it a so-called PV-bus. At a PV bus, the active power $P$ and the voltage $V$ are fixed. In contrast, all other buses are called PQ-buses (implying that the active power $P$ and reactive power $Q$ are fixed). This explains why $S_{bus}(V)$ is a function of voltage, while $S_d$ remains independent of $V$ due to the constant demand.

The three types of power injection, consumption, and transmission are combined in the network equation:

$$g_S(V, S_g) = S_{bus} + S_d - C_g S_g = 0. \tag{3.14}$$

Solving this equation in matrix form will result in a steady state defining all physical properties at the transmission lines and buses. The model used to solve this equation is the AC-CFM model [23], which will be explained in more detail in Section 4.1.4.

# 4 Methodology

In this chapter we will describe in the first section 4.1 the raw data used and the several steps taken to create the datasets which are then used in the machine learning task. In the second section 4.2 we will give an overview over the compared architectures and baselines, before introducing investigated loss functions in section 4.3. Finally we show the procedure used for the hyperparameter studies and cross-validation in sections 4.4 and 4.5.

## 4.1 Data

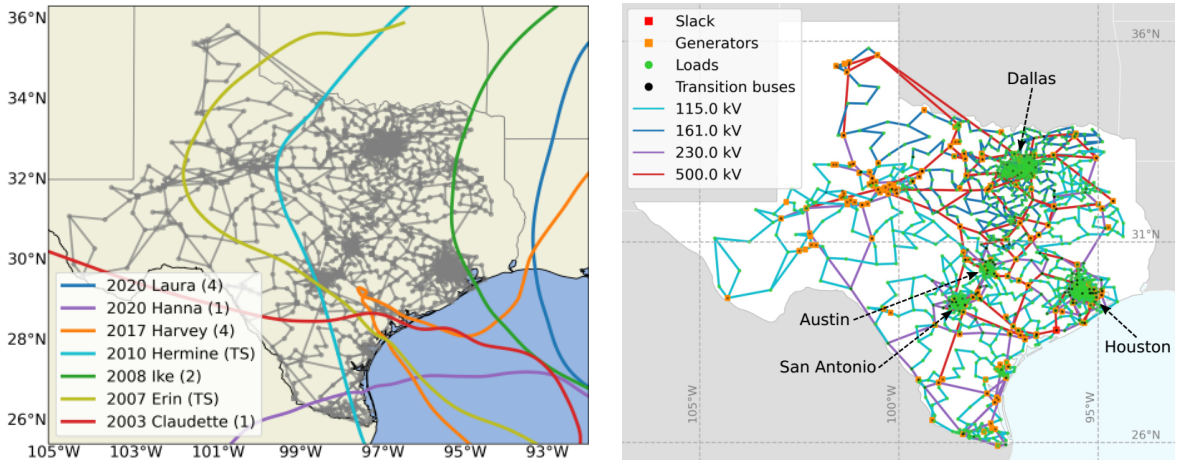
In this section, we will first introduce the raw storm and power grid data in sections 4.1.1 and 4.1.2. We will then explain the wind-induced damage model as well as the used power flow model in sections 4.1.3 and 4.1.4. From there, we will continue with the necessary data processing and feature selection, as well as the resulting datasets in sections 4.1.6 and 4.1.7. Finally, we will describe the applied normalization procedure in section 4.1.8.

### 4.1.1 Wind Data

The primary damages considered in this thesis are the failed transmission lines due to the wind, resulting from simulations using the wind data of 7 hurricanes (namely Claudette, Erin, Hanna, Harvey, Hermine, Ike, and Laura), that passed over Texas between 2001 and 2020. The hurricane center coordinates and maximum sustained wind speeds are obtained from the International Best Track Archive for Climate Stewardship (IBTrACS) [16]. This data is then utilized in CLIMADA [3], which calculates the hurricane's time series with a temporal resolution of 5 minutes and a spatial resolution of 0.1°. Subsequently, the wind speeds obtained are used to determine the wind load on transmission lines along the hurricane's path, allowing for the assessment of transmission failure probabilities. The trajectories of the storms can be seen in figure 4.1a.

### 4.1.2 Power Grid Data

The data utilized for the power grid is derived from the ACTIVSg2000 dataset [28]. This freely accessible synthetic data represents the power grid of Texas and comprises over 2000 buses, in excess of 2345 transmission lines, 771 transformers, and an extensive set of parameters such as branch capacities or upper and lower bounds for active and reactive power. A visualization of the power grid can be seen in figure 4.1b.




**(a)** **(b)**

**Figure 4.1:** Figure 4.1a shows the trajectories of the seven Hurricanes, of which the wind data has been used to calculate the sequences of destroyed lines in the power grid. The number indicates the strength of the hurricane according to the Saffir-Simpson hurricane scale. 'TS' implies that the storm is a Tropical Storm (instead of a Hurricane). Figure 4.1b shows the plot of the power grid data with the different base voltage levels and bus types. (Figures: Julian Stürmer [27])

### 4.1.3 Wind Induced Damage Model

The utilized probabilistic model for considering wind-induced damages shares similarities with the fragility model introduced by Winkler et al. [31]. This fragility model assesses the probability of a transmission line segment $k$ breaking within the time interval $\tau_i = [t_i, t_i + \Delta t)$ and is defined as follows:

$$p_k(v, l) = \min\left(\frac{F_{wind,k}(v, l)}{F_{brk}} r_{brk} \Delta t, 1\right), \tag{4.1}$$

where, $F_{wind,k}(v, l)$ represents the wind force at segment $k$ recorded at the beginning of the time step. $F_{brk}$ signifies the maximum permissible perpendicular force that a segment can withstand, as indicated by the employed dataset (refer to section 4.1.2). $r^{brk}$ denotes the collapse rate, which is defined as the reciprocal of the "time to failure", reflecting the speed at which a transmission line segment deteriorates once the wind load reaches $F_{brk}$. The "time to failure" is chosen so that the damages per unit time align with the historical hurricane data. Furthermore, $l$ represents the length of segment $k$, which is the distance between two transmission towers. This is uniformly set to 161 meters for all segments [1]. Lastly, $v = v(t, k)$ denotes the local 3-second gust wind speed at segment $k$ during time step $\tau_i$, derived from the 10-minute sustained wind speed.

[1] Note that a transmission line usually consists of multiple transmission line segments thus the probability of a line to collapse still depends on the length of the line via its number of segments.

### 4.1.4 AC Power Flow Model

The power flow simulations used to create the initial data were performed using the AC Cascading Failure Model (AC-CFM) [23]. This model utilizes MATPOWER [35] to solve the AC power flow network equation 3.14.

AC-CFM has been validated following the suggested approach of the IEEE PES group on cascading failures [4], demonstrating stability even for large grids with more than 2000 buses. The model employs a recursive approach to identify causalities among generations within the cascading failures of the system, which can occur due to the propagation of overloads or insufficient power supply after the initial damages.

The model incorporates various protection mechanisms that, in addition to primary damages, can result in isolating parts of the system when constraints cannot be met. This isolation is referred to as *islanding*, and the isolated parts are known as *islands*. Each island that emerges due to the implemented protection mechanisms is treated separately. The protection mechanisms are recursively applied to each island until their conditions meet the specified tolerances or until the entire island is disconnected from the system.

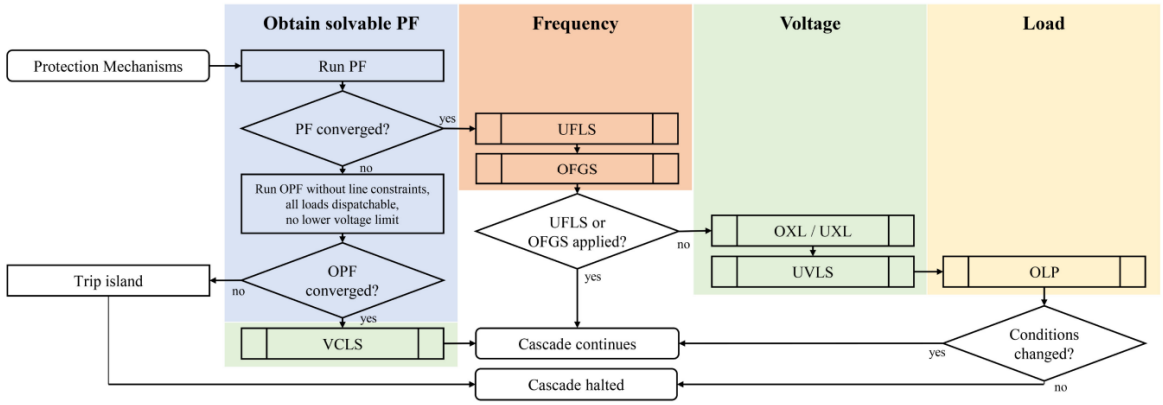



**Figure 4.2:** The flowchart of the recursive application of the different protection mechanisms of the AC-CFM algorithm [23] is shown (Figure: [23]).

The implemented protection mechanisms, in the order of application, include Under-/Over-Frequency Load-/Generator-Shedding (UFLS/OFGS), Excitation Limiters (U/OXL), Under Voltage Load Shedding (UVLS), Overload Protection (OLP), and Voltage Collapse Load Shedding (VCLS) [23]. Figure 4.2 illustrates a flowchart depicting these protection systems. The purpose of these mechanisms is to enable the system to restore solvable conditions (primarily through load shedding) when the power flow does not converge, rather than assuming a complete blackout.

VCLS is activated when the power flow fails to converge due to a significantly higher demand for reactive power than the available generation. In such cases, all loads are dispatched and an Optimal Power Flow (OPF) solver is utilized. The OPF solver determines the minimum load shedding required to bring the power flow back to a solvable

region. If the OPF solver fails to converge, it could indicate either physical limitations leading to a blackout or a mathematical issue with non-convergence.

Under Frequency Load Shedding or Over Frequency Generator Shedding is triggered when there is a discrepancy between the mechanical frequency and electrical frequency, occurring when loads change more rapidly than the generators can adapt. In such situations, the same percentage of loads is shed at each bus, or generators are shed starting from the smallest generator.

Excitation limiters (U/OXL) come into play when the reactive power demand of a generator surpasses its specified limit. The excitation voltage of the generator is adjusted by transforming the PV bus (generator) into a PQ bus (load). The relative power output is fixed to the upper (or lower) relative power limit, and the voltage is adjusted accordingly.

Subsequently, Under Voltage Load Shed (UVLS) is applied if the voltage at a bus falls below the lower limit. The demand is gradually reduced until the voltage returns within the specified limits.

Finally, if a transmission line exceeds its specified rating, it is tripped by Over Load Protection (OLP).

Please refer to Figure 4.2 for a visual representation of the recursive application of the different protection mechanisms in the AC-CFM algorithm [23].

### 4.1.5 Generated Data

Using the wind-induced damage model (section 4.1.3), we conducted Monte Carlo simulations to create 10,000 scenarios of subsequently broken lines for each hurricane in the set. Each scenario has a number of steps with broken lines in the order of 10 (usually between 30 and 70). These scenarios, along with the power grid data (section 4.1.2), were fed into AC-CFM to create the power flow solutions after destroying each line, resulting in a number of power grid states in the order of a few million (7 storms $\times$ 10,000 scenarios $\times$ $\sim$50 steps $\approx 3,500,000$), which are available as training data.

One might question why we put effort into simulating the destroyed lines during the storms but did not utilize the time series properties in the machine learning task. The reason is that, from the machine learning perspective, the goal (as described in the introduction) is to develop a model that can predict nodal power outages in the grid for any destroyed line from any state of the grid. Since these simulations have already been performed in a prior project [24] and are computationally relatively expensive, we decided to use this existing data instead of creating a new, strategically more fitting dataset.

### 4.1.6 Data Processing

The data produced by AC-CFM contains a vast amount of information from which the relevant features must be selected. Since we are dealing with power outages in an AC power flow setting, the chosen features include *Apparent Power Demand* and *Voltage*

*Magnitude* as node features, and *Active Power Flow* and *Reactive Power Flow* , *Status*, *Resistance*, *Reactance*, *Capacity*, and *Initial Damage* as edge features.

The *Status* feature indicates whether the line is working (1) or broken (0), while *Initial Damage* is only 1 for the line that was the initial contingency (broken by the wind) at that step. This differentiation is necessary as the data contains states of the grid with pre-existing broken lines, which must be distinguished from the lines broken by the wind in each step.

Finally the node label is the *Power Outage* of each node after that step. In conclusion this sums up to two node features, seven edge features and one node label. These features have been chosen as they are intuitively the most relevant features to the calculation of the power flow as well to the limitations of the system indicating that a failure will happen if these limits are not met.

### Treatment of Parallel Transmission Lines

Since there are parallel power lines connecting the same nodes we need to combine their attributes according to the underlying physical laws in order to have only singular (bidirectional) edges between each connected pair of nodes with physically correct edge features.

The features *Status* and *Initial Damage* are combined by considering if one of the parallel lines is destroyed, then all parallel lines are also counted as destroyed, which is the same way the initial modelling process treats parallel lines.

For a lines *Capacity* $C$ (maximum power the line can transfer without being damaged), the *Active Power* $P$ and the *Reactive Power* $Q$, the values are simply added [8] so that

$$C_{tot} = \sum_{i}^{N_{parallel}} C_i, \tag{4.2}$$

$$P_{tot} = \sum_{i}^{N_{parallel}} P_i, \tag{4.3}$$

$$Q_{tot} = \sum_{i}^{N_{parallel}} Q_i, \tag{4.4}$$

where $N_{parallel}$ is the number of parallel lines.

Lastly for the *Resistance* and *Reactance* we use some fundamental laws of AC power flow to calculate the total *Resistance* and *Reactance* of two parallel transmission lines. In AC power flow resistance $R$ and Reactance $X$ as well as Conductance $G$ and Susceptance $B$ are linked through Impedance $Z$ and Admittance $Y$ as [8]

$$Y = \frac{1}{Z} = \frac{1}{R + iX} = G + iB \tag{4.5}$$

where $i$ is the imaginary number. Furthermore the Admittance of two parallel lines is additive. Thus

$$Y_{tot} = Y_1 + Y_2 = \frac{1}{R_1 + iX_1} + \frac{1}{R_2 + iX_2} = \frac{R_1 + R_2 + i(X_1 + X_2)}{(R_1 + iX_1)(R_2 + iX_2)} \tag{4.6}$$

holds. Splitting equation 4.5 into its real and imaginary part results in the expressions

$$GX^2 + GR^2 - R = 0 \tag{4.7}$$

and

$$BX^2 - X + BR^2 = 0, \tag{4.8}$$

from which we can easily express $R$ and $X$ in terms of $G$ and $B$ as

$$R = \frac{G}{G^2 + B^2} \tag{4.9}$$

and

$$X = \frac{B}{G^2 + B^2}. \tag{4.10}$$

Keeping in mind that $Y = G + iB$, we can rearrange eq. 4.6 by multiplying with the complex conjugate of the denominator to express $G$ and $B$ through $R$ and $X$ as

$$G = \frac{R_1R_2^2 - R_1X_2^2 + R_2R_1^2 - R_2X_1^2}{(R_1^2 - X_1^2)(R_2^2 - X_2^2)} \tag{4.11}$$

and

$$B = \frac{X_1X_2^2 - X_1R_2^2 - X_2R_1^2 + X_2X_1^2}{(R_1^2 - X_1^2)(R_2^2 - X_2^2)}. \tag{4.12}$$

Now we only need to plug in equations 4.11 and 4.12 into equations 4.9 and 4.10 and we arrive at the *Resistance* and *Reactance* of two parallel lines.

After removing parallel lines in this way the resulting graph for the learning task has 2000 nodes with 2 node features and one node label each and 2667 edges. To account for both flow directions which can have slightly different values, every edge is split into two one directional edges leading to 5334 one directional edges with 7 edge features each.

### Treatment of Inactive Lines

During data generation it can appear that power lines are set to inactive (*Status*=0) but the other features are not set to physically meaningful values since the power flow model would disregard these values anyways as the line is inactive. To make learning a bit easier we change the values for the features if the line is inactive to more meaningful values.

Specifically the *Capacity*, *Active-* and *Reactive Power Flow* are set to 0 and *Resistance* and *Reactance* are set to 1 for inactive lines.

### 4.1.7 Resulting Datasets

As explained in 4.1.5, the raw data generated consists of a large set of a few million training instances. However, a significant portion of these instances contains steps in which no load shed or further damages occur. In fact, only 14% of the instances contain any load shed at all, and most of these instances have only small load shed at a few nodes. To address this bias towards no load shed, a subset of the data was created by limiting the instances with little or no load shed.

To create this subset, instances with more than 10% load shed were kept, as they represent significant damage. For instances with less than 10% load shed, instances were randomly selected until they represent 25% of the dataset (figure 4.3a). This process was done separately for each hurricane to maintain the proportions of the different storms in the full dataset. Since each hurricane has a different impact on the power grid, not selecting instances separately for each storm could lead to the overrepresentation of a weaker storm in the low damage instances.

While limiting the low damage instances may seem like a loss of useful data, the distribution of nodal outages (figure 4.3b) shows that the large majority of nodes still have close to 0 or 0 outage. This is because, especially for lower total load sheds in the grid, many nodes remain untouched even if something happens in the grid.

The final subset consists of 17,578 instances and will be referred to as the *full set*. This subset is used for training and evaluation in the machine learning task.

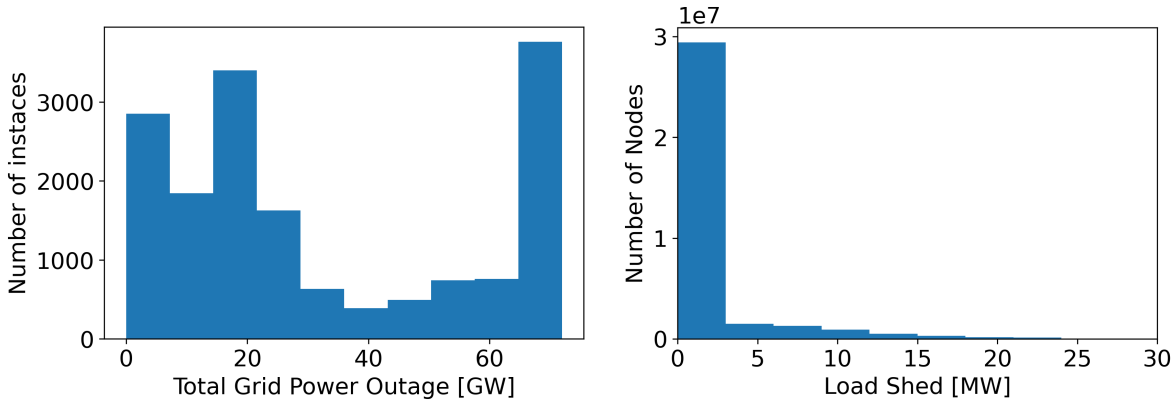



**Figure 4.3:** Figure 4.3a shows the distribution of total outage in the grid for each step of the used data set. Figure 4.3b shows the distribution of the node label (power outage at each node) of the used data set.

Additionally to the *full set*, which includes data from all the storms, we produced a different subset that excludes data from the storm 'Claudette'. This subset has 16492 instances. The first set is intended to be used for the cross-validation (section 4.5) and the second set for the hyperparameter studies (section 4.4).

We decided to split the datasets by hurricane to ensure that the data in every set is independent of the data of the other sets. At the same tim it is a more intuitive and arguably a more realistic way to split the data as in a potential use case of trying to predict the power outage of a Hurricane one would have trained the model on data of different hurricanes trying to predict the outage of a separate hurricane.

### 4.1.8 Normalization

To normalize the features, two methods are used: standardization and logarithmic normalization. The standardization is followed by normalization to ensure that all features are within the interval $[0, 1]$.

For standardization, the following equation is applied to each feature $j$ of every node $i$:

$$|x_{ij}| = \frac{1}{|\max_i(x_{ij})|} \frac{x_{ij} - \bar{x}j}{\sigma_j}, \tag{4.13}$$

where $\bar{x}j$ is the mean of feature $j$ over all nodes $i$, $\sigma_j$ is the standard deviation of feature $j$, and $|\max_i(xij)| = \frac{\max_i(xij) - \bar{x}_j}{\sigma_j}$ is the standardized maximum of feature $j$. The equation for logarithmic normalization is

$$|x_{ij}| = \frac{\log(x_{ij} + c)}{\log(\max_i(x_{ij}) + c)} \tag{4.14}$$

where $c$ is a constant added to prevent undefined values encountered by $x_{ij} <= 0$.

The decision of which method to apply to each feature is based on the skewness of the initial distribution. The goal is to achieve distributions that are as close as possible to the normal distribution for all features. This normalization process ensures that all features are on a comparable scale, which can improve the performance of machine learning models.

As can be seen in Figure 4.4, the only features that are not skewed are the active and reactive power flow (Figures 4.4c and 4.4d). Therefore, standardization is applied to both active and reactive power flow in the edge features. Since the status and the initial condition can only take the values 0 or 1, no normalization is needed, and the same applies to all other features as well as the node labels (Figure 4.3b). Logarithmic normalization has been applied to the rest of the features. The resulting normalized feature distributions can be seen in Figure 4.5.

Normalization has been applied in a way that the maximum values and standard deviations used for normalization were calculated only from the training data. This ensures that the validation and testing data are also normalized using the values from the training data, creating a more realistic scenario where a pre-trained model should be able to predict and infer from values outside the range it has trained on, and also outside the interval [0,1].

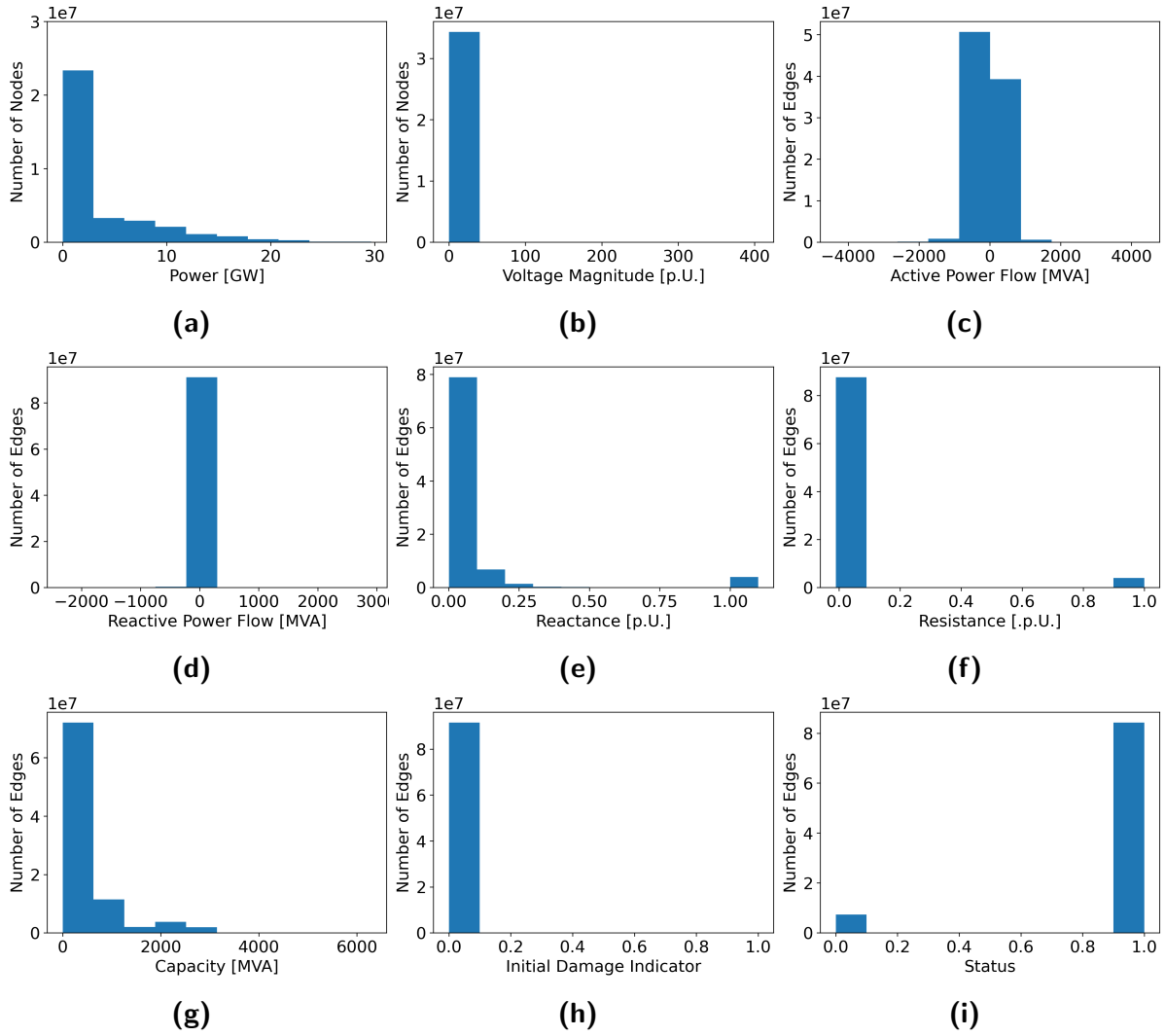



**Figure 4.4:** The figures show the distributions of the unnormalized node and edge features of the dataset using all the storms, as described in section 4.1.7. Figures 4.4a and 4.4b are node features, and figures 4.4c to 4.4i are edge features. Note that the x-limits indicate the existence of values up to this range, but they are too few to be seen.

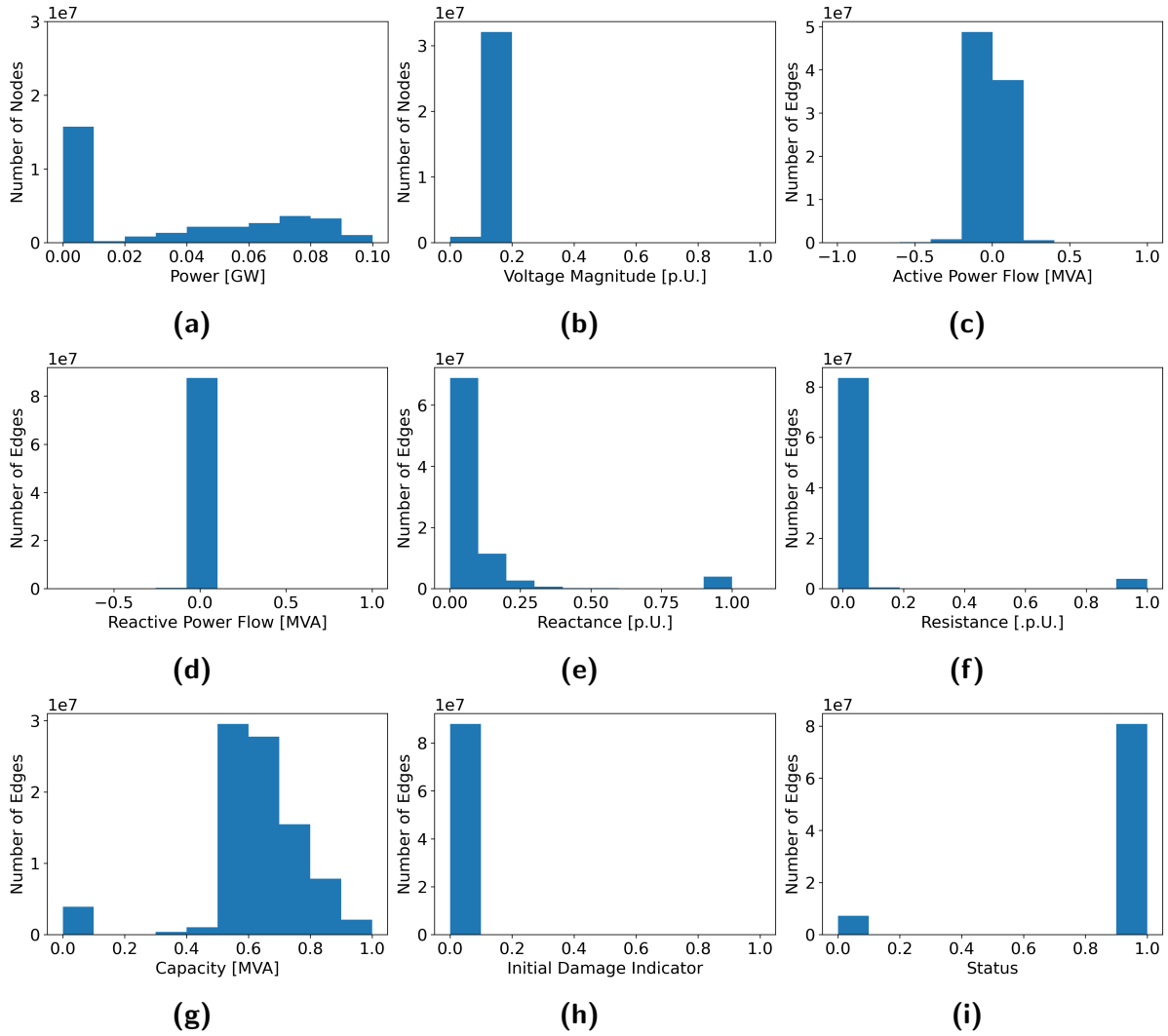



**Figure 4.5:** The figures show the distributions of the normalized node and edge features of the dataset using all the storms, as described in section 4.1.7. Figures 4.5a and 4.5b represent node features, while figures 4.5c to 4.5i represent edge features. The active and reactive power flow (Figures 4.5c and 4.5d) have been normalized using standardization with consecutive normalization, whereas the rest have been normalized with logarithmic normalization with $c = 1$, except for status and initial damage indicator (Figures 4.5h and 4.5i), which have not been normalized due to their binary nature. Zoomed-in plots of the distributions can be seen in Figure 7.2

## 4.2 Models

In this section, we will introduce the studied architectures of the investigated models, as well as the used baselines. All the models presented in this thesis are implemented using PyTorch 1.9.1 [25] and PyTorch Geometric 2.0.3 [7]. The version of Python used is 3.9.12. All applied layer types utilize a Leaky Rectified Linear Unit with a gradient of -0.02 for the negative slope, serving as the activation function.

### 4.2.1 Baselines

To gauge the effectiveness of the applied graph neural network architectures, we compare the results with four simple baselines, which we describe in the following sections. The chosen baselines are as follows:

- Mean Baseline: This baseline always predicts the mean power outage for each node.
- Node-only Multi-Layer Perceptron (MLP): A multi-layer perceptron that solely utilizes the node features.
- Node2Vec-Enhanced Multi-Layer Perceptron (MLP): A multi-layer perceptron that incorporates both the node features and an embedding created by the Node2Vec algorithm.
- Node-wise Ridge Regression: This baseline involves training a separate model for each node using ridge regression.

We opted for these baselines since, to the best of our knowledge, no benchmark models exist for predicting cascading failure outages. These baselines offer straightforward and intuitive benchmark results for comparison.

#### Node Mean

The Node Mean baseline always predicts the node-wise mean power outage for each node. Therefore, the output is calculated as follows:

$$\hat{y}(v) = \frac{1}{N_{\text{instances}}} \sum_{i=1}^{N_{\text{instances}}} y(v_i), \tag{4.15}$$

where $v$ represents node $v \in V_{\text{Training Set}}$, $N_{\text{instances}}$ is the number of instances in the training set, and $y(v_i)$ is the node label of node $v$ for instance $i$.

We specifically chose the node mean over the total mean since it appears feasible for network operators to calculate the nodal mean outages. Thus, to achieve better performance than the state-of-the-art possibilities, using the node mean serves as a more reasonable minimum baseline compared to outperforming the total mean outage.

### Multi Layer Perceptron

The multilayer perceptron (MLP) used as a baseline consists of $N$ layers, with each layer containing $D$ neurons. Each linear layer is followed by batch normalization. Additionally, a dropout is applied before feeding the output into the next layer, and a skip connection is utilized. The skip connection takes the mean of the output of the full previous layer and the output after the batch normalization of the following layer. Figure 4.6 provides a visualization of this architecture.

The batch normalization, dropout, and skip connections are all optional but are applied consistently for either all layers or none of them. These components are subject to the hyperparameter studies explained in section 4.4.

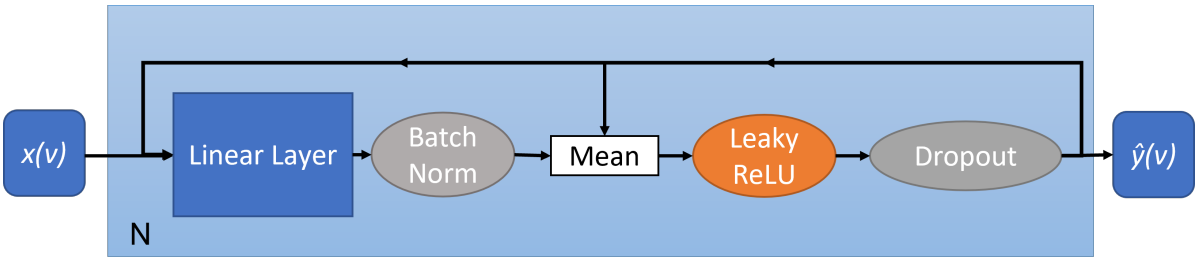



**Figure 4.6:** Representation of the applied MLP architecture. The batch normalization, skip connections, and dropout layers are all optional and are subject to the applied hyperparameter studies explained in section 4.4. $\boldsymbol{x}_v$ represents the node features of one node, which are used as input for the model.

This multilayer perceptron architecture is part of two of the used baselines. The first baseline is applied only to the node features of individual nodes. The second baseline is trained using both the node features and node embeddings created by the Node2Vec algorithm [10], as implemented in PyTorch Geometric. The parameters used are shown in table 4.1. Node2Vec is a graph embedding technique that learns representations for nodes in a network. It achieves this by optimizing random walks to balance between capturing local neighborhood structures and exploring the broader graph connectivity. Contrary to Graph Neural Networks it does not make use of any features and thus represents purely the topology of the graph.

### Ridge Regression

The last baseline is a ridge regression that takes the node features of all $n = 2000$ nodes of one graph as input simultaneously. Therefore, the total number of input features amounts to $N_{f,\text{in}} = N_f \cdot n = 4000$, where $N_f = 2$ is the number of features per node.

This baseline has been chosen to assess whether a single, very simple model per node can achieve better performance than the other approaches involving more complex models applied to arbitrary nodes.

**Table 4.1:** Parameters used to create the Node2Vec embedding.

| Parameter | Value |
|---|---|
| Embedding Dimension | 20 |
| Walk Length | 20 |
| Walks per Node | 1 |
| Context Size | 2 |
| Number of Negative Samples | 2 |
| p | 1 |
| q | 1.75 |

### 4.2.2 Graph Neural Network Architectures

In the following paragraphs, the studied graph neural network architectures are explained. The specific layer types have been selected as they delivered good results for comparable tasks or have intuitively good properties for the task at hand. All layer types that can not process at least one edge feature have been eliminated as the cascading failures are triggered by the failure of a single line, which we deem a crucial information to the GNN.

#### Graph Isomorphism Network with Edge Features

The Graph Isomorphism Network (GIN) has demonstrated promising results on a task similar to ours [13]. The paper originally used the GIN layer [32], which cannot process edge features. However, since the GINE version was introduced in [12], we opted to use this layer type, as it allows for the inclusion of edge features. A more detailed explanation of the GINE layer can be found in section 7.2.

The applied Graph Isomorphism Network architecture uses the PyTorch GINE implementation. Each convolutional (GINE) layer consists of two linear layers with 32 hidden features, with batch normalization applied to the first layer before activation. A visualization of this GINE layer can be seen in figure 4.7a.

The convolutional layers, built as described, are then combined into a graph neural network with $N$ layers. These layers take both the node and edge features as input and use a rectified linear unit as the activation function. An optional dropout between the layers is implemented to prevent overfitting. The convolutional layers can be used with skip connections, which feed the mean of the output of layer $i$ and the output after activation function and dropout of layer $i-1$. The output of the last convolutional layer is fed into the regression head, consisting of $M+1$ linear layers with a rectified linear unit as the activation function. The last linear layer, without an activation function, produces the final output. A visualization of the network can be seen in figure 4.7b.

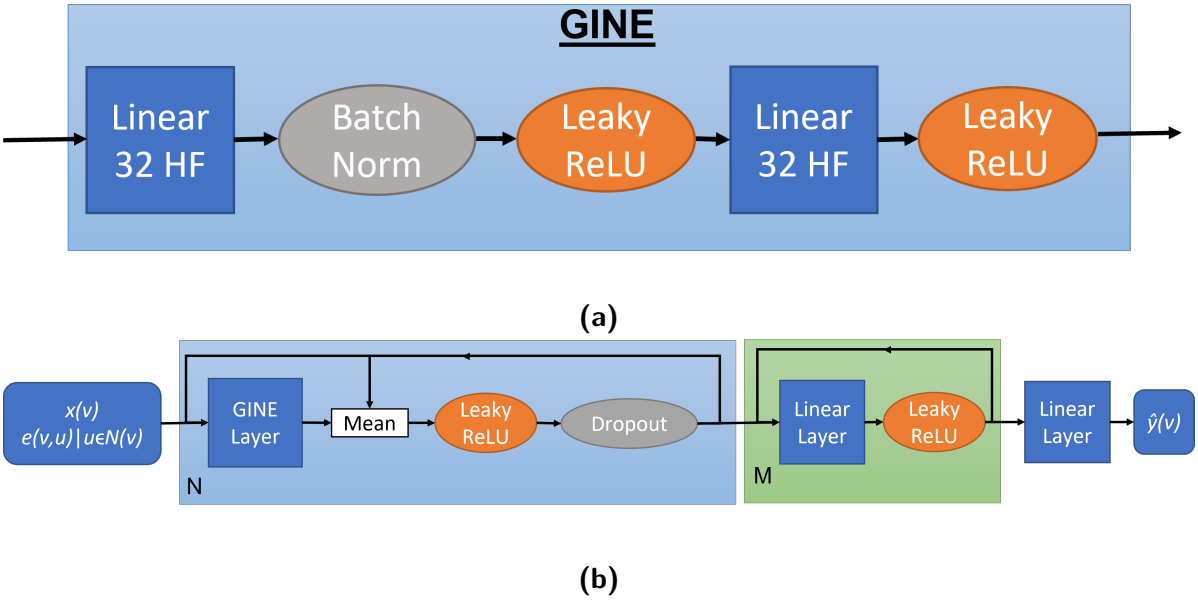



**Figure 4.7:** Figure 4.7a illustrates the architecture of a single GINE layer as described in 7.2 where the layer learns the functions $f$ and $\phi$. Both inner linear layers have 32 hidden features. Figure 4.7b shows the architecture built from the single GINE layers. The architecture is split into the convolutional layers, which repeat $N$ times and can be used with a skip connection with mean aggregation and the regression head which consists of $M+1$ linear layers. The skip connections and the dropout are both optional and subject to the hyperparameter studies as explained in section 4.4.

### Graph Attention Network and Topology Adaptive Graph Convolutional Network

The GAT layer's attention mechanism intuitively possesses advantageous properties, allowing nodes to weigh information from different neighboring nodes differently. In the context of cascading failures, this feature could be helpful, as certain nodes and edges play a more critical role in the system than others. Consequently, applying higher weights to nodes of greater importance in the grid structure than to less important nodes can be advantageous when predicting power outages resulting from cascading failures.

Similarly, the Topology Adaptive Graph Convolutional Network (TAG) appears to be a strong contender due to the global impact that cascading failures can have, despite being triggered locally. The ability of TAG to incorporate information from beyond the neighborhood of nearest neighbors is promising, as it enables the model to capture relevant information from distant regions of the network. This feature is particularly useful as it allows for the construction of more shallow networks without losing information from further away in the network, which is a practical advantage.

For more in-depth explanations of the GAT and TAG layers, please refer to sections 7.4 and 7.3. These architectures are selected based on their intuitive properties that align well with the challenges posed by predicting cascading failures in power grids.

The applied Graph Attention Network and Topology Adaptive Graph Convolutional

Network architecture use the PyTorch layers *gat2conv* [30] and *TAGconv* [6] respectively. The inputs for the Graph Attention Network are the node features and edge features, while the Topology Adaptive Graph Convolutional Network uses only the node features and one edge feature. As the edge feature which we chose the initial damage, as it is intuitivly the most meaningful, since without knowing where the damage has happend, it should be impossible to predict the caused power outages.

Both architectures follow the following scheme: Each model consists of $N$ convolutional layers. Each convolutional layer is followed by optional batch normalization before the activation function. After the activation function a possible dropout is applied. The convolutional layers can be used with skip connection using the mean of layer $i$ after batch normalization and the output after dropout of layer $i-1$. The convolutional layers are followed by a regression head consisting of $M+1$ linear layers. A visualization of the architecture can be seen in figure 4.8.

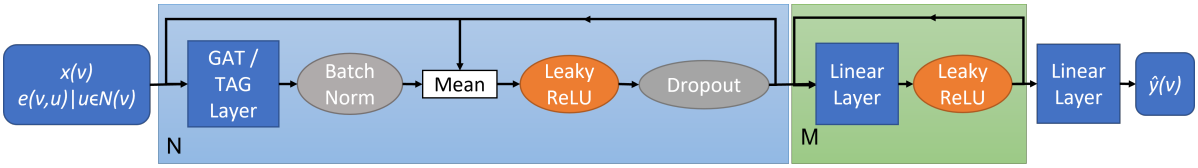



**Figure 4.8:** The applied Graph Attention Network architecture consisting of $N$ convolutional layers followed by $M+1$ linear layers as regression head. All layers have a rectified linear unit as activation function. Each convolutional layer is followed by optional batch normalization and after the activation funtion an optional dropout. The convolutional layers can also be used with an optional skip connection taking the mean of the output of layer $i$ after batchnormalization and layer $i-1$ after dropout. The batch normalization, skip connections and dropout are all optional and subject to the hyperaparameter studies as explained in section 4.4.

## 4.3 Loss Functions and Loss Masking

During the course of the thesis, we experimented with multiple loss functions $l(\boldsymbol{y}, \hat{\boldsymbol{y}})$. The primary loss function used was the regular mean squared error, defined as:

$$l(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{4.16}$$

where $\boldsymbol{y}$ represents the label vector, $\hat{\boldsymbol{y}}$ is the output vector, and $n$ is the number of instances.

We also experimented with custom loss functions that apply weights to the contribution of different nodes. One of these custom loss functions weights the contribution by the variance of the node label of each node, defined as:

$$l_{var}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\sigma_{y_i}^2, \tag{4.17}$$

where $\sigma_{y_i}$ is the standard deviation of the node label of node $i$ within the training data set. However, this loss function performed poorly during prototyping and was not applied during the hyperparameter studies or cross-validation.

On the other hand, the second custom loss function weights the contribution of each node with a fixed factor if the label of that node is not 0 for that sample. Thus the loss is calculated as

$$l_{non-zero}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 f_i, \tag{4.18}$$

where

$$f_i = \begin{cases} f, & \text{if } y_i > 0 \\ 1, & \text{otherwise} \end{cases}, \tag{4.19}$$

thus either diminishing ($f < 1$) or amplifying ($f > 1$) the contribution of nodes with non-zero labels.

Finally, we also investigated loss masking, where the contribution to the total loss of some nodes is set to 0. The decision of whether a node is considered during loss calculation is based on attributing each node a probability. The probability for each node is calculated as follows:

$$p_i = \frac{\sigma_i^2}{\max_i(\sigma_i^2)} + c, \tag{4.20}$$

where $\sigma_i$ is the standard deviation of the node label of node $i$ within the training set, and $c$ is a constant. The calculated probability $p_i$ represents the likelihood of a node being considered during loss calculation. The loss masking is achieved by drawing random numbers from a Bernoulli experiment using the calculated probabilities. Consequently, the masking amplifies the impact on the loss of nodes with higher node label variance. The constant $c$ is used to scale the total number of nodes considered on average during loss calculation, influencing the overall masking effect.

## 4.4 Studies

The hyperparameter studies were conducted using the Python library Ray Tune 2.3.0 [18], specifically employing the Bayesian Optimization (BayesOpt) with the Asynchronous Successive Halving Scheduler (ASHA) [17]. The maximum number of epochs for each study was set to 100, with a grace period of 10 epochs. Additionally, the ReduceLROnPlateau learning rate scheduler from PyTorch was used in all cases, with a factor of 0.1, patience of 10 epochs, and a threshold of 0.0001.

The batch sizes for all models were initially set to 100, except for the Graph Attention Network (GAT) (batch size = 25) and the Topology Adaptive Graph Convolutional Network (TAG) (batch size = 75) due to memory constraints during training. To address this issue, gradient accumulation was applied in these cases.

For all models, the training set was shuffled during training. The hyperparameter studies were conducted on the dataset excluding storm Claudette, as described in section 4.1.7, with the data from storm Hanna used as the validation set.

The search spaces were set up so that parameters shared between the models had the same search space in all models. The configurations of the search spaces for the different models are presented in table 4.2.

Each hyperparameter study consisted of 30 trials to comprehensively explore the hyperparameter space and find the optimal configurations for each model.

**Table 4.2:** Searchspace of the conducted hyperparameter studies

| Parameter | Interval | Ridge | MLP | Node2Vec +MLP | GINE | GAT | TAG |
|---|---|---|---|---|---|---|---|
| Learning Rate | $[10^{-9}, 10^{-3}]$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Weight Decay | $[10^{-9}, 10^{-3}]$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Number of Layers | $[1, 5]$ | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Hidden Features | $[16, 512]$ | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Dropout | [0.0,0.35] | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Gradient Clipping | $[0.02, 1]$ or None | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Batchnormalization | [True,False] | × | ✓ | ✓ | × | ✓ | ✓ |
| Skip Connections | [True,False] | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Loss Masking | [True,False] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Masking Bias | [0.0,0.5] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Loss Type | [MSE, Weighted Loss] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Loss Weight | [0.5,10] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Regression Head Layers | [1,3] | × | × | × | ✓ | ✓ | ✓ |
| Regression Head Hidden Features | [64,300] | × | × | × | ✓ | ✓ | ✓ |
| Attention Heads | [1,5] | × | × | × | × | ✓ | × |
| GAT Layer Dropout | [0, 0.5] | × | × | × | × | ✓ | × |
| TAG Jumps | [1,30] | × | × | × | × | × | ✓ |

## 4.5 Cross-Validation

To validate the results a 7-fold cross-validation was applied in which each fold consists in the use of the data of one storm as the test data and the data of the rest of the storms as training data. This is conducted on the full data set as described in section 4.1.7.

Since storm Claudette was completely excluded for the hyperparameter studies the cross-validation contains a test set which has not been seen prior by the model. This ensured that the model's performance was evaluated on unseen data to provide a robust assessment of its generalization capabilities.

# 5 Results

In this section, we will present the results of the hyperparameter studies and the cross-validation. We will investigate common metrics and compare the performances of the different models in these metrics with each other and the baselines.

## 5.1 Study Results

The optimal parameters obtained from the hyperparameter studies conducted, as explained in Section 4.4, are summarized in Table 5.1.

## 5.2 Cross-Validated Results

The results presented in Table 5.2 show the mean MSE and $R^2$ scores of the 7-folds of the cross-validation, with each fold using one storm as the test set (as explained in Section 4.5). We focus on the $R^2$ error as the main metric because the data (as described in Section 4.1.7) mostly consists of nodes with 0 load shed. Therefore, small errors or high accuracy are less meaningful, as a model always predicting zero would achieve very high accuracy and low error. Instead, we are interested in accurately predicting non-zero values, which is better reflected through the $R^2$ score. Like the MSE The $R^2$ score is calculated for every model between its output and the labels on every test set of the cross validation. The values are then averaged and the standard deviation is calculated.

As observed, the best performing baseline is Node2Vec+MLP, closely followed by MLP, while the Node Mean still achieves an $R^2$ greater than 0. However, the Ridge model performs much worse, with an $R^2$ of 0. Among the Graph Neural Networks (GNNs), both the Graph Isomorphism Network (GINE) and the Topology Adaptive Graph Convolutional Network (TAG) show very similar performances, close to Node2Vec+MLP and MLP, while the Graph Attention Network (GAT) performs even worse than the Node Mean baseline.

Comparing the GNNs to the baselines using their $R^2$ scores, we find that only the GINE slightly outperforms the best baseline (Node2Vec+MLP), but their scores fall within the standard deviations. The same applies to the MSE on the test set for the best performing model and baseline (GINE and Node2Vec+MLP). While TAG still outperforms the Node Mean, GAT only outperforms the Ridge regression which performed particularly poorly.

To assess the significance of the performance differences, we conducted paired $t$-tests comparing the GNNs against the baselines using the test $R^2$ scores obtained during

**Table 5.1:** Optimal parameters of the models, derived from the hyperparameter studies conducted as explained in section 4.4.

| Parameter | Ridge | MLP | Node2Vec +MLP | GINE | GAT | TAG |
|---|---|---|---|---|---|---|
| Learning Rate | 2.5e-5 | 5.0e-4 | 1.3e-5 | 7.9e-4 | 2.0e-6 | 2.0e-5 |
| Weight Decay | 4.0e-6 | 1.0e-6 | 1.3e-6 | 7.9e-7 | 1.3e-7 | 2.0e-7 |
| Number of Layers | - | 5 | 4 | 1 | 1 | 1 |
| Hidden Features | - | 459 | 460 | 449 | 460 | 70 |
| Dropout | - | 0.28 | 0.26 | 0.013 | 0.06 | 0.17 |
| Gradient Clipping | × | 0.88 | 0.78 | 0.55 | 0.78 | 0.6 |
| Batchnormalization | - | × | × | - | ✓ | ✓ |
| Skip Connections | - | × | ✓ | ✓ | ✓ | × |
| Loss Masking | × | × | × | ✓ | × | ✓ |
| Masking Bias | × | × | × | 0.18 | × | 0.032 |
| Loss Type | MSE | MSE | MSE | MSE | MSE | MSE |
| Loss Weight | × | × | × | × | × | × |
| Regression Head Layers | - | - | - | 1 | 1 | 2 |
| Regression Head Hidden Features | - | - | - | 274 | 141 | 141 |
| Attention Heads | - | - | - | - | 4 | - |
| GAT Layer Dropout | - | - | - | - | 0.48 | - |
| TAG Jumps | - | - | - | - | - | 22 |

A hyphen (-) implies that the parameter has not been studied whereas an '×' implies it has been studied and the result is to not apply that feature. Some parameters have not been studied for some models as they are not present in these models.

cross-validation. The corresponding $p$-values of these tests are presented in Table 5.3. It is important to note that the $R^2$ score of GAT was consistently lower than that of the baselines, except for Ridge. Therefore, if a $p$-value indicates a significant difference for GAT, it implies that GAT is outperformed by the baseline models (except for Ridge).

The results indicate that none of the performance differences between the Node Mean and the GNNs are significant. However, Ridge regression clearly gets outperformed by all GNNs.

The improved performance of GINE over MLP and Node2Vec+MLP is statistically significant. Similarly, the better performance of Node2Vec+MLP over TAG is also significant, but the difference in performance between MLP and TAG is not significant.

It is noteworthy that Ridge regression is by far the worst performing model, with notably large errors. These errors suggest that Ridge performs better on some folds than others, indicating inconsistency across different folds (i.e., storms) in terms of data size, feature and label distributions. The poor performance of Ridge could be attributed

**Table 5.2:** The table shows the mean performance of the investigated architectures (section 4.2) with tuned hyperparameters (section 4.4) during cross-validation (section 4.5) on the full set (section 4.1.7). The best values for the baselines and the graph neural networks are in bold.

| | **Train MSE [e-2]** | **Test MSE [e-2]** | **Train $R^2$** | **Test $R^2$** |
|---|---|---|---|---|
| **Node Mean** | $5.69 \pm 0.03$ | $5.97 \pm 0.17$ | $0.320 \pm 0.003$ | $0.313 \pm 0.017$ |
| **Ridge** | $9 \pm 2$ | $9 \pm 2$ | $0.0 \pm 0.2$ | $0.0 \pm 0.2$ |
| **MLP** | $\mathbf{5.44 \pm 0.01}$ | $5.43 \pm 0.01$ | $\mathbf{0.350 \pm 0.005}$ | $0.351 \pm 0.005$ |
| **Node2Vec+MLP** | $5.48 \pm 0.05$ | $\mathbf{5.38 \pm 0.04}$ | $0.345 \pm 0.011$ | $\mathbf{0.357 \pm 0.011}$ |
| **GINE** | $3.90 \pm 0.01$ | $\mathbf{5.37 \pm 0.01}$ | $0.428 \pm 0.004$ | $\mathbf{0.358 \pm 0.004}$ |
| **TAG** | $\mathbf{3.160 \pm 0.007}$ | $5.44 \pm 0.01$ | $\mathbf{0.445 \pm 0.005}$ | $0.350 \pm 0.005$ |
| **GAT** | $6.31 \pm 0.03$ | $5.96 \pm 0.02$ | $0.287 \pm 0.004$ | $0.246 \pm 0.004$ |

**Table 5.3:** The table shows the $p$-values calculated with a paired $t$-test from the $R^2$-results of the test sets of the 7-fold cross-validations.

| | **GINE** | **TAG** | **GAT** |
|---|---|---|---|
| **Node Mean** | 0.051 | 0.10 | 0.2 |
| **Ridge** | 2e-8 | 2e-8 | 1.0e-10 |
| **MLP** | 1.0e-5 | 0.13 | 7e-10 |
| **Node2Vec+MLP** | 0.001 | 0.005 | 0.017 |

to its approach of taking whole graphs as input and training one model for each node. This node-specific modeling may encounter difficulties during cross-validation when the folds have varying sizes and distribution characteristics. Additionally, the general sample size for the Ridge model can be considered significantly smaller, as every model only considers one node per instance instead of all 2000 nodes in the graph.

## 5.3 Analysis of Similar Performance of GNNs and Baselines

For the unsatisfying performance of the GNNs in comparison to the baselines we found four possible reasons:

1. The number of trials in the hyperparameter study was not high enough.
2. The optimal parameters are not within the investigated search spaces.
3. The training period was too short.
4. The data representation was not good enough.

In the following sections we will investigate these four possibilities by taking a closer look at the studies, the output of the models and their performance measures. We will also put a focus on the comparison of GINE and TAG to the MLP and Node2Vec+MLP baselines as they show very similar performances even though the GNNs should intuitively outperform them by a large margin.

### 5.3.1 Number of Trials and Optimal Parameters

As evident from the exemplary history plot of the Bayes optimization of the Graph Isomorphism Network (GINE) shown in Figure 5.1, the hyperparameter search does not substantially improve the performance of the network. This observation is consistent across all the studied architectures (see Figure 7.3).

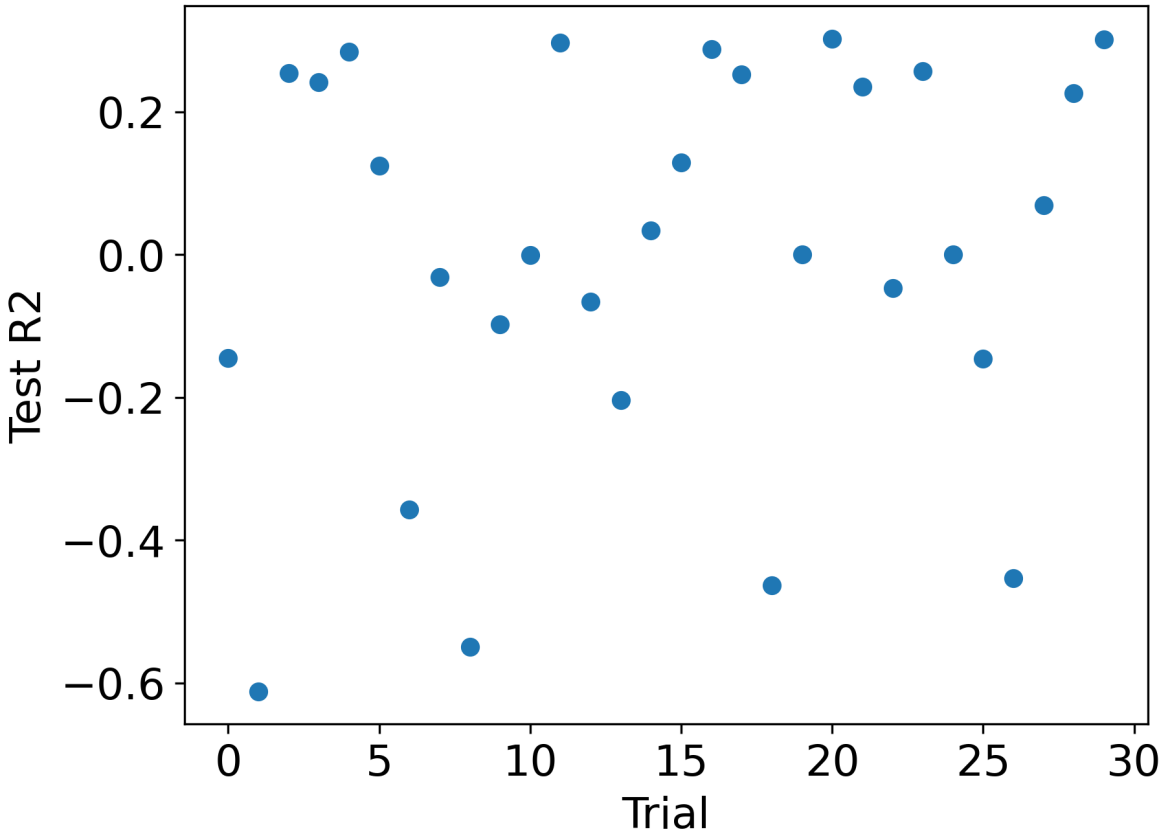



**Figure 5.1:** The plot shows the history of the hyperparameter study on the graph isomorphism network architecture. The history plots of the studies of the other models can be found in figure 7.3.

To further investigate this, we conducted an exemplary ablation study on the GINE architecture to explore possible hints at the edges of the search space that could potentially enhance its performance. The ablation studies were performed using the optimal

parameters evaluated in the full study (as discussed in Section 5.1), except for the parameter under study. The parameters studied were within the same search spaces as in the full study, except for the regression head hidden features (as shown in Figure 5.2h) and the convolutional layers hidden features (as shown in Figure 5.2d), both of which were extended to [64, 1024]. The number of convolutional layers (as shown in Figure 5.2c) was also extended to [1, 8].

However, none of the ablation studies exhibited a clear trend or maximum, and the results appeared to be more or less random, except for the learning rate (as shown in Figure 5.2a) and the dropout (as shown in Figure 5.2e). For dropout values close to 0.1, achieving better performance seemed more reliable, but the model did not consistently perform better, as higher values still achieved similar $R^2$ scores at times. Regarding the learning rate, the curve appeared to saturate around $10^{-5}$, and going to values higher than $10^{-3}$ would likely result in unstable learning. These findings suggest that the search space for hyperparameters may not significantly impact the performance of the GINE architecture.

The results of the ablation study, combined with the low $R^2$ scores ($< 0.36$) obtained during the cross-validations and the absence of any improvement within the conducted thirty trials of the hyperparameter studies, lead us to believe that increasing the number of trials would be unlikely to yield significant improvements in the models' performance.

Furthermore, the ablation study indicates that expanding the search spaces (or using more complex models) would also have little to no effect on the models' performances.

### 5.3.2 Training Period

Regarding the possibility that the training time was too short for the Graph Neural Networks (GNNs) to learn properly, we examined the learning curves of the cross-validations. Figure 5.3 demonstrates that the length of the training period is unlikely to be the reason for the GNNs' performances. The learning curves show saturation for the GNNs (figures 5.3a to 5.3c) around 50 epochs, and for TAG, it saturates even around 30 epochs. As for the baselines, the MLP also appears to saturate around 50 epochs, while the Node2Vec+MLP and Ridge regression do not seem to have reached the learning limit. This observation could be part of the explanation why Ridge performed so bad as it could further improve performance if trained longer. It also has important implications on the comparison between Node2Vec+MLP and the GNNs.

Since we have now also ruled out the option of too little training time being the reason for the Node2Vec+MLP, MLP and the GNNs having similar performance, we will now take a closer look on the networks outputs and what they actually learn and compare the networks to each other. We will particularly focus on the comparison between the MLP and GINE and TAG since their performances are very similar even though GINE and TAG are much more suitable to the task and can profit from much more of the contained information.

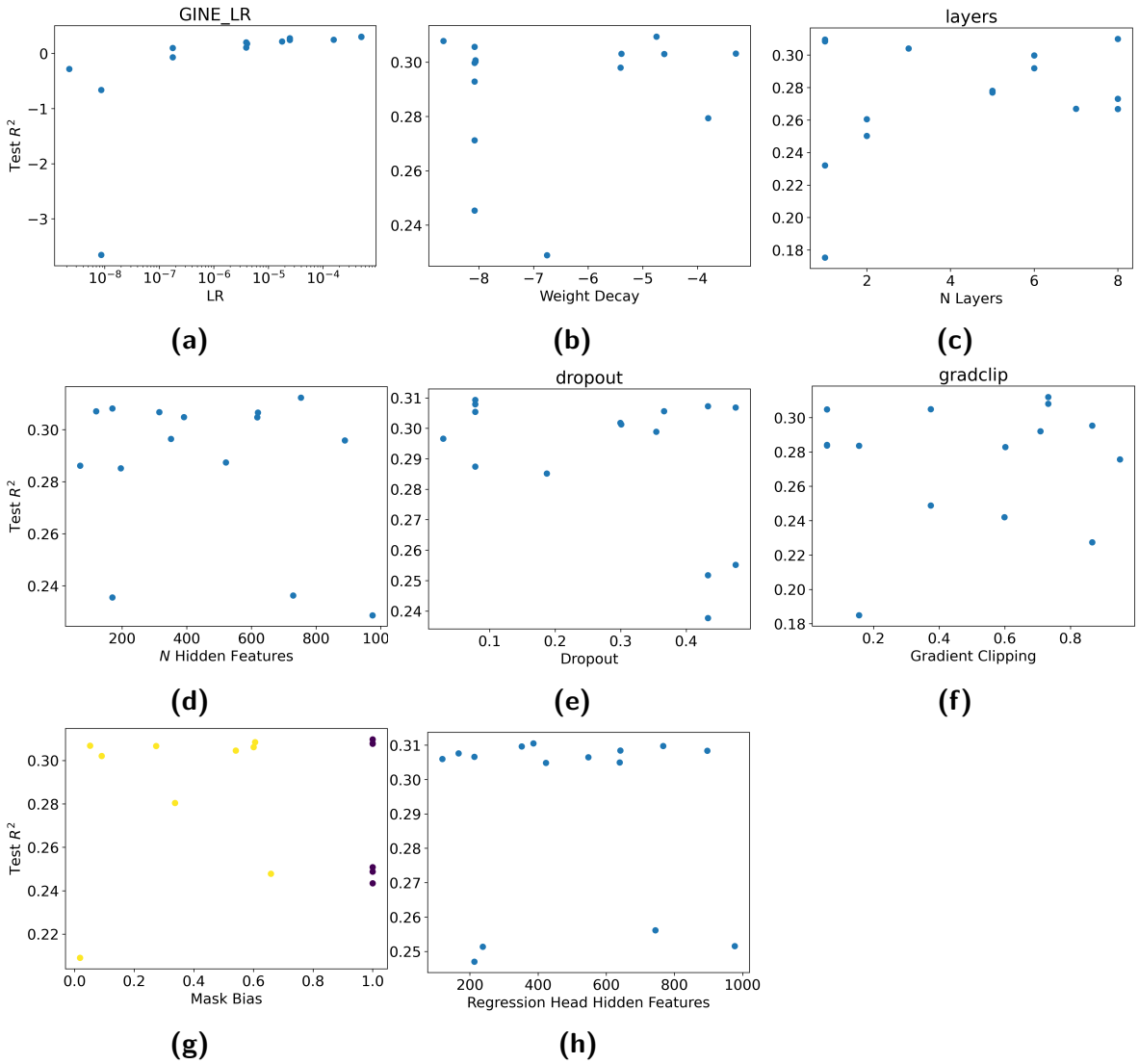
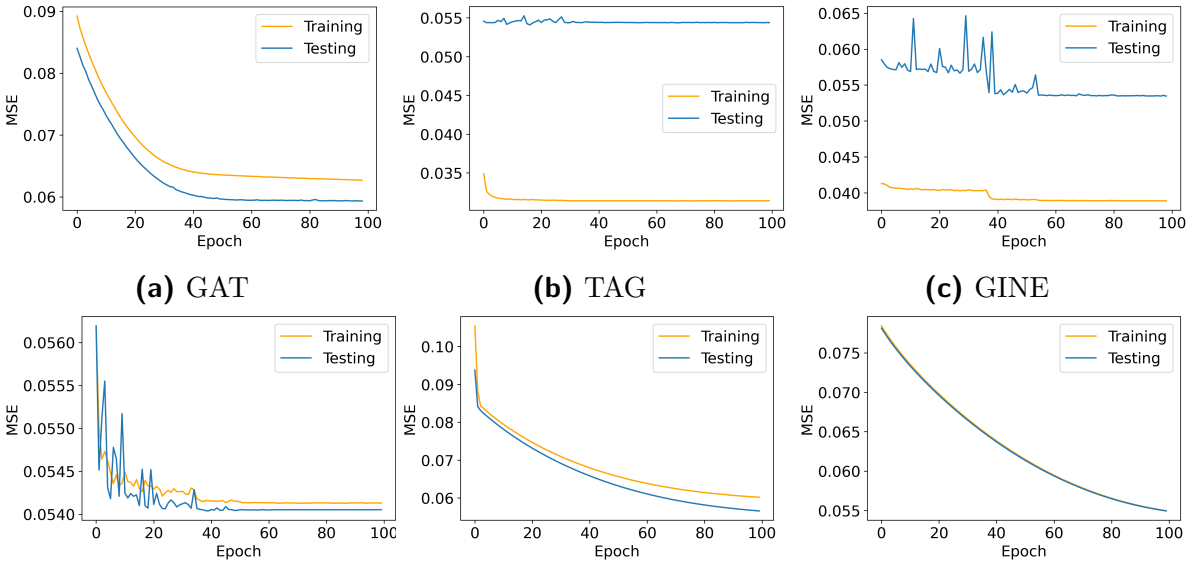



**Figure 5.2:** The figures show the results of the ablation studies for the different hyperparameters of the Graph Isomorphism Network (GINE). The values for Learning Rate and Weight Decay are the exponents used. The search spaces during the ablation studies are the same as in the original study except for both hidden features 5.2d and regression head hidden features 5.2h where it was extended to [64, 1024] and the number of layers 5.2c where it was extended to [1, 8]. All of the ablation studies use the constant optimal parameters evaluated through the full study (sec. 5.1) except for the studied hyperparameter. The purple points in figure 5.2g had the masking disabled which is equivalent to setting the mask bias to 1 (see Section 4.3).

### 5.3.3 Data Representation

The similar performance of Node2Vec+MLP, MLP, TAG, and GINE could suggest that the data representation may not provide enough detail for the GNNs to effectively utilize




**(a)** GAT **(b)** TAG **(c)** GINE

**(d)** MLP **(e)** Node2Vec+MLP **(f)** Ridge

**Figure 5.3:** The figures show the learning curves of the different models taken from the cross-validation fold with Claudette as test set.

the graph properties. This might cause the GNNs to learn a function that maps node features to the output in a manner similar to the MLP, while only making use of the topology of the graph as Node2Vec does but not the features of connected nodes and edges. To verify this, we will conduct a thorough investigation of the network outputs using common analysis tools and measures. This analysis will help us gain insights into how the models are processing the data and whether the graph structure is being effectively utilized.

### Confusion Matrices

We will begin by examining the confusion matrices of all models to classify the output of the test set into 11 classes in the interval $[0, 1]$. The confusion matrix shows how the instances of each label class have been classified by the model. The classification is done by flooring the predicted regression values to the first digit after the decimal point. This results in a seperate class for labels or predicted values of exactly 1.

Figure 5.4 displays the confusion matrices for all models, showing that they predict 0 relatively well but underpredict all the classes with outages. This underprediction is particularly evident in the normalized plots in figure 5.5. When comparing the general shapes of the confusion matrices, we observe that all models exhibit a similar classification pattern, classifying most 0 labels as 0 and some up to 0.5, while limiting the output to values up to 0.5 even for labels with higher load shed values.

This similarity in behavior (as well as the similar multiclass recalls) indicate that both the baselines and the Graph Neural Networks (Gine and TAG) learn similar representations, and the GNNs do not effectively utilize the graph properties to learn more

advanced representations.

Again, the Graph Attention Network (GAT) (figures 5.4g and 5.5g) appears to have more difficulty in identifying the classes compared to the other networks. The classification outputs of GAT are more diffused, spreading across a wider range between 0.1 and 0.4, while other models exhibit more linear connections between the labels and the predicted outputs. For example, GINE may classify the class of labels between 0.7 and 0.8 into the classes 0.3 and 0.4, with a large majority falling into 0.3, whereas GAT has outputs distributed more evenly between 0.1 and 0.4.

Given the cross-validation results, we would have expected to see different behavior for Ridge regression. However, since we only evaluate the fold where Claudette constitutes the test set, the performance of Ridge is very similar, reporting an $R^2$ value of 0.34 on both the test and train sets, along with an MSE of 0.055 on both sets.

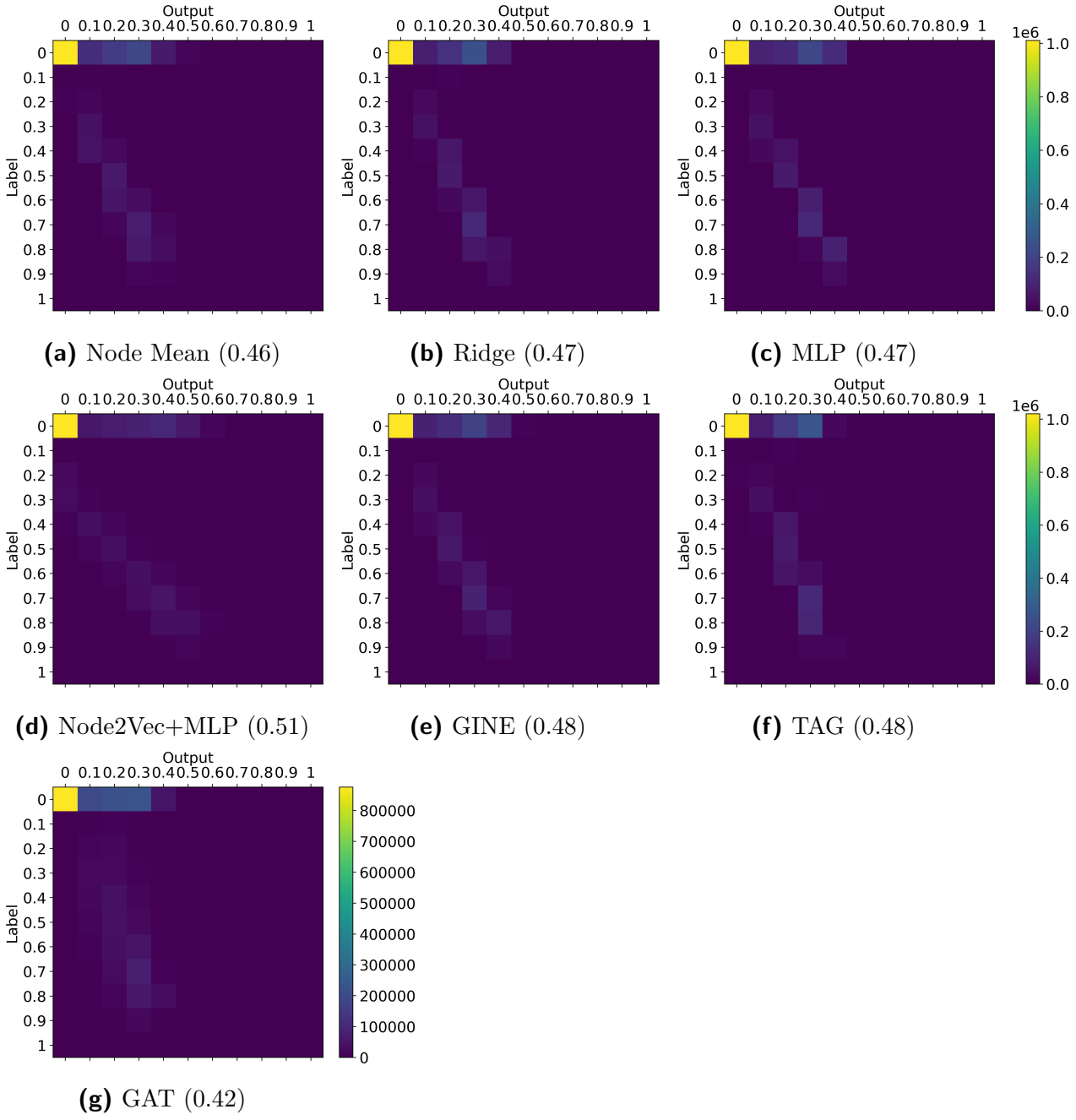
### Inter Model $R^2$ Scores

To further test the coherence of predictions between the models we calculate the mean $R^2$ scores between the different model outputs on the test sets by taking the output of one model as the 'ground truth' and the output of the other model as the modeled values. The mean of the $R^2$ scores is taken over the 7-folds of the cross-validation and the standard deviation is calculated.

The idea is that if the $R^2$ between two models is close to 1 they predict (almost) the same implying that they learned the same underlying structure. In particular if the GNN architectures have a high $R^2$ with the MLP it implies that the GNNs do not make use of the graph structured data as they learn the same structure as the MLP which only takes the node features as input. The results can be seen in table 5.4.

The table shows that the outputs of GINE and TAG have an $R^2 \geq 0.95$ with the outputs of the MLP implying that they learn very similar structures. Node2Vec+MLP also achieves $R^2 > 0.95$ with MLP, GINE and TAG. Again implying that the GNNs can not make more use of the graph properties as Node2Vec.

GAT only has $R^2 \approx 0.7$ with MLP which is explainable by the general worse performance of GAT which can also be seen in comparison of GAT and the other models.

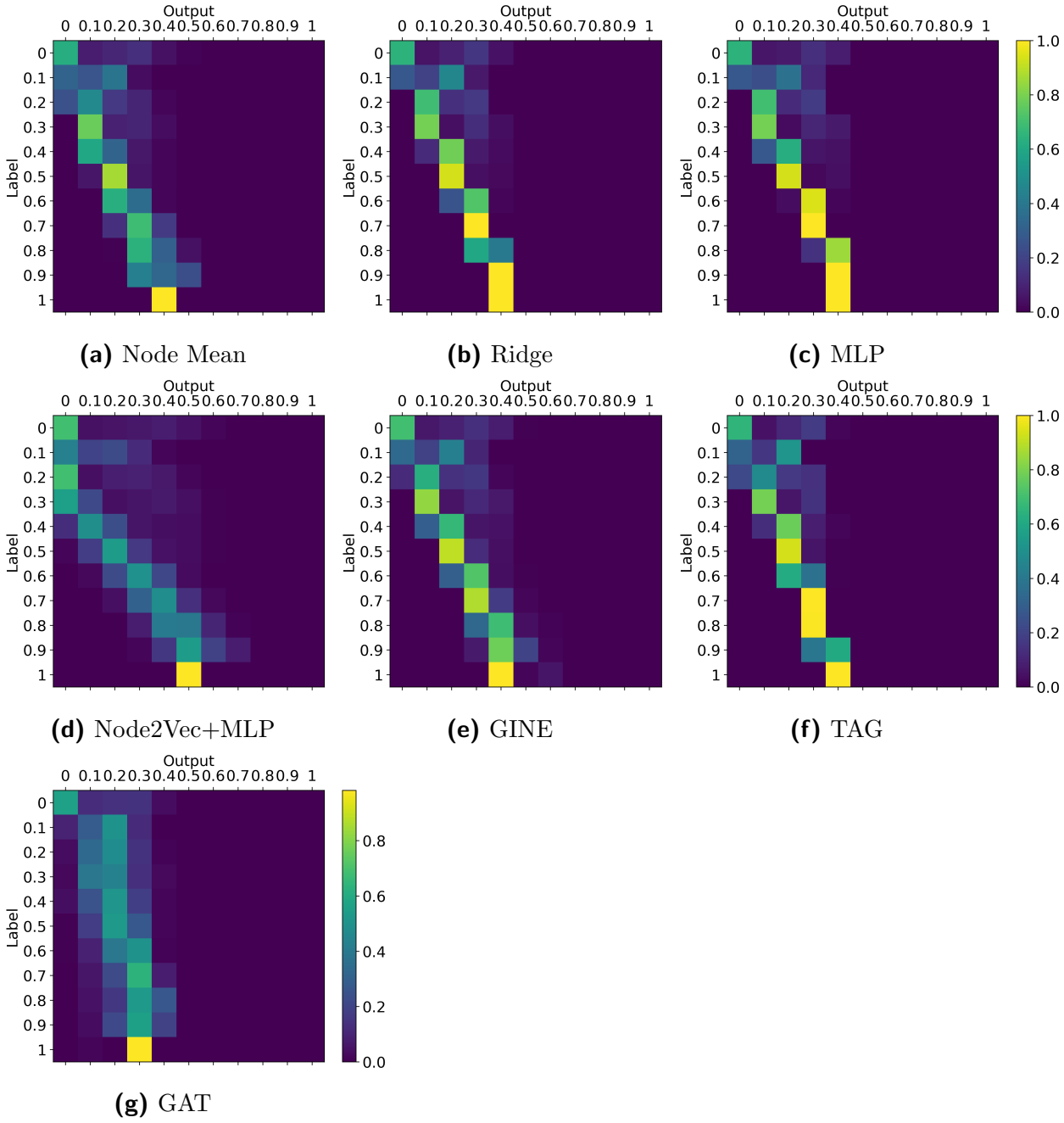



**(a)** Node Mean (0.46) **(b)** Ridge (0.47) **(c)** MLP (0.47)

**(d)** Node2Vec+MLP (0.51) **(e)** GINE (0.48) **(f)** TAG (0.48)

**(g)** GAT (0.42)

**Figure 5.4:** The figures display the confusion matrices of the different models, utilizing multiclass classification with 11 classes. The classification is done by flooring both the output and labels to the first digit after the decimal point. The value provided in parentheses corresponds to the multiclass recall. The colorbar indicates the number of instances per classification.




**(a)** Node Mean **(b)** Ridge **(c)** MLP

**(d)** Node2Vec+MLP **(e)** GINE **(f)** TAG

**(g)** GAT

**Figure 5.5:** The figures display the confusion matrices of the different models, utilizing multiclass classification with 11 classes. The classification is done by flooring both the output and labels to the first digit after the decimal point. The normalization is executed per label class. Thus one row always sum to 1. The colorbar indicates the percentage of classifications within the label class.

**Table 5.4:** The tables show the mean $R^2$ scores between two models. The output counted as 'ground truth' is the output of the model indicating the rows. The mean and standard deviation arise from the different outputs on the different test sets of the cross-validation.

| | **Node Mean** | **Ridge** | **MLP** | **Node2Vec+MLP** |
|---|---|---|---|---|
| **Node Mean** | 1 | -10±30 | 0.88±0.06 | 0.90±0.06 |
| **Ridge** | -1±3 | 1 | -1±4 | -1±3 |
| **MLP** | 0.88±0.06 | | 1 | 0.980±0.004 |
| **Node2Vec+MLP** | 0.89±0.06 | -10±30 | 0.977±0.005 | 1 |
| **GINE** | 0.85±0.06 | -10±30 | 0.95±0.05 | 0.95±0.05 |
| **TAG** | 0.88±0.06 | -10±30 | 0.984±0.007 | 0.970±0.007 |
| **GAT** | 0.71±0.05 | -11±18 | 0.80±0.03 | 0.77±0.03 |

| | **GINE** | **TAG** | **GAT** |
|---|---|---|---|
| **Node Mean** | 0.87±0.06 | 0.86±0.08 | 0.5±0.2 |
| **Ridge** | -1±3 | -1±4 | -2±5 |
| **MLP** | 0.96±0.04 | 0.980±0.009 | 0.66±0.12 |
| **Node2Vec+MLP** | 0.96±0.05 | 0.959±0.011 | 0.6±0.2 |
| **GINE** | 1 | 0.91±0.06 | 0.56±0.2 |
| **TAG** | 0.94±0.05 | 1 | 0.72±0.09 |
| **GAT** | 0.79±0.05 | 0.80±0.02 | 1 |

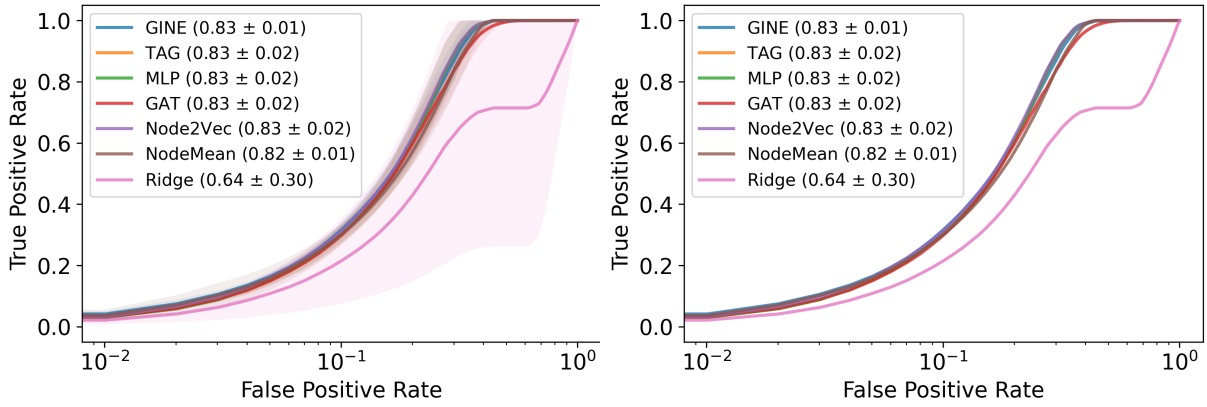
### Binary Classification Analysis

To provide more insight on the performance of the models we will present the Receiver Operating Characteristic (ROC) curves and Precision-Recall curves (PR curves) of the models as well as some fundamental performance measures. The ROC-curves are produced by calculating the True Positive Rate (TPR) and the False Positive Rate (FPR) for all unique thresholds presented by the data. To allow for classification we classify the labels as positive if the label is larger than 0.1. We calculated the ROC curves for every test set of the 7 folds of the cross-validation from where we calculated the average and the standard deviations. The PR curves are calculated in the same manner but using Precision instead of the TPR and Recall instead of the FPR.

The ROC curves and their Areas Under the Curve (AUC) [1] (see Figure 5.6 confirm the similar performance of all models and the relatively worse performance of the Ridge regression. However, it's important to note that the reported high AUCs compared to the $R^2$ scores can be attributed to the confusion matrices and label distribution.

As shown in figure 5.5, the models can choose a threshold just high enough to include almost all positive samples (Label $> 0.1$). As a result, the FPR, which is the ratio of

[1]The AUC summarizes the model's ability to discriminate between positive and negative instances, with higher AUC values indicating better performance.




**(a)** GINE **(b)** TAG

**Figure 5.6:** The figures show the mean ROC curves of the cross-validation of each model. For the binary classification the labels with $y < 0.1$ are counted as a positive labels. The opaque areas show the standard error. Figure 5.6b does not show the standard error for better visibility of the lines. The value shown as AUC is the Area Under the Curve of the model (with standard error). The plots can be found separated for each model in figure 7.4.

False Positives (FP) to the total number of negative values (FP+TN) (FPR = $\frac{\text{FP}}{\text{FP+TN}}$) is dominated by True Negatives (TN) since there are a significantly larger number of negative instances that the model can distinguish accurately. This is why the PR curve provides a more suitable measure in this case.

On the other hand, the ROC curves reveal that the models cannot be effectively used as a classification tool due to the trade-off between true positive rate and false positive rate. Trying to predict cascading failures either results in a high number of wrong predictions for positives or a significant number of missed positives. This limitation makes the models unsatisfactory for predicting cascading failures.

From the PR curves (figure 5.7) we can see that there is no way to achieve reasonable values in both measures the optimal case would be a recall of 1 with a precision around 0.4 ($< 0.3$ for Ridge). Getting a recall of higher than 0.7 is only possible with a precision smaller than 0.01. The curves show again that all models perform very similar as the curves are all very close (except Ridge) and all within each others standard errors.

### Feature Correlations

Finally, we will investigate whether there is any correlation between the node features and the node labels that could be learned by the networks. This is done by calculating the Pearson correlation of the node features and node labels for all nodes in the full set (see Section 4.1.7. The result can be seen in figure 5.8. The plot indicates a correlation of 0.59 between the apparent power at the node and the node outage, which helps to explain the performance of the MLP. It is highly likely that the MLP is learning this correlation between the two values, leading to its performance in the task.

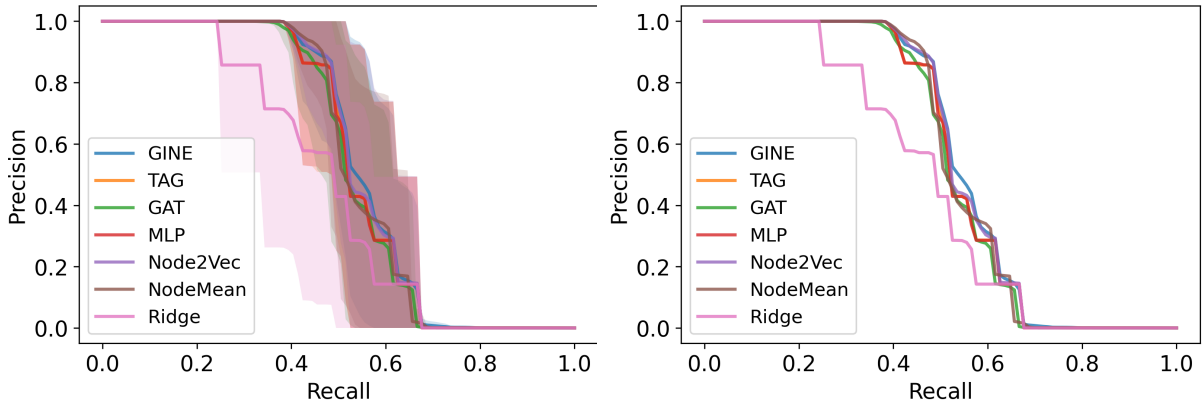



**Figure 5.7:** The figures show the mean precision recall curves of the cross-validation for each model. For the binary classification the labels with $y < 0.1$ are counted as positive labels. The opaque areas show the standard deviations. Figure 5.7b does not show the standard deviations for better visibility of the curves. The separate plots for each model can be found in figure 7.5.

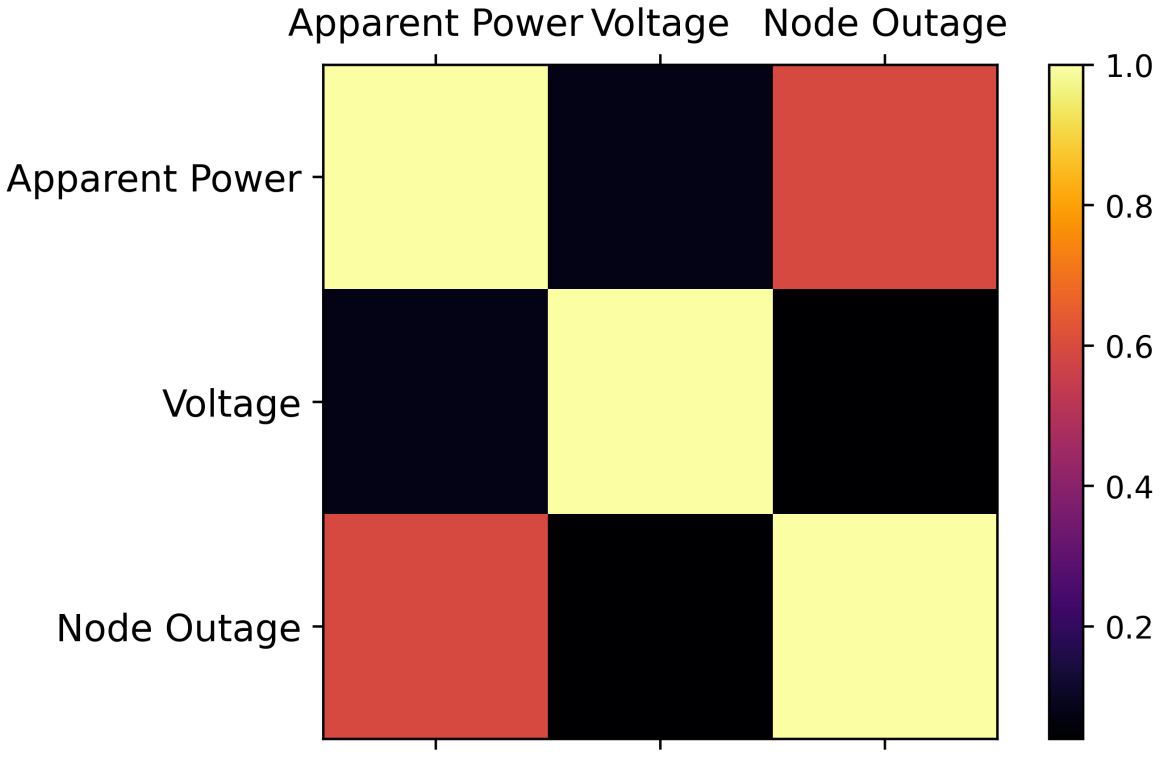



**Figure 5.8:** The plot shows the Pearson correlation between the node features and the node labels of the full data set (see Section 4.1.7).

# 6 Conclusion and Outlook

In conclusion, the goal of successfully predicting the power outage of all nodes in a potentially already damaged power grid following single line failures caused by Hurricanes using Graph Neural Networks can not be considered as reached. None of the investigated Graph Neural Networks (GNN) architectures, namely Graph Isomorphism Network (GINE), Topology Adaptive Graph convolutional network (TAG) and Graph Attention Network (GAT) reported a significantly better performance than all of the baselines presented.

For all applied GNN architectures the improvement in performance over the MLP and Node2Vec+MLP baselines was either statistically insignificant or the improvements were within the standard errors (or both). The only baseline that can be considered as outperformed was the Ridge regression, which performed especially poorly. Notably, GAT was outperformed (statistically significant and regarding the standard deviation) by MLP and Node2Vec+MLP.

The analysis of the hyperparameter studies, combined with an additional ablation study of the optimal GINE architecture, showed that the number of trials, as well as the search spaces, can be considered large enough given the computational resources available for this thesis (section 5.3.1).

The investigation of learning curves suggested that a longer training period would not considerably improve the models' performances, and it is unlikely to be the main reason for the GNNs' poor performance (section5.3.2). Additionally, the learning curves indicated that the Node2Vec+MLP and the Ridge regression model have not reached their saturation point after 100 epochs. Since Node2Vec+MLP is the best performing baseline this implies that it could perform possibly even better if trained longer, which could lead to the Node2Vec+MLP outperforming the GNNs.

Based on the prior investigations, the similar performance of the baselines and GNNs, especially Node2Vec+MLP, MLP, GINE, and TAG, suggests that the chosen data representation (section 4.1) may not be sufficient for the GNNs to effectively leverage the graph structure or differentiate nodes with different labels. The inter-model $R^2$ scores (section 5.3.3) further support this notion, as they indicate that the baselines and the GNNS (Node2Vec+MLP, MLP, GINE, and TAG), produce nearly identical outputs for each instance, with $R^2$ scores greater than 0.95.

This similarity in output implies that these models learn a very similar underlying structure. Since the MLP can only use the node features, it suggests that the GNNs are also primarily learning the connection between node features and labels. This is further supported by the correlation between one of the node features (Apparent Power) and the node labels (section 5.3.3). Additionally, the best-performing baseline, Node2Vec+MLP,

can take advantage of the graph topology, albeit not in relation to node or edge features. The better performance of Node2Vec+MLP compared to other GNNs suggests that more features may be necessary (as Node2Vec creates an embedding of the graph structure resulting in more input features).

Indeed, the performance similarity between Node2Vec+MLP, TAG, and GINE, despite their different capabilities to capture graph topology, suggests that the GNNs may not be fully utilizing the graph structure as expected. While Node2Vec+MLP incorporates topological information indirectly through Node2Vec embeddings, TAG and GINE have explicit mechanisms to process graph topology in their architectures.

This observation again consolidates the notion that the current representation of the relation between topology and features may not be detailed enough for the GNNs to effectively leverage the graph structure.

Regarding the Ridge model used as baseline which has been chosen to be a graph wise model instead of a node wise model to check if it would give a better performance in a trade off for higher computational cost it has clearly shown that the trade off is not worth it (at least considering a simple Ridge regression model) as it performed much worse than all other models in all measures.

Regarding the Ridge model used as a baseline, which was chosen to be a graph-wise model instead of a node-wise model to check if it would yield better performance, despite the higher computational cost. It has clearly shown that the trade-off is not worth it (at least with a simple Ridge regression model) as it performed significantly worse than all other models in all measures.

While we can not confirm with absolute certainty that there is no architecture that is able to successfully predict the power outage of all nodes in a potentially already damaged power grid following single line failures caused by Hurricanes given the data as presented in this thesis, we show strong indicators that this is indeed the case. Therefore other approaches to the data processing should first be considered when trying to solve this problem. Some options for more detailed data representations include adding one-hot encoded node IDs or incorporating additional unused features from the AC-CFM results, such as Voltage Angles or connected Generators with their limitations.

A particularly interesting approach could involve leveraging the time series properties of the underlying data set by adding features and labels of previous time steps to the data or using time series forecasting techniques over the scope of multiple instances. Similarly AC-CFM provides insight on the propagation of each cascade within one instance. This data could be used to make time series forecasting within each instance. This would allow to stay closer to the original goal of predicting the outage of each time step without information of previous time steps. Alternatively, creating a new dataset with a consistent initial state for the power grid in each instance could simplify the learning process and allow the models to focus on learning specific patterns and dependencies related to power outages.

Switching from a node regression setup to an edge regression task, where the power flow is directly predicted, could also be advantageous, as there is more edge data available with additional features. Furthermore, including some of the physical rules in the message passing process could facilitate learning in the models.

Apart from changing the data it could also be worth to try out transformer networks as Verbella et al. [29] and Zhou et al. [34] have been very successful on very similar tasks (even though on smaller power grids) using transformers.

Another considerable option is to pretrain on smaller networks as multiple papers have reported well performing results with this (f.e. Jhun et al.[13], Nauck et al.[22]). Even though these applications were not on AC cascading failures which can be considered more complex and more topology dependent. This approach would possibly allow the creation of datasets more suitable for the task as it would be easier to create large amounts of data with a focus on data with load shed.

We also want to mention that the data used in this thesis only displays one load profile of the Texan power grid. While the topology is likely to have a significant impact on cascading failures, testing with different load profiles could further amplify the effects of topology on failures, potentially facilitating GNN learning of topological effects when trained on data from multiple consumption states.

# 7 Appendix

## 7.1 Code Availability

The code used to perform the experiments can be found in the following repository: https://gitlab.pik-potsdam.de/tobiasoh/DC-CFM-GNN

## 7.2 Graph Isomorphism Network with Edge Features (GINE)

The Graph Isomorphism Network (GIN) is a Graph Neural Network (GNN) developed to generalize the Weissfeiler Lehman test in order to achieve maximum discriminitave power. Keyulu et al. show in [32] that no GNN can be more expressive than the WL test and prove that the developed GIN is a generalization of the Weissfeiler Lehman test and thus a maximally expressive GNN. A maximally expressive GNN is a one that never maps two different neighbourhoods to the same embedding.

To achieve this they define a multiset $S = (c, X)$, where $X$ is the underlying set of unique values in $S$ and $c : S \to \mathbb{N}_{\geq 1}$ gives the multiplicity of the elements. They show that for a multiset there exists two functions $f$ and $\phi$ so that $\boldsymbol{h}(c, X) = (1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)$ is unique for each pair $(c, X)$. Then according to the universal approximation theorem ([11]) this can be learned by multilayer perceptrons (MLPs).

Thus they develop the GIN layer as

$$\boldsymbol{h_i^{(l)}} = \text{MLP}\left(\left(1 + \epsilon^{(l)}\right) \cdot \boldsymbol{h}_i^{(l-1)} + \sum_{j:v_j \in \mathcal{N}(v_i)} \boldsymbol{h}_j^{(l-1)}\right) \tag{7.1}$$

where $\boldsymbol{h}_{i/j}^{(l)}$ are the embeddings of node $i/j$ at layer $l$ respectively. The aggregation at each node is thus a neighbourhood representation learned by an MLP based on the node and its neighbours. The GINE layer is an adaption to the GIN layer introduced in [12] that simply adds the edge features in the aggregation sum so that

$$\boldsymbol{h}_i^{(l)} = \text{MLP}\left(\left(1 + \epsilon^{(l)}\right) \cdot \boldsymbol{h}_i^{(l-1)} + \sum_{j:v_j \in \mathcal{N}(v_i)} \boldsymbol{h}_j^{(l-1)} + \sum_{e_{ij}:v_j \in \mathcal{N}(v_i) \cup v_i} \boldsymbol{h}_{e_{ij}}^{(l-1)}\right), \tag{7.2}$$

where $\boldsymbol{h}_e^{(l)}$ is the embedding of edge $e = e(v, u)$ between nodes $u$ and $v$ at layer $l$.

## 7.3 Topology Adaptive Graph Convolutional Network (TAG)

The Topology Adaptive Graph Convolutional Network (TAG) [6] is a Graph Neural Network (GNN) designed to have topology adaptive graph filters over the $K$ nearest neighbours preventing linear approximation in deeper networks, while at the same time operating on lower computational cost than traditional graph convolutional networks without using approximations [6].

The graph filter for the $m$-th input feature of $N_f$ total input features and $n$-th output feature, of the $l$-th layer $G_{m,n}^{(l)}$ is defined as

$$G_{m,n}^{(l)} = \sum_{k=0}^{K} g_{m,n,k}^{(l)} A^k, \tag{7.3}$$

where $K$ is the number of applied filters and $A = D^{-\frac{1}{2}} \bar{A} D^{-\frac{1}{2}}$ is the normalized adjancency matrix, and $D$ is the diagonal degree matrix $D = \text{Diag}(d)$ with $d_i = \sum_j A_{i,i}$. $\bar{A}$ is the so called graph shift that replaces the feature vectors of the nodes by a linear combination of their neighbouring nodes ${\boldsymbol{x}'_m}^{(l)} = \bar{A}\boldsymbol{x}_m^{(l)}$. Applying these graph filters the $n$-th output of the $l$-th layer is

$$\boldsymbol{y}_f^{(l)} = \sum_{m=1}^{N_f} G_{m,n}^{(l)} \boldsymbol{x}_m^{(l)} + b_n^{(l)} \tag{7.4}$$

where $x_m^{(l)}$ contains feature $m$ for all nodes and $b_n^{(l)}$ is a learnable bias.

It is worth mentioning that the TAG layer does not include edge attributes. TAG is only capable of processing one edge attribute which is introduced through $A$ being a weighted adjacency matrix, where very entry is weighted by the edge attribute.

## 7.4 Graph Attention Network (GAT)

The Graph Attention Network (GAT) [30] is a graph convolutional network designed to allow nodes to weight the different features of their different neighbouring nodes separately, allowing for a very high expressiveness.

The attention mechanism works as follows. For a GAT layer with $N_{f'}$ output features and $N_f$ input features the attention coefficients $\alpha_{ij}$ are calculated as

$$\alpha_{ij} = a(\boldsymbol{W}\boldsymbol{x}_i, \boldsymbol{W}\boldsymbol{x}_j), \tag{7.5}$$

where $\boldsymbol{W}$ is a $N_{f'} \times N_f$ matrix and $x_{i/j}$ are the feature vectors of node $i/j$ respectively. $a$ is the attentention mechanism $a : \mathbb{R}^{N_{f'}} \times \mathbb{R}^{N_{f'}} \to \mathbb{R}$.

The attention mechanism is masked so that $\alpha_{ij}$ is only calculated for nodes $v_j$ if $v_j \in \mathcal{N}(v_i)$ In the original paper [30] the attention mechanism $a$ is realized as a single layer feed forward neural network as

$$\alpha_{ij} = \text{LeakyReLU}(\tilde{\boldsymbol{a}}^T[\boldsymbol{W}\boldsymbol{h}_i || \boldsymbol{W}\boldsymbol{h}_j])), \tag{7.6}$$

where $\tilde{a} \in \mathbb{R}^{2N_{f'}}$ is a weight vector and $||$ is the concatination operator. $\boldsymbol{W}$ is the weight matrix of the feed forward neural network and $\boldsymbol{h}_{i/j}$ are the embeddings of nodes $i/j$ respectively.

Since, according to Brody et al. [5], this formulation leads to a static attention, meaning that the attention given to a node $v_j$ is independent of the query node $v_i$, we use the proposed fixed version GATv2conv which calculates the attention as

$$\alpha_{ij} = \tilde{\boldsymbol{a}}^T \text{LeakyReLU}(\boldsymbol{W}[\boldsymbol{h}_i || \boldsymbol{h}_j]), \tag{7.7}$$

which leads to dynamical atttention. This formulation calculates the attention given to node $v_j$ separately for each query node $v_i$.

In both versions the attention coefficients are normalized using softmax

$$\bar{\alpha}_{ij} = \text{softmax}(\alpha_{ij}). \tag{7.8}$$

This results in the layer output

$$h_i = \sigma \left( \sum_{j:v_j \in \mathcal{N}(v_i)} \bar{\alpha_{ij}} \boldsymbol{W} \boldsymbol{h}_j \right). \tag{7.9}$$

Finally both versions allow for *multi-head attention* which is calculated by concatenating the embeddings created by the multiple attention heads:

$$\boldsymbol{h}_i = \|_{k=1}^{N_{heads}} \sigma \left( \sum_{j:v_j \in \mathcal{N}(v_i)} \bar{\alpha}_{ij} \boldsymbol{W} \boldsymbol{h}_j \right). \tag{7.10}$$

## 7.5 Additional Figures

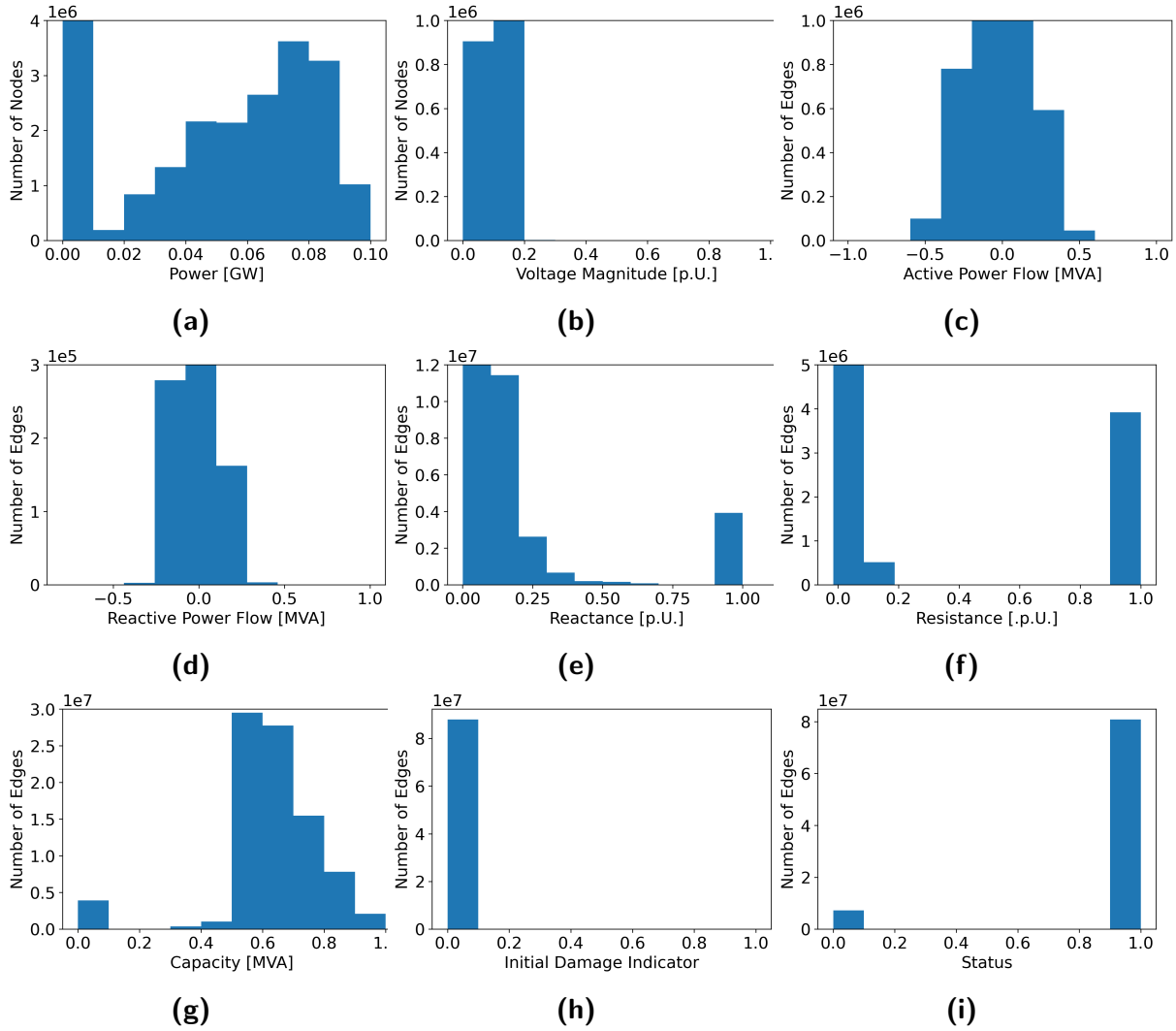



**Figure 7.1:** The figures show the zoomed in distributions of the normalized node and edge features (figure 4.5) of the data set using all the storms as described in section 4.1.7. Figures 7.1a and 7.1b are node features and figures 7.1c to 7.1i are edge features. The active and reactive power flow (figures 7.1c and 7.1d) have been normalized using standardization with consecutive normalization whereas the rest has been normalized with logarithmic normalization except for status and initial damage indicator (figures 7.1h and 7.1i) which have not been normalized due to their binary nature.

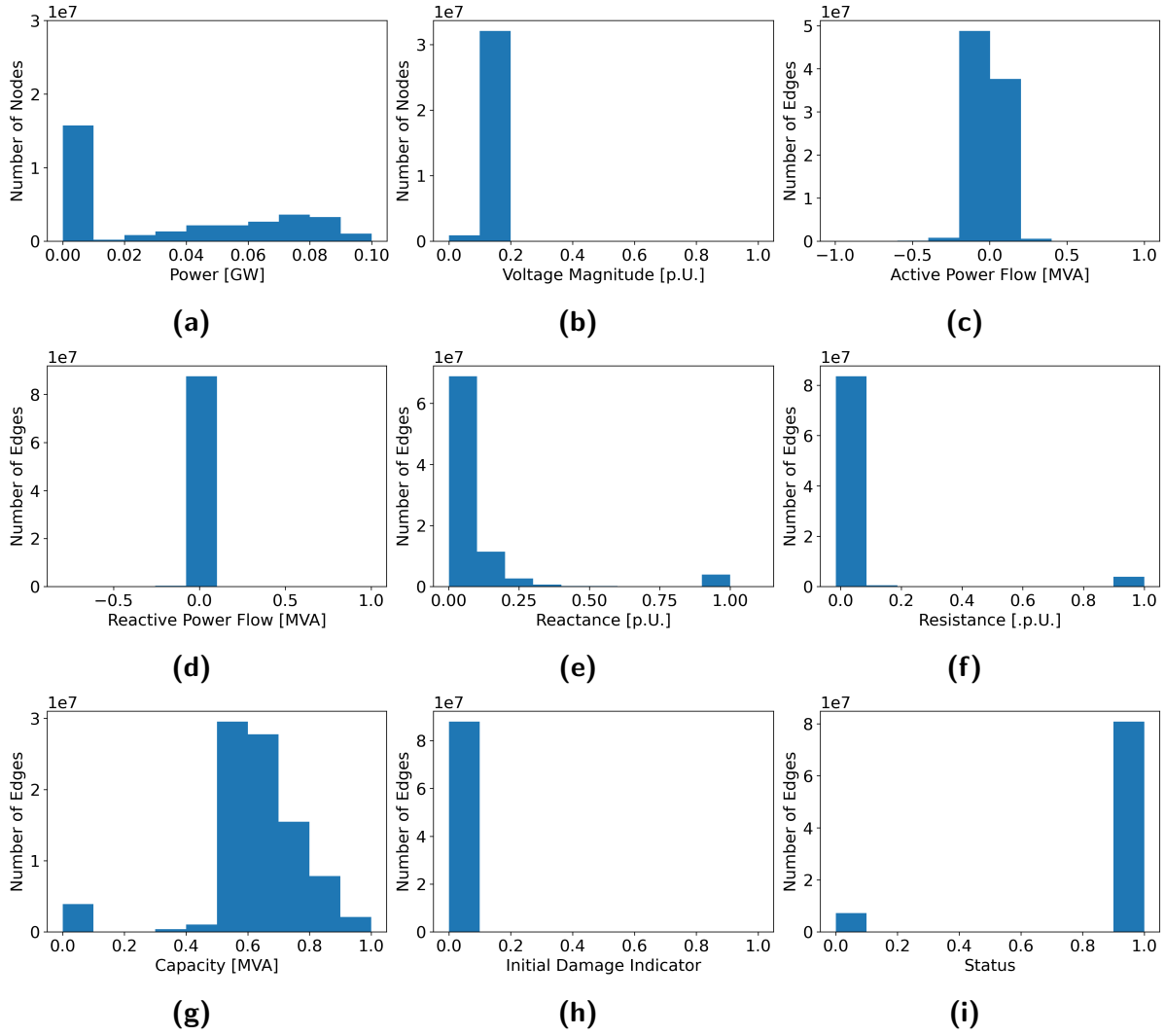
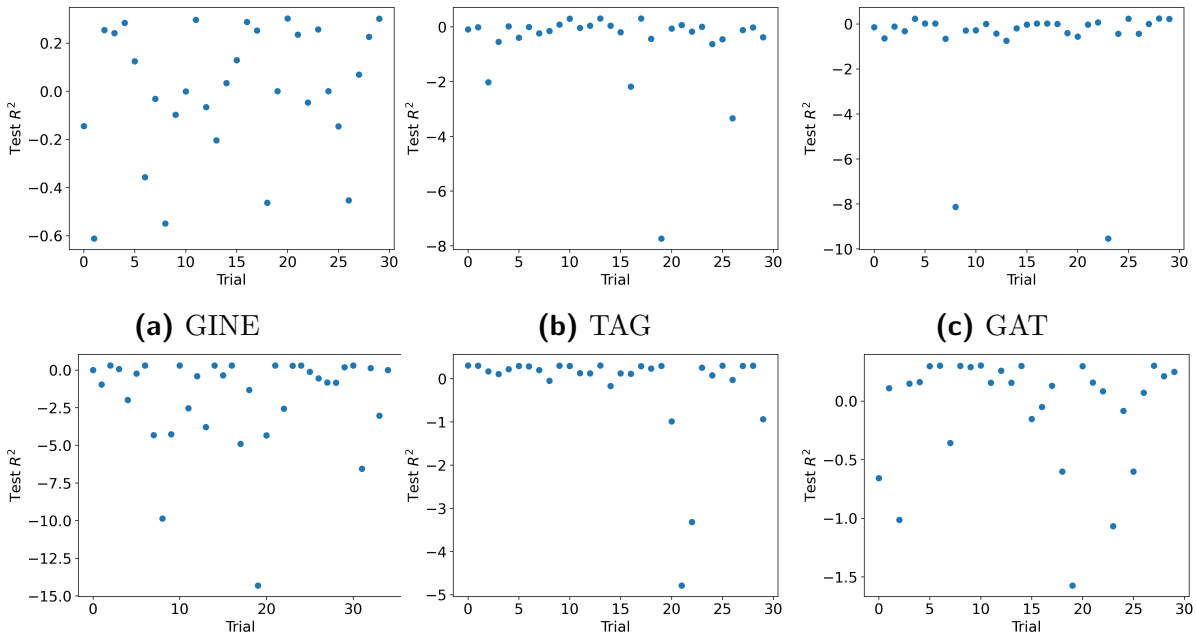


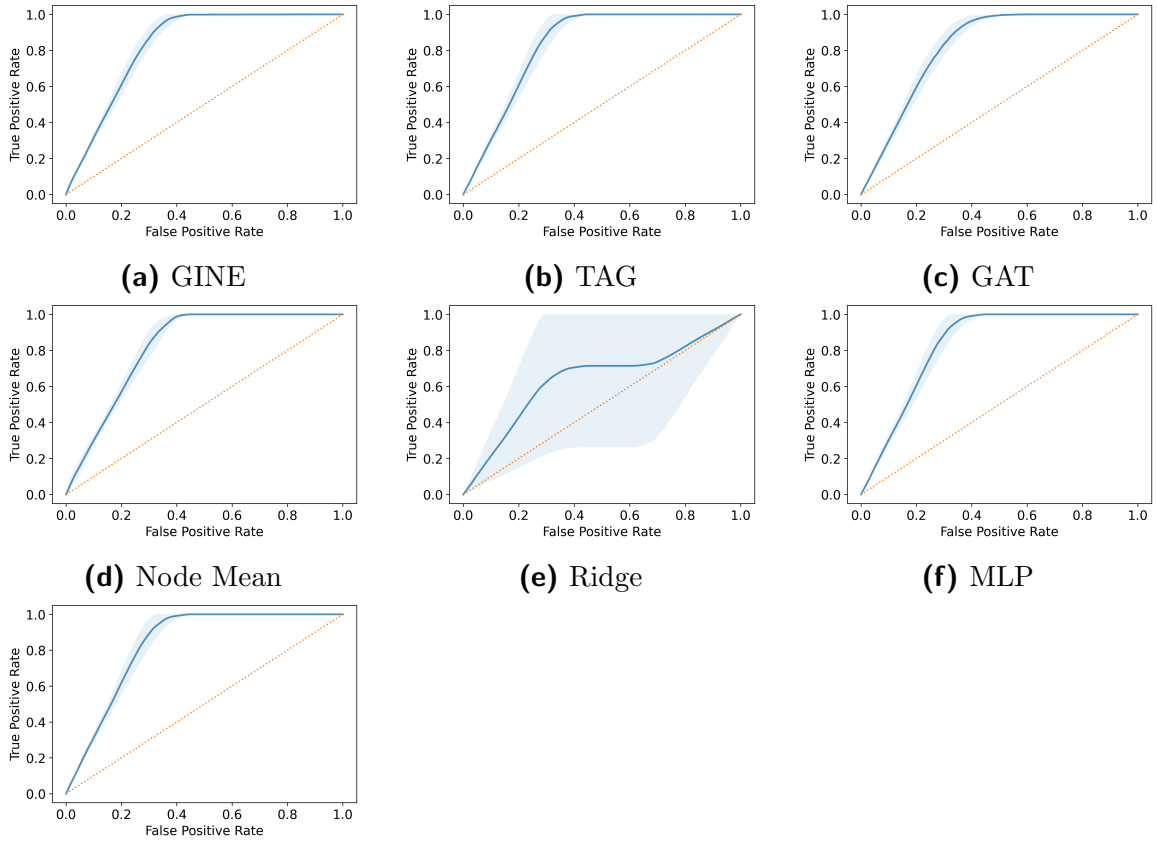

**Figure 7.2:** The figures show the distributions of the normalized node and edge features of the study set using all the hurricanes except Claudette as described in section 4.1.7. Figures 7.2a and 7.2b are node features and figures 7.2c to 7.2i are edge features. The active and reactive power flow (figures 7.2c and 7.2d) have been normalized using standardization with consecutive normalization whereas the rest has been normalized with logarithmic normalization except for status and initial damage indicator (figures 7.2h and 7.2i) which have not been normalized due to their binary nature.




**(a)** GINE **(b)** TAG **(c)** GAT

**(d)** Ridge **(e)** MLP **(f)** Node2Vec+MLP

**Figure 7.3:** The figures show the history plots of the hyperparameter studies (section 4.4) for the different models.

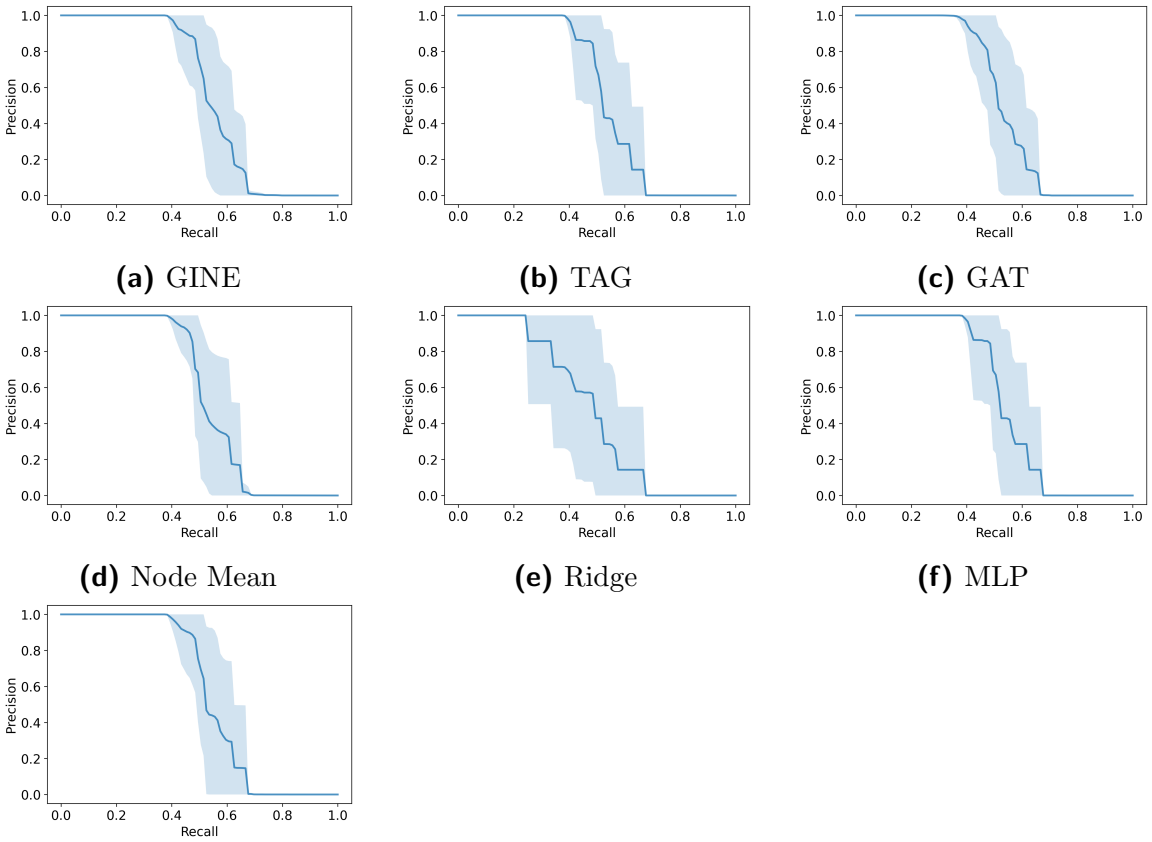



**(a)** GINE **(b)** TAG **(c)** GAT

**(d)** Node Mean **(e)** Ridge **(f)** MLP

**(g)** Node2Vec+MLP

**Figure 7.4:** The figures show the mean ROC curves with standard deviation of the 7-fold cross-validation of the different models.




**(a)** GINE **(b)** TAG **(c)** GAT

**(d)** Node Mean **(e)** Ridge **(f)** MLP

**(g)** Node2Vec+MLP

**Figure 7.5:** The figures show the mean precision recall curves with standard deviation of the cross-validation of the different models. For the binary classification every label with $y < 0.1$ is counted as positive label.

# Bibliography


[1] https://www.50hertz.com/en/grid/systemcontrol/systembalancing. accessed: July 2023.

[2] Ethem Alpaydin. Introduction to machine learning. *MIT press*, 2010.

[3] G. Aznar-Siguan and D. N. Bresch. Climada v1: a global weather and climate risk assessment platform. *Geoscientific Model Development*, 12(7):3085–3097, 2019.

[4] Janusz Bialek, Emanuele Ciapessoni, Diego Cirio, Eduardo Cotilla-Sanchez, Chris Dent, Ian Dobson, Pierre Henneaux, Paul Hines, Jorge Jardim, Stephen Miller, Mathaios Panteli, Milorad Papic, Andrea Pitto, Jairo Quiros-Tortos, and Dee Wu. Benchmarking and validation of cascading failure analysis tools. *IEEE Transactions on Power Systems*, 31(6):4887–4900, 2016.

[5] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[6] Jian Du, Shanghang Zhang, Guanhang Wu, Jose M. F. Moura, and Soummya Kar. Topology adaptive graph convolutional networks, 2018.

[7] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[8] J. Duncan Glover and Sarma. *Power Systems Analysis and Design*. PWS Publishing Co., USA, 1987.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. *Nature*, 521(7553):436–444, 2016.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[12] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020.

[13] Bukyoung Jhun, Hoyun Choi, Yongsun Lee, Jongshin Lee, Cook Hyun Kim, and B. Kahng. Prediction and mitigation of nonlocal cascading failures using graph neural networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(1):013115, 2023.

[14] Cheolmin Kim, Kibaek Kim, Prasanna Balaprakash, and Mihai Anitescu. Graph convolutional neural networks for optimal load shedding under line contingency. In *2019 IEEE Power  Energy Society General Meeting (PESGM)*, pages 1–5, 2019.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[16] Kenneth R. Knapp, Michael C. Kruk, David H. Levinson, Howard J. Diamond, and Charles J. Neumann. The international best track archive for climate stewardship (ibtracs): Unifying tropical cyclone data. *Bulletin of the American Meteorological Society*, 91(3):363 – 376, 2010.

[17] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning, 2018.

[18] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[19] Yuxiao Liu, Ning Zhang, Dan Wu, Audun Botterud, Rui Yao, and Chongqing Kang. Guiding cascading failure search with interpretable graph convolutional network. *ArXiv*, abs/2001.11553, 2020.

[20] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[21] Adilson E. Motter and Ying-Cheng Lai. Cascade-based attacks on complex networks. *Phys. Rev. E*, 66:065102, Dec 2002.

[22] Christian Nauck, Michael Lindner, Konstantin Schürholt, Haoming Zhang, Paul Schultz, Jürgen Kurths, Ingrid Isenhardt, and Frank Hellmann. Predicting basin stability of power grids using graph neural networks. *New Journal of Physics*, 24(4):043041, apr 2022.

[23] Matthias Noebels, Robin Preece, and Mathaios Panteli. Ac cascading failure model for resilience analysis in power networks. *IEEE Systems Journal*, pages 1–12, 2020.

[24] Tobias Ohlinger. Simulation of hurricane-induced cascading failures in the texan electrical network, 2022.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[26] Rezoan A. Shuvro, Pankaz Das, Majeed M. Hayat, and Mitun Talukder. Predicting cascading failures in power grids using machine learning algorithms. In *2019 North American Power Symposium (NAPS)*, pages 1–6, 2019.

[27] J. M. Stürmer. Investigating the risk of hurricane-induced cascading failures in power systems of the u.s. east coast, 2022.

[28] Texas AM University. Electric grid test case repository. https://electricgrids.engr.tamu.edu/electric-grid-test-cases/.

[29] Anna Varbella, Blazhe Gjorgiev, and Giovanni Sansavini. Geometric deep learning for online prediction of cascading failures in power grids. *Reliability Engineering System Safety*, 237:109341, 2023.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[31] James Winkler, Leonardo Dueñas-Osorio, Robert Stein, and Devika Subramanian. Performance assessment of topologically diverse power systems subjected to hurricane events. *Reliability Engineering System Safety*, 95(4):323–336, 2010.

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[33] Chun Zhang, Yansong Li, Zhihong Yu, and Fang Tian. Feature selection of power system transient stability assessment based on random forest and recursive feature elimination. In *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pages 1264–1268, 2016.

[34] Tianxin Zhou, Xiang Li, and Haibing Lu. *Power Grid Cascading Failure Prediction Based on Transformer*, pages 156–167. 12 2021.

[35] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.

# Acknowledgements

I would like to thank my supervisors Prof. Tobias Scheffer from the University of Potsdam and Dr. Mehrnaz Anvari from the Fraunhofer Institute for Algorithms and Scientific Computing SCAI for supervising this project. They have guided me well through the whole project and have always been available for questions.

Additionally I want to thank the working group of Nora Molkenthin and Frank Hellmann at the Potsdam Institute for Climate Impact Reasearch who were hosting me during the time of my thesis and presented a room to discuss ideas and doubts. A special thanks goes out to Christian Nauck who was my main contact for everything regarding the interface of Graph Neural Networks and Power Grids and who was always ready to answer my questions no matter of theoretical or technical nature.

I also want to thank the group of Prof. Tobias Scheffer, specifically Pedro Alonso Campana, Silvia Makowski and Paul Prasse who guided me through the procedure of conducting and writing a machine learning thesis and helped me a lot during the process of figuring out which step to do next. A special thanks goes out to Pedro who assisted me a lot with implementation and prototyping of new approaches.

Further more I want to thank my family for supporting me during all of my studies and assisting me wherever they can.

Finally I gratefully acknowledge the European Regional Development Fund (ERDF), the German Federal Ministry of Education and Research and the Land Brandenburg for supporting this project by providing resources on the high performance computer system at the Potsdam Institute for Climate Impact Research.

**Erklärung**

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die von mir angegebenen Quellen und Hilfsmittel verwendet habe. Wörtlich oder sinngemäß aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht. Die Richtlinien zur Sicherung der guten wissenschaftlichen Praxis an der Universität Potsdam wurden von mir beachtet. Eine gegebenenfalls eingereichte digitale Version stimmt mit der schriftlichen Fassung überein. Mir ist bewusst, dass bei Verstoß gegen diese Grundsätze die Prüfung mit nicht bestanden bewertet wird.

Potsdam, den 28. Juli 2023

(Tobias Ohlinger)