UiT The Arctic University of Norway

Faculty of Science and Technology, Department of Physics and Technology

DC-Approximated Power System Reliability Predictions with Graph Convolutional Neural Networks

Fredrik Marinius Haugseth

EOM-3901 Master's thesis in Energy, Climate and Environment 30 SP - June 2022

Abstract

The current standard operational strategy within electrical power systems is done following deterministic reliability practices. These practices are deemed to be secure under most operating situations when considering power system security, but as the deterministic practices do not consider the probability and consequences of operation, the operating situation may often become either too strict or not strict enough. This can in periods lead to inefficient operation when regarding the socio-economic aspects. With the continuous integration of renewable energy sources to the electrical power system coupled with the increasing demand for electricity, the power systems have been pushed to operating closer to their stability limit. This poses a challenge for the operation and planning of the power system. Research is therefore being invested into finding more flexible operational strategies which operates according to probabilistic reliability criteria, taking the probability of future events into consideration while also aiming to minimize the expected cost and defining limits for probabilistic reliability indicators.

To reliably plan and operate the systems according to a probabilistic reliability criterion, numerical problems such as the Optimal Power Flow (OPF) and the Power Flow (PF) equations are used. These tools are helpful as they are used to determine the optimal way of producing and transporting power. These tools are also used in contingency analyses, where the effect of occurring contingencies is analyzed and evaluated. Due to the non-linearity of the PF equations, the solution is often found through iterative numerical methods such as the Gauss-Seidel method or the Newton-Raphson method. These numerical methods are often computationally expensive, and convergence to the global minimum is not guaranteed either. In recent years, various Machine Learning (ML) models have gathered a lot of attention due to their success in different numerical tasks, particularly Graph Convolutional Networks (GCNs) due to the model's ability to utilize the topology and learn localized features. As the field of GCN is new, extensive research is being committed to identify the GCNs ability to work on applications such as the electrical power system.

This thesis seeks to conduct preliminary experiments where Graph Convolutional Networks (GCN) models are used as a substitution for the numerical DC-OPFs which are used to determine values such as the system load shedding due to contingencies. The GCN models are trained and tested on multiple datasets on both a system- and a node-level, where the goal

is to test the models' ability to generalize across perturbations of different system-parameters, such as the system load, the number of induced contingencies and different system topologies.

The experiments of the thesis show that the GCNs can predict the load-shedding values across multiple system-parameter perturbations such as the number of induced contingencies, increasing load-variation and a modified system-topology with a high accuracy, without having to be retrained for those specific situations. Though, the further the system-parameters were perturbated, the less accurate the model's predictions became. This reduction in accuracy per system-parameter perturbation was caused by a change in the load-shedding pattern as additional parameters were perturbated, which the models were unable to comprehend. Lastly, this thesis also shows that the GCN models are substantially faster than the numerical methods which they seek to replace.

Acknowledgments

First and foremost, I would like to extend a gratitude to my supervisors Sigurd Hofstad Jakobsen and Benjamin Ricaud for their excellent supervision over the course this thesis. Your guidance and knowledge within the field have been invaluable, and the recurring conversations we have had over the course of the year have been nothing but inspiring. I would also like to personally thank SINTEF Energy Research for the opportunity I was given to work on one of their amazing summer projects which sparked the inspiration for this thesis.

I would like to thank all my fellow classmates which has contributed to making the last 5 years a memorable time. Without the collective effort I do not think we would have managed to overcome the road bump that was Calculus 3. To my parents for always supporting me. Lastly, to my fiancée and my son for giving me the motivation to finish what has been 8 long years of studies.

Table of Contents

List of Tables

List of Figures

Abbreviations

AC	Alternating Current
ANN	Artificial Neural Networks
ConvGNN	Convolutional Graph Neural Networks
CNN	Convolutional Neural Networks
DC	Direct Current
FCL	Fully Connected Layer
GCN	Graph Convolutional Networks
GHG	Greenhouse Gases
MC	Monte Carlo
ML	Machine Learning
OPF	Optimal Power
PF	Power Flow
RES	Renewable Energy Sources
TSO	Transmission System Operator

Part I / Introduction

1 Introduction

1.1 Motivation

One of the biggest current global topics of debate is the looming climate crisis and the possible irreversible and calamitous consequences. Currently there is a global consensus among scientist that one of the causing factors for the arising climate crisis is the continuous emission of greenhouse gases (GHG) into the atmosphere [1]. The majority of GHG emission stems from sources such as industry, the transportation section and from the production of electricity and heat. In 2010, electricity generation was the source of 25% [1] of the average global carbon emissions.

In the coming years, the demand for electricity is estimated to increase in countries with developing economies [2], while in countries with more developed economies, the immediate demand for electricity have stagnated over the past years. However, with the continuous electrification of the society, such as within the transportation section [3], this stagnation might cease. In addition to the projected increase in energy demand worldwide, there is also a major shakeup happening within the energy section. In 2015, 196 parties signed a legally binding international treaty called the Paris Agreement, to limit global warming to well below 2 degrees compared to pre-industrial levels [4]. To reach the goals set in Paris Agreement, a drastic reduction of emission within all sectors in the coming decade is imperative. Hence, a heavy dismantling of the current energy-sources using carbon related fuel must take place within a short span of time, replacing the carbon-related energy sources with more sustainable and Renewable Energy Sources (RES).

This continuous growth of worldwide electricity demand combined with factors such as the continuous connection of new intermittent RES to the electrical power system over the last decade, has resulted in the current power systems being pushed to operating closer to their stability limits [5]. The deployment of intermittent energy sources, replacing the already existing energy sources, has led to the power grids being more exposed to disturbances such as voltage collapses or complete blackouts [6]. One of the major perks provided by energy-sources such as gas and coal, that is not prevalent in sources such as wind and solar, is a scalable flow of energy and the use of synchronous generators which provides inertia to the

system. The inertia stabilizes the grid-frequency, which is often used as an indicator for significant changes in either the supply or demand of power onto the grid. The demand and supply of energy is not constant over a longer period, and changes in supply or demand occurs frequently. The inertia allows for a quick extraction of stored energy, which in turn gives a brief leeway if there is a drop in production. This makes the grid more stable to fluctuation of production and consumption, resulting in a more stable power grid [7] thus making it easier to plan and schedule production and load according to an operational strategy.

The operation, planning and scheduling of production and transport of energy is performed by the Transmission System Operators (TSOs), following set operational strategies. The current standard operational strategy that is deployed by the TSOs within electrical power systems is done following deterministic reliability practices, such as the N-1 criterion [8]. The N-1 criterion demands that the system should be able to withstand a failure or an outage of a single component in the electrical power system, being a transformer, generator, load, or a line, such that the system can accommodate to the new operational situation without violating any of the security limits [9]. This deterministic criterion imposes a restriction on the system, resulting in the capability of the transmission lines not being fully utilized [10]. This can in periods lead to an increase in congestion, which can result in an increase in the socio-economic cost of operation and consumption. Intermittent RES such as Solar- and Wind parks are often located far away from the area of consumption, thus requiring large transmission cables to transport the energy from the area of production to the area of consumption. If the flow of the transmission cables is constrained due to a restrictive operational strategy, then the utilization of the energy produced by the energy sources is limited as well.

Under stable weather conditions, the N-1 criterion is a good criterion for most operating situations when considering power system security. Exceptions can be during periods of rough weather, such as a storm, where not even the N-1 criterion is cautious enough. Despite the security benefits, the deterministic criterions have a few drawbacks when considering socio-economic costs due to its restrictive operation strategy. Research is therefore being invested in finding alternative and more flexible Reliability Management strategies which operates according to a probabilistic reliability criteria [10] [11] [12]. Such a criteria would take the probability of future events into consideration, while also aiming to minimize expected societal costs and define limits for the probabilistic reliability indicators. This can

however prove to be a daunting task, as the number of possible optimal or semi-optimal operating states that exists is vast. Finding the optimal operating state which minimises the socio-economic cost can be a difficult and time-consuming process, as a change of a single variable can lead to hundreds of new possible operating states.

In the Operation Planning context of Reliability Management, the scheduling and planning of the power production is done on a short timeframe ahead, often on a day-to-day basis. The objective is to ensure that the power demand is met without breaching any system violations while being resilient towards possible contingencies and while minimizing the socio-economic cost of operation. Here, a contingency is defined as an unplanned outage of one or more primary equipment components, such as a line outage [13]. As such, the operation planning is dependent on flexible, quick, and accurate algorithms and simulation tools that can find the optimal operating state which optimizes the cost of operation for a given set of conditions in a short time frame, while adhering to the reliability indicators.

There currently exist various methods that are used for reliability assessment, such as Contingency Analyses. The Contingency Analyses simulates and analyses the impact occurring contingencies have on the electrical power system, using tools such as the Monte Carlo (MC) simulation. These tools are mostly used for research purposes but can give a decent indication of real-life events. These tools work by simulating the system, either sequentially over a time-period, or non-sequentially by provoking the system by inducing contingencies, either deterministically or following a probabilistic distribution. This allows for stress- and limit testing of the system. At each step in the simulation, the state of the system is evaluated, and the consequence of any occurring contingencies is analysed. The consequences are often evaluated based on parameters such as the amount of interrupted power and the duration of the interruption, the energy not supplied during the period, the shed load, or the rescheduling and redispatch of power due to contingencies.

To find the optimal operating state given a set of conditions and constraints, the simulations call on the Optimal Power Flow (OPF) problem to determine the optimal way of generating and transporting power while minimizing a set of variables, such as the generation cost etc, while subjected to constraints. To determine the optimal way of generating and transporting power, a set of Power Flow (PF) equations are used. The PF equations are derived from Kirchhoff's current law and are non-linear. Due to the non-linearity of the equations, the solution of the PF equations is often found through numerical approximation methods such as

the Gauss-Seidel method or the Newton-Raphson method [14]. For smaller networks, these iterative methods often converge after a reasonable time, yielding solutions for the PF equations. However, for larger networks the iterative methods can be slow, and convergence to the global optimum is also not guaranteed if the objective is to find the most optimal state. The iterative methods also suffer from high computational complexity when considering several contingencies and how they will affect the grid. The high computational complexity makes the iterative method slow, rendering them useless for screening methods.

In recent years, Machine Learning (ML) has become a prominent substitution for many numerical problems. Recently, various Artificial Neural Networks (ANNs) has been proposed to compute and solve a series of power-flow problems. This includes the Multi-Layered Perceptron (MLP) models, and more recently, advanced model such as Graph Convolutional Network (GCN) models. GCNs have seen a variety of success within different applications, such as document classification [15], a variety of chemistry and biology related tasks [16] [17] and various applications within the power system field [18] [19] [20] [21]. Some of the work in the mentioned papers is similar to the work being conducted in this thesis, however most of the papers use less advanced ANN-models, such as the MLP.

The disadvantage of using MLP models is that they are strictly data driven, meaning that no prior knowledge of the physics of the system nor the topology of the grid has been utilized. For electrical power system, the structure of the system is an essential feature which must be included in the process. It is therefore more natural to represent the data in a graph-format, retaining the information about the topological structure while allowing for the inclusion of features.

1.2 Aim and Objective

To evaluate the socio-economic cost of multiple operating states and contingency scenarios, simulation tools such as the Monte Carlo simulation are often used. The simulation tools deploy the use of OPFs at each iteration to evaluate the state of the system and to find the optimal operating state given a series of conditions and constraints. To achieve a realistic representation of the power flow, the simulations utilize Alternating Current (AC) -PF equations which are non-linear and non-convex. For large networks, the OPFs can be slow, and a solution is not always guaranteed. To alleviate these issues, the PF equations in the OPFs are often simplified by a set of assumptions, leading to the Direct Current (DC)

approximation. These simplification makes the PF-equations linear, guaranteeing a solution. While the DC-equations are faster to solve compared to the AC-equations, they are often too simple to accurately represent the dynamics of the power system. This is problematic, as either option comes with a drawback.

A proposed solution is to use ML-models as a substitution for the numerical estimations in the contingency analysis. Optimally, the ML-model should be inductive, meaning that the model is useable across a multitude of topologies, load conditions and contingencies. If the ML-models are non-inductive, scores of models must be trained and tested for every iteration of the power system. Models that can adapt to small perturbations of the set-up without having to retrain the model are therefore preferable.

To this end, this thesis aims to explore a simple initial approach, by replacing the DC-OPFs in a non-sequential Monte Carlo simulation which calculates the system load shedding values due to occurring contingencies, using advanced GCN-models. The GCN-models will be trained to estimate the load shedding values across multiple system perturbations, such as multiple contingencies, varying- load and -topology, to investigate the model's resilience and adaptability towards system-perturbations. The models will be using system-features that were prevalent before the last OPF in the simulation tool was called. If the ML-models are successful at reproducing the results from the DC-OPFs, the next potential step would be to investigate replacing the AC-OPFs with ML-models, or by introducing data drawn from a sequential simulation instead.

To summarize, the goal of this thesis is to construct multiple GCN-models that are to predict the load-shedding values across system-parameter perturbations, such as multiple contingencies, varying- load and -topology. Some models will also be sequentially trained on additional dataset with differing system parameters to investigate if sequential training has any effect on the models' predictive capabilities across system-parameter perturbations. The thesis will also be exploring a node-level prediction approach and a system-level prediction approach to determine if there are any benefits of doing either. Lastly, this thesis also aims to test out three prominent GCN-operators to determine the best operator.

To reach the goals of the thesis, the following experiments will be conducted. For the system-level, all four experiments are conducted, while only parts of experiment 1 – 3 are conducted for the node-level.

1) Experiment 1, Contingency Perturbation Analysis
    - Train a model for each operator on a dataset where a single contingency was induced at each simulation point. Investigate the models' predictive capabilities on datasets where additional contingencies have been induced at each simulation point.
    - Determine the best convolutional operator for the problems at hand by comparing the speed and accuracy of each operator on the three contingency datasets. Discard all but the best performing operator.
    - Investigate how the model's behaviour changes after sequential training on datasets with additional contingencies induced at each simulation point.
2) Experiment 2, Load Perturbation Analysis
    - Train a model on the 1-contingency dataset and then test the trained model on datasets where the system load values have been increased by adding a scaling factor k = 1.05, 1.10, 1.20 to the load for each of the simulated datasets, respectively. Investigate the model's predictive abilities across the load-perturbation datasets.
    - Investigate how the model's behaviour changes after sequential training on the datasets with the highest scaled load.
3) Experiment 3, Topology Perturbation Analysis
    - Train a model on the 1-contingency dataset with the original topology of the system and then test the model's predictive abilities on additional datasets where the topology of the test-system has been slightly altered and the number of induced contingencies per simulation point increases.
    - Investigate how the model's behaviour changes after sequential training on the datasets with the modified topology.
4) Experiment 4, Case Study
    - Perform a case-study where the speed and accuracy of the best-performing ML-model is tested against the OPF which the models seek to replace.
    - Investigate the impact the batch-size of the test-set has on the model's run-time.

1.3 Thesis Structure

The thesis is divided into five parts. Part I Background and Motivation, Part II Theoretical background, Part III Method and Model set-up, Part IV Results and Discussion, Part V Conclusion.

In Part I, the motivation and background of the problem at hand is introduced.

Part II gives a brief introduction to the theory behind Power System Reliability Analysis, Graph Theory, and various Machine Learning models.

Part III Introduces the system, its set-up, its features and how the data was generated using a non-sequential Monte Carlo simulation. The ML-model, its prediction level, parameters, features and how it was constructed is covered. The section also briefly talks about certain parameters such as the batch-size.

Part IV Presents the results of the experiments and discusses the results accordingly.

Part V Concludes the work done in the thesis and give some insight on possible future pathways.

Part II / Theoretical Background,

The theoretical section of the thesis will delve into some preliminary theory about the problem at hand and the means to solve it. Section 2 explains the goal and challenges of Reliability Management, while Section 3 introduces relevant ML- & Graph theory. The theoretical ML-section will not cover all the basic, internal process within the different ML-models. The reader is however encouraged to read up on the different processes if there are unclear points, however this should not strictly be necessary to understand the results of the thesis.

More specific, each sub-section is as following:

Section 2.1 Introduces the concept of Reliability Management and its context.

Section 2.2 Introduces the concept of reliability practices and the current deterministic approach.

Section 2.3 Introduces the concept of probabilistic reliability assessment.

Section 2.4 Introduces the concept of Contingency Analyses.

Section 2.5 Introduces the Sequential – and Non-Sequential MC simulation and motivates the use of non-sequential method.

Section 2.6 Introduces the Power Flow Analysis, the PF equations, and the AC- and DC-approach.

Section 3.1 Introduces the MLP – model.

Section 3.2 Introduces the Convolutional method and the Convolutional Neural Network model.

Section 3.3 Gives an extensive introduction to Graph theory.

Section 3.4 Gives a brief motivation for the use of Graph Convolutional Networks, defines the taxonomy used in this thesis and introduces the various prediction levels for GCNs.

Section 3.5 Introduces the Convolutional Graph Convolutional Networks sub-group and the Spatial and Spectral approaches.

Section 3.6 Introduces three convolutional operators, the GCN, GAT and GraphConv operators, all which are based on the Spatial and Spectral approaches.

2 Power System Reliability Analysis

2.1 Reliability Management

The overarching goal of power system reliability management is to plan, schedule and operate the system in a way such that the supply of electricity from the producer to the end-user is on a near continuous basis. This is often done in tandem with minimizing certain socio-economic cost values of operation, by means such as minimizing the possible interruptions that can occur over an extended period [9]. The TSO must often take decision as early as conveniently possible, ranging from just a few minutes ahead to decades ahead. To ensure that the decision taken are sufficiently reliable, the system is safeguarded through a set of reliability criterions which acts as a set of constraints. These criterions give the system an indication if the reliability level of the power system is sufficient and allows the operator to plan and schedule the supply of electricity accordingly.

Due to the varying time horizon, the decisions addressed in reliability management can be divided into a series of contexts depending on the time-horizon of the planning [21], as shown in **Figure 1**.

Figure 1: The different reliability management contexts and their corresponding time horizon *[21]*. This thesis will be focusing on operation planning.

Reliability management can further be classified into two brackets, assessment, and control. Reliability assessment is used to estimate the potential consequences of a decision over a period, such as the potential socio-economic cost and the corresponding reliability indicators [21]. Reliability assessment can be seen as a simulation step used to assess possible pathways and solutions for the problems at hand. Reliability Control on the other hand is used to

compute decisions in line with the reliability criterion over the period. When attempting to find the optimal decision, the set of potential candidates can be vast, making the control problem much more complex than the assessment problem. This introduces a trade-off, in which the time and accuracy must be balanced to reduce the cost of operation while ensuring that the reliability criterion is met.

2.2 Reliability Criteria

Traditionally, the operating and planning of power systems is done following deterministic reliability practices, such as the N-1 criterion. As mentioned in the introduction, the deterministic approaches act by a principle which states that the system should be able to withstand a failure or an outage of a single component in the electrical power system, being a transformer or a line, such that the system can accommodate to the new operational situation without violating any of the security limits. While often viewed as a safe reliability practice, this form of management has its shortcomings, especially considering the socio-economic costs of operation. In recent years there has been a surge of studies which proposes the use of a probabilistic reliability criteria as an alternative to the deterministic criteria [22]. The probabilistic reliability criteria aim at taking decisions which minimizes the expected cost of operation, rather than only minimizing the risk of operation. Such a transition will allow for a better balance between reliability and cost, leading to more cost-optimal planning and operation of the system.

One way of approaching the probabilistic reliability criteria is to define a set of preventive- and corrective measures which lead to the lowest expected system operational cost [23]. The preventive measures are a set of actions that are taken prior to the operation of the system. These measures aim to ensure that the system should adhere to a reliability criterion, based on the potential threat exposure and the predicted future system state [24]. Corrective measures define an action which ensure that the system's compliance with a reliability criterion based on both an observed and the estimated present system state, coupled with the threat exposure [24]. These actions are taken after contingencies have occurred to stabilize the system. An example of a preventive measure can be the restriction of transmission capacity between market areas. Such a restriction ensures that the flow is within the bounds set by the operating criterion following contingencies. Ample restrictions can result in high congestion costs, as the possibility of transporting energy from an area with surplus to an area with a shortage is

restricted. On the other hand, no restrictions of the flow of energy can lead to a loss of security, resulting in potential high cost of interruption.

2.3 Probabilistic Reliability Assessment

The probabilistic reliability assessments often base themselves on risk-based reliability indicators. These indicators attempt to directly assess the socio-economic- or physical risk for consumers, estimating the total cost of operation or the risk of possible contingencies. Typically, reliability analyses attempt to answer the following questions [25].

1) What can go wrong?
2) How likely is it to happen?
3) What are the consequences?

These questions are asked and analysed to assess how vulnerable the system is to the various potential threats, the probability of occurrence and the potential cost if any of the threats occurs. Potential threats to the power system can range from natural causes, such as a storm, earthquakes, or strong weather, to human errors during planning and/or operation, components in the power system wearing down or even terror and/or sabotages. The analysis also evaluates the system's susceptibility and coping capacity, checking the likelihood of a contingency cascading out of control and how effective the system deals with the cascading threats, which can lead to disruptions and high socio-economic costs.

2.4 Contingency Analysis

To evaluate the consequences potential contingencies can have on the system, a contingency analysis is performed. The contingency analyses can be performed either sequentially over a period, or non-sequentially, using simulations tools to analyse the system under varying conditions, such as different load- and generation conditions and/or by provoking contingencies to occur. At each simulation step, the state of the system is evaluated, and the consequences of any occurring contingencies are analysed [26] [27]. The consequences for each contingency are often evaluated based on socio-economic factors such as the amount of interrupted power and the duration of the interruption, the energy not supplied during the period or the rescheduling and redispatch of power due to contingency [28]. The contingency analyses allow the operator to stress the system, test out different preventive methods and to find the optimal corrective approaches when contingencies occur in the system. The analyses

can also reveal if any occurring contingencies led to a system overload, which often results in the rescheduling of generated power and the shedding of load to accommodate to the new system state. The contingency analyses can be used both for short-term and long-term planning. For long-term planning, the simulations will over time reveal what areas are most vulnerable to occurring contingencies, by evaluating socio-economic factors such as the shed load and the rescheduled power.

The two main tools used to select the contingencies for the contingency analyses are the Monte Carlo Simulation Method and the Analytical Method [29]. In the Analytical Approach, the contingencies are selected through a screening technique. The contingencies are then based on set failure criteria [30]. The analytical approach allows for handpicking of the most severe contingencies. The MC approach selects the contingencies from random samplings rather through a manual selection process [31], opening for numerous potential combinations of contingencies.

Both suggested methods have their strengths and weaknesses. Not all potential threats to the system are as likely to occur, and not all the contingencies are as potentially dangerous for the system. Some contingencies have a larger impact on the cost-of operation compared to other. However, using a method such as the Analytical Approach requires expert knowledge within the field to determine which contingencies are more important. As the Monte-Carlo simulation chooses the contingencies at random, knowledge about the contingencies is therefore not necessary. This thesis will therefore be focusing on using the Monte-Carlo approach for quicker screening of multiple potential contingencies.

2.5 Monte Carlo Simulation

In this thesis, a distinction is draw between two types of MC simulation methods, the sequential and non-sequential MC simulation.

2.5.1 Sequential Monte Carlo Simulation

In the sequential simulation, select components in the system, such as the generators and inter-connecting lines, is modelled with a probability of failure. The simulation then attempts to draw the time to failure for each component. One way of doing this is to base the time to failure in a set distribution, such as an exponential distribution. The sequential analysis then does a time-step simulation, such as every hour, until a contingency occurs in the system.

After one or several components in the system have been marked for failure, the system disconnects the component(s) to repair them. Similar with the failure-estimation, the repair-estimation can be estimated by an exponential distribution just with different base parameters. After the component(s) have been disconnected, the system is then simulated until the component(s) has been repaired. During this time-period in which the component(s) are being repaired, other components can fail, resulting in a disconnection of multiple components. This form of modelling allows for several contingencies to occur simultaneously and is a more natural way of simulating the system.

The sequential simulation follows a time-step, in which each datapoint in the simulation is dependent on the previous time-step and the events that occurred at that time-step. This means that each succeeding data point is correlated with the previous data point. If the data were to be used for a ML-model, the model would also have to consider the dependency of each data point. This would drastically reduce the number of potential models that would be compatible with the data.

2.5.2 Non-Sequential Monte Carlo Simulation

As opposed to the sequential method, the non-sequential MC simulation does not perform a time step simulation. The model therefore does not model the failure-rate of the components following a time-to-failure distribution. Instead, the non-sequential simulation deterministically induces contingencies and analyses the state of the system based on the occurred contingency. The choice of contingencies can either follow a procedural list or be chosen at random.

The non-sequential simulation allows for easier testing of contingencies and for screening of multiple contingencies and their effect on the system. As the contingencies can be chosen deterministically, it is easier to couple contingencies to explore the effects. As the non-sequential data is time-independent, each simulation step is independent of each other. This opens the number of ML-models that can be used for the problem.

The difference between the non-sequential and the sequential MC-simulation can be seen in **Figure *2***, where **Figure *2*** a) depicts the non-sequential procedure, while **Figure *2*** b) depicts the sequential procedure. Each dot represents the status of a line, with a black dot indicating an occurring contingency while a white dot represents a normal state. For the sequential figure, arrows indicate a change of status.

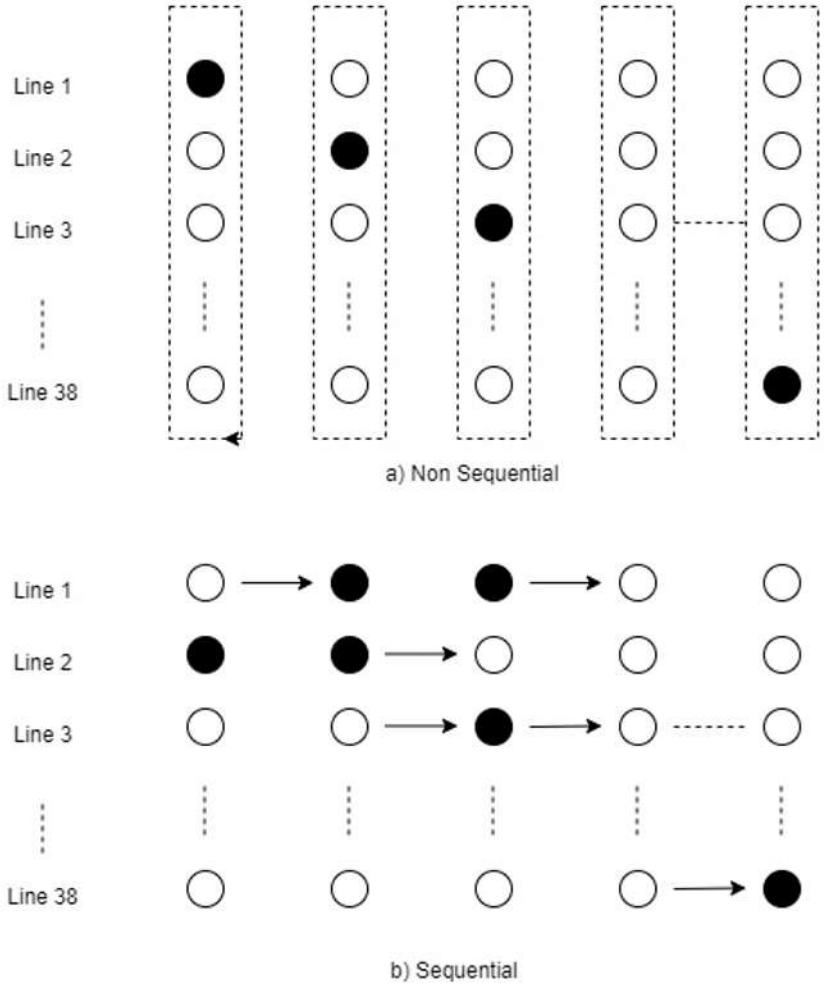


Figure 2: Illustration of a) A non-Sequential and b) Sequential MC simulation. Black dots indicate an occurring contingency for the affected line, while a white dot represent a normal state. Arrows indicate a change in status.

2.6 Power Flow Analysis

At each simulation step, the OPF and the PF equations are used to evaluate the state of the system and to analyse the effect of any potential contingencies in the contingency analysis.

2.6.1 Optimal Power Flow

The OPF is a power flow optimization problem which attempts to determine the currents, voltages, and the real and reactive power flows in a system under a given load condition while minimizing a set of variables, such as generation cost, loss, emission etc. [32] [33]. The OPF also ensures that the system constraints are not violated when finding the optimal solution, being cable current limits, voltage magnitude limits on generator or load nodes etc. A basic formulation for the OPF and its constraint is shown below.

$$\min \sum_{i \in gen,sgen,load} P_i * f_i(P_i)$$

$$Subject\ to$$

$$Load\ flow\ equations$$

$$Branch\ Constraints$$

$$Bus\ Constraints$$

$$Operational\ Power\ Constraints$$

There exist various variants of the OPF, each with a predefined objective. The goal of the OPF used in this thesis was to minimize the socio-economic cost, such as the cost of shedding load, while ensuring that the line and production constraints were met.

2.6.2 Power Flow Equations

The PF equations are used to find the voltage angle and magnitude for the buses in the system, based on the load conditions and the real power and voltage conditions of the generators. Once these values are estimated, both the real and reactive power flow for each branch and the reactive power output of each generator can be analytically determined. The PF-function can also be used to indicate any overloads in the system after a contingency has occurred. If an overload occurs, the OPF is called to optimize the generation and flow of the power based on the new system state.

As the OPF is called multiple times per iteration of the Monte Carlo simulation, it must be quick to solve, accurate and guarantee a solution for the system state. As the OPF relies on the PF equations, the same holds true for the PF-equations. The most realistic representation of the PF equations are the Alternating Current (AC) equations, while the Direct Current (DC) equations are a more simplified representation. The DC-representation uses various estimations and assumptions to simplify the equations and to reduce the complexity and computation time. As the OPF relies directly on the PF, the simplifications from AC to DC are made to reduce the complexity, such that the calculation is faster and that the system guarantees a solution, thus making it solvable.

2.6.2.1 AC Power Flow Equations

The AC power flow equations assumes that the electrical power system is in 'steady state' and attempts to find the system state in which power equilibrium is achieved for a given load and the amount of power generation in the system. The power system is not technically in a steady

state as there are always external factors such as changes in the load, switching actions and weather conditions etc. which makes the system non-static. However, as these variations often are small in a short time frame, the not-time varying model of the power system can be justified. By assuming steady state, the oscillation of the voltage is not accounted for, nor any transition between states. This simplification therefore rules out any possible violations of the voltage oscillating past the operation criterion. As such, the ability to model a potential fault in the system due to voltage oscillations is lost.

For a simple Power Flow model, the active and reactive powers flowing from bus *i* to *j* can be defined as shown in Equation ***(1)*** [34]. Here, p_{ij} and q_{ij} is the active- and reactive flow from bus i to j, respectively, r_{ij} is the resistance of the line connecting bus i and j, v_i is the bus voltage and δ_{ij} is the voltage angle difference between the buses.

$$\begin{aligned} p_{ij} &= \frac{1}{r_{ij}^2 + x_{ij}^2}\left[r_{ij}\left(v_i^2 - v_i v_j \cos\left(\delta_{ij}\right)\right) + x_{ij}\left(v_i v_j \sin\left(\delta_{ij}\right)\right)\right] \\ q_{ij} &= \frac{1}{r_{ij}^2 + x_{ij}^2}\left[x_{ij}\left(v_i^2 - v_i v_j \cos\left(\delta_{ij}\right)\right) + r_{ij}\left(v_i v_j \sin\left(\delta_{ij}\right)\right)\right] \end{aligned} \quad \textbf{(1)}$$

As can be observed from Equation ***(1)***, the AC power flow equations are non-linear and non-convex. Because of the non-linearity of the power flow problem, the power balance equations cannot be solved analytically. Instead, iterative solutions such as the Gauss-Seidel iterative method and the Newton-Raphson method are used. Out of these two methods, the Newton-Raphson method is the most common. However, none of these iterative methods can guarantee convergence. As the number of buses and branches in the network increases, the AC-power flow equations become increasingly computationally expensive and time consuming. Further simplifications are therefore preferred, leading to the DC-power flow estimation.

2.6.2.2 DC Power Flow Equations

For the DC power flow equation, the equations have been linearized through the following assumption [35].

1) The line-resistance is negligible, thus making $r_{ij} \approx 0$. If we apply this to Equation ***(1)***, we get Equation ***(2)***.

$$p_{ij} = \frac{1}{x_{ij}}\left(v_i v_j \sin\left(\delta_{ij}\right)\right)$$
$$q_{ij} = \frac{1}{x_{ij}}\left(v_i^2 - v_i v_j \cos\left(\delta_{ij}\right)\right) \quad \textbf{(2)}$$

2) The bus voltage magnitudes V_i to be approximated to 1 per unit (p.u). If we apply this to Equation ***(2)***, we get Equation ***(3)***.

$$p_{ij} = \frac{\sin\left(\delta_{ij}\right)}{x_{ij}}$$
$$q_{ij} = \frac{1 - \cos\left(\delta_{ij}\right)}{x_{ij}} \quad \textbf{(3)}$$

3) During certain conditions, such as light load conditions, the difference in voltage phasor angle between two buses connected by a line is assumed to be small. This assumption makes the voltage phasor angle to be estimated as $\sin \delta_{ij} \approx \delta_{ij}$ and $\cos(\delta_{ij}) \approx 1$. By applying these assumptions to Equation ***(3)***, we get Equation ***(4)***.

$$p_{ij} = \frac{\delta_{ij}}{x_{ij}}$$
$$q_{ij} \approx 0 \quad \textbf{(4)}$$

The linearization of the system allows for an analytical solution of the problem, guaranteeing that the system converges, allowing for much faster estimations. The drawbacks of the linearization are heavy simplifications of the power flow, resulting in the DC approximation being inaccurate as information about the reactive flow and the voltage is lost. As such, there is no feasible way to check if the voltage in the system violates the system operational requirements. Despite these shortcomings, the DC PF-equations are preferred over the AC PF-equations due to the improved calculation speed.

3 Machine Learning

The following section will cover some preliminary theory behind the most popular Artificial Neural Networks (ANN) and graph theory.

Section 3.1 introduces the most basic form of ANNs, being the Multilayer Perceptron (MLP).

Section 3.2 introduces the Convolutional Neural Networks (CNN) and the convolutional process.

Section 3.3 Introduces Graph Theory and important concepts.

Section 3.4 introduces Graph Convolutional Networks and the taxonomy used for this thesis.

Section 3.5 Introduces the Convolutional Graph Neural Networks (ConvGNN) and its subgroups, the spatial- and spectral approaches.

Section 3.6 Introduces three popular operators based on the ConvGNN approaches.

3.1 Multilayer Perceptron

Within the field of ML, there exist a broad variety of different types of ANNs useable for numerous classification and regression tasks. The most basic ANN model is the MLP model, which has shown profound strength as a predictive tool for various tasks, both regressive and classification.

Integral for the structure of all MLP models, are pre-defined set of connected layers, each containing a set of neurons which represent the feature space at each specific layer. The structure of MLPs can be generalized into three main layers: an input layer, a set of inner layers often called hidden layers, and an output layer, as shown in **Figure 3**. The neurons in each layer are the computational units of the MLP model which estimates an output based on the sum of the weighted input signal from previous neurons and a bias. A standard design for MLP-models is to have each neuron in layer l being connected to every neuron in the proceeding layer $l + 1$. This type of MLP-model often goes under the name of a Fully Connected Layer model (FCL). For each connection between the neurons, there is a trainable weight-value attached. This weight-value determines the strength of the connection and allow the model to recognize patterns. MLP models are flexible in its way as it can be designed and specified layer-wise.

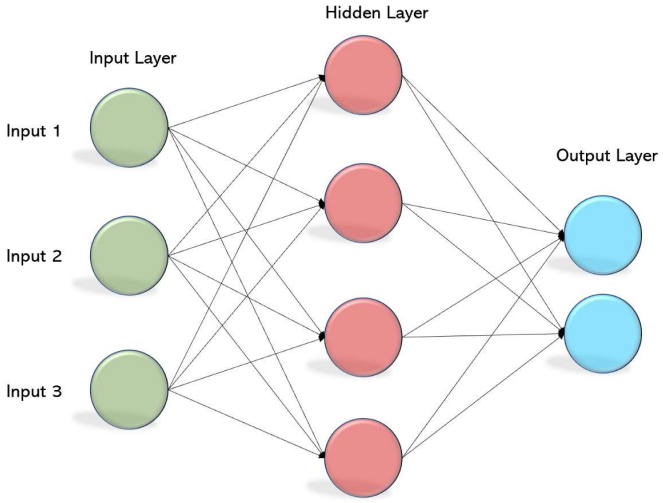


Figure 3: A general architectural structure for an MLP-model *[36]*. Depending on the task of the model, the output layer can be designed to have one or several neurons.

Except for the input layer, each neuron often has a non-linear activation function tied to it, allowing the MLP-model to separate and classify non-linearly separable data. The activation

function is a mapping of the weighted input to the neurons and is responsible for telling the neuron whether to send a signal to its connected neurons or not. The core machinery of an MLP-model is governed by a process called feedforward. The mathematical formulation of the feedforward process [37] is given in Equation ***(5)***.

$$\begin{aligned} \boldsymbol{h}^{(0)} &= \boldsymbol{x} \\ \boldsymbol{h}^{(l)} &= \sigma^{(l)}\left(W^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \;\; l = 1,2,\dots L \\ \boldsymbol{y} &= \sigma^{(L+1)}\left(W^{(L+1)}\boldsymbol{h}^{(L)} + \boldsymbol{b}^{(L+1)}\right) \end{aligned} \tag{5}$$

Here, $\boldsymbol{x} \in \mathbb{R}^{F_0 \times 1}$ is the feature input to the MLP-model, $\boldsymbol{h}^{(l)} \in \mathbb{R}^{F_l \times 1}$ is the hidden-state feature vector at the *l*-th layer, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{F_l \times F_{l-1}}$ is the weight-matrix at the *l*-layer, $\boldsymbol{b}^{(l)} \in \mathbb{R}^{F_l \times 1}$ is the bias-vector at the *l*-th layer, $\sigma^{(l)}$ is the activation function at the *l*-th layer and $\boldsymbol{y} \in \mathbb{R}^{F_{out} \times 1}$ is the output-vector from the output layer. The dimension F_l of the various vectors are equal to the number of neurons in each layer.

3.2 Convolutional networks

While the MLP-models can be used on a broad number of tasks, the MLP-models have no information about the local structure or the geometry connecting the features, which may contain essential information. For some problems at hand, local information is cardinal as the information it provides can be decisive for the success of the ML-model. It can therefore be advantageous to utilize ML-models that can learn and utilize both the geometry of the data and the feature values of the input data. Convolutional Neural Networks (CNNs) are networks designed to work on data with a Euclidian grid-like structure and local parameters [38]. Due to the structure of the input, where classification must be invariant to the translation (or other geometric transformation) of important patterns, the CNNs are often used in more advanced computer visions tasks such as image classification [39], speech recognition [40] and object detection [41].

3.2.1 Feature Learning

To estimate and obtain the local feature-information, the CNN models utilize an operation called convolution in its feature learning process. The convolutional method is a form of linear operation which takes a tensor, a multidimensional array of data as an input, and performs a convolution with a multi-dimensional weight matrix called a kernel or a filter. To obtain the local feature-information, the kernel or the filter in the CNN models is often made

to be smaller in dimensional size compared to the input. The convolutional method takes two functions as input and produces a third function. This third function explains how the shape of one function affects the shape of the other. Equation **(6)** displays the convolutional process between two arbitrary functions *f* and *w*.

$$(\boldsymbol{f} * \boldsymbol{w})(\boldsymbol{t}) = \int_{-\infty}^{\infty} f(\tau) w(t - \tau) d\tau \quad \textbf{(6)}$$

The result of this convolutional process is a trainable feature map containing the summarized information of the feature presence in the input. This process can be thought of the network reusing local filters with learnable parameters, by applying the filters to all input positions [42]. This can also be viewed as a 'sliding' filter, where the kernel is sliding across the input, calculating the convolution at each step. In the MLP models, each neuron in a layer is connected to every neuron in the succeeding layer. This is not the case for CNN models, where only a fraction of the neurons (the neighbours in the geometrical structure) in the current layer connects to the neuron in the succeeding layer. This method allows for local connectivity, making it easier to find patterns despite its location in the data.

3.2.2 Pooling Layers

After the feature map is created from the convolutional step, a pooling method, such as Min-, Max- or Average pooling, is often used to produce new and more condensed feature maps from the convolutional layer. This reduces the spatial dimension and the computational cost by lowering the number of parameters. The pooling layers also has the effect of reducing the chance of the model overfitting to the training data [43]. This process can then be repeated by adding additional layers to the model.

3.2.3 Classification

Depending on the desired outcome, the final process of the CNN model can be classification by SoftMax or a numerical output. This is done by adding one or several flatting fully connected MLP-layers at the end of the network. This process flattens the input from a higher dimension to 1D. **Figure 4** display a general architecture of a CNN model that takes an image as an input and classifies the image accordingly.

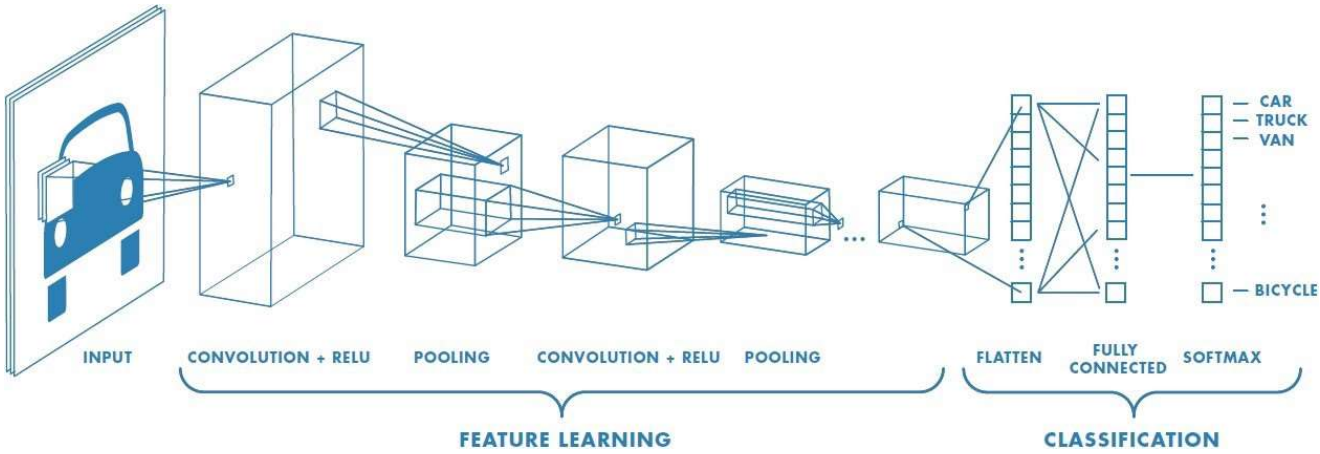


Figure 4: An example of a general architectural structure for an image classification CNN model *[44]*. **The model takes an image as an input, performs feature learning, and then classifies the image accordingly by flatting the feature map and by using a SoftMax function.**

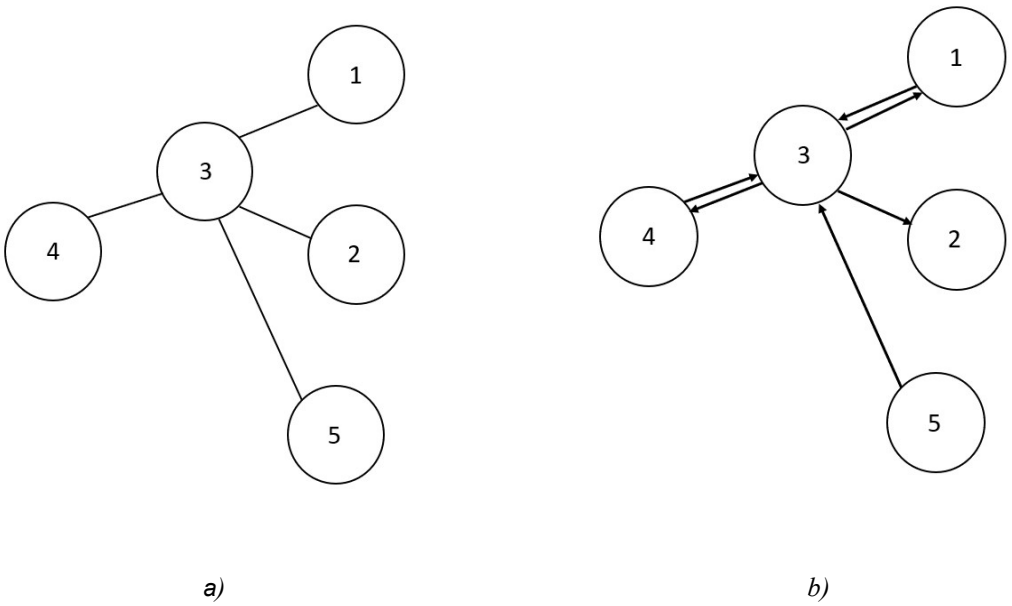
3.3 Graph Theory

In some cases, it is preferable to preserve the geometric structure of the data, as the structure itself retain essential information. For electrical power systems, the structure of the system is essential, and disregarding it would remove an important feature. This is problematic as it is not possible to construct a valid Euclidean input of the topology without the loss of the geometric structure. Hence, a more natural way of modelling the system instead would be as a mathematical graph. This representation would preserve the geometric structure of the system, while also allowing for the inclusions of features. To better understand how the convolutional method on a graph structure works, some preliminary within Graph theory is necessary. In this section, the standard graph structure and the different methods of representation is presented.

A graph is a mathematical structure used to model pairwise relations between objects. A graph is made of a set of nodes (**V**), connected by edges (**E**) as seen in Equation **(7)**

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{\phi}) \tag{7}$$

Where $\boldsymbol{\phi}: \boldsymbol{E} \rightarrow \{(x, y) | (x, y) \in V^2 \; and \; x \neq y\}$ is the incidence function mapping every edge to an ordered pair of vertices. *Graphs* can be either undirected *graphs*, where edges link two nodes symmetrically, or directed *graphs*, where edges link two nodes unsymmetrically [45]. **Figure** ***5*** illustrates the difference between an undirected graph vs a directed graph. Each node in a graph can have multiple edges, representing several connections between a single node or multiple nodes. This often goes under the general term of **directed multigraph or undirected multigraph**.



a) *b)*

Figure 5: a) Example of an undirected graph. b) Example of a directed graph with an asymmetric adjacency matrix.

A *graph* can also be fully represented in a more convenient and mathematical form by its adjacency matrix $\boldsymbol{A}$ and the degree matrix $\boldsymbol{D}$. The adjacency matrix $\boldsymbol{A}$ is a $n \times n$ square matrix, where $\boldsymbol{A_{ij}}$ specifies the number of connections from node *I* to node *j*. For the graphs in **Figure 5**, the adjacency matrixes are as following

$$a)\ A_{undir} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad b)\ A_{dir} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

As can observe from the adjacency matrixes, the adjacency matrix for the directed graph is non-symmetrical as some of the nodes only have edges pointing a single direction, while the adjacency matrix for the undirected graph is symmetrical.

The adjacency matrix $\boldsymbol{A}$ can be further modified by adding a connection from node *I* to node *I,* defined as a self-loop, giving the matrix $\widehat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. Here, $\boldsymbol{I}$ is the identity matrix. This alteration of the adjacency matrix allows a node to pass information to itself.

The degree matrix ***D*** is a $n \times n$ diagonal matrix which describes the degree of each node, giving information about the number of edges connected to each node. The corresponding degree matrix to the self-looped adjacency matrix is shown in Equation ***(8)***.

$$\widehat{\mathbf{D}}_{\mathbf{ii}} = \sum_{\mathrm{j}=0} \widehat{\mathbf{A}}_{\mathbf{ij}} \tag{8}$$

The degree-matrixes for the graphs in **Figure 5** are the following.

$$a)\ D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad b)\ D_{in} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

An important note for the degree matrix for the directed graph is that the direction of the degree must be specified beforehand, as the degree can symbolize an edge going either *out* of the node or *in* to the node. The correct notation would be to label the undirected degree-matrix as D, the directed degree-matrix of outgoing edges as D_{out} and the directed degree-matrix of ingoing edges as D_{in}.

When working with graphs it can be of interest in knowing the neighbouring nodes to a specific node, I.E., the connected nodes for each node. For a node v, the connected nodes are often defined as 'The Neighbourhood' of the node. In an undirected graph, the neighbourhood is denoted as $\mathcal{N}_u$, and is defined as a subgraph of the set of nodes which have connecting edges to node v. The mathematical formulation is $\mathcal{N}_u = \{z \mid \{v, z\} \in E\}$. The neighbourhood for each node can be determined directly from the adjacency matrix. For two connected nodes, such as v & z, if $A_{vz} = A_{zv} = 1$, the two nodes are defined as being in a neighbourhood. If the neighbourhood does not include the node v itself (self-loop), it is defined as an open neighbourhood. Otherwise, if the neighbourhood does include the node v, it is defined as a closed neighbourhood. Like with the degree-matrix, one can also distinguish between the in-going neighbours and the out-going neighbours of a node. The neighbourhood for node 3 in the undirected graph in **Figure 5** is $\mathcal{N}_3 = \{1, 2, 4, 5\}$, while for the directed graph the in- and out neighbourhoods are $\mathcal{N}_3^{in} = \{1, 4, 5\}$, $\mathcal{N}_3^{out} = \{1, 2, 4\}$.

The graph can be further extended by defining a feature-vector *k* to each node in the graph. These feature vectors can describe the current state or the properties of the node.

Similarly, each edge in the graph can have a corresponding feature-vector $\boldsymbol{e}$, describing the state/properties of the edges. These features can either be weights describing the strength of the connection, or a feature describing the state of the connection (active / disabled).
Weighted graphs are used when pairwise connections have different weights, giving each connection a magnitude of importance. The node-wise formulation for the edge-weight degree is given in Equation ***(9)***.

$$\hat{d}_i = \sum_{j \in \mathcal{N}(v) \cup \{i\}} e_{j,i} \tag{9}$$

Where $e_{j,i}$ denotes the edge weight from source node *j* to target node *I* [15]. A weighted graph will be reflected in the adjacency matrix, where each '1' value is replaced with the corresponding edge weight.

3.4 Graph Convolutional Networks

As the graph-setup is non-Euclidean and has a varying structure, the traditional convolutional method introduced the CNN section is not compatible. Furthermore, ML-models that do not utilize graph-inputs assume that instances are independent on each other [46]. This is not valid when working with graphs, as a connection between nodes directly correlates with dependency. Hence, a modified version of the convolutional method is required to work with graphs. The terminology for a model which work strictly on non-Euclidean graph-input rather than traditional Euclidean input values is a Graph Convolutional Network (GCN). Similar as with CNN models, GCN models have Feature Learning steps, Pooling Layers, and linear output layers.

3.4.1 Prediction Level

The success of the GCN models stems from the added complexity gained from adding the topological structure, and the GCN models being highly modifiable, allowing them to work at various levels of classification and regression, such as Node-level, Edge-level, and Graph-level.

- At Node-level, the analytic task is to predict and determine the node value for classification or regression by neighbourhood representation. This can either be classification to determine what the node represent, or regression for a set of feature-value(s).
- At Edge-level, the goal is to predict or classify the strength of an edge between two nodes in a graph. The edge-level prediction can also be used to check if an edge should exist.
- At Graph-level, the tasks include graph-classification, graph clustering or graph embedding.

  For graph-clustering, the goal is to divide the graph into a set of clusters, like other clustering methods such as K-means. The division can be based on the edge-weights, the edge distance between nodes or the neighbourhoods.

  For graph classification, the goal is to classify the graphs based on its current setup and features.

For graph embedding, the models map and compress the graph input into a vector while preserving the information of the graph. The size of the vector can be chosen by the user.

3.4.2 GCN Taxonomy

Within the field of GCN, there is no defined unanimous taxonomy. As the field is rapidly evolving due to extensive research being done, methods may quickly become outdated, and general terms may change based on new discoveries. This lack of a universal taxonomy can therefore lead to confusion, as similar convolutional methods and techniques can be presented with different names. As such, this thesis will be following the presented taxonomy in [46], which categorizes the GCN models into 4 main groups: Recurrent Graph Neural Networks (RecGNNs), Convolutional Graph Neural Networks (ConvGNNs), Graph Autoencoders (GAEs) and Spatial-temporal Graph Neural Networks (STGNNs). Within some of the presented main groups there are also subgroup divisions.

3.5 Convolutional Graph Neural Networks (ConvGNN)

In this thesis, the scope has been on the ConvGNN group. The ConvGNN group has two main approaches, the Spatial- and the Spectral approach [47]. The spectral approach bases its approach on techniques found in the signal-processing field, while the spatial approach bases its approach on a form of message passing between the nodes in the graph. In the following two subsections, both approaches are introduced.

3.5.1 Spectral Approaches

The Spectral approaches of the ConvGNNs work by defining a spectral representation of the graph. This form of representation has seen extensive use within the field of Signal Processing, where the graph-convolution utilizes various filters to remove noise from graph signals. A general assumption is also that the graph is undirected, resulting in the Adjacency matrix being symmetric. When going from a standard domain to the Fourier domain, a convolutional operation becomes a multiplication operation, which is often easier to compute. Spectral Approaches utilize this through graph Fourier transforms, which are the eigenvectors of the graph Laplacian [48]. Here, the graph Laplacian matrix is estimated by

$$\boldsymbol{L = D - A}$$

where $\boldsymbol{L}$ is the Laplacian matrix, $\boldsymbol{D}$ is the degree matrix and $\boldsymbol{A}$ is the adjacency matrix, both introduced in the Graph Theory chapter. This form of representation is heavily simplified and has a few limitations. Whenever the convolutional process is performed, the information for a node $\boldsymbol{v}$ is estimated by summing up the feature vectors for all neighbouring nodes. However, this excludes the information of the node $\boldsymbol{v}$. To fix this, the adjacency matrix is modified by adding the identity matrix, allowing for self-loops. Additionally, the Laplacian is also normalized as to prevent the scale of features to change substantially when performing several multiplications of the matrixes. These changes can be seen in Equation ***(10)*** The modification of the Laplacian matrix result in the following normalized Laplacian matrix, which is real symmetric positive and semidefinite [46].

$$\mathbf{L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}} \tag{10}$$

This form of representation allows the Laplacian matrix to be factored as seen in Equation ***(11)***

$$\mathbf{L = U \Lambda U^{T}} \tag{11}$$

Here, $\boldsymbol{U = [u_0, u_1, \cdots, u_{n-1}] \in R^{n \times n}}$ is the matrix of eigenvectors of the normalized Laplacian matrix L, ordered by the eigenvalues and $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues i.e., $\boldsymbol{\Lambda_{ii} = \lambda_i}$.

In signal processing, a graph signal $\boldsymbol{x \in R^n}$ is defined as a feature-vector of all nodes in a graph, where x_i is the value for node *i*. The *graph Fourier Transform* to a signal x is defined as $\boldsymbol{\mathcal{F}(x) = U^T x}$, while the inverse- graph Fourier transform is defined as $\boldsymbol{\mathcal{F}^{-1}(\hat{x}) = U\hat{x}}$ where $\boldsymbol{\hat{x} = \mathcal{F}(x)}$. The idea of the graph Fourier transform is to project the graph-signal input to an orthonormal space where the basis is formed by the eigenvectors of the normalized graph Laplacian found in Equation ***(10)***. Based on this, the graph convolutional for an input signal x with a filter $\boldsymbol{g \in R^n}$ can be defined as shown in Equation (12).

$$\begin{aligned} \mathbf{x} *_{\mathrm{G}} \mathbf{g} &= \boldsymbol{\mathcal{F}^{-1}}(\hat{\mathbf{x}})\big(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})\big) \\ &= \mathbf{U}(\mathbf{U}^{\mathrm{T}}\mathbf{x} \odot \mathbf{U}^{\mathrm{T}}\mathbf{g}) \end{aligned} \tag{12}$$

Here, $\odot$ denotes the element-wise product and $*_G$ denotes the graph-convolutional operator. The spectral graph convolution can be further simplified by changing the denotation for the filter as $g_\theta = diag(\boldsymbol{U}^T \boldsymbol{g})$, as shown in Equation ***(13)***.

$$\mathbf{x} *_{\mathrm{G}} \mathbf{g}_{\boldsymbol{\theta}} = \mathbf{U}\mathbf{g}_{\theta}\mathbf{U}^{\mathrm{T}}\mathbf{x} \quad (13)$$

The definition for the spectral graph convolution in Equation ***(13)*** is baseline for most, if not all the popular Spectral-based ConvGNN methods. The methods differ in their choices for the filter g_θ. Later work has improved the spectral graph convolutional method by improving the filter to yield more non-spatially localized filters. This was achieved by approximating the filters by a Chebyshev expansion of the graph Laplacian [49]. This approximation also removed the necessity of estimating the eigenvectors of the Laplacian, thus drastically reducing the run-time of the model as the process of estimating the eigenvectors is high computation-wise.

What has been thought of as one of the major disadvantages of the spectral-approaches and the popular convolutional method using a spectral approach, is that the filters directly depend on the Laplacian Eigenbasis. As the Eigenbasis are found directly from the Adjacency matrix, any perturbations to the graph will lead to a change in the Eigenbasis [46]. Furthermore, as the spectral filters are domain-dependent, it is thought that the models are unapplicable on graphs with differing structures. The models using the spectral approach were thought to be unable to generalize to graphs with a graph-structure differing from that of the trained graph. However, there are studies that show that spectral graph convolutional models are transferrable (inductive), and that they are usable across graphs with varying topologies [50].

3.5.2 Spatial Approaches

While the spectral models base their convolutional methods on the Fourier transform and by using different forms of filters in their convolutional methods, spatial methods work through the aggregation of neighbourhood information. A common approach for the Spatial ConvGNN can be thought of the models utilizing the graph structure alongside an aggregative and propagating node information process with a convolutional method [47]. An example of how the message-passing process work can be seen in **Figure 6**.

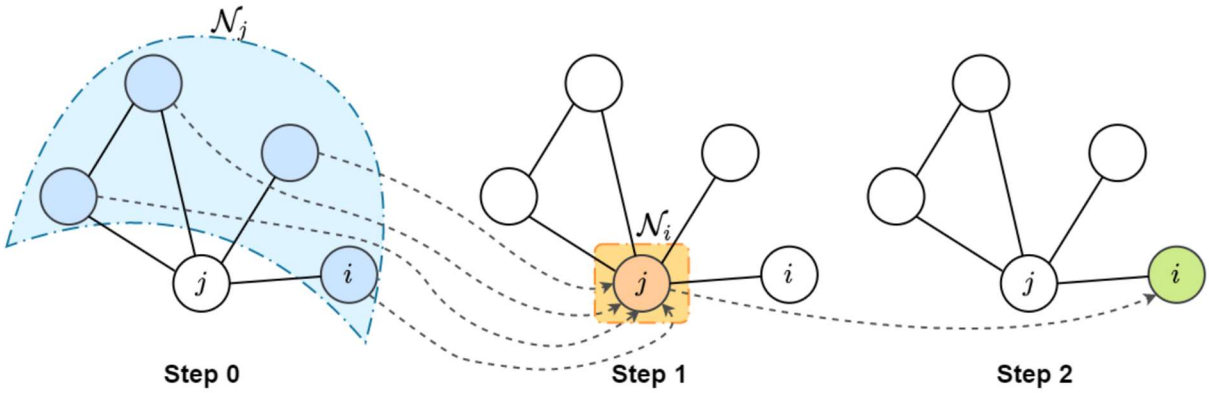


Figure 6: **Illustrative figure** *[51]* **of how the propagation steps in a GCN allows for information passing between nodes not directly connected to each other. From step 0 to step 1, the features of node j are updated by aggregating the feature-information of its neighbouring nodes. From step 1 to step 2, the features of node i are updated by utilizing the information from the connected node j, which in turn contain information from all neighbouring nodes, including those of node i itself.**

As with the spectral approach, there are numerous ways of performing the spatial convolutional method. A general methodology can be thought of as each node in the graph having a corresponding feature-state vector $\boldsymbol{H^{(l)}}$ which represent the state of the node after $\boldsymbol{l}$ iterative steps. At initialization, this feature-state vector is $\boldsymbol{H^{(0)}} = \boldsymbol{X}$, where $\mathbf{X}$ is the feature-input vector. These feature-state vectors are then updated by iteratively passing node-state information with its closest neighbours. At each iterative step, the feature-vector for each node can be updated by using the nodes current feature-vector alongside the information from its closest neighbours and the features of the edges connecting the nodes.

One of the challenges the spatial approach faces is maintaining a weight sharing property when nodes in the graph have a varying degree of neighbouring nodes. One solution is to utilize a trainable weight matrix for each node degree [52]. The various spatial convolutional methods having different approaches in which they define the weight of the message passing between nodes [53]. A general, albeit simple approach, is to apply the same weight to each node. Other convolutional methods modify this by using edge attributes, node weights [15] or trainable attention weights [42] for each set of weights in the network. This makes the model more localized and can increase the performance of the network.

The terminology "steps" or "hops" are often used when talking about spatial convolutional methods. A step or hop indicate a passing of message from a node to its connected neighbours. For ConvGNN models, additional steps can be added by introducing more layers

to the network. Each additional layer will add another set of message-passing. With enough steps, information from a single node $\boldsymbol{v}$ can be passed over the whole graph, depending on the graph's connectivity. Such a construction is not necessarily preferable, as too many steps can result in the dilution of the local node-specific information.

3.6 Convolutional Methods

3.6.1 GCN

The Graph Convolutional Network operator (GCN), which is not to be confused with the general term of Graph Convolutional Networks, is a layer-wise propagation rule operator introduced by Kipf & Welling [15]. The GCN-operator takes root in the spectral approach, however the operator includes several simplifications introduced in the spectral chapter, such as restricting the filters to operate in a 1-step neighbourhood around each node.

In its simplest form, the layer-wise propagation rule can be defined as shown in Equation *(14)*

$$\mathbf{H}^{(\mathbf{l}+\mathbf{1})} = \mathbf{f}\left(\mathbf{H}^{(\mathbf{l})}, \mathbf{A}\right), \;\; \mathbf{f}\left(\mathbf{H}^{(\mathbf{l})}, \mathbf{A}\right) = \boldsymbol{\sigma}(\mathbf{A}\mathbf{H}^{(\mathbf{l})}\mathbf{W}^{(\mathbf{l})}) \tag{14}$$

Where $\boldsymbol{\sigma}$ is the activation function, $\boldsymbol{A}$ is a mathematical description of the graph topology in matrix form, like the adjacency matrix, $\boldsymbol{W}^{(l)}$ is the weight matrix for the $\boldsymbol{l}$-th layer of the neural network, $\boldsymbol{H}^{(l)}$ is the feature state vector for the l-th layer and $\boldsymbol{f}(\cdot)$ is an arbitrary transformation function. As with the spectral approach, this simple representation comes with its limitation. The method does now allow for self-passing of information. This is easily fixed by adding the identity matrix to the adjacency matrix, giving us the $\widehat{\boldsymbol{A}}$ matrix. Additionally, as the graph gets bigger, the scale of the adjacency matrix will increase unevenly. As the nodes in the graph have a varying amount of connected neighbouring nodes, it can lead to significance difference between the values in the adjacency matrix. This can cause the gradients to become unstable, either vanishing or exploding. It is therefore preferred to normalize the adjacency matrix $\widehat{\boldsymbol{A}}$, by multiplying it with the inverse of the degree matrix $\widehat{\boldsymbol{D}}$. These changes yield the following layer-wise propagation rule introduced by Kipf and Welling, as seen in equation *(15)*.

$$\mathbf{f}(\mathbf{H}^{(\mathbf{l})}, \mathbf{A}) = \boldsymbol{\sigma}(\widehat{\mathbf{D}}^{-\frac{\mathbf{1}}{\mathbf{2}}}\widehat{\mathbf{A}}\widehat{\mathbf{D}}^{\frac{\mathbf{1}}{\mathbf{2}}}\mathbf{H}^{(\mathbf{l})}\mathbf{W}^{(\mathbf{l})}) \tag{15}$$

The GCN-operator is a quick and powerful operator which combines local graph structure and node-level features. The operator however is limited by the usage of the adjacency matrix in its propagation rule, hindering a single operator to be used on multiple topologies. This makes the operator useless in inductive problems, where the operator must work on unseen graph structures.

3.6.2 Graph Attention Network

The Graph Attention Network (GAT) is a spatial convolutional operator. The GATs leverage a mechanism called self-attention layers to specify the weights to different nodes in a set neighbourhood, instead of generalizing this as the GCN-method does. In general, the spatial operators aggregate features node-wise across neighbourhoods as shown in equation ***(16)***

$$\vec{\mathbf{h}}_\mathbf{i}' = \boldsymbol{\sigma}\left(\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} \boldsymbol{\alpha}_{\mathbf{ij}}\vec{\mathbf{g}}_\mathbf{j}\right) \tag{16}$$

Where σ is an activation function and α_{ij} is the weighting factor between node i and j. In the GATs, this weighting factor is defined by using the self-attention mechanism over the features. To transform the input features to higher-level features, the operator requires at least one learnable linear transformation. To achieve this, the GAT operator introduces a shared linear transformation parametrized by a weight matrix **W** at the initial step. The operator then performs self-attention on the nodes by a shared attentional mechanism *a* which computes a set of attention coefficients *e,* as shown in equation ***(17)***. These coefficients indicate how important the features of a node i is to node j.

$$\mathrm{e}_{\mathrm{ij}} = \mathrm{a}(\mathbf{W}\overrightarrow{\mathrm{h}_\mathrm{i}}, \mathbf{W}\overrightarrow{\mathrm{h}_\mathrm{j}}) \tag{17}$$

The operator also disregards the topological structure of the system by a process called masked attention. In this process, the graph structure is injected by computing e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the neighbour for any node *i* in the graph. Furthermore, to make the coefficients more easily comparable across nodes, the attention coefficients are normalized for all choices of j by using the SoftMax function as shown in equation ***(18)***.

$$\alpha_{\mathrm{ij}} = \mathrm{softmax}_\mathrm{j}(\mathrm{e}_{\mathrm{ij}}) = \frac{exp\ (\mathrm{e}_{\mathrm{ij}})}{\sum_{\mathrm{k}\in\mathcal{N}_\mathrm{i}} exp\ (\mathrm{e}_{\mathrm{ik}})} \tag{18}$$

The GAT operator's attention mechanism *a* is a single-layer MLP network. The network is parametrized by a weight vector ***a*** and uses the LeakyReLU activation function. This gives the following equation for the weighting factor, as shown in equation*(19)*.

$$\alpha_{ij} = \frac{exp\ (\text{LeakyReLU}\left(\mathbf{a}^{\mathrm{T}}\left[\mathbf{W}\mathrm{h}_{\mathrm{i}}||\mathbf{W}\mathrm{h}_{\mathrm{j}}\right]\right))}{\sum_{k\in\mathcal{N}_i} exp\ (\text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{W}\mathrm{h}_{\mathrm{i}}||\mathbf{W}\mathrm{h}_{\mathrm{k}}]))} \tag{19}$$

Where the $.^{T}$ is the transposing operator and || is the concatenation operation [42].

The GAT operator has several interesting perks. The operator can be applied to graph nodes with varying degree by simply specifying arbitrary weights to the neighbours. Furthermore, the GAT operator does not utilize any form of matrix operators, nor does it require any knowledge about the global topology of the system [42]. This allows the operator to be used on inductive problems, where the model must adapt to unseen graphs. The biggest drawback of the GAT operator is the intensive computations needed to compute the weighting factors.

3.6.3 GraphConv

This thesis also utilized an operator named GraphConv, a spatial ConvGNN operator [54]. The GraphConv operator, while not as popular as the two previously mentioned operators nor as complex as the GAT operator, displayed profound strength in solving the problems faced in this thesis. The GraphConv operator work similarly as other spatial operators through message-passing. The operator estimates the node-features by aggregating the nodes features x_i with the neighbouring features x_j. The operator does a nodewise screening, where each nodes' features are multiplied with a weight factor $\boldsymbol{\theta_1}$, while the neighbouring node's features are scaled with a weight factor $\boldsymbol{\theta_2}$ and with the connecting edge weight $e_{j,i}$, as shown in Equation *(20)*. These weight factors are learnt through training of the model.

$$\mathbf{x}_{\mathbf{i}}' = \boldsymbol{\theta}_{\mathbf{1}}\mathbf{x}_{\mathbf{i}} + \boldsymbol{\theta}_{\mathbf{2}}\sum_{\mathbf{j}\in\boldsymbol{\mathcal{N}}(\mathbf{i})}\mathbf{e}_{\mathbf{j},\mathbf{i}}\cdot\mathbf{x}_{\mathbf{j}} \tag{20}$$

As the operator does not utilize the graph-structure in the estimation of the node-features, the operator is usable on inductive problems. Because of the low complexity of the operator, the speed of calculation is significantly lower than that of the GAT-operator.

Part III / Methodology

In the following section, the methodological procedure used to create the datasets and the ML-models are presented. In the first chapter, the test-system and the general methodology used to produce the datasets is covered. Information about the test-system, such as the topology and relevant features is catalogued and discussed, and the non-sequential MC simulation procedure that was used to perform the contingency analyses is described in detail. The section also describes how parts of the MC-simulation was altered to tailor the simulation tool for each experiment conducted in this thesis.

The methodology part also describes how the ML-models used for this thesis were constructed, the system parameters and the prevalent hyperparameters. The model-section also gives a brief introduction of the concept of batch-sizes. The model's prediction-level and how the model's output was changed for each prediction level to accurately represent the model's performance is covered. Lastly, the section presents the prevalent model-features and how the features were pre-processed before they were used in the ML-models.

In summary, the following will be covered in the methodology chapter:

Section 4.1 Introduces the IEEE-24 Bus System, its topology, and the essential information about the system.

Section 4.2 Introduces the Non-Sequential Monte Carlo simulation process and how the simulation process was tailored to create datasets for each experiment conducted in this thesis.

Section 5.1 Introduces the ML-model, how it was constructed and the relevant packages. The section also covers a few important system parameters, such as the loss-function and the learning-rate scheduler.

Section 5.2 Introduces the two prediction levels of the model used in this thesis, and how the output was tailored to give a proper representation of the model's performance for each prediction level.

Section 5.3 Gives a thorough overview of the model's parameters and hyperparameters and display the chosen hyperparameters for all models.

Section 5.4 Introduces the features of the model, derived from both the test-system and from the data-generation process.

Section 5.5 Covers the Training- and test-data split, introduces the concept of batch-size and introduces the concept of data-pre-processing, various ways of performing pre-processing and which method used for this thesis.

4 Test System and Data Generation

The following section introduces the test-system, its features and the procedural used to generate the dataset that was used for the ML-models. Section 4.1 introduces the test-system, its topology, and the features of the system. Section 4.2 introduces the data-generation process of the non-sequential Monte Carlo simulation, the simulation procedure and how the data was stored. The subsequential subchapters explains how the simulation procedure was tailored to each experiment.

4.1 IEEE-24 Bus System

The datasets in this thesis are based on the IEEE 24 bus-system. The IEEE-24 bus-system is one of many reliability test systems developed by IEEE, and is a fictional system used for testing of various reliability analysis methods [55]. The topology for the test-system can be found in **Figure** *7*.

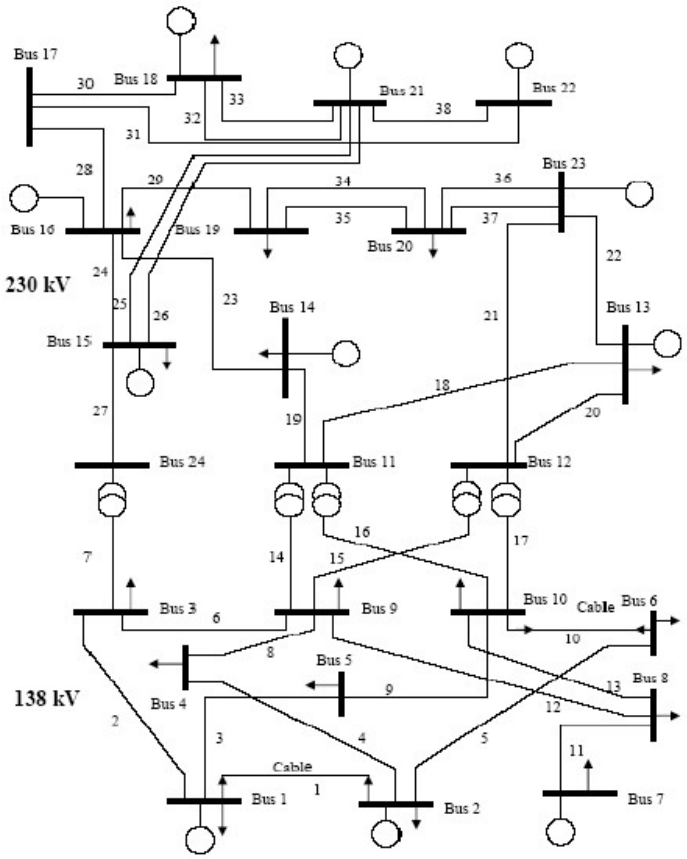


Figure 7: One Line Diagram showing the topology for the IEEE-24 bus test system *[56]*

The system consists of 24 buses, in which there are 10 PV buses, 13 PQ buses and one Slack bus. The bus-type indicate what variables are known at each bus. In the PQ busses, the real (P) and reactive power (Q) are specified, while the bus voltage and angle are unknown. In the PV-bus, the real power (P) and the voltage magnitude (V) is specified, while the phase angle and the reactive power is unknown. The slack bus, also known as the reference bus, is used to balance the active and reactive power flow in the system. The slack bus also defines the system's reference angle. The buses in this case represent an area which can have both a production and a load, though this is not guaranteed as some of the buses have a production of power but no load, while some buses have a load but no production. Further information about the buses and their affiliated values can be found in **Table *15*** and **Table *16*** in Appendix A.

The test-system has 38 interconnecting lines and 34 generators. The generators in the system consist of a mix of different energy sources, such as coal, gas and nuclear, with a varying production potential and a specific cost-value specified in $/MWh. The cheapest generators produce energy at $5/MWh, while the most expensive generator produce energy at 25$/MWh. At average for all the generators in the system, the cost for producing energy is at $15.4/MWh. The cost of rescheduling the power of any generators will be equal to the normal operating cost the specific generator. Further information about the generators can be found in Appendix A in **Table *17*** and **Table *18***.

In the system, each load-bus also have an affiliated cost-value for shedding the load. These cost-values represent the severity for the loss of load in that area. The higher the cost, the more significant the loss of load is. Industrial areas, farms and hospitals are typically areas where load-shedding can be highly expensive, while the loss of load in residential areas are not as severe. In the test-system, the area where it is cheapest to shed load, the cost is at 3662.3$/MWh (Load 9), while the most expensive area for load shedding is at 9599.2$/MWh (Load 4). **Figure *8*** shows the cost of shedding load for all load-buses.

The shedding of load is around 427 times more expensive compared to running the regular generators or to reschedule any power. Any potential outages in the system where the system cannot accommodate will therefore lead to substantial socio-economic costs. The system therefore prioritizes cutting the load in areas where the cost of doing so is the cheapest. This will be heavily reflected in the predictions of the model.

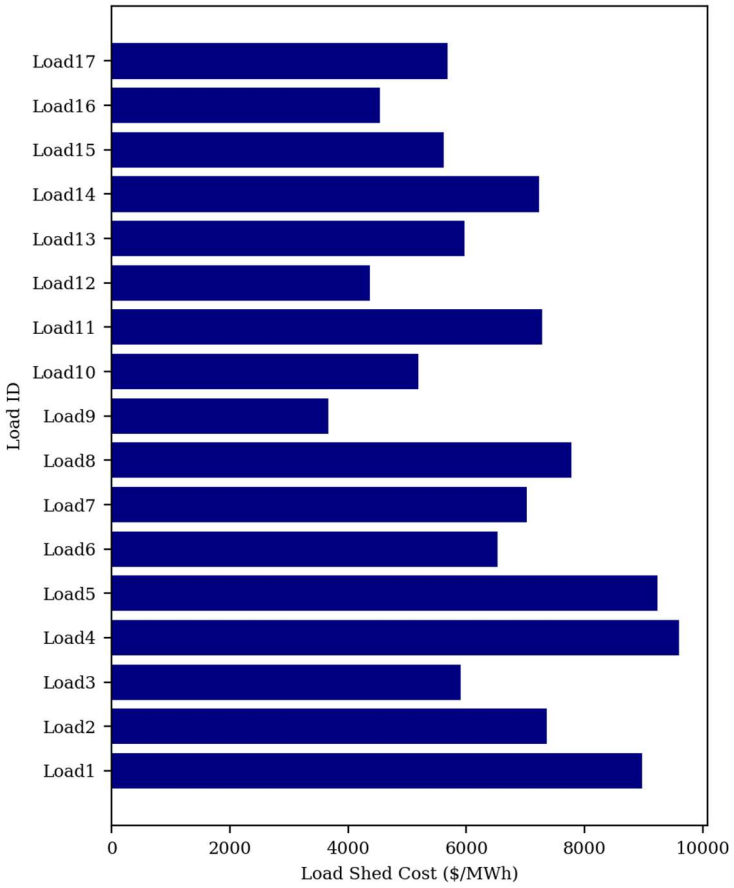


Figure 8: The Load Shed Cost given in $/MWh for each load-bus.

4.2 Data Generation

As this thesis sought to test out multiple various system-perturbations, it became convenient to create a new data-generation simulation tool that was specialized for the experiments conducted in this thesis. The simulation tool was made in such a way that it was easy to change the system-parameters. The datasets used for the experiments of this thesis were created through a contingency analysis, inducing k-number of contingencies at each simulation step using a non-sequential Monte Carlo simulation and the IEEE-24 bus system. The non-sequential method was chosen as occurrences of contingencies in a power system are rare, especially cascading contingencies. If the contingencies were to be drawn from an exponential distribution, only a few select datapoints would include cases where a contingency occurred, and the occurring contingencies would not always have a profound effect on the state of the system either. Some contingencies can occur in areas where the system is resilient to outages, while other contingencies might occur in less resilient areas. By inducing a contingency at each simulation step, the probability of the occurrences of load-

shed would increase, making for more interesting analyses. For this thesis, the type of contingency was narrowed to **only** consisting of line disconnections. However, the type of contingency can be anything from failing generators, defect components, a disconnected or overloaded line etc.

4.2.1 Data Generation Process

The implementation of the contingency analysis and the Monte Carlo simulation was done using the python package *pandapower* [57], a powerful power system analysis toolbox. Pandapower also has a vast library of integrated power systems, including the IEEE-24 bus system, making the implementation of all the system's variables much easier. The toolbox of pandapower also included all the necessary functions to perform the Contingency Analysis of the system, such as the OPF and PF.

The general data-generation procedure is show below:

1) Define load condition.
2) Solve DC-OPF with the set load conditions.
    - If the system does not converge, discard the load conditions, and proceed to the next.
3) Store set-point values for the generators, loads and lines.
4) Produce $\boldsymbol{k}$ contingencies by disconnecting $\boldsymbol{k}$ different components at random.
5) Check if any of the induced contingencies lead to a system overload by estimating the DC-PF equations.
6) If the system was overloaded, solve a DC-OPF with the new system state induced by the contingencies. If not, continue.
    - If the system does not converge during the OPF, discard the load conditions and contingencies, and proceed to the next set of load conditions.
7) Reconnect the disconnected component(s).
8) Estimate any potential load-shedding values by taking the difference between the set-point load values for all buses before and after the OPF in (6) took place.
9) Repeat.

In both cases where the OPF had to be solved, a safeguard was introduced to ensure that the system had a solution. The integrated Pandapower OPF algorithm tended to not converge

despite the solution existing. The occurrence of this incident increased as the load-demand of the system increased, and the cases where it occurred were dropped.

When the OPFs did converge, the state of the system and its values such as the system load, the generated power, the line-flow, the shed-load values, and the rescheduled generator values were stored in data-frames, which was later used to construct the feature data to the ML-model. As the ML-model was thought to replace the last OPF in the contingency analysis (part 6), the model had to be trained with features that were prevalent before the last OPF estimation. Hence values such as the line-flow were stored before the last OPF was initialized. This meant that the line-values which the ML-model received did not reflect the state of the system after the contingency had occurred, and the model had no information about the occurrence of the contingency in the system. To include this information for the ML-model, the status of all lines after the contingencies had occurred were stored. This would allow the inclusion of the line-status to the affected nodes of the system.

4.2.2 Datasets

The purpose of the thesis was to investigate the model's predictive abilities across multiple system-perturbations to investigate the model's resilience to changes. The performance of the model would dictate whether the model would become a useful tool for contingency analyses. As an initial step, the induced perturbations were small to investigate if the models were generalizable across minor system perturbations.

To realize the objectives of the thesis, multiple datasets for each system-perturbation had to be created, where each dataset further perturbated the system parameter. As the scope of the thesis were on the contingency-, load- and topology parameters, three sets of datasets were made, one for each system parameter. For each set, the system-parameters were increasingly perturbated. This was achieved by increasing the number of induced contingencies per simulation step for the contingency-objective, or by increasing the system load for each dataset for the load-objective. By making multiple dataset with different system perturbations, it opened the possibility of evaluating the model across multiple parameter perturbations. By changing only one parameter at the time, it became easier to determine which perturbated system-parameter the model was and was not resilient against. This also opened for the possibility of sequentially training the model on additional datasets with further system perturbations.

4.2.2.1 Experiment 1, Contingency Perturbation Simulation

The first objective of the thesis was to explore the model's predictive ability across multiple contingencies. The thesis sought to explore the model's ability to predict the load-shedding values for cases with varying number of contingencies without being trained specifically for those cases. The objective was also to explore how the behaviour of the model changed after sequentially training on the dataset with the highest number of induced contingencies per simulation point. As such, three datasets were created with each dataset having an increased number of induced contingencies at each simulation step in the MC simulation. The load of the system was kept identical across all three datasets, and the topology was held static. The first dataset induced a single line-contingency at each simulation step, the second dataset two line-contingencies and the third dataset induced three line-contingencies at each simulation step. Each simulation, independent of the number of contingencies, followed the same simulation-procedure and the contingency points was chosen at random.

4.2.2.2 Experiment 2, Load Perturbation Simulation with Contingencies

The second objective of the thesis was to explore the model's predictive ability across system-load perturbations. To this end, the topology of the system was held static, and a single contingency was induced at each simulation step. The set-values for the load were initially equal across the dataset, but the values were increased by adding a scaling factor k = 1.05, 1.10, 1.20 to the load for each of the simulated datasets, respectively. In total, three additional datasets were made with the scaling load factor, increasing the system load with 5%, 10% and 20%.

4.2.2.3 Experiment 3, Contingency Perturbation Simulation with modified topology

The third objective of the thesis was to explore the model's predictive ability across slight perturbation of the system topology and induced contingencies. To this end, the test-system was altered by adding two additional lines in the system. The addition of the extra lines was not chosen at random, but rather through a selective process in which the buses that were most vulnerable to line outages were chosen. The first line was added between bus 7 and bus 8, and the added line adopted the same features as the already existing line. The placement of the first line was chosen as bus 7 was the only bus that only had a single connecting line between itself and the rest of the system. By adding another line, the bus would be better

protected against any possible contingencies. The second line that was added to the system was laid between bus 1 and bus 3. This line also adopted the same features as the already existing line between the two buses. The altered topology can be seen in **Figure *9***.

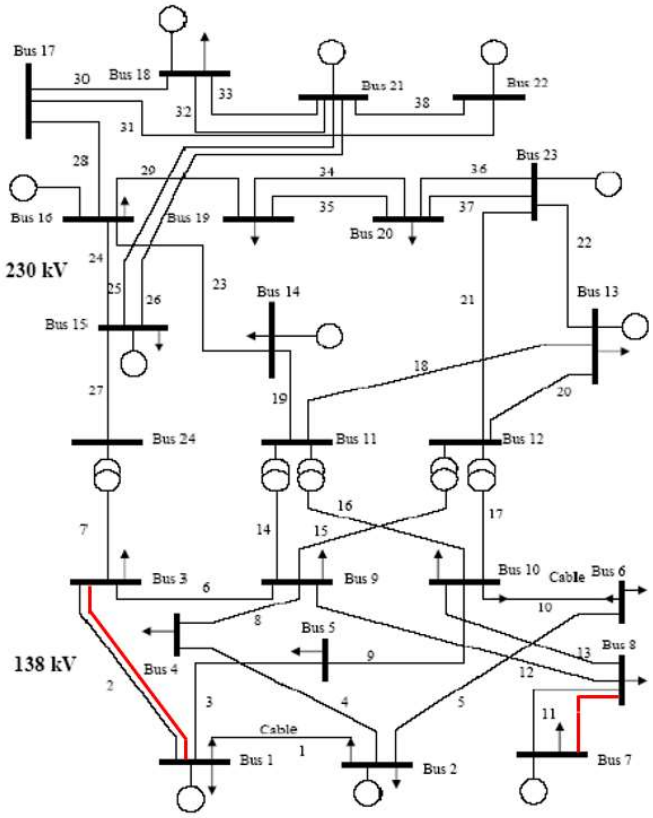


Figure 9: Altered One Line Diagram showing the topology for the IEEE-24 bus test system.

With the modified topology, a total of three datasets were made following the processes in experiment 1 (section 4.2.2.1), where each dataset was simulated with an increasing number of induced contingencies per simulation step (1-, 2- and 3-contingencies).

4.2.2.4 Experiment 4, Case Study

The last experiment of the thesis conducted a case study to investigate multiple sub-objectives. The first sub-objective was to compare the run-time between the ML-model and OPF the model sought to replace. The second sub-objective was to compare the load-shedding values between the OPF and the predicted values of ML-model. The last sub-objective was to investigate the impact the batch-size of the model's test-dataset had on the run-time of the model.

To this end, a single dataset was simulated using the original topology of the system and a single contingency per simulation step. In total, 5000 datapoints were simulated, each point containing information about the adjacency matrix, the node-features, and the edge-weights. As the dataset was simulated, the run-time of the last OPF in the data-generation procedure was estimated per simulation step and added up. As the last OPF was only called when an overload occurred in the system, the run-time could be zero if no overload occurred. To determine the run-time of the model, the run-time was only estimated when the model was making the predictions and did not include the process of creating the input-dataset. After the model had made its prediction, the predicted values were then compared against the load-shedding values estimated by the OPF. To determine the impact the batch-size of the training-set of the ML-model could have on the predictions, two ML-models were trained with the training-sets having two different batch-sizes.

5 Model

In this section, the construction and the key parameters of the ML-models is introduced. Section 5.1 introduces the packages used to construct the model, and the model's internal functions. Section 5.2 explains the prediction level of the model, and how the output of the model was tailored for each prediction level. Section 5.3 introduces the key parameters of the model, the concept of hyperparameters and the chosen hyperparameters for the models used to produce the results. Section 5.4 introduces the features of the system and the key differences between the features of standard ANN models and a ConvGNN-model. Section 5.5 discusses the split between training- and test data, the concept of batch-size and briefly discusses the pre-processing steps taken to prepare the features.

5.1 Model Construction

To construct the ConvGNN-models used in this thesis, the python libraries PyTorch and PyTorch Geometric [58] were used. The libraries offer well defined convolutional methods from published papers in its calculations, such as the ones introduced in chapter 3. The library also includes a range of in-built functions, such as the optimizing function of the model, loss-function, learning-rate scheduler etc. The built-in functions allow for a more precise and quicker construction of each model and easier tuning of the many parameters.

5.1.1 Optimizer Function

The objective of the optimizing function in the model is to adjust the parameters of the model such that the error in each training step is reduced. PyTorch's library offers a vast number of optimizers, with the standard method being the Adam-optimizer, a first-order gradient-based stochastic optimization function [59]. This was also the chosen optimizer for all ConvGNN-models used in this thesis.

5.1.2 Learning Rate Scheduler

The learning-rate scheduler is a tuneable function that decreases the learning rate by an amount per epoch to reduce fluctuation of error. The rate at which the learning rate is reduced per epoch can be tuned before training by changing the gamma-parameter of the scheduler. A decent choice of the gamma parameter can assist the model to converge.

5.1.3 Loss Function

The ML-models used in this thesis fall under the category of supervised learning, which learn by estimating a loss function which is then used to optimize the weights of the model. The loss function work as an evaluation tool for the model, describing how well the model works on the given data. The loss value is estimated by comparing the predicted output value $\check{y}$ with the target value y. This can be done in several ways, depending on the goal of the model, such as regression or classification. For regression, which was the prediction goal of the models used in thesis, the more popular loss-functions are the Mean Square Error (MSE) and the Root Mean Square Error (RMSE). The difference between these two methods is how the loss function penalizes outlying and deviating values, with the MSE-function penalizing outlying values much more as compared to the RMSE-function. For this thesis, the RMSE-loss function was used. The mathematical formulation for the loss-function is given in equation ***(21)***.

$$\boldsymbol{RMSE} = \frac{\sum_{i=1}^{n} \sqrt{(y_i - \hat{y}_i)^2}}{n} \tag{21}$$

5.2 Prediction Level

As mentioned in Section 3.4.1, ConvGNN-models can work at different prediction levels, such as Node-level, Edge-level, and Graph-Level. Each prediction level specializes in extracting certain information about the features or the graph. The scope of the thesis has been to design and construct models on both a system-level, meaning that the models output a single value for the whole system, and on a partial node-level. For the system-level, the predicted output value would be the sum of shed load in the system. On a partial node-level, the number of outputs is equal to the number of buses in the system which has a corresponding load. For the IEEE-24 RTS system, the system has 24 buses, but only 17 of them has a load. The prediction level has been dubbed ‘partial’, as it is not beneficial to include the nodes which does not have any form of load-shedding. Thus, the number of outputs for partial node-level would be 17.

5.2.1 Estimation of the Load Shedding values

The target value for the ML-models of this thesis was the estimated load shedding values from the MC simulation. The shed load values were the values of lost load to end-points that

was deliberately cut to prevent system-failures when the system's capacity was under strain. The load-shedding values were estimated by taking the difference of the load values in the load-buses before and after the last OPF in point (6) was run. As not all contingencies led to a system overload, the value for the shed load was sometimes zero.

As the scope of the thesis included both a system-level prediction and a node-wise prediction for the ML-models, the load-shedding values were estimated on a different basis for each prediction level to properly display the model's performance. It is important to distinguish the difference between the estimated output values per simulation point, which is used to visualize the model's performance per simulation point, and the total error for the dataset.

For the system-level prediction, the load-shedding values were estimated as a total sum of shed load per simulation point, as shown below. This representation would allow for a comparison of the model's predicted value vs the target value for each simulation point.

$$\mathbf{Load_{system}} = (\mathbf{Load1_{shed}} + \mathbf{Load2_{shed}} + \ \ldots + \ \mathbf{Load17_{shed}})$$

The prediction-error at the system level was estimated as an average of the prediction error for each simulation point, as shown below.

$$\boldsymbol{Error_{system}} = \frac{\boldsymbol{Error_1} + \boldsymbol{Error_2} + \ldots + \boldsymbol{Error_N}}{\boldsymbol{N}}$$

For the node-level prediction, the load-shedding value was not estimated as a sum, rather each load-bus value was presented separately. For the node-level, the results were not given per simulation point, rather, the mean over all values for each node was estimated, as shown in the following equation.

$$\mathbf{Load_{node}} = [\frac{\sum_{i=1}^{n} \mathbf{ShedLoad1_i}}{\boldsymbol{n}}, \frac{\sum_{i=1}^{n} \mathbf{ShedLoad2_i}}{\boldsymbol{n}}, \ldots, \frac{\sum_{i=1}^{n} \mathbf{ShedLoad17_i}}{\boldsymbol{n}}]$$

This allowed for a per-node representation, displaying how well the model's prediction were on a per node basis.

Another way of representation would be to aggregate all the values into a single value for each simulation point. One way of achieving this is to estimate the mean of all load-buses per simulation point as shown in the following equation. This enabled the possibility of plotting

the predicted-value vs the target value for each simulation point, though this was not utilized in this thesis.

$$\mathbf{Load_{node}} = \frac{(\mathbf{ShedLoad1} + \mathbf{Load2_{shed}} + \ \ldots + \ \mathbf{Load17_{shed}})}{\text{Nr. of load buses}}$$

When estimating the error at the node-level, it was imperative that the error was estimated on a node level at first, before averaging over the RMSE for each node.

5.3 Model Parameters

Common for all ML-models are pre-tuneable parameters called hyperparameters. Hyperparameters are not to be mistaken for standard parameters, which are internal configuration variables for the model, often estimated based on the input data to the model. The standard parameters are used by the model to make predictions and define how well a model is performing. The parameters are values such as the internal learning weights of the ML-model, or the loss function used to define how well the model is performing, and these are not manually defined.

The hyperparameters on the other hand are defined by the user before a model is trained and are decisive factors for the performance of a model. The importance of each hyperparameter varies depending on the model, but the most significant values are the Learning Rate, the Dropout Rate, the number of layers and some specific parameters for models, such as the number of heads, a specific parameter for the GAT model. **Table *1*** includes the parameters and hyperparameters prevalent in the model and a brief description of each parameter.

Table 1: The parameters and hyperparameters of the ConvGNN-models.

Parameter	Description
Layers	Number of hidden layers in the network
Hidden Channels	Number of neurons in each hidden layer
Loss	Loss function used during training
Epoch	Number of iterative steps that a model is trained on a dataset
Learning Rate	The step size during the backpropagation process during training
Decay of Learning Rate	The reduction of learning rate per epoch during training
Dropout Rate	The rate of dropout between each propagation
Batch Size	The batch dimension of the dataset

As the hyperparameters are decisive for the outcome of the models, choosing the correct values for the hyperparameters are essential to optimize the models. Within the field of ML, there are multiple ways of going by this. Common approaches include methods such as grid search or random search, where the hyperparameters are defined within a range, and the hyperparameters are systematically or randomly chosen from that range. The best performing combination of parameters are then stored. While this can allow the model to be tested on a wider range of possible hyperparameters, the process is often long and tedious. Training a model with a set of hyperparameters for 100 epochs, which guarantees at least a degree of convergence, can sometimes take hours.

As this thesis intended to perform several experiences with multiple operators, the manual tuning of the parameters was preferable over the random approach due to the short amount of time available. The manual search used personal knowledge and experience to identify the most relevant hyperparameters and the optimal range for each parameter. While the manual search method may not necessarily grant the most optimal values for the hyperparameters, the process is significantly faster. The difference in error from the manual choice of hyperparameters and the optimal choice of hyperparameters is often not detrimental for the result of the model, and the results given by the manually chosen hyperparameters should give a good indication about the model's ability. **Table *2*** introduces the chosen parameters used by the models to produce the results displayed in the thesis.

Table 2: The hyperparameters for the models used in this thesis.

Parameter	GAT	GraphConv	GCN
Layers	3	3	3
Hidden Channels	64	64	64
Dropout Rate	0.2	0.2	0.2
Epoch	100	100	100
Learning Rate	0.0001	0.0005	0.0001
Decay of Learning Rate	0.95	0.95	0.95
Batch Size	15	15	15
Heads	4	-	-

5.4 Features

When training ML-models, the models depend on associated features which describes the characteristic of the data. These features allow the ML-models to extract the information of the dataset, which is subsequently used to enhance the model's predictive abilities. The quality of the features therefore has a major impact on the performance of the models. Choosing features that are informative, discriminating, and independent of each other is crucial for the model to understand the nuances of the data. For the ConvGNN-models, the prevalent type of features differs slightly compared to other ANN-models. The features of a ConvGNN model can be separated into three groups, being the topology of the system, the edge-weights, and the node-features.

The topology describes the system set-up and is used to construct the neighbourhood-matrix which the models use for message-passing between nodes. As the topology is not a traditional feature, it does not require any form of pre-processing. Rather, the topology is used to construct the neighbour-matrix which was introduced in section 3.3. The edge-weights in ConvGNN-models defines the strength of the connection between nodes in the graph. The edge-weights, albeit different from the normal features, are pre-processed in the same way as to normalize the connection strength between nodes. The regular features of a ConvGNN-model are managed in the same way as for other ANNs. A distinction is made between features describing a certain value and features describing a state. The features describing a state are often one-hot encoded, meaning that the values are given either as 0's or 1's. This can be interpreted as an active state (1) or an inactive state (0), though this varies depending on the feature. The features describing a state are not pre-processed either.

The features describing a value do not typically have a predisposed range, but the values are often normalized or standardized through pre-processing methods. **Table *3*** includes a list of the features of the models. The content of the table describes what the features represent and where they can be found in the graph-system. The Load Flow the Power load and the Generated Power are dynamic values, either predisposed or generated in the Monte Carlo simulation. These values differ for each simulated scenario. The Active Power Demand and active Power Generation are static values given by the creators of the test system and can be found at [60].

Table 3: Descriptive table of all the features of the graph. All values are given in a per unit scaled on the system's MVA.

Feature	Description	Location
Topology	Description of the setup of the system.	Edge Index
Load Flow	The flow of power between the nodes in the graph.	Edges
Power Load	The load demand at each bus.	Nodes
Generated Power	The amount of generated power at each node with a generator.	Nodes
Active Power Demand	The Active Power Demand at each node with a load.	Nodes
Active Power Generation Potential	The Active Power Generation Potential at nodes with generators	Nodes
Line Status	Status of the node, used to check if a node has a connecting line which has been disconnected.	Nodes

5.5 Training- and Test Data

Within the field of ML, one typically distinguishes between three types of datasets: training, validation, and test datasets. The training datasets' sole purpose is to train the ML-model, improving its prediction by updating the internal parameters. Most of the available data is often used for training purposes. The validation set is often used to give an unbiased evaluation of the model's hyperparameters as the model is being trained and is often small compared to the other sets. The model never learns anything directly from the validation set, but the model becomes increasingly more biased towards the validation set the more changes that are done on the hyperparameters based on the performance of the validation set. Lastly, the test-set is used to evaluate the model's performance after training has concluded. For ML-models to provide an unbiased prediction of any dataset, the datasets used for testing purposes must be independent of the dataset used to train the ML-models. It is therefore essential that

the datasets are split before any form of pre-processing is done on the datasets. For this thesis, no validation sets were used due to limited time and this thesis only being an exploration study. As such, the data was split solely into training and test-sets instead. All datasets were split following the 80/20 distribution, in which 80% of the data was given to the training-set and 20% of the data was given to the test-set.

5.5.1 Data Preparation

Because the range of the features in the system varied significantly, the features had to be pre-processed to bring all values in the same range. This was done to ensure that none of the features were more dominant than other features simply because of the natural range of values. Most values that were simulated from the Monte-Carlo Simulation was given in a per-unit value, where most of the values ranged from -1 to 1. Other features, such as the active power generation per generator was not given in a per-unit range and therefore had a much broader range. A pre-processing step was executed to bring all the feature-values in the same scale. In ML, there exist many different tools to transform the data, the most popular being the standardization method and the normalization method. For this thesis, the normalization method was used on every feature, bringing all values in the range from [-1, 1]. The pre-processing method is given in equation ***(22)***.

$$\boldsymbol{y'} = \frac{\boldsymbol{y} - \boldsymbol{min}(\boldsymbol{y})}{\boldsymbol{max}(\boldsymbol{y}) - \boldsymbol{min}(\boldsymbol{y})} \tag{22}$$

The pre-processing was performed using functions provided by scikit-learn, a powerful ML-package. The package allows the user to call for two methods, being the fit-transform and the transform method. The fit-transform method uses the dataset to estimate the scaling parameter, while the transform method only uses the already estimated scaling parameters to process the data. When pre-processing the data, it is vital that the training set and the test-set is completely separated and independent of each other. As such, when pre-processing, the fit-transform function should only be called on the training data, ensuring that the scaling parameters are only estimated by the training-set. The test-set should then be processed by only calling the transform method, using the parameters from the training set.

5.5.2 Batch Size

To speed up the training-process of the ConvGNN-models, the datasets is occasionally given in batches instead of feeding the whole dataset to the model at once. By feeding the dataset in batches, the model does not have to store an error-value for each datapoint in the dataset, only for each batch. The model also updates its parameters for each batch-point based on the estimated error. By using a large batch-size, the models' parameters are updated based on the average error of the batch-size, rather than for each value. This saves a lot of memory making the model run substantially faster, as each sample of the batch are processed in parallel with each other.

Part IV / Results & Discussion

This section covers the experiments conducted in the thesis and assesses the results of each experiment. The result-section is split into two parts, the first part presenting the system-level prediction results and the second part presenting the partial node-level results. The system-level has four sub-sections, where each section presents the results of an experiment. The node-level has three sub-sections, where each section presents the results of an experiment. Each of the experiments conducted in this thesis sought to test the models in different conditions to examine the flexibility of the models by perturbating key-parameters of the system, such as the number of contingencies that had occurred, the system load and the topology of the system.

The contingency-perturbation was achieved by adding additional contingencies in each simulation step of the system in MC-simulation. This was done to check if the models were able to predict the load-shedding values across multiple contingencies without being trained for those situations. The load-perturbation was achieved by adding a percentage of additional load to the base-values in the system. The goal of the load-perturbation was to determine how resilient the models were to future load-perturbations, to check if the models must be retrained every year, every 5^{th} year etc. because of load variations. The last parameter was the change of topology. As the power system evolves, the addition of extra loads, generators and connecting lines are inevitable. Having a flexible model that can easily implement these small perturbations of the system set-up is preferrable, as the model does not have to be retrained for each addition.

In this thesis, each dataset originally simulated 20 000 datapoints. However, as some points are discarded due to a lack of convergence, the number of simulated datapoints may vary between datasets.

Dataset Taxonomy

In the following sections some confusion regarding the difference between the datasets can occur. As the model is to experiment by perturbating system-parameters between each experiment, it is essential to understand which parameter that was perturbated, and which was kept static.

- The **1-Contingency** dataset refers to the datasets being simulated with 1 contingency being induced at each simulation step in the Monte Carlo simulation. The contingency point is randomly chosen.
- The **2**- and **3-Contingency** dataset refers to the datasets created with test-system being simulated with 2 and 3 contingencies induced at each data point, respectively. All contingencies are chosen at random, and they cannot be the same.
- The **5%**, **10%** and **20%** load-perturbation datasets, referred to as the **low-, medium-** and **high-load datasets,** was created by increasing the load in the system by a flat percentage from the original values, as indicated with the %. The topology and number of contingencies that was simulated per datapoint was kept static.
- Some experiments also include multiple scenarios in which a model is first trained solely on a single dataset until convergence, and then sequentially trained on a percentage of a second dataset until convergence. **Scenario 1** indicates the model being trained on **20%** of the second dataset of the respective experiment, while **Scenario 2** indicates the model being trained on **80%** of the second dataset.

6 System-level Predictions

This section of the thesis presents the system-level predictions of the load-shedding values, achieved by setting the output of the ConvGNN-models to be a singular value. As each experiment sought to display the evolution of the RMSE for each test-set as the models were being trained, the model's had to be retrained for each experiment which explains any potential deviation for the base-models predictive results.

Section 6.1 display the Contingency-perturbation experiment, testing the model's ability to predict across contingencies.

Section 6.2 display the load-perturbation experiment, testing the model's ability to predict across varying load.

Section 6.3 displays the topology-perturbation experiment, testing the model's ability to predict across contingencies and topologies.

Section 6.4 introduces a case-study to compare the model's run-time and estimation against the run-time and results of the MC-simulations.

6.1 Experiment 1, Contingency Perturbation

The objective of the first experiment was to investigate if the ConvGNN-models could predict the sum of load-shedding in the system across datasets with a varying number of induced contingencies per simulation point. For this purpose, three datasets were simulated and constructed using the Monte Carlo Simulation tool. The topology and the load of the simulated system was held static across the datasets, but the number of contingencies per simulated step increased for each dataset. The first simulated dataset included a single contingency, the second simulation two contingencies and the third dataset had three contingencies per simulation step. The only difference in the graph-features between the induced contingency datasets was the node-feature list which stored information about line-outages. The first experiment also sought to test three different ConvGNN-operators to assess the accuracy and speed of the operators. The operator that showed the most prominent results in the first experiment was henceforth used as the baseline model for the remaining experiments, while the other operators were discarded.

Briefly summarized, the objectives of the first experiment are to:

1) Construct models with different ConvGNN-operators, check if the models can predict the load-shedding values for the dataset with a single contingency.
2) Compare runtime and accuracy between the ConvGNN-operators.
3) Investigate the model's capability of predicting the load-shedding values for the cases with two- and three-line contingencies while only being trained on the case with a single line contingency.
4) Investigate the change of behaviour and accuracy of the best performing model after being trained on multiple datasets with a differing number of induced contingencies.

6.1.1 Experiment 1.1, Comparative Test of ConvGNN Operators

In this first subsection, the results of the three models using the GAT, GCN and GraphConv operator are displayed. To establish a ground of comparison between the operators, each model was trained and tested solely on the **1-contingency** dataset set for 100 epochs. For each model, the time-to-train was estimated. To compare the accuracy of operators, the RMSE value for each model tested on the test-sets of the **1, 2 and 3-contingency** dataset was stored alongside the average run-time of the test-sets. These values can be found in **Table *4***. The table is presented to give the reader an insight into the average test-error of the operators and

the time it took to train and test each operator. As some of the operators converged before the 100-epoch mark, the time-to-train has been estimated to a 100-epoch mark based on the converged epoch number and the time it took before convergence.

Table 4: The RMSE Error for all operators on the test-sets of the three-contingency dataset. The table also includes the train- and test time for all models on the 1-contingency dataset after 100 epochs of training with a batch size of 15.

Model	Error (RMSE) 1-Contingency	Error (RMSE) 2-Contingency	Error (RMSE) 3-Contingency	Train Time (s)	Test Time (s)
GraphConv	**0.1470**	0.4445	1.0712	**1340**	**3.031**
GAT	0.2405	**0.3462**	**0.9157**	3882	8.728
GCN	0.2806	0.4407	1.0669	1858	3.935

As seen in the table, the two most outstanding operators were the GraphConv and the GAT operator. Overall, the GAT operator's predictive results were better than that of the GCN and GraphConv operator. This was especially apparent for the 2- and 3- contingency dataset, though the GraphConv operator performed best on the 1-contingency dataset. Despite the GAT operator being superior on the 2- and 3-contingency dataset, the GraphConv operator was faster at both training and testing compared to the other operators. The GAT-operator performance was formidable, but the operator used almost three times as long to train and test compared to the other operators, rendering the model obsolete. Based on both the error across the contingency-datasets and the time-to-train of the models, the GraphConv-operator was deemed to be the better operator for the remaining experiments of this thesis. The explanation behind the speed and accuracy of the GraphConv operator could how it made use of the local and global feature information combined with the edge-weights. While all the operators used this information, the simple approach of the GraphConv's operator to the learned weight for both nodal and local feature-information may have been decisive for both its speed and accuracy in this case.

Figure 10 displays the evolution of the RMSE-error for all three datasets as the GraphConv operator model was trained. After each epoch of training, the parameters of the model were updated based on the training-error. Each test-set was then fed to model to determine the

model's accuracy and did not affect the training of the model in any way. From the figure it is observable that the error of the 1- and 2-contingency dataset was initially unstable with the error fluctuating in the initial epochs before stabilizing. The model did not fully converge to what is probable a local minimum after 100 epochs, but with additional training the model would most likely have converged in the same error range. A notable observation is that some of the spikes in error as the model was being trained occurred in all test-sets simultaneously, indicating a form of correlation between all three contingency-datasets. The magnitude of the spikes decreases with each contingency, indicating a stronger correlation between the 1- and 2-contingency dataset compared to the 1- and 3 contingency dataset.

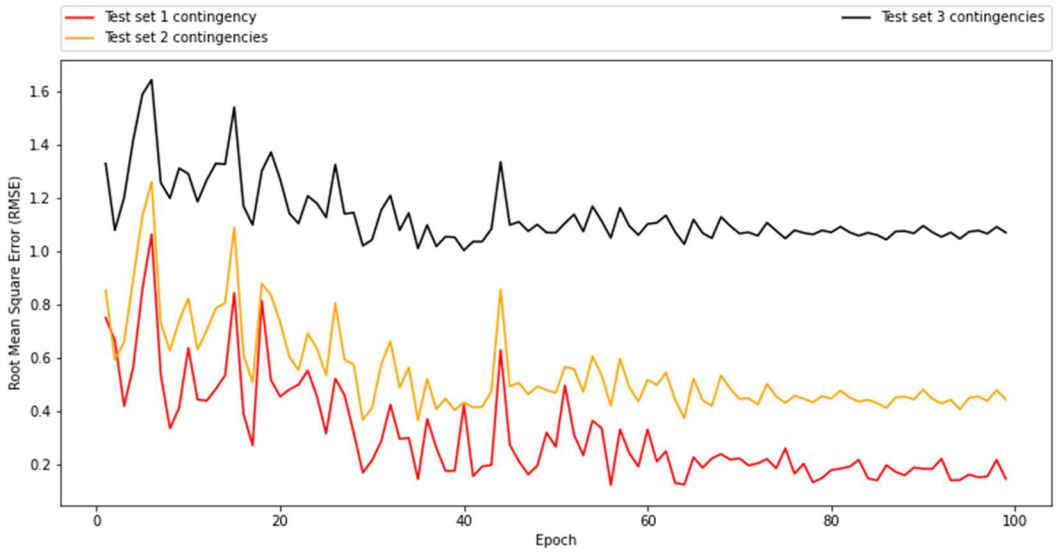


Figure 10: The evolution of the RMSE per epoch for all three contingency datasets for the GraphConv.

Figure 11 displays the target- vs the predicted-value scatter plot for the GraphConv operator. Similar plots for the GAT- and GCN operator can be found in Appendix B. The plotted red line indicates where the predicted value $\hat{y}$ is equal to the target value y. A large deviation from red line indicates a poor prediction, while points close to the red line indicate good predictions. As is observable from the plots in **Figure *11***, the deviation from the red line increases with the number of contingencies, though the magnitude of deviation for the majority of the scenarios is small. For the 2-contingency dataset, the model is consistently underpredict most load-shedding values. As for the 3-contingency dataset, the model had a tendency of overpredicting many of the values in the range of [0, 20]. As the shed system-load passed the 20-value mark, the model began to consistently underpredict the values instead. A possible explanation for the increase in deviation from the target value could be

that the model was unable to comprehend that additional contingencies would lead to an increase in the shedding of system load. Despite this, there is a good overlap of points on the prediction line across all datasets, and the Graph- conv model is capable to a high degree of predicting both small and large load-shedding values across contingencies.

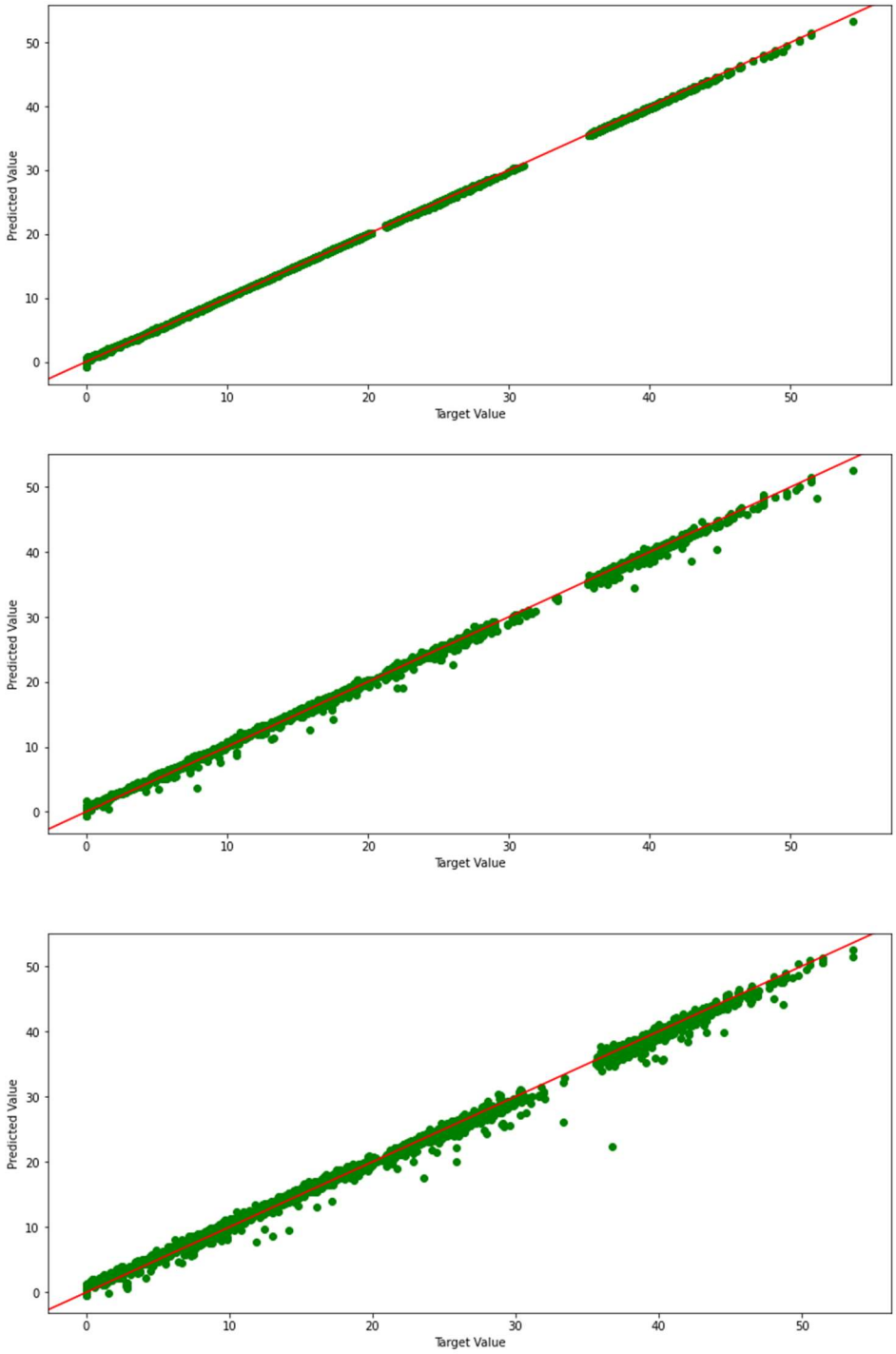


Figure 11: Scatter plots for the target value vs the predicted value for the GraphConv model. The plots are shown in an ascending order of contingencies, with the results of the 1-contingency dataset displayed at the top.

In the following experiments, the number of ConvGNN-operators have been reduced from three to one. The GraphConv operator was chosen as the baseline operator for the following experiments as it was the fastest operator, while having decent predictive qualities.

6.1.2 Experiment 1.2, GraphConv model trained on multiple datasets.

This section displays the results where the GraphConv model was sequentially trained on two datasets, the 1-contingency and 3-contingency dataset. The model was first trained solely on the 1-contingency training-dataset for 100 epochs. The model was then trained on 20% (Scenario 1) and 80% (Scenario 2) of the 3-contingency dataset for another 100 epochs. While the models are set to train for a pre-defined number of epochs, an early stop mechanism has been added to cease training if the model show no sign of improving with sequential training.

Table 5 displays the summed RMSE of the model after being trained on the 1-contingency (pre-RMSE) and the RMSE after being trained on the 3-contingency dataset (post-RMSE). The table also shows the percentage of the 3-contingency dataset the model was trained on, indicated with their corresponding scenario. The summed post- and pre-RMSE value for this experiment was calculated by adding up the model's RMSE value on all three test-datasets after training had concluded, as shown in equation ***(23)***.

$$\mathrm{RMSE}_{\mathrm{tot}} = \mathrm{RMSE}_{\mathrm{Test\ cont1}} + \mathrm{RMSE}_{\mathrm{Test\ cont2}} + \mathrm{RMSE}_{\mathrm{Test\ cont3}} \quad \textbf{(23)}$$

The total RMSE value was calculated to check if training the model on subsequential datasets would yield a lower total error. The difference in RMSE is estimated based on the difference between the post-RMSE and the pre-RMSE.

The results shown in the table indicates that both scenario 1 and scenario 2 gave a negative difference between the post- and pre-$RMSE_{tot}$, meaning that subsequentially training the model on the 3-contingency dataset improved the model's ability to predict the load-shedding values across contingencies, with scenario 2 being the superior approach to reduce the overall RMSE.

Table 5: Table displaying the Pre, Post and RMSE difference between the two scenarios, indicated with the percentage of training data of the 3-contingency dataset. A negative RMSE difference indicate a reduction of total RMSE.

Scenario	Pre-$RMSE_{tot}$	Post-$RMSE_{tot}$	$RMSE_{tot}$ difference	Percentage trained
1	1.6627	1.3316	-0.3311	20%
2	1.6627	1.2717	-0.3910	80%

The evolution of the RMSE after being trained on both datasets can be observed in **Figure 12** and **Figure 13**, respectively. The dashed line marks the transition from the model being trained on one training-set to the other. The position of the dashed line can change from experiment to experiment due to the implementation of the early stop mechanism. From both **Figure 12** and **Figure 13**, one can observe that after the change in training-data, the error for both the 2- and 3-contingency test-set dropped instantaneously while the error for the 1-contingency increased slightly.

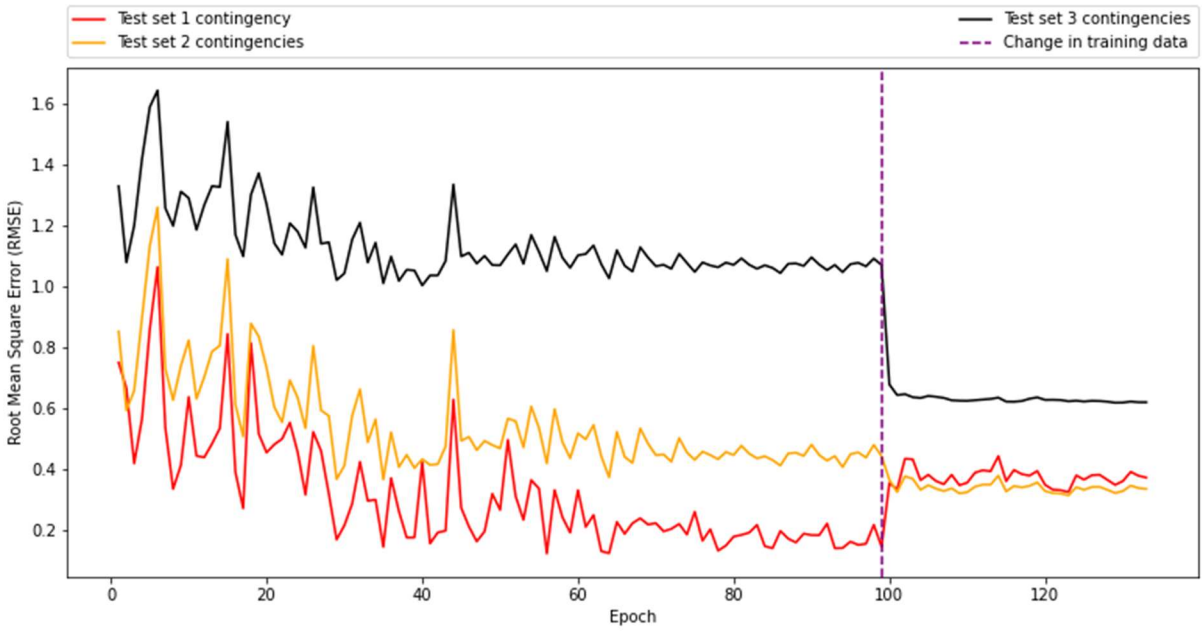


Figure *12*: The evolution of the RMSE per epoch for all three contingency datasets before and after the model was trained on 20% of the 3-contingecy dataset. The purple dashed line indicates the change in training data.

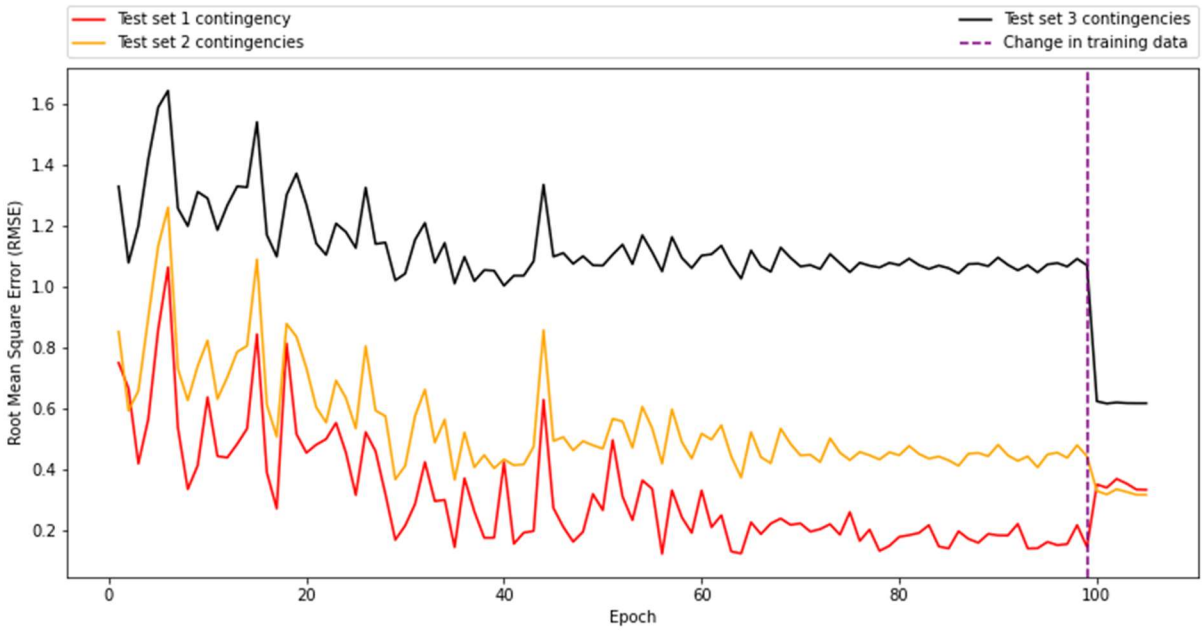


Figure 13**: The evolution of the RMSE per epoch for all three contingency datasets before and after the model was trained on 80% of the 3-contingecy dataset. The purple dashed line indicates the change in training data.**

6.2 Experiment 2, Load Perturbation, 1-Contingecy.

In the second experiment, the load of the system was gradually increased by a percentage for each simulated dataset to investigate the GraphConv model's resilience against load-perturbations. In the experiment, the topology was kept static, and a single contingency was induced at each simulation step. The set-values for the load were initially equal across the dataset, but the values were increased by adding a scaling factor k = 1.05, 1.10, 1.20 for each of the simulated datasets, respectively. In total, three additional datasets were made with the scaling load factor, increasing the system load with 5%, 10% and 20%. To avoid any confusions, the datasets are renamed to **low-load** (5%), **medium-load** (10%) and **high-load** (20%), respectively.

The objectives of the second experiment were to:

1) Investigate the model's capability of predicting the load-shedding values for the cases with 5%, 10% and 20% extra load while only being trained on the case with the standard load condition.
2) Investigate the change of behaviour and accuracy of the model after being trained on multiple datasets with a differing load. **Scenario 1** is the model trained on **20%** of the **high-load** data, while **Scenario 2** is the model trained on **80%** of the **high-load** data. In both scenarios, the model was first trained on the normal load-condition dataset.

6.2.1 Experiment 2.1, Prediction Across Varying Load Without Additional Training

The first sub-section of experiment 2 displays the results of the GraphConv-model trained solely on the 1-contingency dataset with the standard load-condition. After training had concluded, the model was subsequentially tested on the test sets of all the load-condition datasets. **Table 6** shows the error of the trained model on said test-sets. As with the contingency-perturbation experiment, the error of the model increased slightly as the system load was further perturbated, although the discrepancy of error between the lowest and highest error is lower for the load-perturbation experiment compared to the contingency-perturbation experiment.

Table 6: **Table displaying the RMSE error of experiment 2.1 for the GraphConv operator on the test-set for all four load-condition datasets after being trained solely on the normal load-condition dataset.**

Dataset	Error (RMSE)
Normal Load Condition	0.1765
5% Additional Load	0.1813
10% Additional Load	0.1865
20% Additional Load	0.2047

Figure *14* displays the evolution of the RMSE per epoch for the test-set of all four load-condition datasets as the model was being trained. From the figure it is observable that the RMSE of each load-dataset decreases sharply in the initial epochs with some initial fluctuation. As the model was further trained, the error began to stabilize though the model did not fully converge after 100 epochs. Additional training would most likely cause the model to converge with roughly equal error.

A notable observation from **Figure *14*** is that the spikes in error as the model is being trained occurs in all test-sets simultaneously. This indicates a strong correlation between the load-datasets, which may explain why the error-discrepancy is much lower for the load-perturbation datasets compared to the contingency-perturbation datasets. Because of the strong correlation between the error-fluctuation of the datasets, it can mean that further load-

perturbation does not change the load-shedding pattern in the system. Rather, only the magnitude of shed load changes. This is a stark contrast compared to **Figure *10***, where there is a slight correlation between the spikes in the error, but the spikes are not equally large across all three datasets.

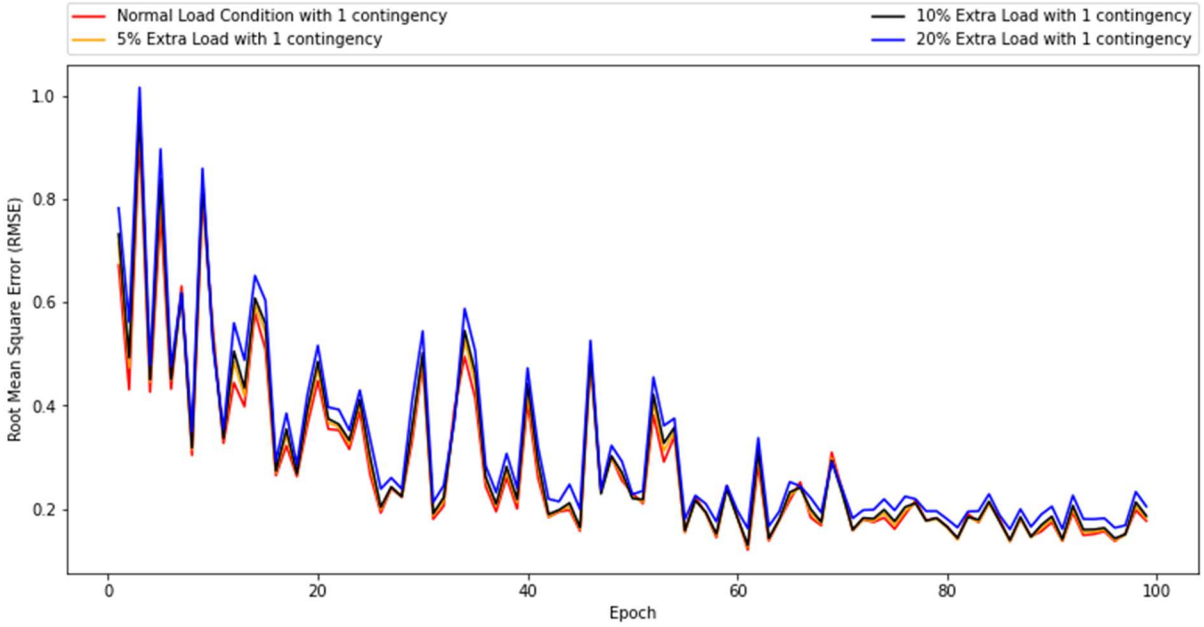


Figure 14*: The evolution of the RMSE per epoch for the four load-datasets using the GraphConv operator.*

Figure *15* and **Figure *16*** displays the target-value vs the predicted value scatter plot for the **Normal-load** condition and the **High-load** condition, respectively. As can be observed from both figures, there is a great overlap on the red line for both the normal- and the high-load datasets. As the load in the system increased, the mean for the load-shedding value increased as well. For the **Normal-load** dataset, the mean was situated around the 20-22.5 MW mark, while for the **High-load** dataset the mean was situated slightly higher, around the 25-30 MW mark. For the **High-load** dataset, there was also a few additional datapoints which exceeded beyond the 50 MW-mark which the model was able to predict with a high accuracy. This shows that the model can predict the load-shedding values in higher load-conditions.

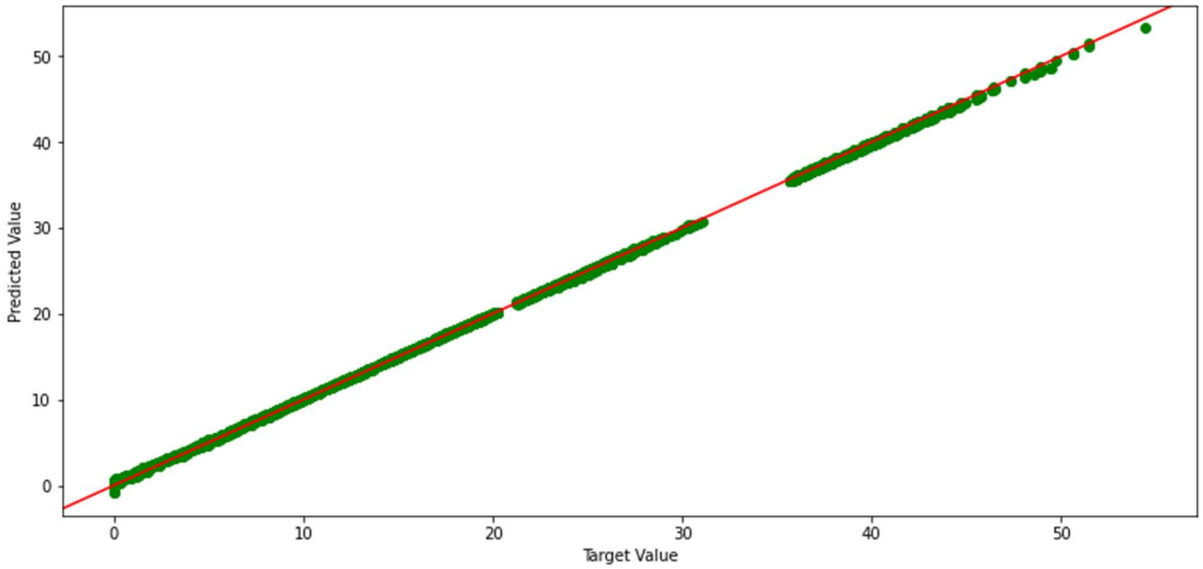


Figure 15*:* Scatter plots for the normal load-condition datasets displaying the target value vs the predicted value.

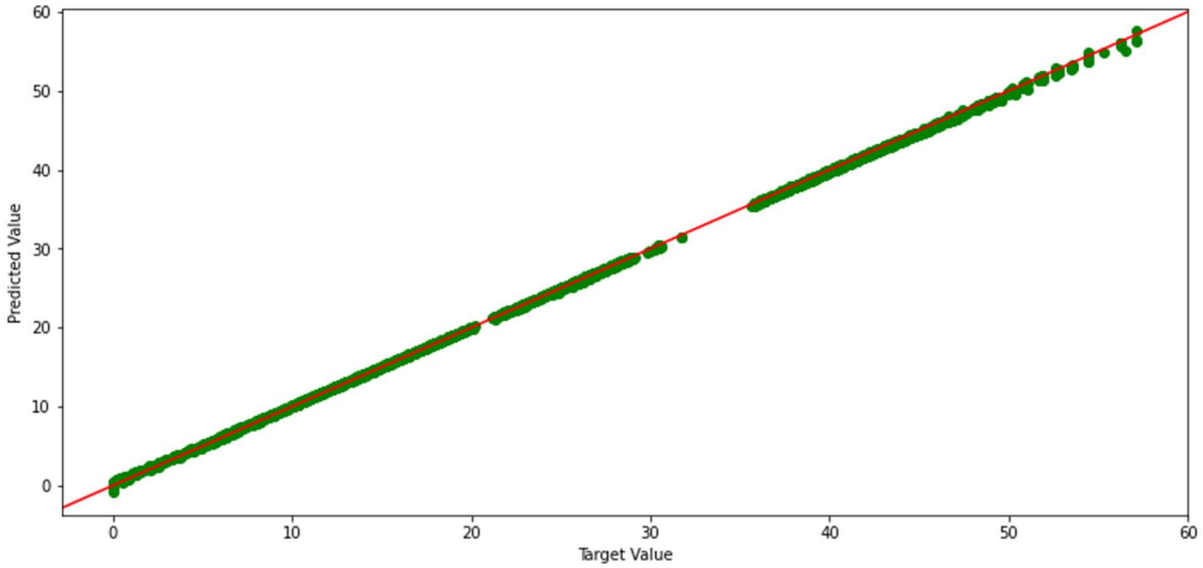


Figure 16: Scatter plots for the high-load datasets displaying the target value vs the predicted value.

6.2.2 Experiment 2.2, Prediction Across Varying Load With Additional Training

This section displays the results where the GraphConv model was sequentially trained on two datasets, the **normal-load,** and the **High-load** dataset. This section includes two scenarios, **Scenario 1** where the model is trained on **20%** of the **high-load** data, while **Scenario 2** is the model trained on **80%** of the **high-load** data. In both scenarios, the same model trained on the normal load-condition dataset was used. ***Table 7*** displays the total RMSE of the models before (pre-RMSE) and after (post-RMSE) being trained on the second dataset for each corresponding scenario. The RMSE was calculated following equation ***(23)***, although slightly altered as the experiment includes all four load conditions. This alteration is reflected in Equation ***(24***).

$$\mathrm{RMSE}_{\mathrm{tot}} = \mathrm{RMSE}_{\mathrm{Test0\%}} + \mathrm{RMSE}_{\mathrm{Test5\%}} + \mathrm{RMSE}_{\mathrm{Test10\%}} + \mathrm{RMSE}_{\mathrm{Test20\%}} \quad (24)$$

Table 7: Table displaying the Pre, Post and RMSE difference between the two scenarios, indicated with the percentage of training data of the High-load dataset. A negative RMSE difference indicate a reduction of total RMSE.

Scenario	Pre-$\mathrm{RMSE}_{\mathrm{tot}}$	Post-$\mathrm{RMSE}_{\mathrm{tot}}$	RMSE difference	Percentage trained
1	1.6203	1.6291	0.0088	20%
2	1.6203	1.5999	-0.0548	80%

Although the difference between the post-RMSE and the pre-RMSE for both scenarios is small, the model saw a slight reduction in the total RMSE in scenario 2. The evolution of the RMSE after being trained on both datasets can be observed in **Figure *17*** and **Figure 18**, respectively. The dashed line marks the transition from the model being trained on one training-set to the other. As can be observed from the figures, the convergence point is vastly different between the two scenarios. The model in scenario 2 converged after only 6 epochs, while the model in scenario-1 needed 89 epochs to converge. Despite the extra training time, the model in scenario 1 saw no improvement in accuracy.

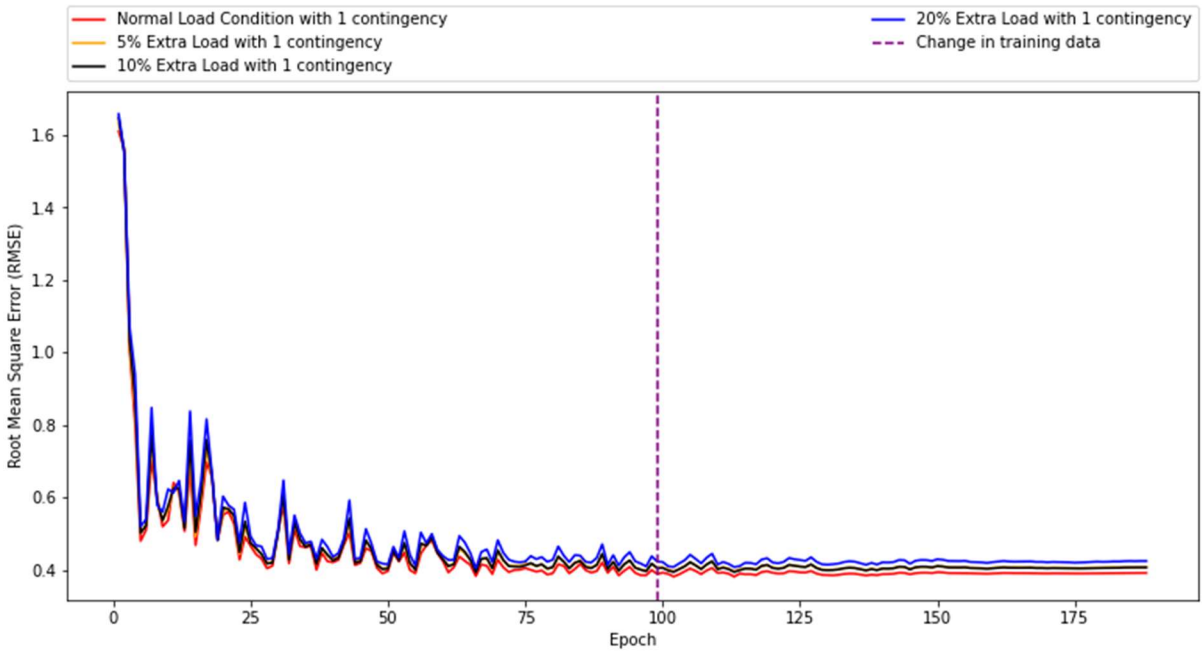


Figure 17: The evolution of the RMSE per epoch for all four datasets before and after the model was trained on 20% of the High-load dataset. The purple dashed line indicates the change in training data.

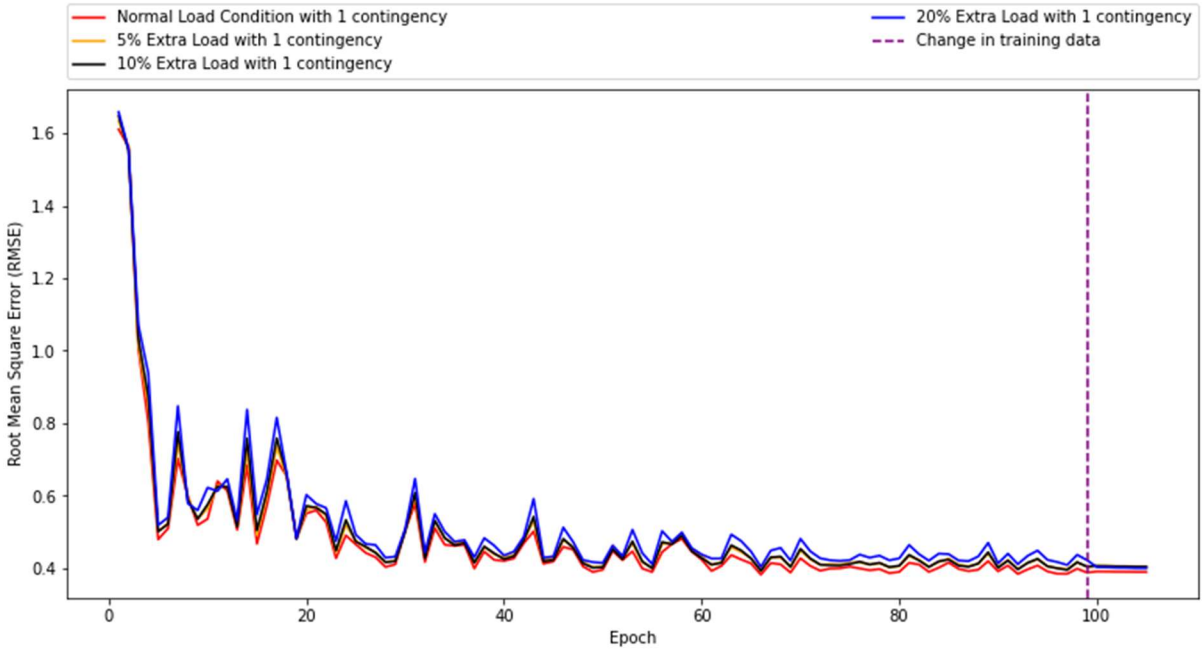


Figure 18: **The evolution of the RMSE per epoch for all four datasets before and after the model was trained on 80% of the High-load dataset. The purple dashed line indicates the change in training data.**

6.3 Experiment 3, Topology and Contingency Perturbation

In the third experiment of the thesis, the topology of the system was altered by adding two additional lines to the system. With the modified topology, three datasets were simulated with a varying number of contingencies (1, 2 and 3-contingencies), similar as in Experiment 1. The model was first trained solely on the 1-contingency dataset with the original topology of the system, and then subsequentially tested on all test-sets to investigate the model's ability to predict across topologies. In the last experiment, the model was then subsequentially trained on 20% (Scenario 1) and 80% (Scenario 2) of the dataset for the 3-contingency with the modified topology.

In a more condensed form, the objectives of the second experiment were to:

1) Investigate the model's capability of predicting the rescheduling values across two topologies while only being trained on one topology.
2) Investigate the change in behaviour and accuracy of the model after being trained on multiple datasets with different topologies. **Scenario 1** is the model trained on **20%** of the 3-contingency data with the modified topology, while **Scenario 2** is the model trained on **80%** of the 3-contingency data with the modified topology. In both scenarios, the model was first trained on the 1-contingency dataset with the original topology.

6.3.1 Experiment 3.1, Prediction Across Topologies Without Additional Training

The first sub-section of experiment 3 displays the results of the GraphConv-model trained solely on the 1-contingency dataset with the original topology. After training had concluded, the model was then subsequentially tested on the datasets with the modified topology and increasing number of induced contingencies. **Table 8** shows the error of the prediction of the model on all test-sets. A notable observation from said table is that the prediction-error on the 1- and 2-contingency dataset with the modified topology is higher as compared to the dataset with the original topology and the same number of contingencies (Experiment 1). A possible explanation for this phenomenon could be that the inclusion of additional lines to the most vulnerable areas in the system made them more robust to contingencies. The model expected the most vulnerable areas to shed load due to the occurred contingencies, but the new connections made it so that the load was shed in different places instead. This change of pattern confused the model, which led to the high prediction error.

Table 8: **Table displaying the RMSE error of experiment 3.1 for the GraphConv operator on the test-set for the 1 to 3-contingency dataset with both the original and the modified topology**

Dataset	Error (RMSE)
1-Contingency, original topology	0.1730
1-Contingency, modified topology	0.5222
2- Contingency, modified topology	0.6214
3- Contingency, modified topology	1.0141

Figure *19* displays the evolution of the RMSE for all four datasets over 100 epochs as the model is being trained on the 1-contingency training set with the original topology. The RMSE of each dataset decreases in parallel with training, with the error stabilizing as the model is being trained. For all test-sets, the fluctuation in the error as the model is being trained occurs simultaneously at some points. Though, the magnitude of the fluctuation varies heavily between the test-sets. While the simultaneous fluctuation indicates a correlation between the datasets, the correlation is not as strong in this experiment as with the load-

perturbation experiment. This further indicates that a change occurred in the load-shedding pattern after the topology of the system was modified.

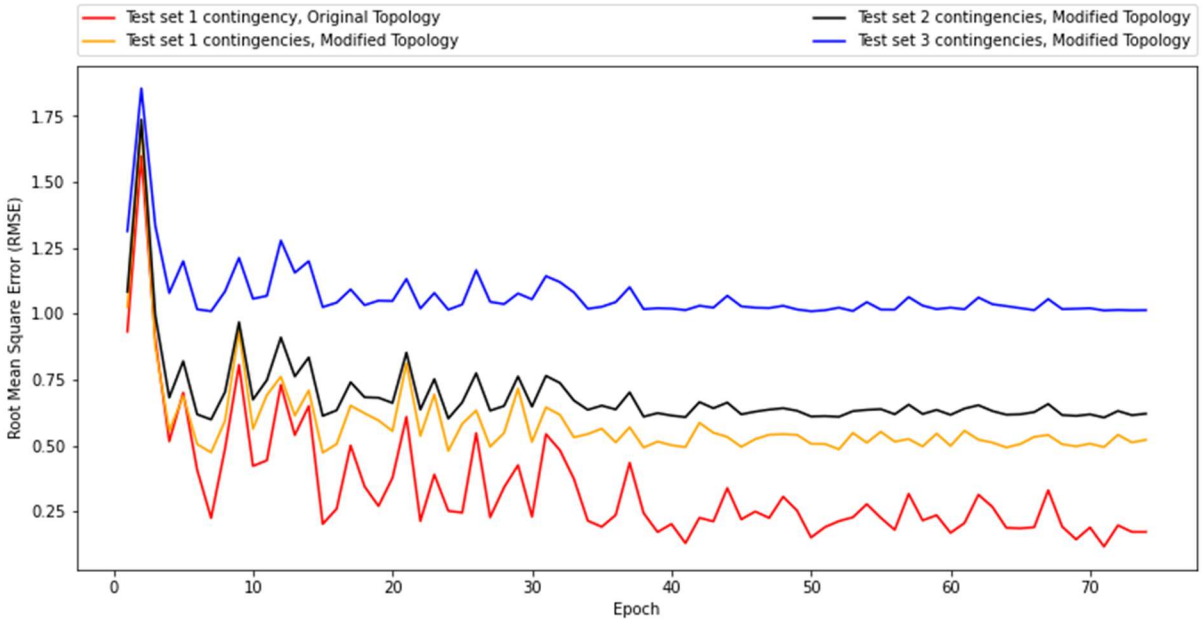


Figure 19: The evolution of the RMSE per epoch for the four contingency datasets using the GraphConv operator.

Figure *20* displays the target-value vs the predicted load-shedding value scatter plot for the datasets used in this experiment. As with the previous plots, the red line indicates where the predicted value $\hat{y}$ is equal to the target value y. As is apparent from studying the figures, the error of the 1- and 2-contingency set is now higher compared to the same contingency-cases in **Figure *11***. From **Figure *20***, the suspicion that there has been a change in the load-shedding pattern for some of the simulated scenarios is further validated. Though, from the figures one can observe that the load-shedding pattern did not completely change for all the simulated scenarios. This can be observed by the parallel dots above the red line for the 1- and 2-contingency datasets, which shows that there are now two patterns for shedding load. The model is only capable of predicting one of the load-shedding pattern. As for the error for the 3-contingency dataset, the overall prediction is decent, but there are now quite a few situations in which the model predicts the load shed to be substantially lower than the actual values.

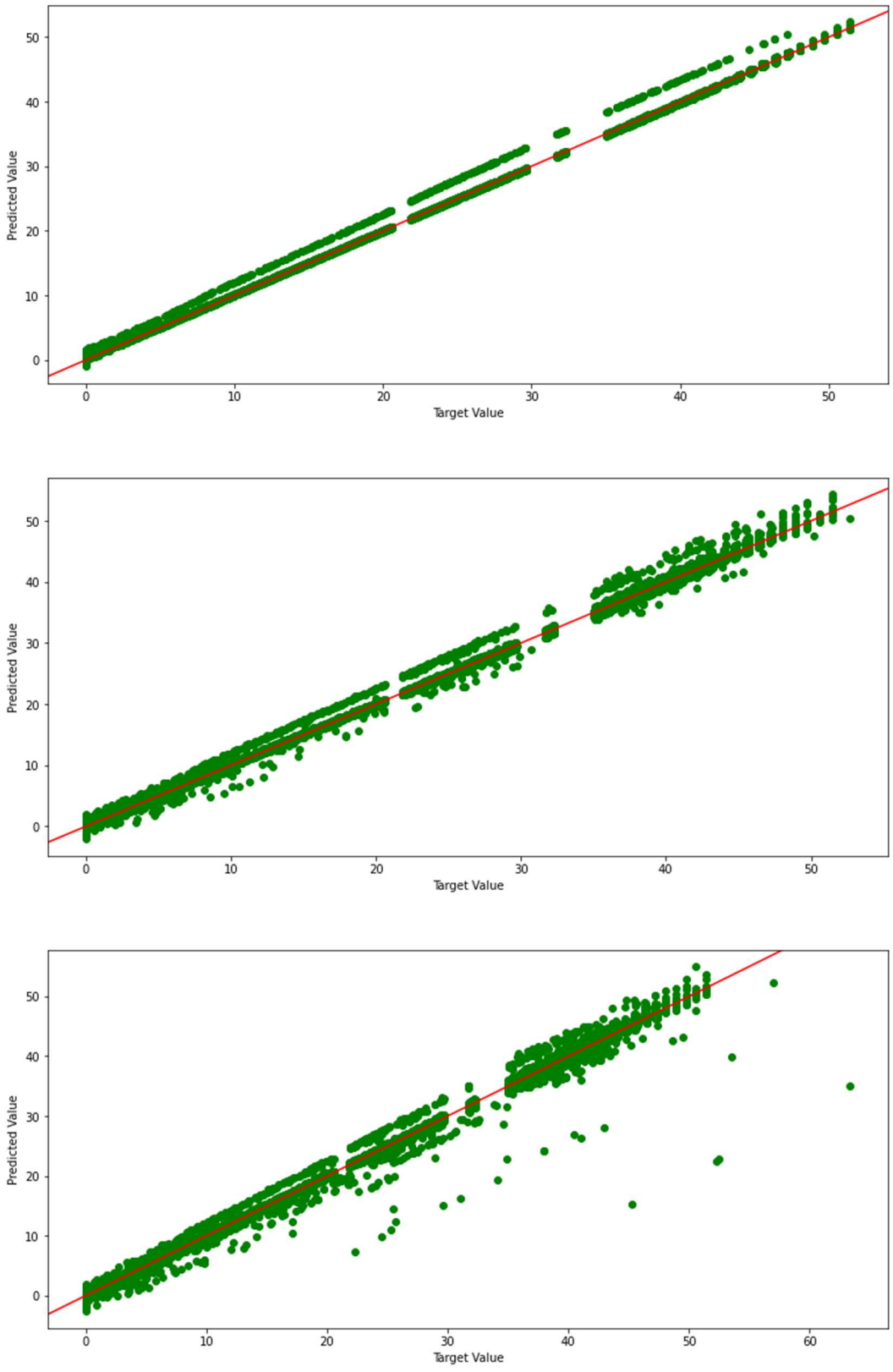


Figure 20: Scatter plots for the 1-,2- and 3-contingency datasets with modified topology displaying the target value vs the predicted value. The red line indicates where the target value equals the predicted value.

6.3.2 Experiment 3.2, Prediction Across Topologies with Additional Training

In this section, the trained model from Experiment 3.1 is sequentially trained on two portions of the 3-contingency dataset with the modified topology, indicated by their respective scenario. **Scenario 1** is the model trained on **20%** of the 3-contingency data with the modified topology, while **Scenario 2** is the model trained on **80%** of the 3-contingency data with the modified topology. In both scenarios, the exact same model trained on the 1-contingency dataset with the original topology was used.

Table 9 displays the total RMSE of the model before (pre-RMSE) and after (post-RMSE) being trained on the 3-contingency dataset with the modified topology for each corresponding scenario. The RMSE was calculated following equation ***(23)***.

Table 9: Table displaying the Pre, Post and RMSE difference between the two scenarios, indicated with the percentage of training data of the 20% load dataset. A negative RMSE difference indicate a reduction of total RMSE.

Scenario	Pre-$\mathbf{RMSE_{tot}}$	Post-$\mathbf{RMSE_{tot}}$	$\mathbf{RMSE_{tot}}$ difference	Percentage trained
1	2.3307	2.0925	-0.2382	20%
2	2.3307	2.0606	-0.2701	80%

As can be seen from the table, both scenarios gave a reduction in the total RMSE, with scenario 2 giving the largest decrease in the total error. This meant that subsequential training of the model led to an improvement of the model's predictions. **Figure *21*** and **Figure *22*** displays the evolution of the RMSE as the model was being trained for both Scenario 1 and Scenario 2, respectively. While the model was trained on the 3-contingency dataset in both scenarios, indicated with the blue line, the error of the test-set of the specific dataset did not change substantially with additional training. A normal assumption would be that the model's prediction-error on a specific case would decrease as the model was trained for that case. However, this was seemingly not the case. As for the 1- and 2-contingency dataset with the modified topology, both saw a sharp decrease in their error, resulting in a reduction of the

total error for all datasets. For both scenarios, the error of the 1-contingency dataset with the original topology saw an increase in error after the change of training data.

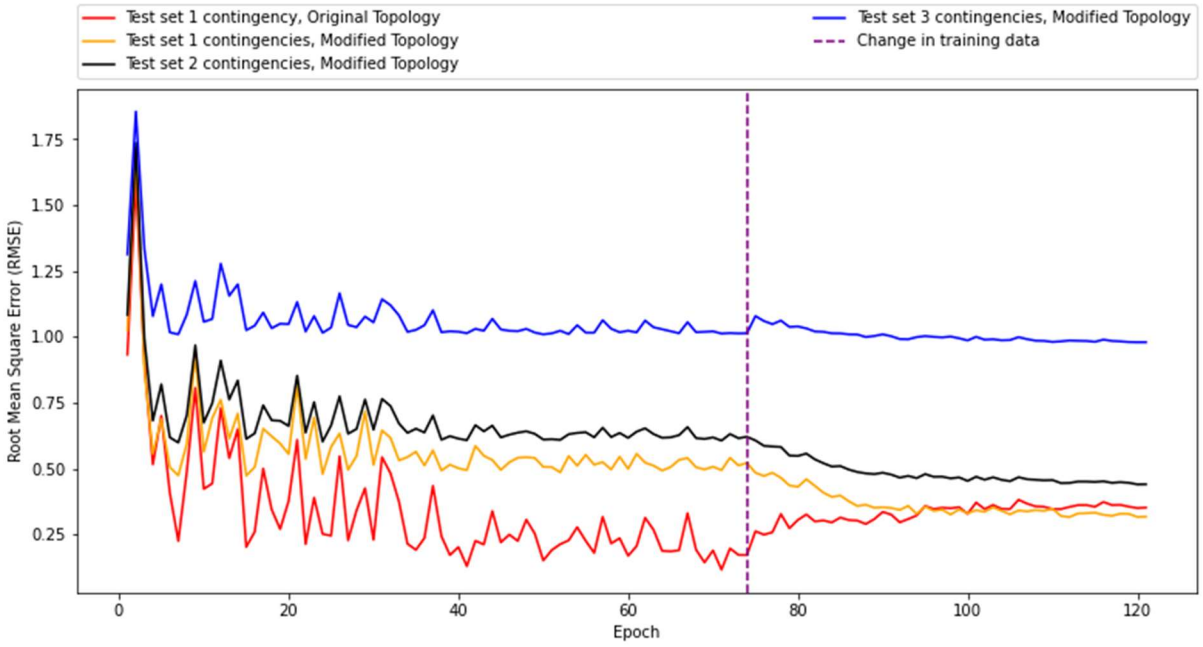


Figure 21: The evolution of the RMSE per epoch for all 4 datasets before and after the model was trained on 20% of the 3-contingency dataset. The purple dashed line indicates the change in training data.

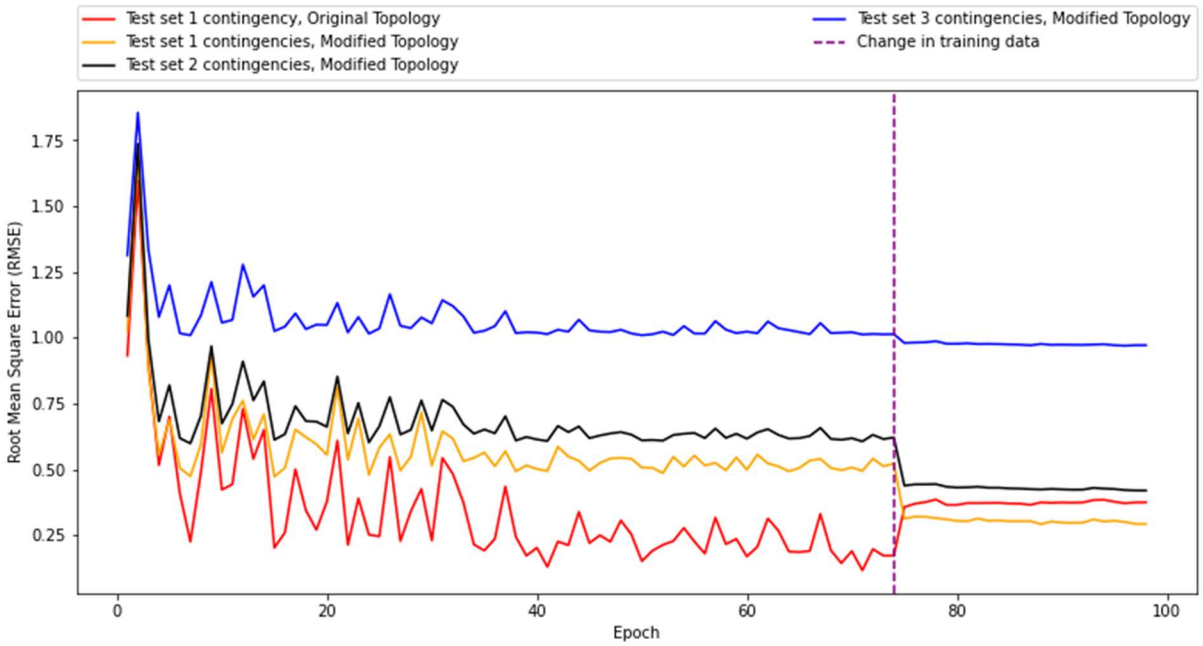


Figure 22: The evolution of the RMSE per epoch for all 4 datasets before and after the model was trained on 80% of the 3-contingency dataset. The purple dashed line indicates the change in training data.

6.4 Experiment 4, Case Study

In the fourth and last experiment, the best performing model from experiment 1 was used in a side-by-side case study with the OPF in the Monte Carlo simulation. The goal of the case-study was to compare both the accuracy and the run-time of both the MC-simulation and the best-performing ML-model. The case-study also wanted to investigate the difference in run-time between different batch-size of the input dataset to the ML-model. To this end, two different batch-sizes were used for the input dataset.

In the experiment, 5000 datapoints were simulated using the Monte Carlo Simulation. Additionally, a new set of load values were used to create a different load-situation to test the model. These values did not differ substantially from the load-values used in the previous experiments. For each simulation point, the resulting load-shedding values, and the features of the system, such as the generated power, line flow, line status etc were stored and used to create the new dataset for the trained ML- model. The run-time for the MC simulation was only estimated for the last OPF in the simulation, which was only called when an overload occurred in the system due to the contingencies. Thus, the run-time would often be zero due to some scenarios not leading to system overloads. The run-time for the ML-model was estimated based on the time it took for the model to return the predicted values and did not include the run-time for creating and pre-processing the dataset.

As with the previous simulated dataset, some of the load-scenarios led to a non-converged state for the OPFs, and these scenarios were discarded. However, to get the correct estimated speed-time for both the model and the OPF, it was imperative that the estimated run-time was based only on the cases in which the OPF did converge. Therefore, the estimated time for both the OPF and model was based only on the number of cases in which the OPF did converge. As for the model, the run-time is dependent on the batch-size of the dataset fed to the model. The larger the batch-size, the faster the model is.

Table *10* displays the run-time of both the OPF and the ConvGNN-model and the RMSE prediction-error of the model. The training-set of the model used a batch-size of 15, while the test-set used a batch-size of 5. The RMSE error was calculated based on the predictions of the model and the actual target value produced by the MC simulation.

Table 10: The run-time for the OPF and ConvGNN-model, and the prediction error of the model using a batch-size of 15.

Model	Run time (s)	Prediction Error (RMSE)
OPF	236.51	0.162
GraphConv	0.8042	

Based on the results of the case-study, the model is around 294 times faster than the OPF at calculating the load-shedding values. As for the accuracy, the predicted value vs the target value is displayed in **Figure *23***. From the figure one can observe that the ML-model's predictions are almost equivalent to that of the OPF.

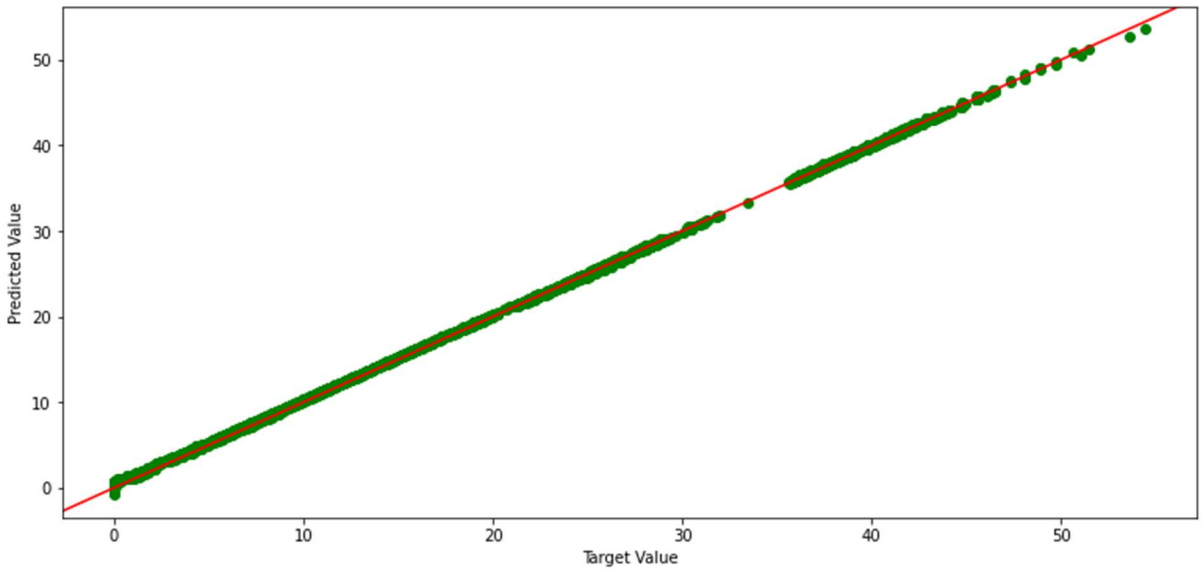


Figure 23: Scatter plots for the predicted load-shedding value made by the ML-model and the target-value estimated by the MC-simulation using a batch-size of 15. The red line indicates where the target value equals the predicted value.

To investigate the difference in prediction-speed with differing batch-sizes, the input dataset was recreated using a batch-size of 1 instead of 5. The same trained model was used, the only difference was the batch-size of the input dataset. The new run-time of the model can be found in **Table *11***. With the batch-size of 1, the model was now only around 80 times faster than the OPF, making it around 3.675 times slower than with the batch-size of 5. As can also be observed from the table, the estimation-error did not change with the batch-size.

Table 11: The run-time for the OPF and ConvGNN-model, and the prediction error of the model using a batch-size of 2.

Model	Run time (s)	Prediction Error (RMSE)
OPF	236.51	0.162
GraphConv	2.94	

The reason for the conducted experiment with differing batch-sizes is to display the difference when screening multiple contingency-cases versus a single-contingency case. If the model is to only predict single scenarios, the gained speed of replacing the OPF with the ML-model is much lower than if the model was to be used for predicting multiple scenarios at once. This means that the optimal use of the model is not in real-time operational use, but rather to investigate multiple scenarios simultaneously in a Contingency Analysis.

As a disclaimer, the speed of the ML-model is highly dependent on the python script used to estimate the model's prediction values. Further optimization of the Python code can thus lead to an improved run-time, which can further increase the discrepancy between the run-times of the OPF and the ML-model.

7 Node-level Predictions

This part of the thesis presents the results of the partial node-level predictions of the models. The purpose of this section is to display that the model works on both a node-level and system-level, showing the flexibility of the model. One major flaw with the system-level prediction is that no information is given regarding which bus that shed load based on the system-state. Rather, only information that load was shed in the system is given, which is not particularly useful in most cases except if the goal is to investigate the total system cost of different long-term investments. Through node-level predictions it can become easier to understand why the prediction-error increases as the key-parameters of the system is increasingly perturbated, by investigating the node-wise error rather than the system error.

To this extent, the same datasets were used in these experiments as with the previous section. Additionally, due to time-constraints, the node-level section will not explore the use of different operators as the previous section did, nor will it be conducting the experiments in which the model was trained sequentially on different datasets.

7.1 Experiment 1, Contingency Perturbation

As with Experiment 1 for the system-level, the model using the GraphConv operator was trained on the 1-contingency dataset and then subsequentially tested on all the three-contingency datasets. The result of the model is shown in **Figure *24***, where the result is shown on a per-node basis rather than as a single value, and the mean error for each contingency dataset is shown in **Table *12***.

Both the predicted and target value was estimated by taking the mean of each nodes' values, averaging over the predictions and target-values, following the equations given in Section 5.2.1. The RMSE was estimated by averaging over each node's prediction-error. As is apparent from **Figure *24***, The model was able to predict the correct load-shedding value per node to a high degree, especially for the 1- and 2-contingency dataset. The model's prediction started to waver for the 3-contingency dataset, where the overlap between the predicted-values and target-values became slightly worse.

As the number of induced contingencies increased, the load-buses 3-7 began to shed more load for each induced contingency, while bus 15 shed less. A potential explanation for this occurrence is that the system prioritized shedding load in the cheapest areas, but it is only able

to shed a finite amount before all the load in the area has been shed. As the number of contingencies increased, the system had to shed additional load to compensate for the contingencies. As the cheapest areas were unable to shed more load, the system had to begin shedding load in additional areas which it previously did not shed in. This changed the load-shedding pattern, which the model was unable to pick up.

Table 12: The prediction error for the node-level model using the GraphConv operator for all three contingency datasets.

Dataset	Error (RMSE)
1-Contingecy	0.1412
2-Contingencies	0.2552
3-Contingencies	0.3831

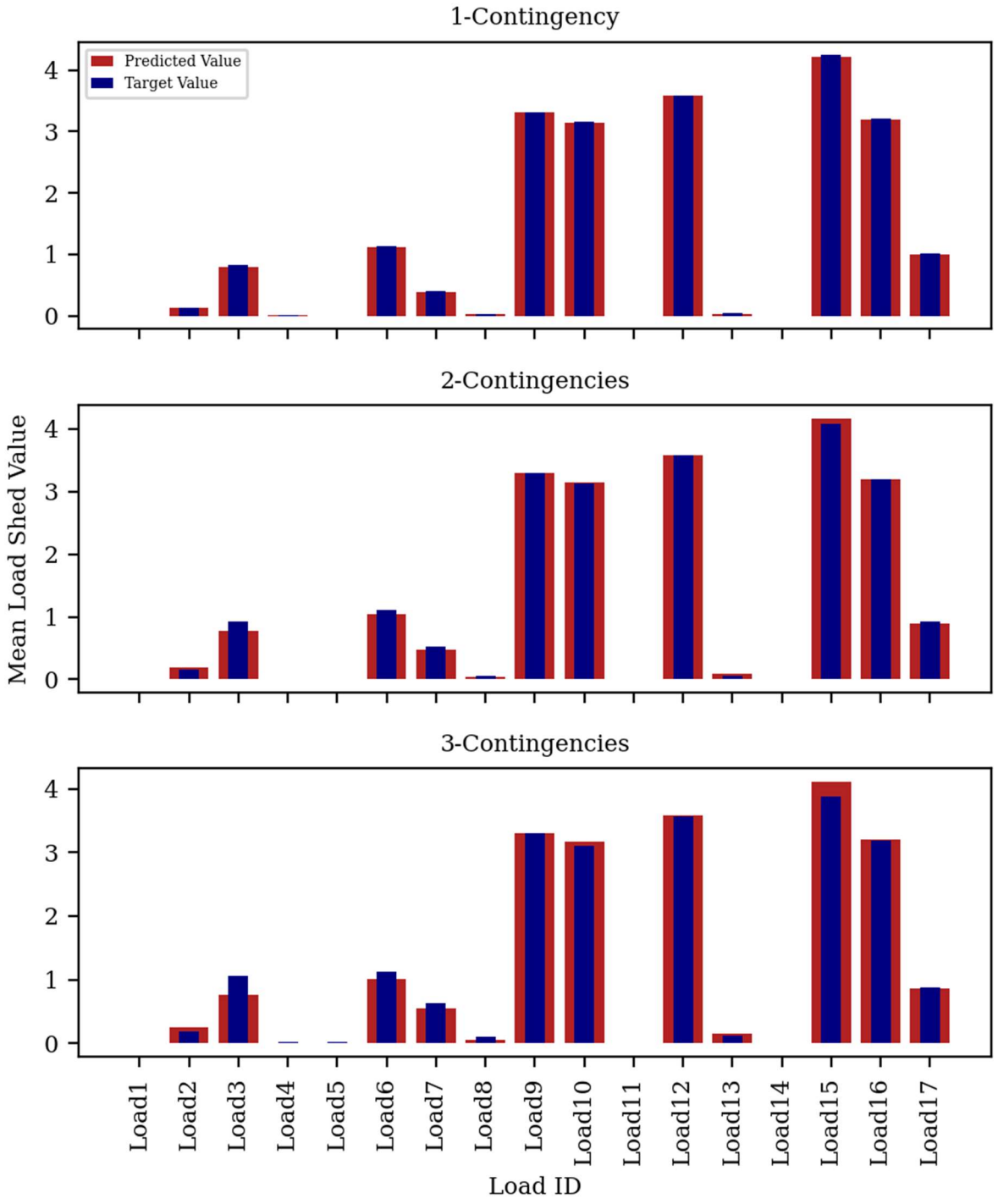


Figure 24: The predicted load-shedding value vs the target value per load-bus in the system for all three contingency datasets.

7.2 Experiment 2, Load Perturbation

For the varying load-prediction experiment, a model using the GraphConv operator was trained on the normal load dataset that induced a single contingency per simulated datapoint and then subsequentially tested on the three additional load datasets, similar as with Experiment 2 on the system-level. The result of the model is shown in **Figure *25***, where the result is shown on a per-node basis, and the error per dataset is given in **Table *13***.

As with the system-level model, the node-level model predicted the load-shedding values across load-perturbations with a high accuracy. As the load in the system gradually increased, the error of the model increased accordingly, though the increase was fractional. This was further validated in **Figure *25***, which displays a great overlap between the predicted values and the target values for all three datasets.

From the figure, one can observe that there are no clear changes in the load-shedding pattern as the load in the system increases. The only visible change as the load in the system increased, is an increase in the difference between the predicted vs target-value for load-bus 2 and 13. As the load in the system increases, the system sheds additional load at these two buses which the model is unable to pick up.

Table 13: The prediction error for the node-level model using the GraphConv operator for all four load-condition datasets.

Dataset	Error (RMSE)
Normal Load Condition	0.1412
5% Additional Load	0.1484
10% Additional Load	0.1596
20% Additional Load	0.1740

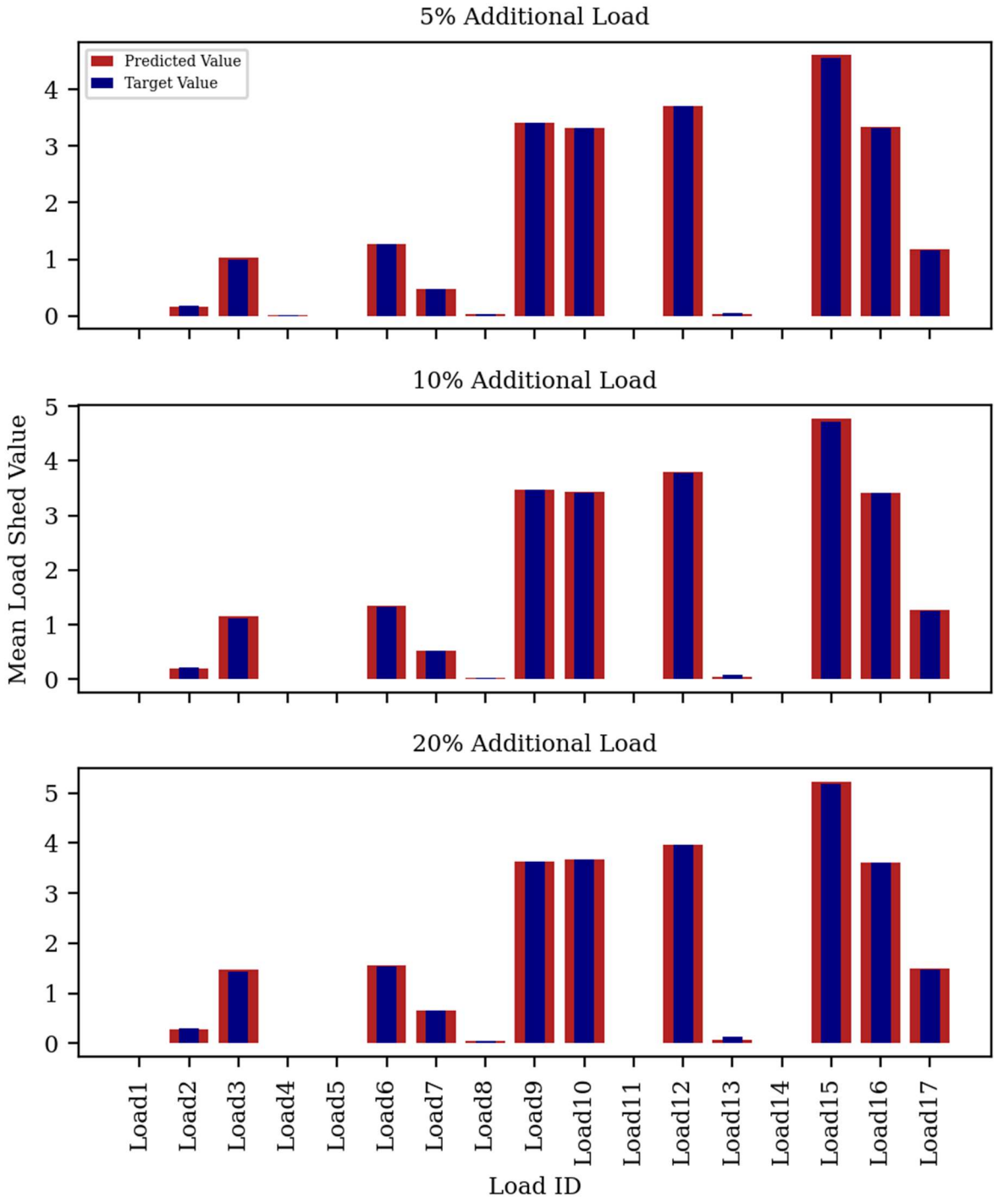


Figure 25: The predicted load-shedding value vs the target value per load-bus in the system for the three datasets with additional load.

7.3 Experiment 3, Topology and Contingency Perturbation

As with Experiment 3 for the system-level, the model using the GraphConv operator was trained on the 1-contingency dataset and then subsequentially tested on all three-contingency datasets with the modified topology. The result of the model's predictions is shown in **Figure *26***, where the predictions are displayed on a per-node basis rather than as a single value. The mean error for each contingency dataset is shown in **Table *14***

As can be observed when comparing **Figure *24*** and **Figure *26***, the modification of the topology had a slight impact on the model's load-shedding pattern. The most notable change was that load-buses 7 and 8 shed less load for each contingency induced. This occurred as an additional line was connected between the two buses, making the two buses more robust to contingencies which led to a reduction of the average shed load in the two buses. Buses 2 and 6 also shed less load after the topology of the system was modified, although none of the two buses had an additional line connected to them. This meant that the modification of the system had an indirect effect on bus 2 & 6, making them shed less load. For the other buses, the pattern was similar as with the contingency-experiment with the original topology, shown in Section 7.1.

Based on the observed changes in pattern in both Experiment 1 and 3, the increase in system-parameter perturbation, such as the induced contingencies per simulation step and the topology, influences the load shedding pattern. Though, the effect varies.

Table 14: The prediction error for the node-level model using the GraphConv operator for all four contingency-dataset with both the original and the modified topology.

Dataset	Error (RMSE)
1-Contingency, original topology	0.1412
1-Contingency, modified topology	0.2602
2- Contingency, modified topology	0.3614
3- Contingency, modified topology	0.4899

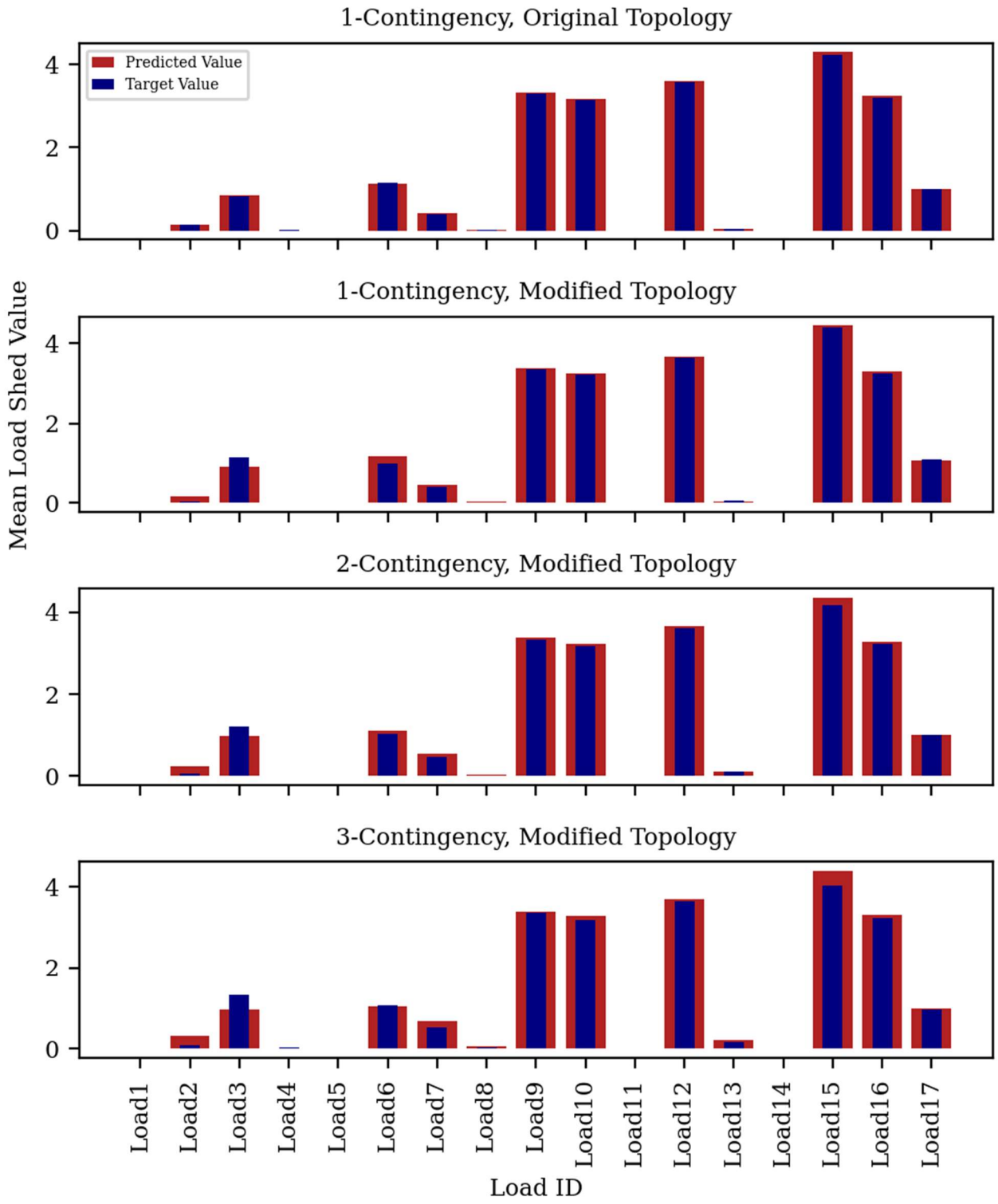


Figure 26: The predicted load-shedding value vs the target value per load-bus in the system for all three contingency datasets with the modified topology.

8 Discussion

8.1 Model Results

In the result-section it was found that the constructed ConvGNN-models were able to predict the load-shedding values across minor system-perturbations such as the number of induced contingencies per simulation point, load-perturbations, and slight changes in the topology of the system with a high accuracy for both the system- and node-level. As the perturbation of the system-parameters increased, the model's accuracy decreased with a varying degree depending on the parameter.

Predictions across system-load perturbations were the easiest task for the models, while the addition of extra induced contingencies per simulation step was the hardest for the model, both with and without the modified topology. An interesting found is that additional system-parameter perturbations did not alter the prediction-pattern of the models. The models' prediction patterns were mostly consistent across all datasets, however the load-shedding pattern of some of the datasets changed as the system-parameters were perturbated. This meant that the bigger the perturbation was, the larger the predicted values deviated from the ground-truth values.

8.1.1 System Level

From the experiments conducted on a system-level, it was found that the GraphConv operator was the best performing operator compared to the GCN and GAT operator. This conclusion was based on an evaluation of the operators' predictive abilities and the time-to-train and time-to-test run time. While the GAT-operator performed slightly better on the 2- and 3-contingency datasets in Experiment 1, the GraphConv operator was superior in both terms of overall speed and prediction accuracy on the 1-contingency dataset. The large discrepancy in speed between the GraphConv and GAT operator was the main motivator as for why the GraphConv operator was chosen as the baseline model for the remaining experiments of the thesis.

The difference in prediction speed between the operators was most likely caused by the complexity of each operators' message-passing method. Both the GCN- and the GraphConv operators has a much simpler method of message-passing compared to the GAT-operator. While it is hard to pinpoint the exact cause for success, a plausible explanation as for why the

GraphConv's operator prediction were so successful for the 1-contingency dataset could be in the how the operator made use of the local and global feature information combined with the edge-weights. While all the operators utilized this information, the approach of the GraphConv to the learned weight for both nodal and local feature-information may have been decisive for its success on the 1-contingency dataset. Though, the GAT operator showed signs of generalizing better to the datasets with additional contingencies induced.

For the GraphConv model, prediction across load-perturbations was the easiest task for the GraphConv model, while the contingency-perturbation experiment was the most difficult. The addition of extra contingencies in the system saw the accuracy of the model decrease sharply per contingency added.

On the system-level, it was not always apparent as to what caused the increase of prediction-error as the system-perturbations increased. The models' predictions tended to deviate both higher and lower compared to the actual value between experiments. For both the contingency-perturbation experiment (Experiment 1, section 6.1) and the topology contingency-perturbation experiment (Experiment 3, section 6.3), the models consistently underpredicted the load-shedding values as the number of induced contingencies increased. For the load-perturbation, there were no visible changes in the prediction pattern as the system load increased.

An explanation of this phenomenon could be that the models are trained on data that follows a set pattern based on the number of induced contingencies in the system. The system shed load according to the number of occurring contingencies, and the load-shedding patterns changes with the number of induced contingencies. As the models were only trained on a single-contingency dataset, the models were only equipped to deal with the single-contingency patterns. For the load-perturbation case, the load-shedding pattern did not change as the load increased, which could explain the superb performance of the models.

For the contingency-perturbation experiment and the modified-topology experiments, sequential training on both the 1- and 3-contingency dataset led to a reduction in the total prediction-error across all relevant datasets for the models in both experiments. The sequential training on the 3-contingency dataset led to an increase in the prediction-error on the 1-contingency dataset, however both models saw an instantaneous decrease in the prediction-error on the 2- and 3-contingency datasets. The additional training on the 3-

contingency dataset gave the models insight into the load-shedding pattern with multiple induced contingencies. This made the models more generalizable, thus making it more robust to any potential perturbation in the system. As for the load-perturbation experiment, additional training saw a tiny increase in the overall prediction accuracy.

In the conducted case-study, it was found that the ConvGNN-model was significantly faster than the opposing OPF. Depending on the batch-size of the test-set fed to the model, the model was around 80-290 times faster. The model was also able to predict the load-shedding values with a superb accuracy, while being much faster than the OPF.

The case-study also found that the batch-size of the training-set had a significant impact on the model's run-time. The initial model used in the case-study was trained using a batch-size of 15 on the training-set, and the test-set created during the case-study used a batch-size of 5. As the batch-size of the model was reduced, the run-time of the model increased accordingly. The difference in the run-time for the model makes for an interesting question as to how one should approach the testing of the model with regards to the batch-size. A larger batch-size would allow for faster screening of more test-cases. However, if the model is to be used for real-time prediction where only singular cases are evaluated, the gained speed would be much smaller. This means that the optimal use of the model is not in real-time operational use, but rather to investigate multiple scenarios simultaneously in a Contingency Analysis.

8.1.2 Node-level

As with the models on the system-level, the node-level models were able to predict the load-shedding values across all experiments with a high accuracy. Similar as with the system-level, the prediction across varying load was the easiest task, while the contingency-perturbation prediction was the hardest. As with the System-level, additional system parameter perturbations led to an increase in the prediction-error for all three experiments on the node-level.

By evaluating the results at the node-level, it became easier to observe why the model's prediction error increased in par with the system-perturbation. As was stated in the system-level discussion, it was assumed that the model's increase in prediction error with further parameter-perturbations was caused by the load-shedding pattern of the system changing with further perturbations. This assumption was validated by the node-level results.

At node-level, the ML-models predicted that the load-shedding pattern it had been trained on would continue with further system-perturbation. As an example, as the number of contingencies for each simulation step increased, the model predicted that the system would continue to shed load in the buses which already had were shedding load, as this is what the ML-model was trained to do. However, the ML-model did not account for the saturation of the load-shed. The load-buses are only able to shed a finite amount of load. As the number of contingencies in the system increased, some buses maxed out the possible shedding of load, which lead to other load-buses having to shed load. This change in the load-shedding pattern was not something the model was able to pick up, which led to a higher prediction error.

9 Conclusion

This thesis has examined the possibility of using advanced ConvGNN-models to partially replace the slow and cumbersome numerical OPFs used in Contingency Analyses and Power System Reliability analyses. Within the Contingency Analyses, the OPFs have multiple areas of usage such as evaluating the impact that occurring contingencies have on the system. This is often done by estimating values such as the load-shedding and the generator rescheduling after occurring contingencies using a set of Power Flow equations. This allows for the estimation of the socio-economic cost of operation and to access the most optimal way of dealing with any occurring contingencies. As the PF equations are often non-linear, numerical approximations are used to solve them. The numerical approaches are often slow and tedious, and convergence is not guaranteed either. This is the motivating factor behind the introduction of ML and the ConvGNN models. However, as the Electrical Power System evolves, changes to system-parameters such as the topology or the system load is inevitable. The thesis therefore sought to explore the ML-models resilience towards small system-parameter perturbations, and if any sequential training on multiple dataset would enhance the models' predictive capabilities.

To this end, multiple ConvGNN-models were made utilizing the system's topology alongside features such as the power flow, load consumption, generated power, line status etc. to learn the patterns that were prevalent to estimate the load-shedding values after a set of contingencies occurred. The thesis sought to explore the model's predictive capabilities at both a node-level and at a system-level. Thereupon, multiple datasets were made with each dataset perturbating a system-parameter, such as the induced contingencies per simulation step or the system load. The results of this thesis showed that ConvGNN-models were able to predict the system load-shedding to a high degree across multiple system-perturbations, such as the number of induced contingencies per simulation point, load-perturbation, and ***small*** topology perturbations. For both prediction-levels, increasing perturbation of the system-parameter led to a decrease in the model's predictive accuracy, which was expected. Through a case-study, it was found that the ML-models were up to 290 times faster than the OPF at calculating the load-shedding values. It was also confirmed that the batch-size of the model has a noticeable impact on the run-time of the model, which was expected.

In the first experiments of the thesis, several ConvGNN-operators were tested and compared on the same dataset to determine the best performing operator that was to be used for the remaining experiments. In total, three ConvGNN-operators were tested, being the GCN, GAT and GraphConv operator. Out of these three operators, the GraphConv operator was the deemed the best operator for the problems in this thesis based on both its speed and predictive accuracy, and ultimately became the baseline operator for the other experiments conducted in the thesis. The GAT-operator showed great strength in its ability to generalize across the contingency-datasets, however the operator was far slower than that off the GraphConv, which rendered the operator useless.

The easiest conducted experiment for the ConvGNN-models were the prediction of load-shedding values across system-load perturbations, while the most challenging task was the prediction across multiple induced contingencies. In the discussion section of the thesis, it was concluded that a possible explanation for this varying performance was thought to be a change in the load-shedding pattern in the data as multiple contingencies were induced. The model assumed that the load-shedding trend it had been trained on would continue, which was not the case as revealed by the node-level predictions. Due to limited time, the thesis did not seek to explore sequential training on dataset with increased system-parameter perturbation on a node-level. This would be beneficial as it would explore if said training would give the model insight into the new load-shedding pattern with additional induced contingencies.

To conclude, this thesis sought to explore the usage of ML-models as a substitution for the numerical DC-OPFs used in Contingency Analyses which was used to estimate the shed system load due to deterministically induced contingencies. The thesis explored the models' resilience towards system-parameter perturbations. The constructed ConvGNN-models of the thesis were equally capable of predicting the load-shedding values across multiple system-parameter perturbations both at a system prediction-level and at a node prediction-level. The prediction-accuracy between system-parameters perturbations varied, with the system-load being an easier task, while the contingency-perturbations being more difficult.

9.1 Future Work

In this thesis, the ConvGNN-models showed prominent results at predicting the load-shedding values in the test-system. However, for the ConvGNN-models to fully replace the OPFs as a tool in the analytical processes, the models must also be able to predict additional values such as the rescheduled power of generators. Over the course of the thesis, several attempts were made to get the models to predict said values, though with no success. A plausible cause could be a fault in the way the values were stored in the MC-simulation. Other potential faults could be that the occurrences of cases where the system had to reschedule production were rare, and when it did occur, the values were too small.

Future work should initially include getting the model to predict the rescheduling generator values. This would make the model a step closer to being able to replace the OPF in the analytical tools. Future work should also be building upon the node-level, performing similar experiments as was done on the system-level. By studying the node-level, it can be easier to observe if sequential training on datasets with additional system-parameter perturbations will influence the model's prediction pattern, which was at fault for the high error in some cases. Other steps could be to test the model on vastly different topologies to explore how the model fare. The perturbation of topology in this thesis was small, and there is no guarantee that the models are usable across topologies which differ by a large amount.

Future work could also include an instance system development for planning, either long-term or short-term. This could be done by performing a case-study in which the system-load is incrementally increased over a period. The model could then be used to evaluate different investment alternative, by checking if the model is able to correctly depict which investment alternative would be cheapest if the system load is estimated to increase in the future, or if the probability of contingencies increases with the coming years.

To this end, the ConvGNN-models must be more complex than the one used in this thesis. The models used in this thesis were trained on non-sequential data. This meant that the models were not capable of simulating how the state of the system evolved over time, nor accounting for evolving contingencies etc. For the models to see use in a real-life scenario, additional work must be made to make the models more realistic and complex. This can be achieved by making the model work on data created using the AC PF-equations rather than the DC-equations. An interesting approach would be to test if a model trained with DC-data is

able to work on cases with AC-data. Other approaches could be to make the models work on sequential time-data. As the input of the models' become more complex, additional relevant features are necessary for the models. By including additional features to the models, the models could become more flexible, thus increasing the area of usage. Additional features could include relevant bus-values, line-values, external factors such as wind, air temperature around the lines etc.

10 References

[1] IPCC, “Climate Change 2014: Synthesis Report. Contribution of Working Groups, I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate change,” Geneva, 2014.

[2] IEA, 2020. Internation Energy Agency, “Outlook for energy demand. When and where will energy use recover?,” 2020.

[3] IEA, 2020. International Energy Agency, “Global EV Outlook 2020: Entering the decade of electric drive?,” June 2020.

[4] Paris Agreement Article 2, 2015. United Nations Framework Convention on Climate Change, “Adoption of the Paris Agreement - Paris Agreement 2015,” U.N. Doc., Dec. 12, 2015.

[5] L. Yang, K. Sun, R. Yao and B. Wang, “Power System Time Domain Simulation Using a Differential Transformation Method,” 2019.

[6] A. E. M. Operator., “Black System South Australia 2016 - Final Report, March 2017.,” 2017.

[7] P. Denholm, T. Mai, R. W. Kenyon, B. Kroposki and M. O'Malley, “Inertia and the Power Grid: A Guide Without the Spin,” NREL, National Renewable Energy Laboratory, Denver, 2020.

[8] GARPUR Consortium, “D1.2: Current practices, drivers and barriers for new reliability standards,” 2014.

[9] GARPUR Consortium, “D1.1: State of the art on reliability assessment in power systems,” 2014.

[10] K. Uhlen, G. H. Kjølle, G. G. Løvås and Ø. Breidablik, “A Probabilistic Security Criterion for Determination of Power Transfer Limits in a Deregulated Environment,” 2000.

[11] E. Karangelos and L. Wehenkel, “"Probabilistic Reliability Managment Approach and Criteria for Power System Real-Time Operation,” *2016 Power Systems Computation Conference (PSCC),* pp. 1-9, 2016.

[12] M. Nooij, B. Baarsma, G. Bloemhof, J. Slootweg and H. Dijk, “Development and application of a cost-benefit framework for energy reliability using probabilistic methods in network planning and regulation to enhance social welfare: the N-1 rule.,” *Energy Economics,* vol. 32, pp. 1277-1282, 2010.

[13] S. J. Hofsmo, E. H. Solvang and I. B. Sperstad, “State of the art on modelling operational strategies including preventive, corrective and restorative actions,” 2020.

[14] P. Murty, Power System Analysis, Butterworth-Heinemann, 2017.

[15] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” 2017.

[16] W. L. Hamilton, R. Ying and J. Leskovec, “Inductive Representation Learning on Large Graphs,” 2018.

[17] M. Zitnik, M. Agrawal and J. Leskovec, “Modeling Polypharmacy side effects with graph convolutional networks,” *Bionformatics,* vol. 34, pp. i457-i466, 2018.

[18] B. Donnot, I. Guyon, M. Schoenauer, A. Marot and P. Panciatici, “Fast Power system security analysis with Guided Dropout,” vol. 1, no. 1, pp. 1-6, 2018.

[19] B. Donon, I. Guyon, B. Donnot and A. Marot, “Graph Neural Solver for Power Systems,” *IJCNN 2019 - International Joint Conference on Neural Networks,* 2019.

[20] B. Donon, R. Clement, B. Donnot, A. Marot, I. Guyon and M. Shoenauer, “Neural networks for power flow: Graph neural solver,” *Electric Power Systems Research,* vol. 189, 2020.

[21] L. Duchesne, “Machine Learning of Proxies for Power Systems Reliability Managment in Operation Planning,” *ULiège - Université de Liège,* 2021.

[22] E. Heylen, M. Ovaere, S. Proost, G. Deconinck and D. Van Hertem, “A multi-dimensional analysis of a reliability criteria: From deterministic N-1 to a probabilistic approach,” *Electric Power System Research,* vol. 167, pp. 290-300, 2019.

[23] O. Gjerde, S. J. Hofsmo and E. F. Bødal, “Probablistic Operational Planning using Dynamic Programming with Time-Domain Simulations,” 2021.

[24] S. J. Hofsmo, “Definitions and classification of power system reliability control actions,” 2021.

[25] M. Rausand, “Risk assessment: Theory, methods, and applications,” vol. 115, 2013.

[26] N. Balu, T. Bertram, A. Bose, V. Brandwajn, G. Cauley, D. Curtice, A. Fouad, L. Fink, M. G. Lauby, B. Wollenberg and J. N. Wrubel, “On-line power system security analysis.,” *Proceedings of the IEEE,* pp. 262-282, 1992.

[27] A. O. Ekwue, “A review of automatic contingency selection algorithms for online security analysis.,” *Power System Monitoring and Control, Third International Conference,* 1991.

[28] A. Khaled, N. Badra and A. Y. Abdelaziz, “Optimal Power Flow Methods: A Comprehensive Study,” *2016.*

[29] G. H. Kjølle and O. Gjerde, “The OPAL methodology for reliability analysis of power systems,” 2012.

[30] R. Billinton, “Composite system adequacy assessment-the contingency enumeration approach.,” *IEEE Tutorial Course Reliability assessment of composite generation and transmission systems, course text 90EH0311-1-PWR,* 1989.

[31] L. Wenyuan and P. Choudhury, “Probabilistic Transmission Planing,” *Power and Energy magazine, IEEE,* vol. 5, pp. 46-53, 2007.

[32] R. Bacher, “Liberalized Electric Power Systems and SmartGrids,” 2017.

[33] A. K. Khamees, N. M. Badra and A. Y. Abdelaziz, “Optimal Power Flow Methods: A Comprehensive Survey,” 2016.

[34] L. Mones, “A Gentle Introduction to Optimal Power Flow,” Invenia Blog, 18 June 2021. [Online]. Available: https://invenia.github.io/blog/2021/06/18/opf-intro/. [Accessed 18 May 2022].

[35] Ø. S. Laengen, “Application of Monte Carlo Simulation to Power System Adequacy Assessment,” 2018.

[36] A. Mohanty, “Becoming Human AI,” 15 May 2019. [Online]. Available: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f. [Accessed 9 February 2022].

[37] E. Alpaydin, Introduction to Machine Learning, Fourth Edition, Massachusetts: MIT Press, 2020.

[38] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA, 2016.

[39] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Communications of the ACM,* vol. 60, no. 6, pp. 84-90, 2017.

[40] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn and D. Yu, “Convolutional Neural Networks for Speech Recognition,” *IEEE/ACM Transactions on Audio, Speech and Language Processing,* vol. 22, no. 10, 2014.

[41] R. Girshick, J. Donahue, T. Darrell and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 580-7, 2014.

[42] P. Velickovic, A. Casanova, P. Lio, Cucurull Guillem, A. Romero and Y. Bengio, “Graph Attention Networks,” in *ICLR*, 2018.

[43] H. Gholamalinezhad and H. Khosravi, “Pooling Methods in Deep Neural Networks, a Review,” 2020.

[44] S. Saha, “Towards Data Science,” Medium, 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 22 February 2022].

[45] R. J. Trudeau, Introduction to Graph Thery, Kent, Ohio: The Kent State University Press, 1976.

[46] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems,* vol. 32, no. 1, pp. 4-24, 2019.

[47] S. Zhang, H. Tong, J. Xu and R. Maicejewski, “Graph convolutional networks: a comprehensive review,” 2019.

[48] J. Bruna, A. Szlam, W. Zaremba and Y. LeCun, “Spectral Networks and Deep Locally Connected Networks on Graphs,” 2014.

[49] M. Defferrard, X. Bresson and P. Vandergheynst, “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering,” *Proceedings of the 30th International Conference on Neural Information Processing Systems,* pp. 3844-3852, 2016.

[50] R. Levie, W. Huang, L. Bucci, M. Bronstein and G. Kutyniok, “Transferability of Spectral Graph Convolutional Neural Networks,” *Journal of Machine Learning Research 22,* vol. 22, pp. 1-59, 2021.

[51] J. Berg Hansen, “Power Flow Optimization with Graph Neural Networks,” 2021.

[52] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gomez-Bombarelli, T. Hirzel, A. Aspuru-Guzik and R. P. Adams, “Convolutional Networks on Graphs for Learning Molecular Fingerprints,” *Advances in neural information processing systems,* vol. 28, 2015.

[53] T. Danel, P. Spurek, J. Tabor, M. Smieja, L. Struski, A. Slowik and L. Maziarka, “Spatial Graph Convolutional Networks,” 2020.

[54] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan and M. Grohe, “Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks,” 2018.

[55] IEEE Reliability Test System Task Force of the Applications of Probability Methods Subcommittee, “IEEE reliability test system,” *IEEE Transactions on Power Apparatus and Systems,* vol. 98, no. 6, pp. 2047-2054, 1979.

[56] IEEE-RTS Task Force of APM Subcommittee, “IEEE Reliability test system,” pp. 2047-2054, IEEE PAS 98, 1979.

[57] L. Thurner, A. Scheidler, F. Schafer, J. Menke, J. Dollichon, F. Meier, S. Meinecke and M. Braun, “pandapower - An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems,” *IEEE Transactions on Power Systems,* vol. 33, no. 6, pp. 6510-6521, 2018.

[58] M. Fey and J. E. Lenssen, “Fast Graph Representation Learning with PyTorch Geometric,” 2019.

[59] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization,” *International Conference on Learning Representation,* 2014.

[60] MatPower, “Case24_ieee_rts,” Matpower, 26 Janaury 2015. [Online]. Available: https://matpower.org/docs/ref/matpower5.0/case24_ieee_rts.html. [Accessed 29 April 2022].

[61] R. D. Zimmerman and C. E. Murillo-Sanchez, “Matpower User's Manual, Version 7.1,” 2020.

Appendix A

Table 15: Bus Data *[61]*

Name	*Description*
BUS_i	Bus number
BUS type	Bus type (1= PQ, 2=PV, 3=ref)
PD	Real Power Demand
QD	Reactive Power Demand
GS	Shunt Conductance (MW demanded at V = 1.0 p.u.)
BS	Shunt Susceptance (MW demanded at V = 1.0 p.u.)
Bus_area	Area Number
Zone	Loss Zone (positive Integer)
VM	Voltage Magnitude (p.u.)
VA	Voltage Angle (p.u.)
VMAX	Maximum Voltage Magnitude (p.u)
Vmin	Minimum Voltage Magnitude

Table 16: Bus information for the IEEE-24 bus test system.

Bus	Type	Pd	Qd	Gs	Bs	Area	Vm	Va	BaseKV	Zone	Vmax	Vmin
1	2	108	22	0	0	1	1	0	138	1	1.05	0.95
2	2	97	20	0	0	1	1	0	138	1	1.05	0.95
3	1	180	37	0	0	1	1	0	138	1	1.05	0.95
4	1	74	15	0	0	1	1	0	138	1	1.05	0.95
5	1	71	14	0	0	1	1	0	138	1	1.05	0.95
6	1	136	28	0	0	1	1	0	138	1	1.05	0.95
7	2	125	25	0	0	2	1	0	138	1	1.05	0.95
8	1	171	35	0	0	2	1	0	138	1	1.05	0.95
9	1	175	36	0	0	2	1	0	138	1	1.05	0.95
10	1	195	40	0	0	1	1	0	138	1	1.05	0.95
11	1	0	0	0	0	2	1	0	230	1	1.05	0.95
12	1	0	0	0	0	3	1	0	230	1	1.05	0.95
13	3	265	54	0	0	3	1	0	230	1	1.05	0.95
14	2	194	39	0	0	3	1	0	230	1	1.05	0.95
15	2	317	64	0	0	3	1	0	230	1	1.05	0.95
16	2	100	20	0	0	4	1	0	230	1	1.05	0.95
17	1	0	0	0	0	4	1	0	230	1	1.05	0.95
18	2	333	68	0	0	4	1	0	230	1	1.05	0.95
19	1	181	37	0	0	4	1	0	230	1	1.05	0.95
20	1	128	26	0	0	3	1	0	230	1	1.05	0.95
21	2	0	0	0	0	3	1	0	230	1	1.05	0.95
22	2	0	0	0	0	4	1	0	230	1	1.05	0.95
23	2	0	0	0	0	4	1	0	230	1	1.05	0.95
24	1	0	0	0	0	3	1	0	230	1	1.05	0.95

Table 17: Generator Data Description

Name	*Description*
Gen_bus	Bus number
Pg	Real Power Output (MW)
Qg	Reactive Power Output (MVar)
QMax	Maximum reactive power output (MVar)
qmin	Minimum reactive power output (MVar)
vg	Voltage Magnitude Setpoint (p.u.)
Mbase	Total MVA base of machine, default to baseMVA
status	Machine status, > 0 in service, <= 0 out of service
Pmax	Maximum Active Power Output (MW)
Pmin	Minimum Active Power Output (MW)

Table 18: Generator Information for the IEEE-24 Bus Test System

Bus	Pg	Qg	QMax	QMin	Vg	mBase	Status	Pmax	Pmin
1	10	0	10	0	1.035	100	1	20	0
1	10	0	10	0	1.035	100	1	20	0
1	76	0	30	-25	1.035	100	1	76	0
1	76	0	30	0	1.035	100	1	76	0
2	10	0	10	0	1.035	100	1	20	0
2	10	0	10	-25	1.035	100	1	20	0
2	76	0	30	-25	1.035	100	1	76	0
2	76	0	30	-100	1.035	100	1	76	0
6	0	0	0	0	1.025	100	1	1	0
7	80	0	60	0	1.025	100	1	100	0
7	80	0	60	0	1.025	100	1	100	0
7	80	0	60	0	1.025	100	1	100	0
13	95.1	0	80	0	1.02	100	1	197	0
13	95.1	35.3	80	0	1.02	100	1	197	0
13	95.1	0	80	0	1.02	100	1	197	0
14	0	0	200	-50	0.98	100	1	0	0
15	12	0	6	0	1.014	100	1	12	0
15	12	0	6	0	1.014	100	1	12	0
15	12	0	6	0	1.014	100	1	12	0
15	12	0	6	0	1.014	100	1	12	0
15	12	0	6	0	1.014	100	1	12	0
15	155	0	80	-50	1.014	100	1	155	0
16	155	0	80	-50	1.017	100	1	155	0
18	400	0	200	-50	1.05	100	1	400	0
21	400	0	200	-50	1.05	100	1	400	0
22	50	0	16	-10	1.05	100	1	50	0
22	50	0	16	-10	1.05	100	1	50	0
22	50	0	16	-10	1.05	100	1	50	0
22	50	0	16	-10	1.05	100	1	50	0
22	50	0	16	-10	1.05	100	1	50	0
22	50	0	16	-10	1.05	100	1	50	0
23	155	0	80	-50	1.05	100	1	155	0
23	155	0	80	-50	1.05	100	1	155	0

23	350	0	150	-25	1.05	100	1	350	0

Appendix B

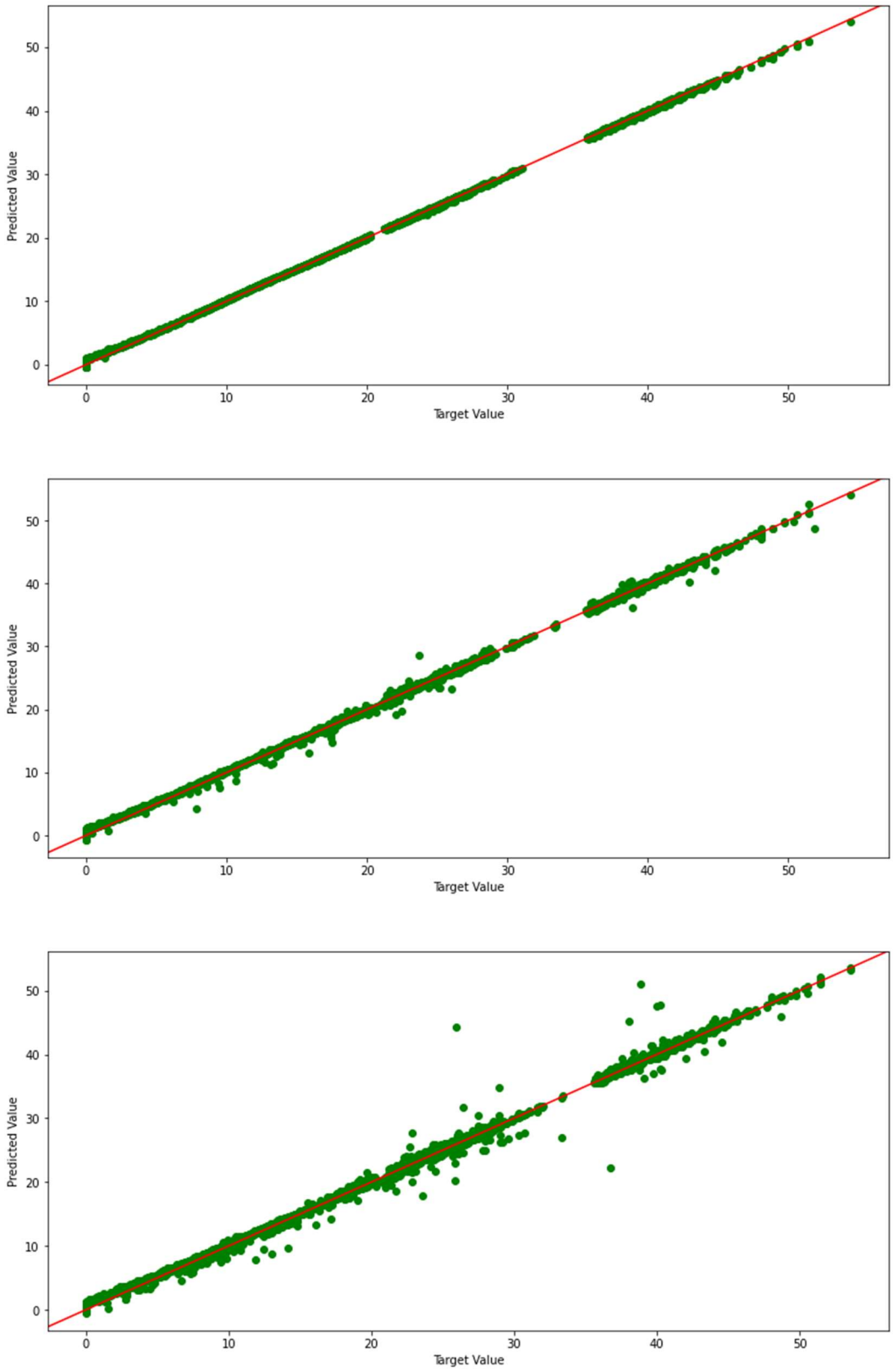


Figure 27: The following figures shows the predicted value vs the target value for the GCN model tested on the test-set of the 1-line, 2-line, and 3-line contingency datasets, respectively.

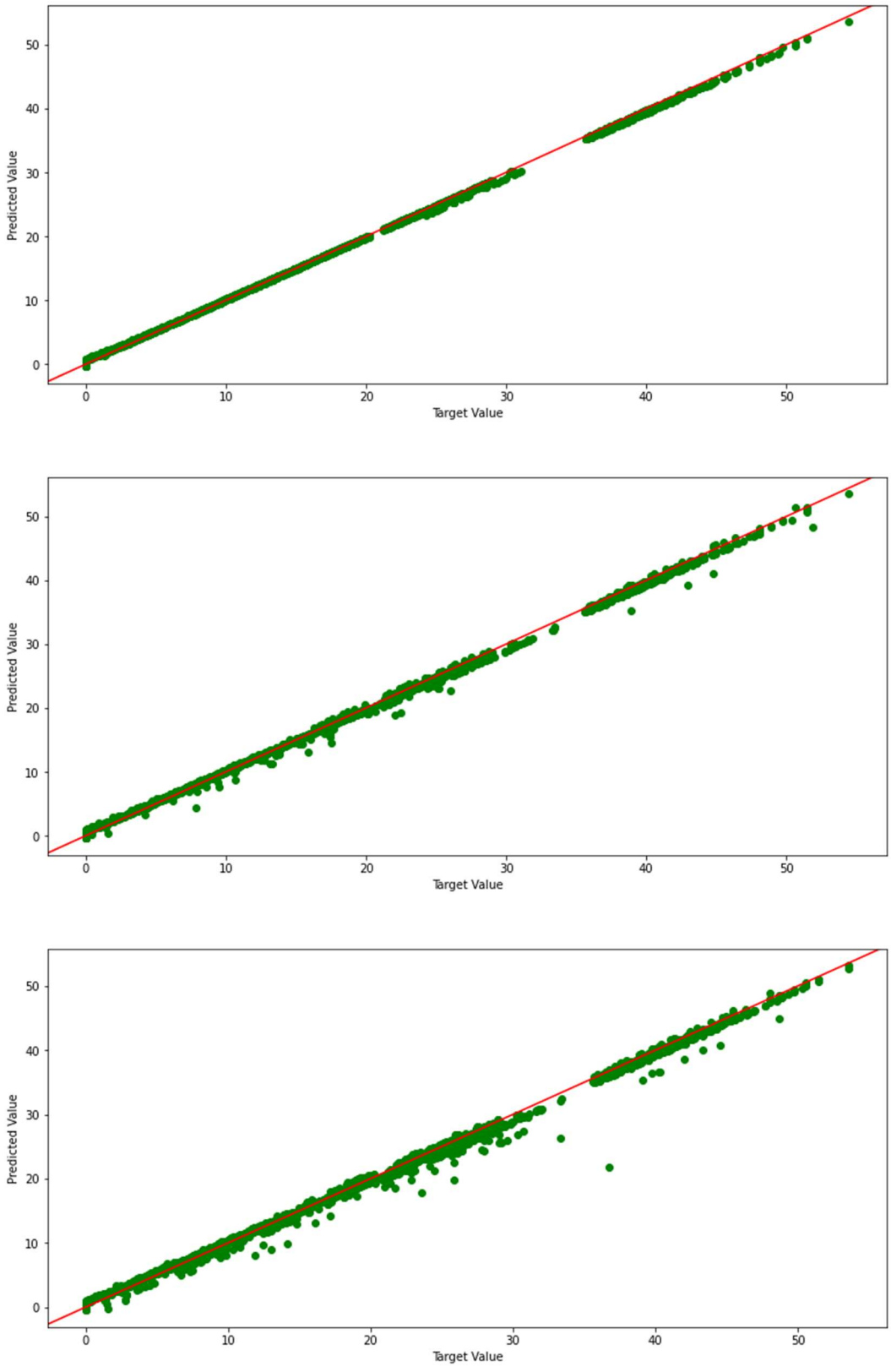


Figure 28: The following figures shows the predicted value vs the target value for the GAT model tested on the test-set of the 1-line, 2-line, and 3-line contingency datasets, respectively.