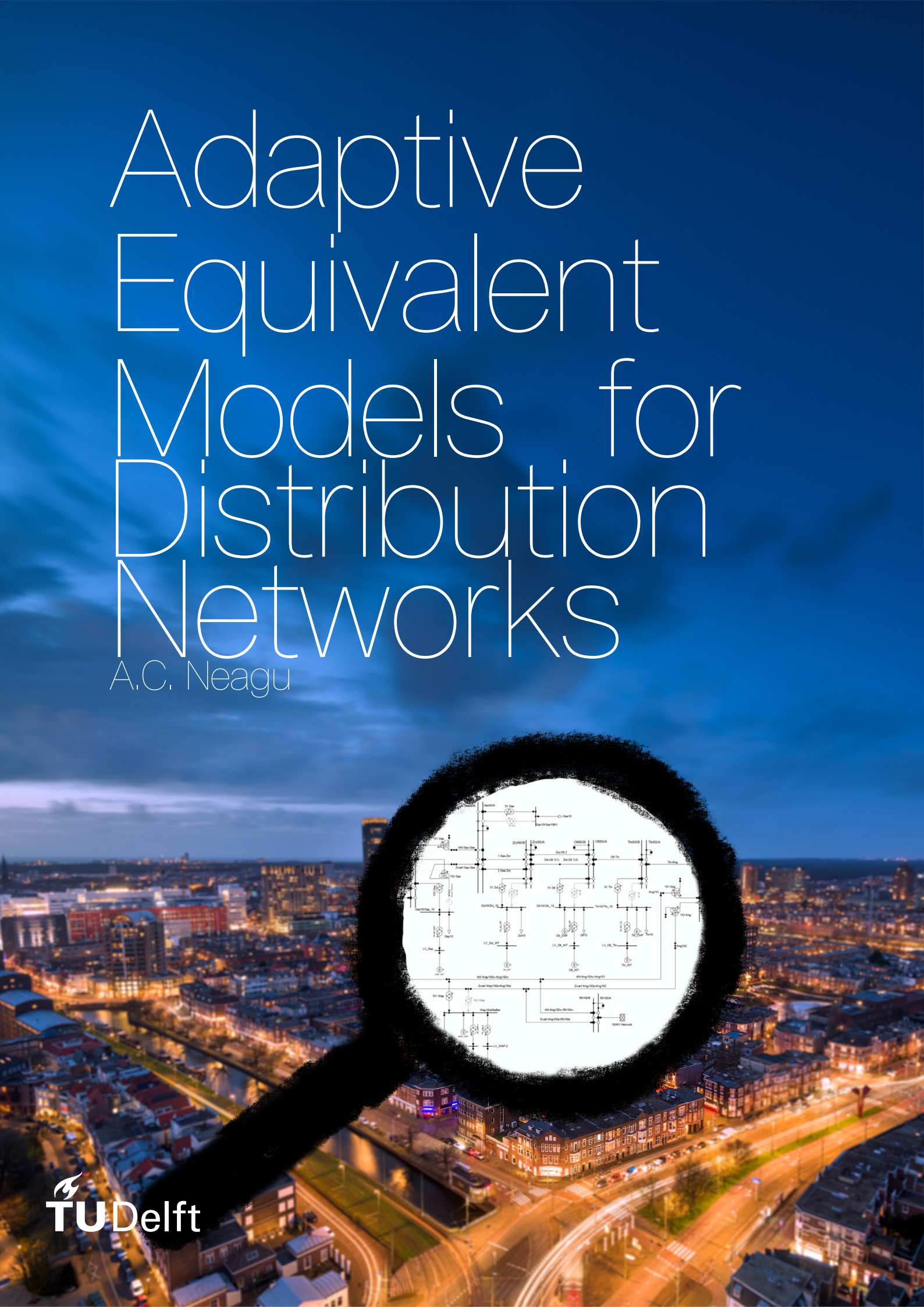
Adaptive Equivalent Models for Distribution Networks

A.C. Neagu



TU Delft

Adaptive Equivalent Models for Distribution Networks

by

A.C. Neagu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday 28 March 2024 at 10:00.

Student number: 4703723
Project duration: 10 January 2023 – 28 March 2024
Thesis committee:
Dr. Ir. José. L. Rueda Torres, TU Delft, Thesis Advisor and Committee Chair
Dr. Jochen L. Cremer TU Delft, Daily Supervisor
Demetris Chrysostomou, MSc TU Delft, Daily Co-Supervisor
Dr. Ir. Gautham Ram Chandra Mouli, TU Delft

TU Delft

Preface

This work represents over a year of dedicated effort, navigating multiple challenges covering scientific exploration, health and personal growth. At times, I questioned my ability and decisions, yet I persevered. The journey was not without distractions, but they became sources of invigoration and inspiration. What I liked the most about the thesis project was the opportunity to apply my academic knowledge to a topic of personal interest and explore it in a direction I shaped. I also learned new skills along the way. I became better at conducting scientific research and asking critical questions. I learned to be patient and push through if something does not work out. I became more proficient in coding, without which this project would not have succeeded. I developed my planning skills to combine my thesis with handball, my work for PowerWeb and my passion for photography and creative work. Although this thesis concludes my studies, it marks the start of a lifelong learning journey, with these acquired skills supporting me along the way.

I owe immense gratitude to Jochen Cremer for his invaluable mentorship, critical insights, and support in overcoming unexpected hurdles throughout this project. My thanks also go to Jose Rueda Torres for his role as my chair and for ensuring the scientific quality of my work from start to end. I am also thankful to Gautham Ram Chandra Mouli for his willingness to join my committee, especially on such short notice.

I sincerely thank Demetris Chrysostomou for his direct involvement with this project. Our weekly discussions were invaluable in shaping and refining the work when needed, alongside his motivational support. Having such steadfast support meant a lot to me during this journey.

I thank my fellow PhD and MSc students for our cherished memories. Your companionship on this journey has made it more enjoyable. I want to thank Mert Karaçelebi for his insightful questions during the project's conception, Olayiwola Arowolo for his assistance in understanding the GNN, and Ali Rajaei for his fast and helpful responses to my questions.

Last, I want to express my heartfelt thanks to my friends, family, and parents for their unwavering support. Your encouragement has been fundamental to my success.

A.C. Neagu
Voorburg, March 2024

Abstract

As the power system grows more complex and active, equivalent models have become a solution for modelling parts of the network that have limited observability or are confidential or too complex to simulate otherwise. In the past decade, this topic has also made its way to distribution networks because of its transition towards an active network, which was not the case before. The current grey- and black-box techniques for equivalent modelling create models that are fitted to the average dynamic response of the system or have limited applicability. Also, the existing methods lack extensive verification under different system conditions, and these works rarely focus on active distribution networks (ADNs) with multiple points of common coupling (PCCs). This thesis aims to develop an equivalent model that can estimate the dynamic response of an active distribution network with multiple PCCs and under diverse operating conditions and topological changes.

The proposed approach is based on a graph-time convolutional neural network (GTCNN) that relates available PMU measurements inside the distribution network on a graph structure $\mathcal{G}_{GTCNN}$. The graph $\mathcal{G}_{GTCNN}$ is obtained by taking a modified line graph of the graph representation of the power system and is expanded using the Cartesian product graph rule to include the temporal dependencies of nodes on their past values. The inputs of the equivalent model are the voltage magnitude $|V|$ and angle θ at the PCC and the initial power injections P_0 and Q_0 at non-PCC nodes, while the model outputs the active $\hat{P}$ and reactive $\hat{Q}$ power at the PCC nodes. The GTCNN explicitly considers the initial power injections P_0 and Q_0 at non-PCC nodes to help the model learn how different operating conditions and topological changes impact the dynamic response. The DSO trains the equivalent model using simulation data or collected PMU measurements. The model is exchanged with the TSO every month, who can use the equivalent in co-simulation with their transmission network model to perform transient stability studies.

The GTCNN-based equivalent model showed promising performance as an equivalent model for transient stability. The GTCNN was benchmarked against two state-of-the-art Long Short-Term Memory (LSTM)-based equivalent models and a hybrid GTCNN-LSTM model. The evaluation was performed on a real Dutch distribution network using three datasets, each focussing on a different system condition: different fault events, different operating conditions and different *hidden* topological changes. The GTCNN-based equivalent model had a mean-squared error (MSE) below 0.02 for each dataset, which means it can accurately reproduce the dynamics. This accuracy is comparable to the LSTM-based equivalent models, but the GTCNN could train 4x faster. The GTCNN also showed good generalisation performance, as its accuracy did not decrease on the validation and test sets. A study on scaling performance suggested that the MSE of the GTCNN-based equivalent model increases slower than that of the LSTM-based models while its training time increases faster. Therefore, the GTCNN-based equivalent model trains faster for smaller ADNs but will be more accurate with more measurement nodes. However, the proposed GTCNN has difficulty learning the response at different close-by PCC terminals if the dynamics are different.

The developed GTCNN-based equivalent model can predict the dynamic response accurately under changing topologies and operating conditions at a similar performance level to existing LSTM-based approaches. However, its training time is much faster, which can result in a more accurate equivalent model by a more frequent model exchange between the DSO and TSO or a more extensive dataset being used to train the model. In future research, the GTCNN performance will be evaluated on a more comprehensive dataset containing all three system conditions to establish how much data is needed to train the equivalent model accurately. Also, the system frequency will be considered as an additional input. Moreover, its scaling performance will be evaluated more extensively and with a more efficient coding implementation. Furthermore, a heterogenous graph convolutional operator will be implemented to learn the connection per relational type (source node - edge type - target node). Finally, the co-simulation interface between the equivalent model and popular simulation tools will be explored.

Contents

Nomenclature

List of abbreviations

Abbreviations	Definition
ADAM	Adaptive Moment estimation
AI	Artificial Intelligence
AND	Active Distribution Network
ANN	Artificial Neural Network
CNN	Convolutional Neural Networks
DFL	Direct Feedback Linearisation
DG	Distributed Generation
DGT	Differential Geometric Technique (type of linearisation for nonlinear models)
DN	Distribution Network
DSO	Distribution System Operator
ERL	Enhanced Reinforcement Learning
ERM	Exponential Recovery Model
EV	Electric Vehicles
EXP	Exponential model
FCNN	Fully Connected Neural Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GRNN	Graph Recurrent Neural Network
GRU	Gated Recurrent Units (type of RNN)
GTCNN	Graph-Time Convolutional Neural Networks
HV	High Voltage
IBG	Invertor-Based Generators
LSTM	Long Short-Term Memory network
MG	Microgrid
ML	Machine Learning
MSE	Mean Squared Error
PCC	Point of Common Coupling
PMU	Phasor Measurement Units
PV	Photovoltaics
RES	Renewable Energy Sources
RNN	Recurrent Neural Network
SG	Synchronous Generator
SGD	Stochastic Gradient Descent
SLT	Standard form of Linearisation Technique using Taylor series approximation
STGNN	Spatiotemporal Graph Neural Networks
TCNN	Temporal Convolutional Neural Networks (CNNs used to represent time such as 1D-CNN)
TN	Transmission Network
TSO	Transmission System Operator
VPP	Virtual Power Plant
WTG	Wind Turbune Generator
ZIP	Constant impedance, current and power load model

List of Figures

List of Tables

1 Introduction

The power grid is humankind's most remarkable machine. In 2021, the transmission system had an estimated 4.7 million km, while the distribution grid trivialises that with a total length between 88 and 104 million km [1]. If we were to untangle the distribution network, we could create one long electricity cable that could power Mars when it is closest to Earth's orbit and still have 30 million km of wire left.

It is truly remarkable how ubiquitous and vital electricity has become, especially considering how novel the technology is. In 1752, Benjamin Franklin was experimenting with lightning [2]. It took 79 more years for Michael Faraday to discover the basic principles of electricity generation in 1831 and 49 more years for Thomas Edison to patent the incandescent lightbulb. Around that time, the first electricity grid was born in New York City, powering lights in houses of the privileged few. Samuel Insull, who started as Edison's assistant, transformed electricity into an economically feasible undertaking and built the foundation of the modern power grid. *"Samuel Insull did for electricity what Henry Ford did for the automobile—he turned a luxury product into an affordable part of everyday life for millions of Americans."* [3] One hundred forty years later, electricity has become (almost) a basic human necessity. In 2020, 21 % of the generated energy was electricity [4], which is only rising with the electrification of transportation and heating towards our goal of becoming carbon neutral by 2050.

The first power grid was relatively straightforward to manage as there were few generating units whose purpose was to light up houses. Nowadays, many distributed generators with different properties power the grid. The interaction between other units and the physical laws that govern them give rise to more complex dynamics in the power system. For example, all units are coupled through the electrical infrastructure. Traditional machines have a rotating generator which produces fluctuating power; the rate is known as the system frequency. A change in the output of one generator creates a deviation in the system frequency, in turn affecting other units. The frequency cannot deviate too much from its nominal value to prevent equipment damage, e.g. by speeding up the rotating core past safe values. Other power quality issues, such as voltage surges, power outages and noise, also need to be prevented, some of which are related to frequency instability [5]. Therefore, it is crucial to investigate system dynamics thoroughly to keep humankind's most remarkable machine alive.

1.1. Problem definition

Investigating power system dynamics is challenging because the equations that describe the components are nonlinear differential equations. For example, a ninth-order model is required to fully describe the behaviour of synchronous generators (SGs) [6]. Using these models when analysing big systems is impractical because of the high computational burden. Therefore, a reduced-order model that ignores some dynamics of the generator is used. Even then, the simulations can be computationally expensive as at least a fourth-order model might be needed to reproduce the dynamics of interest accurately [7].

Another challenge for dynamic studies is that the network is expanding. In Europe, transmission system operators (TSOs) want to reach the EU Green Deal using *"a pan-European approach to electricity planning"* [8]. One of the consequences has been the coupling of the electricity markets in North-West Europe [9]. This impacts system dynamics as there is a cross-border flow of power, in turn requiring the use of a bigger and, thus, more computationally expensive network model.

The transmission network (TN) delivers power to distribution networks (DNs). Since the DNs are directly exchanging power with the TN, they must be included in the TSO's model. This is usually done through equivalent models where the response of the DN is aggregated. This aggregation is necessary as DNs are bigger than TNs [1], [10]. DNs are bigger since they consist of tens to thousands of nodes, depending on the model's voltage levels included in the model [11], [12]. Moreover, a TN network connects multiple DNs, so the model size would quickly explode in terms of dimensionality and computational complexity if the detailed DN models were included in the TSO model. By using equivalent models, the TSOs can have a lightweight representation of the DN that they can use for power system analysis.

Nowadays, the distribution network has become active by the addition of distributed generation (DG) [13]. Solar panels, for example, are interfaced with the DN through an inverter, adding dynamic behaviour to the network. By aggregating these DG units with storage and controllable loads, virtual power plants (VPPs) [14] or microgrids (MGs) can be formed [15]. VPPs and MGs are active clusters in the power systems and can provide services like frequency support to the TSO [14]. At the same time, the power system is encountering a digital transformation which enables new interactions between assets. For example, the Dutch TSO TenneT experiments directly accessing the network's flexible storage capacity, e.g., electric vehicles (EVs), through the blockchain-based Equigy platform [16]. If successful, this could result in direct interactions between the TSO and the DN. To model these types of dynamic behaviour, new equivalent models are needed.

1.2. Literature review

The issue of dynamic equivalent modelling is a relatively recent topic in literature. An overview of techniques for dynamic equivalents can be found in [17], where the authors distinguish between *conventional system reduction based* and *measurement based* approaches. The former is so-called white box modelling, where a full network model is available, and system reduction techniques can be applied to derive an equivalent model. In the latter, both grey and black box approaches are found, depending on the available knowledge on the configuration of the DN (grey box) or not (black box).
The field is evolving rapidly, especially with the expansion and maturation and expansion of data-driven techniques like machine learning. Moreover, approaches from different fields have potential for equivalent modelling, such as learning the probability distribution to predict the most likely dynamic response [18], [19] or the usage of holomorphic embedding to predict voltage stability [20].
To make a better distinction between the measurement-based approaches, this thesis subdivides the methods between white, grey and black box modelling:

- **White box modelling:** This type of modelling assumes that an accurate system model is available, including the control systems.
- **Grey box modelling:** There is some prior knowledge about the model in this approach, but it is far more limited than white box modelling. For example, the type of components in a DN are known, but their exact configuration is not. The prior knowledge can be exploited to create simple and accurate models even when the entire dynamic model is unavailable.
- **Black box modelling:** This method uses no prior knowledge about the network to construct an equivalent model since it relies on (collected) measurement data to reproduce the dynamic response. Machine learning and other curve-fitting methods fall under this category.

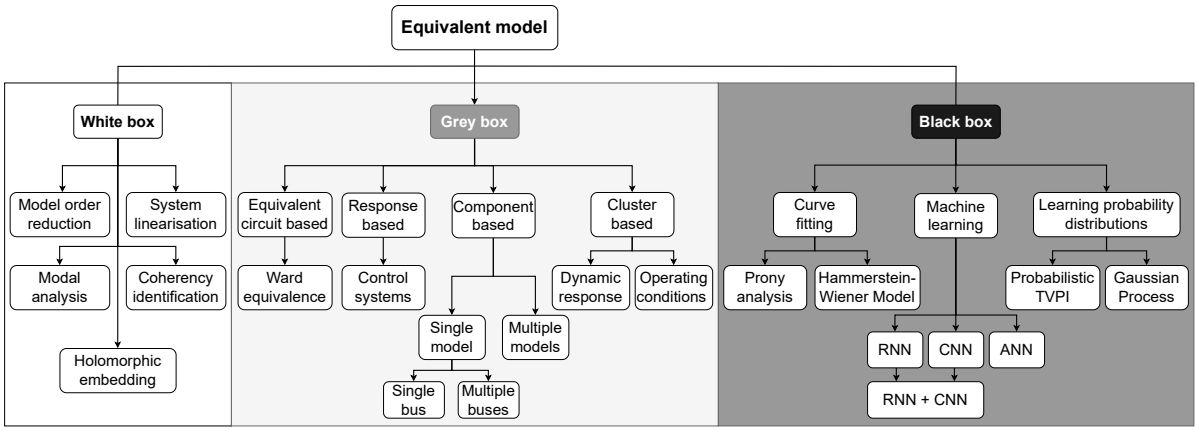


Figure 1.1: Overview of common approaches for dynamic equivalent modelling

Using three groups to classify equivalent models allows for a clear separation between methods solely based on measurement data (i.e. black box) and methods that use prior network knowledge (grey box). The approaches and their submethods are also illustrated in Figure 1.1. In the remainder of this section, each approach is elaborated more and compared to the others. Moreover, scientific gaps are identified.

1.2.1. White box modelling

White box modelling is considered the conventional approach to equivalent modelling [17]. The starting point of these methods is the full dynamic network model, full of non-linear differential equations. The model includes the full-order model of the synchronous generator and its control systems, such as the AVR, PSS and GOV, which are computationally expensive to run.

Initial approaches to achieve model reduction are model order reduction and system linearisation. It was demonstrated in [7] that the full generator model can be reduced to a fourth-order model while still preserving the necessary dynamic response. The second option is to perform system linearisation, among which SLT (standard form of linearisation technique using Taylor series approximation), DFL (direct feedback linearisation) and DGT (linearisation of nonlinear model-based on differential geometric technique) are the popular techniques [21]. These methods are old but effective and are applied in other fields, such as robotics and control engineering (especially linearisation). One downside is that they achieve a limited model reduction, which is insufficient for big networks. Another drawback is that the system linearisation is highly dependent on the operating condition around which it is performed and is usually only accurate for small disturbances [17].

There are other reduction-based approaches based on modal analysis. In modal analysis, the system state-space representation is linearised first. After that, eigenvalue decomposition is performed, which yields information about the oscillatory modes in the system or, in other words, the dynamics that can occur in the network. With this information, it is possible to eliminate modes that hardly contribute to dynamics while losing very little accuracy [22].[23] extends on the idea by proposing a method that can be applied when part of the system is unavailable. Other modal analysis approaches are elaborated in [17], each retaining different system properties in the reduced model. Modal analysis has several drawbacks [17]. For example, they can be difficult to perform or integrate in simulation software. Also, like linearisation, they depend highly on the operating condition around which the system is linearised. Finally, choosing relevant modes is difficult the more extensive the network becomes.

Coherency identification is an alternative approach where generators are grouped based on their dynamic response to an event. This is usually done based on the rotor angle swings of generators [24]. Two generators whose rotor angles remain close after an event are considered coherent and can be represented by a single generator with different parameters. The parameters are found by matching the generator response to either the time or frequency domain response. The electrical proximity is also considered in [25] to overcome convergence issues. There are a couple of disadvantages of coherency analysis [17]. First, the found parameters of the equivalents is highly dependent on the class of events, meaning that updating the equivalent is difficult. Another critical challenge is that coherency-

based methods rely on synchronicity between generators. While this is the case in conventional TNs consisting of synchronous machines, the situation is different with the advent of invertor-based generators (IBGs) and distributed generation in the DN. This makes the method more difficult to apply, albeit grouping is possible on other aspects.

Holomorphic embedding is used in [20] to assess the voltage stability. The system is decoupled into a set of decoupled two-bus circuits consisting of a separate load or generator connected through a transmission line with the swing bus. The voltage stability is evaluated by plotting the trajectories of the holomorphic function for different operating conditions. While this approach provides an interesting alternative for power system analysis, it is too simple to capture complex system dynamics as this approach has only been proven for static power flows [20].

White box modelling is a straightforward way to achieve reduced equivalent models, and its working is highly explainable. However, approaches of this category require the full system model, which might not be possible because of confidentiality issues or simply because such a dynamic model is not available in the traditionally passive distribution network. Hence, this class of models is not attractive for modelling active distribution networks.

1.2.2. Grey box modelling

Grey box modelling is a measurement-based method for equivalent modelling that incorporates prior knowledge about the network. Even when no model is available for the DN, some knowledge about the system is still available. For example, the operator might know what types of loads are present. Moreover, there will be some knowledge about the share of distributed generation in the system. This knowledge can be used to derive equivalent models, which is similar to system identification theory (the building of mathematical models from observed data). In this thesis, a distinction is made between equivalent circuit-based, response-based, component-based and cluster-based models.

Equivalent circuit-based methods create an equivalent model by using an equivalent circuit representation. Thevenin and Norton equivalents are well-known examples, where at a given observation point, the remainder of the circuit is represented by a source and an impedance. Ward equivalence is used in [26] to create an equivalent model of a multi-terminal power system. In Ward equivalence, the internal circuit is simplified as fictitious lines between the different points of common coupling (PCCs) [26], [27]. Moreover, a fictitious load is added at each PCC. In a way, ward equivalence can be considered as a multi-terminal generalisation of Thevenin's theorem. In the original method, the load is either fixed current or fixed power, while in [26], a ZIP load is used (constant impedance, current and power). This approach is interesting because it considers multiple terminals, something that also occurs with DNs in more urban areas. A drawback is that the method requires different parameters for various operating conditions to produce an accurate response.

Response-based equivalent models construct a model whose response is the same as the full network. Various control systems and schemes are in place to ensure continuous power system operation. These are the primary, secondary and tertiary control schemes. Secondary and tertiary controls are system-level controls whose response can be represented by an equivalent model. System identification theory was used to represent the local secondary control of microgrids in [28]. The fast dynamics part provided by grid-forming converters is kept, while the slower dynamics part representing grid-following converters are replaced with an equivalent model. This approach to equivalent modelling is valid when the lower-level control systems are well-designed and can still ensure this system-level behaviour. Therefore, it is also clear that during contingencies and in poor designed systems, these type of equivalents could lose their accuracy.

Component-based equivalent models are based on knowing which types of components are part of the DN. Households, for example, contain both static loads (e.g. laptops, LED) and inductive loads (e.g. washing machines). The DN could contain different types of inductive loads, as seen in [29]. An equivalent network can be obtained by aggregating components of the same type into one analogous component. An optimisation is performed to find the parameters of the equivalent components that yield a similar output to the full model at the PCC. There exists multiple models that can be used to represent the components. Different types of models are compared in [30] for different ratios of dynamic loads and distribution generation. This includes ZIP, exponential (EXP), EPRI and exponential

recovery model (ERM), and the dynamic extensions of ZIP and EXP. [29] makes a distinction between different types of inductive machines, e.g. residential motors, large and small industrial motors and air conditioning devices, and compares the performance of ZIP and 1st and second-order ERM. [31], [32] are some of the earlier works of component-based modelling and represent the DN by an induction machine, a ZIP (constant impedance, current and power) load and a converter-based SG. [33] uses a ZIP load, directly connected SG and a voltage source connected through an inverter (e.g. photovoltaics, PV) to represent a microgrid. Trajectory sensitivity analysis is performed to optimise only the important parameters of the equivalent. [34] uses the network of [31] but simplifies the problem by discarding non-controllable and non-observable parts from the model. [35] takes a different approach by using Monte Carlo simulations to sample different load and generator parameters and generate dynamic responses of the network to different disturbances. The parameters of the equivalent model are found as 1) the set of randomly drawn parameters whose response is closest to the average of the responses caused by each set of parameters or 2) the set that gives a response that is closest to any other given response. [35] also uses an inverter-based generator that reproduces the typical response of such a component to terminal voltage changes required by most grid codes. This concept is expanded in [36], where the first approach is used to find the parameters, but it is scaled by the standard deviation between responses, and LASSO is applied to reduce the number of parameters in the optimisation. Moreover, multiple operational scenarios can be considered by evaluating the existing equivalent performance, deriving a new set of parameters if necessary, and storing them for later use. [37] considers the spatial uncertainty between renewable energy sources (RES) and uses enhanced reinforcement learning (ERL) to run a probabilistic power flow for TSOs. Sometimes, the components are replaced by equivalents at multiple buses instead of only at the PCC [38], [39]. [38] introduces a partial tripping feature that reproduces the effect of individual IGBs tripping in the DN and provides more insights into the LASSO method for reduction of optimisation parameters. The component-based equivalents are attractive to system operators because of their explainability and can easily be implemented into existing simulation software because they consist of the same components. A drawback is that the equivalent's performance is highly dependent on the found parameters, and its best performance is limited by the underlying (usually fixed) network structure.

Cluster-based equivalent models are a relatively new type of equivalent models that combine the powers of knowing the type of components in the network and deriving useful patterns from the measurement data. Different methodologies for clustering are employed, focusing either on grouping components with similar dynamic responses or on clustering based on varying initial operating conditions.
The first approach follows the concept of coherency identification but uses machine learning clustering methods to find components that behave similarly. Each group is represented by a single component in the equivalent. [40] finds clusters with similar behaviour from real measurement data and fits those responses to modified ERM models. A similar approach is applied in [41], and the performance is compared with a long short-term memory (LSTM) network, a black box approach, and proves better performance.
The second approach is creating equivalent models based on clusters of initial operating conditions. [42], [43] concurrently train a neural network to predict the fitted parameters under different conditions, which improves the generalisation performance. In [42], an artificial neural network is trained per cluster of similar initial conditions that predicts the dynamic model parameters. Prediction is performed by finding the cluster most similar to the current conditions and using the trained network. [43] follows a similar approach but adds Fisher discriminant analysis to combine similar clusters, uses a component-based equivalent model to fit the parameters, and predicts new operating conditions with an Elmann neural network.

Grey box modelling approaches are attractive for equivalent modelling of DNs because there is a tangible base to the model, i.e. underlying components or structure. They have higher explainability than black box models, meaning it is easier to get acceptance for using them. It is no wonder that most research is concentrated in this area. One of the major downsides is that the underlying model inherently limits the maximum accuracy of the dynamic response that the equivalent can produce [30]. Nevertheless, this class of equivalent modelling remains attractive for modelling active distribution networks.

1.2.3. Black box modelling

Black box modelling is a model-agnostic way of creating equivalent models. These methods rely on collected data to fit a model, but, unlike the grey box models, the underlying model is not based on power system elements. The three main types of identified approaches are machine learning, curve fitting, and learning probability distributions.

Machine learning is a fast-developing field where data is used to train (mathematical) models to perform a specific task. One such task is regression, where the model learns to predict one or several outputs based on input data. Reproducing the dynamic response of a system from input data is an example of regression. In [44], [45], recurrent neural networks (RNNs) are used for this task. This type of neural network has the property that it can capture long-term dependencies in the data, which is desired from dynamics since the current value is highly correlated to the previous values. Moreover, the mathematical representation of these networks resembles that of the differential algebraic equations that govern the power system. This makes this type of network suitable for representing the dynamic response of ADNs. [44] uses a long short-term memory (LSTM) RNN sandwhiced between fully connected layers to predict the active and reactive power using the previous bus voltages as inputs. [45] applies gated recurrent units (GRUs) to predict the i_α, i_β currents of a microgrid cluster using the operating points of all clusters and the voltage at the observation point. Moreover, the known differential equations are also used in the neural network to predict the output. Other regression methods are also suitable for this prediction task but might yield lower accuracy. For example, regular neural networks can also perform regression tasks but larger networks are needed to accurately produce a sequence resulting is less computational efficient networks. Convolutional neural networks (CNNs) improve this by deriving new latent features using convolutional filters, yielding better performance with usually a lower dimension. These latter networks can be trained in parallel, something that RNN cannot. However, CNNs do not store their network state for using it in future predictions. Machine learning methods have demonstrated their excellent performance in various tasks. But, a downside is that they need a lot of data to train, whereas data is quite scarce in power systems.

Curve fitting is another way to fit data to a model by learning the best coefficients. It has some similarities with machine learning, but in curve fitting, the mathematical model is often chosen beforehand based on some assumptions about the form of the output. In contrast, machine learning methods often learn this themselves. Prony analysis uses Prony's method to fit several complex exponentials to the measured signal. These exponentials contain the phase, magnitude and damping coefficient of each frequency. Prony analysis is also used as a data-driven modal analysis method because it is possible to estimate the modes from the signal. [32] uses Prony analysis to learn the parameters that reproduce the power output of the system for different events and concludes this approach works well when there is only one dominant mode in the network. [46] expands on this concept by using the pre-disturbance steady state condition and disturbance amplitude to adapt the magnitude term of the Prony exponential, realising a slightly more general model. Still, this method requires the oscillation frequency to be known, making it more suitable for mode identification than reproduction. Hammerstein-Wiener models represent the dynamics of a non-linear system by using a linear transfer function and imposing the non-linearities using input and output functions. Because of their convenient block representation and clear relation to linear models, they are easier to implement than neural networks [47]. [12] uses Hammerstein-Wiener models to learn the relationship between the voltage at the PCC and the power (and current) output of the DN. This model performs better than the EXP load model, but, as the authors also point out, this obtained model is only valid for the cases under which it was trained. This parameter-dependency is similar to that of the grey box models, except that at the basis of those approaches, there were real power system components.

A different approach is to view this task as a probabilistic problem and learn to estimate the most likely output given some past observations and measurements. [18] uses probabilistic time-varying parameter identification to predict the power output by estimating the parameters of a ZIP load and induction motor using past predictions and measurements. This problem is implemented using a deep generative architecture consisting of temporal feature extraction with LSTMs and a generative adversarial network. [19] uses Gaussian Processes to predict the power output. A Gaussian Process is *"a set of finite random variables connected by a joint multivariate Gaussian distribution"* [19]. The prediction is done using a separate process for the active and reactive power output, and the difference in power output is taken to compensate for different levels of DG penetration. The more diverse observations

there are, the better the model will be. The advantage of these approaches is that because the predicted output is probabilistic, it is possible to indicate how confident the model is. However, this can be a double-edged sword because operators are slow to adopt probabilistic approaches and would rather have a deterministic output than a range of possible outputs.

Black box models are attractive for equivalent modelling because they can be applied in situations with limited prior knowledge available about the underlying network structure and its components. Similar to grey box modelling, their performance depends on some fitted parameters to the response of the system. A drawback is that they rely on abundant data for accurate predictions and good generalisation performance [17]. This is because the output is generated by mathematical functions rather than real power system components in grey box modelling. Nevertheless, this class remains attractive because of its excellent prediction performance, even in system-agnostic cases.

1.2.4. Missing research

Equivalent modelling is an ongoing topic of interest because larger systems need to be simulated on finite computational power. In the past decade, this topic has made its way to DN as well because of their transition towards an active network, something which was not the case before. The scientific literature is mostly focused on grey box modelling because of its explainability and ease of implementation in existing simulation software. On the other hand, black box models have been explored more in recent years and have shown promising results. Still, some topics have not been investigated yet.

The problem with existing approaches is that they do not explicitly consider the operating conditions or the topological changes in their models. Instead, other approaches fit parameters that best represent the average dynamic behaviour. For grey box equivalents, where the entire DN is represented by one or several components that produce a dynamic response equivalent to the original network, this approach is logical as the components can have a single set of parameters [31]. The problem with fitting a single set of parameters is limited generalisation performance [46]. As the system conditions shift slightly, e.g. by an increase in generation and consumption, a different set of parameters will yield a better accuracy [29], [41], [42]. This was also seen in [34], where two equivalents were derived for different times of the day. More recent approaches focus on clustering similar dynamic responses and deriving a set of parameters per cluster [29], [41], [42]. However, the real-time selection of the most appropriate clusters is unclear. Moreover, grey-box models have a limit to their accuracy as that will depend on the components making up the equivalent model or the chosen mathematical model [30], and there is no consensus on which one is better. On the other hand, black-box approaches rely on curve fitting techniques and machine learning to reproduce the dynamic response by using large amounts of data to train the models [44]–[46]. Unfortunately, the curve-fitting algorithms suffer the same problem as the grey-box models, meaning their coefficients change under different system conditions [46]. The machine learning approaches can learn any arbitrary function [48] but are usually less interpretable since their internal models differ from the power system model [17]. Also, machine learning models require large amounts of data to cover a variety of scenarios, which has resulted in limited validation of the models under different system conditions [17]. At the same time, this is a necessity for accurate equivalent models [41].

1.3. Objective

Accurate simulation of the now active distribution network is becoming crucial for network operators. The current techniques for equivalent modelling create models that are fitted to the average dynamic response of the system or have limited applicability. Also, the existing methods lack extensive verification under different system conditions. Moreover, these works rarely focus on ADNs with multiple PCCs, something which becomes more common as ADNs become more meshed. This thesis aims to develop an equivalent model that can estimate the dynamic response of an ADN with multiple PCCs and under diverse operating conditions and topological changes. The following research questions need to be answered:

1. How can an equivalent model be created that can estimate the dynamic response of an ADN with multiple PCCs under diverse operating conditions?
2. How can an equivalent model be created that can estimate the dynamic response of an ADN with

multiple PCCs under topological changes?

3. How does the method scale to larger systems?

The expected outcome is a new method for developing equivalent models of distribution networks that can grow with their transformation into active components of the power system. It is more future-proof than alternative approaches because it can handle different network configurations while still allowing for fast simulation of networks with many nodes, something that is unfeasible for full network models.

1.4. Thesis outline

The thesis consists of the following chapters. Chapter 2 presents the theoretical background, including equivalent modelling, power system dynamics and machine learning fundamentals. Chapter 3 contains the proposed methodology. In Chapter 4, several case studies investigate the proposed methodology. The discussion and conclusion are presented in Chapter 5.

2

Background

This chapter discusses the relevant background information on which the proposed approach has been developed. First, section 2.1 elaborates on the role of equivalent models. Second, an overview of power system dynamics is given in section 2.2. Third, three mainstream neural networks are discussed in section 2.3. Finally, in section 2.4, the graph neural network (GNN) is introduced with its temporal variant. This is the GNN used in the developed method.

2.1. Equivalent modelling

Equivalent models (or surrogate models in machine learning) are simplified representations of more complex systems. The purpose of such models is to retain specific characteristics of the entire model while unessential aspects are discarded. The result is a model that is accurate for the desired application but is usually easier to solve.

2.1.1. Applications

The applications of equivalent models are numerous. In engineering, such models represent the fundamental physical interactions and create a reduced design problem with several crucial parameters. In economics, market dynamics and customer behaviour are approximated by mathematical models that allow for the interpretation and prediction of economic trends. Weather models are another excellent example where a simpler model can be used instead of running complex thermodynamic, fluid, and other equations. In circuit theory, the Thevenin and Norton equivalent circuits replace entire circuit parts with an equivalent (voltage/current) source and an impedance, simplifying the analysis of linear, time-invariant circuits. Conceptually, every model made from a real process can be considered an equivalent.

2.1.2. Advantages and disadvantages

Equivalents have several advantages. First, they drastically simplify the underlying mathematical equations for a model. As an effect, simulation is made possible and faster. Second, equivalents could be more cost-effective than full models because fewer resources are needed. Third, sensitive information can be hidden depending on the type of equivalent. Others can still use a black box model for a specific application without seeing its content. This is important when exact designs or sensitive data should not be shared. There are two downsides to equivalent modelling. Their accuracy can be lower than the original model. Moreover, their application is limited, meaning they are less versatile.

2.2. Power system dynamics

Power systems are analysed to ensure their behaviour is within operational limits or satisfies macroscopic needs such as consumed power. When the system is in a steady state, the system is considered stable, and quantities such as voltage, current, active and reactive power and frequency do not change. However, any change in one of these will result in the transient behaviour of the various components

and their control systems as the power system tries to reach a new steady state. Analysis of the dynamics against various disturbances will ensure that the power system is resilient enough to work within limits in real life; alternatively, it will highlight improvement areas.

2.2.1. Different types of dynamics

Various phenomena require distinct timescales in power system studies, as shown in Figure 2.1. On the one hand, electromagnetic transient (EMT) studies assess the effects of fast transients such as lightning strikes and switching behaviour. These EMT studies can be used to determine if equipment limits (e.g. max voltage) are violated to prevent equipment failure or destruction. On the other hand, slow and long-term dynamics are investigated to restore system frequency and optimise dispatch for economical operation.

Figure 2.1: Overview of power system dynamics' time scales (from [49])

The equivalent model developed in this thesis is focused on transient stability. Transient stability (sometimes called dynamic stability) is located somewhere in the middle with a time scale of 10^{-2} and 10^{1} seconds. These types of studies indicate the ability of a system to remain in synchronism with other generators following a large disturbance [50]. In other words, the goal is to investigate the network's resilience against a selected set of disturbances (e.g. faults) and evaluate whether the system moves towards a steady state operation. These events are low probability but have a high impact since they can cause cascading outages and widespread blackouts [51]. Therefore, system operators must maintain transient stability.

2.2.2. Frequency and voltage control

Initially, the grid was dominated by induction and synchronous machines with control systems that behaved in a particular way. To ensure that the components in the system are not damaged and that the desired amount of power is transferred, two quantities have to be controlled: frequency and voltage [50]. Frequency closely affects active power, while voltage regulates reactive power (as demonstrated in Chapter 6.3 of [50]).

The regulation of frequency is done by the automatic generation control (AGC) system. The AGC consists of primary, secondary and tertiary controls. The primary control maintains the power balance in a network using the droop control relation, dictating how the power output of a particular unit must change following a frequency deviation. This is done by the speed governor (GOV) in rotating generators, and the result is a new stable frequency, although different from the nominal frequency. The secondary control will detect that the frequency is different from its nominal value and adjust the set points of the generators to return it to the nominal speed. The tertiary control might again adjust the setpoints to yield more economical operation, e.g. by minimising the transfer losses.

A constant frequency is essential for the satisfactory performance of generators as they depend on the performance of all auxiliary drives associated with the fuel, the feed-water and the combustion air supply systems [50]. Also, a sudden drop in frequency could result in high magnetising currents in induction motors and transformers. Moreover, large deviations could result in generator tripping, load loss and blackouts.

The regulation of voltage is made by automatic voltage regulators (AVR) in generators and other components such as shunt capacitors and reactors, static var compensators (SVCs), series capacitors and regulating transformers [50]. Active systems such as the AVR and SVCs automatically adjust their reactive power output to maintain a preset voltage level of the buses to which they are connected.

Failing to regulate the voltage has several undesired effects. The voltage can fall outside the operational limits of components, causing equipment damage, disconnection, and possibly cascading failures. Poor voltage and reactive power control also adversely impact the system stability [50]. Moreover, a large amount of reactive power can drastically increase transmission losses when present in large quantities in the network.

2.2.3. Equivalent models for transient stability

The frequency and voltage control systems introduce a dynamic power system response every time it is disturbed from its steady state point. Since failing to control the active and reactive power in the network has drastic implications, operators must investigate the network's dynamic behaviour, including the distribution networks, which have now become active [13]. Equivalent models are suitable for modelling active distribution networks since they reduce the computational burden and hide sensitive information such as customer consumption.

Generally, the equivalent model can be represented by Figure 2.2. At the input, it has the controlled quantities V (voltage) and f (system frequency); at its output, there are the controlled quantities P (active power) and Q (reactive power).

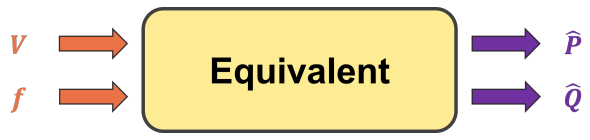


Figure 2.2: General representation of a dynamic equivalent

The difference between the different types of equivalents is the method used to derive them.
White box modelling often reduces the full network model to a simplified representation through techniques such as model order reduction [7], system linearisation [21], modal analysis [22], [23] and coherency identification [24], [25].
Many grey box approaches are a form of system identification where several generic representative components have unknown parameters (see Fig. 2.3). The value of those parameters is determined through fitting techniques and collected measurement data.

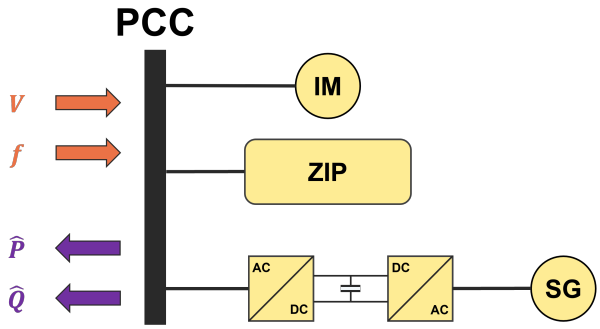


Figure 2.3: Grey box equivalent consisting of one inductive load, a ZIP load and a convertor-based generator (from [31])

Black box approaches are model agnostic methods for creating equivalents: they use no prior knowledge about the network but instead rely on collected measurement data to learn to reproduce the dynamic response. Different approaches can be used; three are shown in Fig. 2.4:

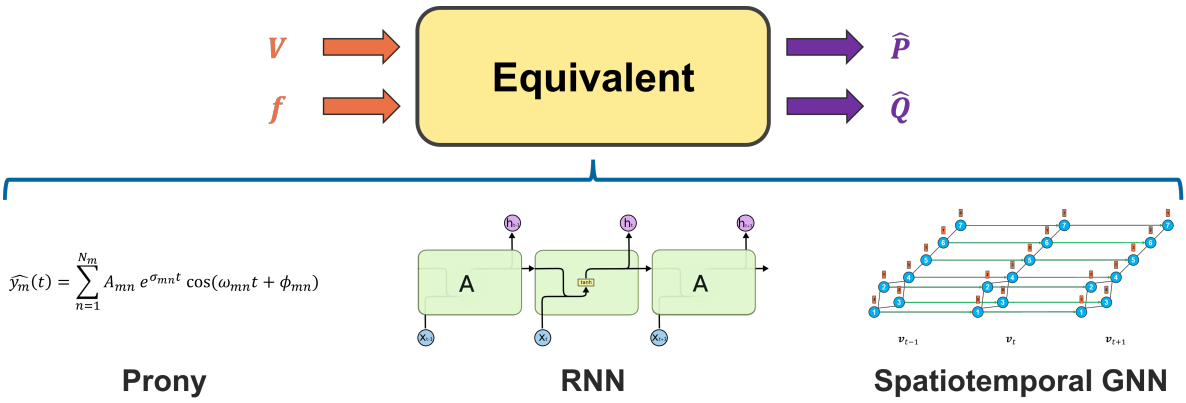


Figure 2.4: Three black box equivalents based on prony exponentials, RNNs (figure from [52]) and spatiotemporal GNNs.

At the core of these black box approaches, there is a mathematical model with different parameters that can be learnt. Using optimisation algorithms such as nonlinear least square optimisation [46], metaheuristics such as particle swarm optimisation or solvers such as stochastic gradient descent (SGD) or adapted moment estimation (ADAM), it is possible to learn the parameters that give an output as close to the actual output as possible. How well a black box model can learn the input-output relation depends on the (mathematical) model's compatibility with the data structure. Prony's method, for example, works well because power system dynamics are a combination of (dampened) oscillatory modes. Similarly, RNNs and spatiotemporal GNNs are designed to learn time series data, making them more suitable to capture dynamics than a simple, fully connected neural network (FCNN).

Grey and black box approaches have advantages and disadvantages when used as equivalent models for transient stability.

Grey box models are explainable because the underlying components are usually power system components. This also means these models should be easier to implement in power system simulation software such as PowerFactory. A big drawback is that the used components and their number limit the accuracy. Suppose two distinct inductive loads are fitted to a single equivalent component (as in Fig.

2.3). In that case, the model should have a lower accuracy than one where an individual equivalent component represents every distinct type of load.

Black box models, on the other hand, use generic mathematical expressions that can approximate any function provided that the network is infinitely large, which is known as the universal approximation theorem in machine learning [48]. Therefore, if the underlying model is chosen correctly, this class of equivalents could generalise better. However, a large amount of data is needed to create accurate equivalents.

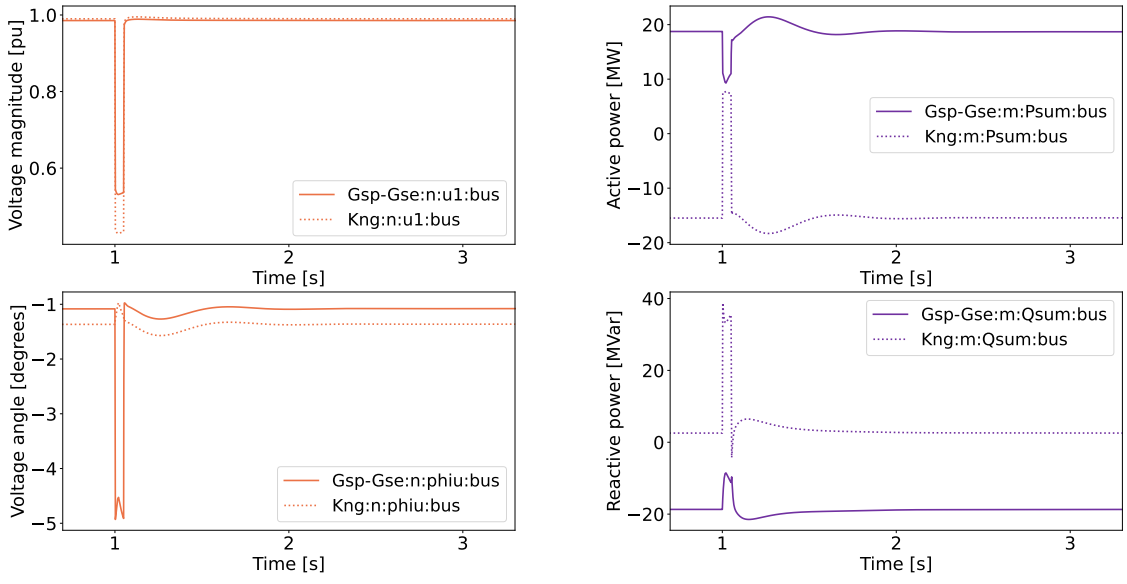
2.3. Neural networks

According to IBM, *"machine learning (ML) is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy."* [53] At the forefront of this field are neural networks, which have gained immense popularity due to their wide range of applications caused by their ability to identify patterns in large data sets. These networks, drawing inspiration from neurons in the human brain, excel in tasks such as image and speech recognition, predictive analytics, and even in creating advanced language models like ChatGPT [1].

In this section, more background is provided on regression with machine learning and three types of neural networks that can be used: artificial neural networks (ANNs), recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

2.3.1. Supervised regression

Regression analysis estimates the relationship between a dependent variable (the label) and one or more independent variables (the features). In the case of the dynamic equivalents of Fig. 2.2, the active P and reactive power Q are the outputs, and the voltage V and frequency f are the inputs. Fig. 2.5 shows an example where only the voltage is considered. For this example, an equivalent for transient stability would have four inputs (features) and four outputs (labels). Note that this example is not a forecaster. This is because the period of the inputs is identical to that of the outputs, whereas a forecaster would use historic inputs to predict future values.



(a) Voltage inputs at two PCCs of an active distribution network (ADN). (b) Active and reactive power response at the PCCs of the same ADN.

Figure 2.5: Example inputs and outputs of an ADN.

[1] This paragraph has been written by ChatGPT 4

The task shown by this example falls under supervised machine learning instead of other categories such as unsupervised and reinforcement learning. In supervised learning, the algorithm is trained on a labelled dataset, meaning that the inputs (in this example, V and frequency f) have known outputs (P and Q). A pair of inputs and outputs is called a sample. Using this category to predict power system dynamics makes sense for two reasons. First, measurement data collected by, e.g. phasor measurement units (PMUs) already contains the desired quantities and, therefore, the labels. Second, the output should be tuned to a particular ADN and its specific dynamics. This requires using labelled data, whereas other categories, such as unsupervised learning, would try to learn some general characteristics.

Various models can be used for supervised regression. Ordinary Least Squares regression is well-known in fields such as economics. It fits a linear combination of the features to minimise the residual sum of squares between the predictions and the targets. The popular Python library scikit-learn supports many models, including Bayesian and logistic regression, support vector machine regression, nearest neighbour regression and decision tree regression. Neural networks are another model type for regression. They can outperform these models because of their ability to model non-linear relationships and their scalability performance.

2.3.2. Artificial neural network basics

This section contains the basics for understanding derivatives such as RNN, CNN and GNNs. The more interested reader is referred to Chapters 6 - 8 of [54] for a more in-depth explanation of (deep) neural networks and their training.

Neural networks are the foundation of many AI tasks. The neurons inspire their structure in the human brain, consisting of the following key components (see Fig. 2.6).

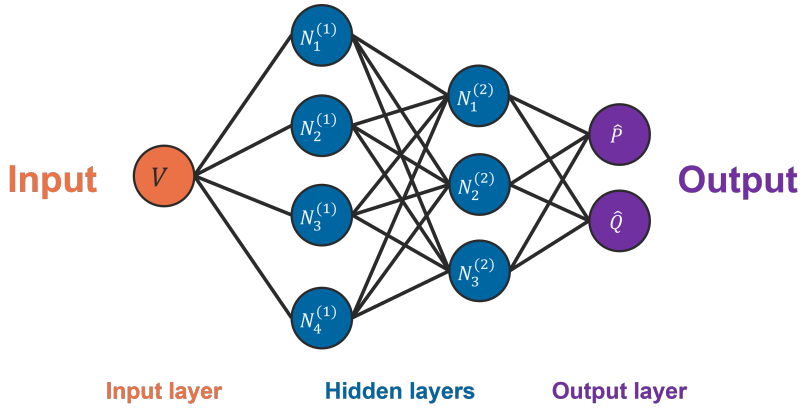


Figure 2.6: A simple neural network.

A **node** is the place where information is collected. Each node receives one or more **inputs** from the features inputted to the NN (in the input layer) or from other nodes (hidden and output layers). Each of these inputs is associated with a certain **weight**, and together with a **bias**, they form the linear output (or embeddings) of a node; an activation function $g(\cdot)$ is often used to introduce a non-linearity to this output.

Nodes in the first layer can be formulated as follows. If the input is given by V, the weights to node $N_i^{(1)}$ by $w_i^{(1)}$, the bias by $b_i^{(1)}$ and the activation function by $g^{(1)}$, it is possible to write the output of $N^{(1)}$ as:

$$N_i^{(1)}(V) = g^{(1)}(w_i^{(1)} \cdot V + b_i^{(1)}) \tag{2.1}$$

If all nodes of the same layer are aggregated, the following equations are obtained:

$$N^{(1)}(V) = g^{(1)}(W^{(1)} \cdot V + b^{(1)}) \tag{2.2}$$

The outputs of all nodes from the same layer $N^{(1)}$ are passed as inputs to the nodes in the next layer,

and the process is repeated with $N^{(1)}$ as the input:

$$N^{(i)}(V) = g^{(i)}(W^{(i)} \cdot N^{(i-1)}(V) + b^{(i)}) \tag{2.3}$$

Finally, the node embeddings from the last hidden layer are passed through another activation function, $g^{(out)}(\cdot)$, in the output layer to form the network's output. If there are L hidden layers, the output can be formulated as:

$$f(V) = g^{(out)}(W^{(out)} \cdot N^{(L)}(V) + b^{(out)}) \tag{2.4}$$

The choice for this activation function depends on the task at hand [55]; for regression, this is a linear activation function, and this simplifies Eq. 2.4 into:

$$f(V) = W^{(out)} \cdot N^{(L)}(V) + b^{(out)} \tag{2.5}$$

The output $f(V)$ is the predictor for P and Q, but there will be some error because $\hat{P} \neq P$ and $\hat{Q} \neq Q$. This is expected because all parameters θ (the weights and biases) are not yet optimised but have some random values. A **loss function** is introduced to quantify this error. For regression, the mean squared error (MSE) loss is a useful metric which is computed as follows for n input-output pairs:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2 \tag{2.6}$$

In Eq. 2.6, Y_i represents the true output (P or Q), and $\hat{Y}_i$ is the predicted values by the output layer ($\hat{P}$ or $\hat{Q}$). The optimal parameters θ^* are the ones that minimise the loss function of Eq. 2.6 for all n samples. This means that, on average, the squared error between the predicted output and the true output is as small as possible.

Unfortunately, neural networks usually do not have closed-form solutions and, therefore, require an iterative solver such as SGD to approximate the optimal parameters θ^*. This is done in a couple steps. First, a forward pass is done to compute the outputs $\hat{P}$ or $\hat{Q}$. Next, the loss is computed using a loss function such as in Eq. 2.6. Subsequently, backwards propagation takes place. In this step, the gradients of the loss function are calculated with respect to each network parameter. The computed gradients are then used to update the network parameters using an optimisation algorithm such as SGD. This will slowly move the current weights towards the optimum θ^*. This process is repeated for a predetermined duration, number of updates using the entire dataset (epochs) or until the accuracy is not improving anymore.

As stated by the universal approximation theorem, ANNs can, in theory, learn to represent any function if their growth is unbounded. In reality, computational power limits what can be achieved. This has resulted in developing more specialised networks, each with distinct properties and suitable for different types of data. The next sections will discuss three types: RNNs, CNNs and GNNs.

2.3.3. Recurrent neural networks

Humans do not perform regular tasks only relying on information available at the time of execution; they use prior information they have learnt earlier in their life. Moreover, even mundane tasks such as communication with others do not consist of individual pieces of information but are instead based on a connected sequence. Many other, more specific, tasks also consist of connected sequential data, as is the case for time-series data such as a power system dynamics response. This means that the input-output pairs are no longer independent of each other but follow a sequential order.

Recurrent neural networks exploit this property of data by introducing loops in the network structure, as seen in Fig. 2.7. As opposed to the ANN in Figure 2.6, the output predicted by the RNN depends not only on the current input but also on the output of a previous prediction [2]. This results in the introduction of a memory element in the NN, resulting in a more suitable structure for learning sequential data.

[2]The output of the RNN, in Figures 2.7 to 2.9 denoted as h_t, are the hidden output of the network. This is not necessarily the same as the actual output (e.g. P_t or Q_t). This depends on the exact network structure as the outputs of the RNN could be passed through additional layers, such as a fully connected ANN.

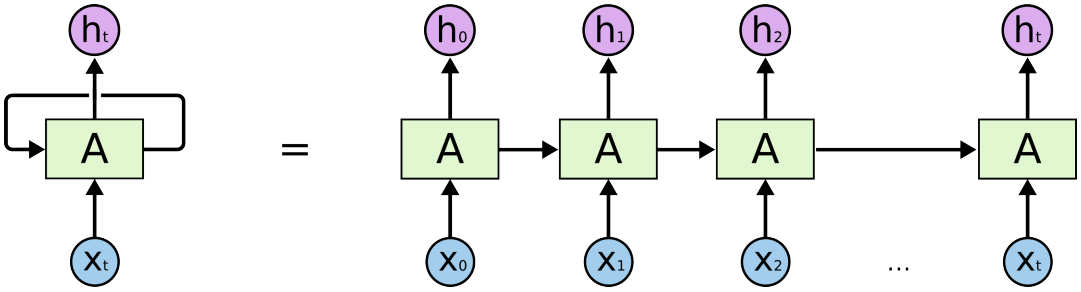


Figure 2.7: Unrolled RNN (figure from [52])

The simplest version is seen in Fig. 2.8. Here, a single hyperbolic tangent activation function is used, and the output is directly combined with the input in the following prediction step.

The problem with this RNN is that it is elementary. The network does not have a separate state for the memory element but shares this with the previous output. This also means that the memory of the network is limited to only the previous state.

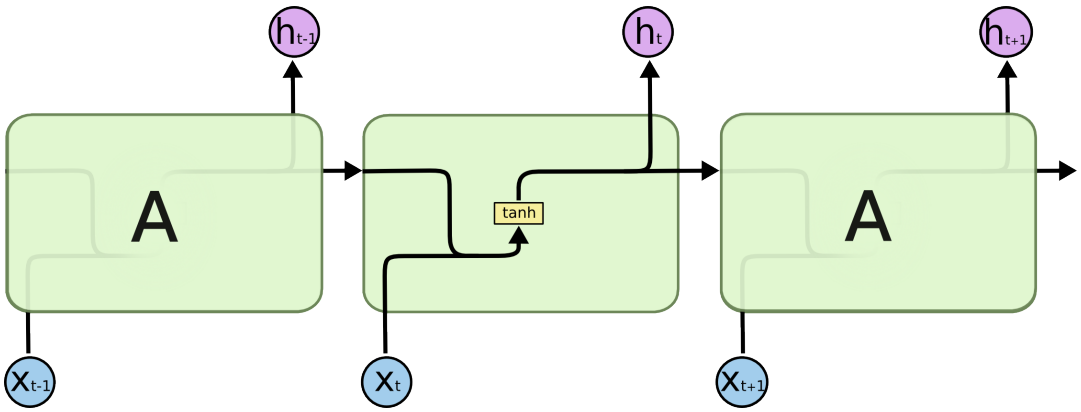


Figure 2.8: A basic RNN cell (figure from [52])

A long short-term memory (LSTM) network solves this problem by introducing a separate memory state called the cell state C_t. In Figure 2.9, this is the output at the top that flows from one step to another. The LSTM network can change the cell state by subtracting or adding information.

In the first step, a sigmoid layer called the forget layer removes information from the cell state C_{t-1}. A sigmoid function has an output between 0 and 1. By looking at the current input X_t and previous output h_{t-1}, this layer determines what information to keep and remove from the cell state. This gives Equation 2.7a.

In the second step, information is stored in the cell state. This consists of two parts. First, a sigmoid layer is used to decide which parts of the cell state to update (Eq. 2.7b); similar to the forget layer. Then, a hyperbolic tangent layer is used to create the new values $\tilde{C}_t$ to add to the cell state, as seen in Eq. 2.7c.

The cell state is then updated using Equation 2.7d by multiplying the previous cell state C_{t-1} with the value from the forget layer (Eq. 2.7a) and multiplying the value from the input layer with the new values for the cell state (Equations 2.7b and 2.7c respectively).

The output of an LSTM cell h_t is computed based on a filtered version of the new cell state C_t. First, a sigmoid function is used to choose what parts of the cell state to output (Eq. 2.7e). Next, the output is computed with Equation 2.7f, using the cell state C_t passed through a hyperbolic tangent function and the previously calculated output multiplier (Eq. 2.7e).

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.7a}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.7b}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.7c}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.7d}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.7e}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{2.7f}$$

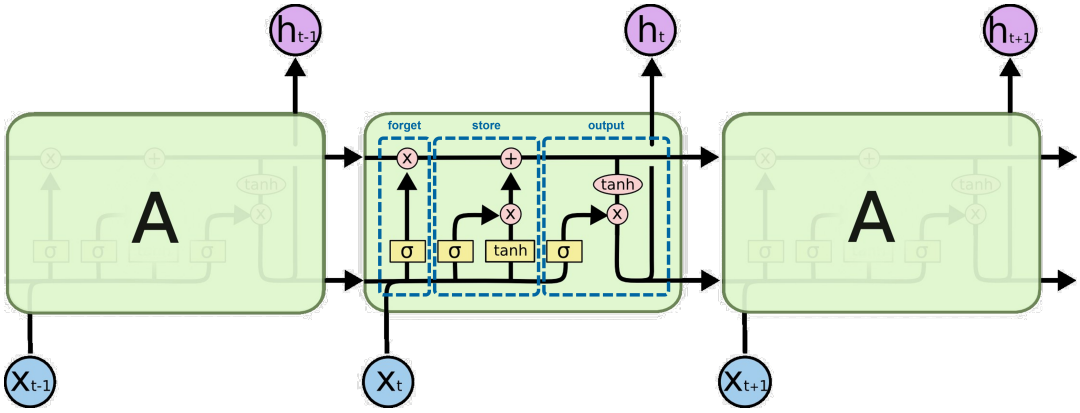


Figure 2.9: LSTM cell(figure adapted from [52])

RNNs and LSTMs are suited for representing power system dynamics because of their ability to learn sequences in data. An LSTM-based equivalent has been developed in [44] based on the mathematical similarity between LSTMs and the discretised differential equations.

There also exist several variations of the RNN and LSTM. One notable mention is the gated recurrent unit (GRU). This type combines the forget and store steps of the cell state and merges the cell and (hidden) output state of the LSTM. A network based on gated recurrent units has been used by [45] to learn the dynamic behaviour of unknown components in a microgrid.

2.3.4. Convolutional neural networks

The human brain is excellent at filtering large amounts of information and identifying only the relevant bits for a task. An excellent example is vision. Incandescent light from our field of view falls on the retina and reaches the receptors. At its core, this is just light falling on a tiny surface. However, our brains process this information to interpret things such as shapes, objects, and colours and even fill in missing details at the periphery of our vision.

Vision and vision data work with grid-like structures. As an example, the number 1 is drawn in Fig. 2.10 on an 8x8 field using black (value 1) and white (value 0) squares[3]. Neural networks such as the ANN of Fig. 2.6 would treat each field or pixel individually, introducing several challenges such as the need for a deep network, an extensive dataset and difficulty in learning patterns.

[3]Although this is an elementary example, this is how cameras and computers reproduce images, albeit at much higher resolutions and include multiple channels (red, green and blue for colour and an alpha channel for brightness).

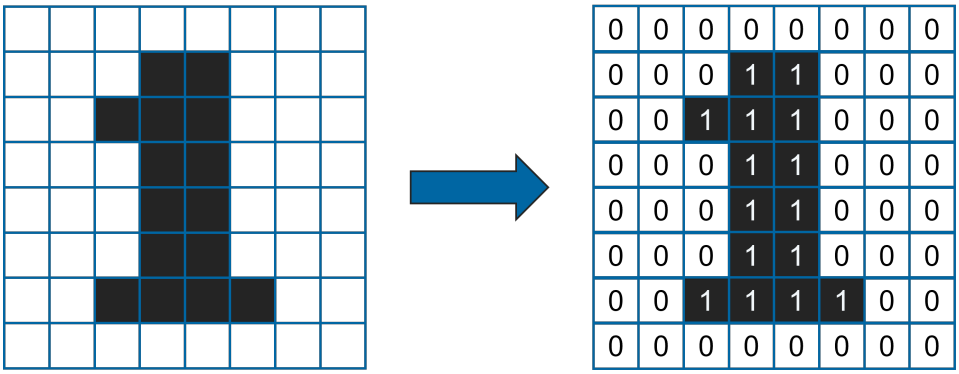


Figure 2.10: An 8x8 greyscale representation of the number 1.

CNNs are inspired by this process of learning to detect helpful information and are designed to work on grid-like structures. They typically consist of three types of layers: a convolutional layer, a pooling layer and a fully connected layer, as is also seen in Fig. 2.11. These layers are arranged to detect simpler patterns first (e.g. edges, corners and colour gradients) and more complex patterns later on (shapes, numbers, etc.)[4].

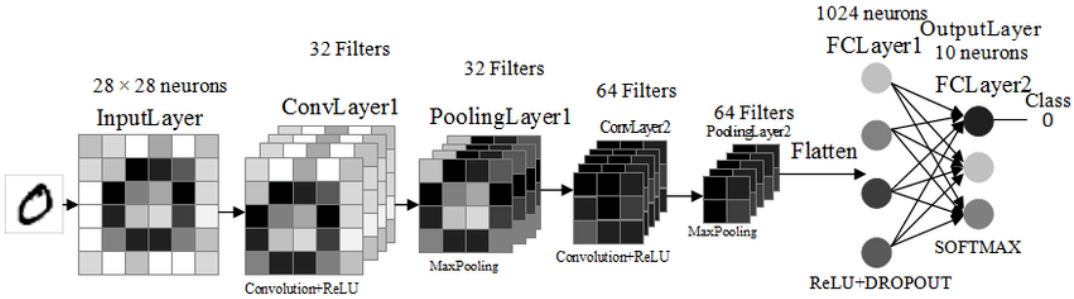


Figure 2.11: Example of a seven-layered 2D CNN network architecture for handwritten digit recognition (from [56])

The convolution layer is one of the fundamental building blocks of CNNs. In this layer, a kernel (or filter) is passed over the original data (or image) and transforms the data by performing convolution between the original input and the values in the kernel. Mathematically, this is given by:

$$(f * h)(t) := \int_{-\infty}^{\infty} f(\tau)h(t-\tau)\,d\tau \tag{2.8}$$

In Equation 2.8, f is the input function, and h is the kernel function. The multidimensional, discretised form of this equation becomes:

$$G[x,y] = (f * h)[x,y] = \sum_{j=0}^{L_{kernel}} \sum_{k=0}^{L_{kernel}} h[j,k] \cdot f[x-j, y-k] \tag{2.9}$$

In Eq. 2.9, $G[x,y]$ is the computed output at the grid coordinates x and y, and L_{kernel} is the kernel width. The kernel parameters $h[j,k]$ are usually optimisable parameters when training the CNN. When applying Eq. 2.9 to the number of Fig. 2.10 with a so-called outline kernel[5] of $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$, the following result is obtained:

[4] The early layers extract simple patterns and features. These values propagate towards the deeper layers of the CNN, which then learn combinations of them.

[5] This type of kernel is chosen for visualisation purposes. Normally, the kernel consists of learnable parameters by the network as it tries to detect useful features. However, when known kernels are applied to an input, certain effects can be obtained. This type of kernel is called an outline kernel because it can detect the outer edges of a shape. In Figure 2.12, it can be seen that the kernel can approximately learn the location of the vertical edges of the 1. If the image was of higher resolution, the detected edges would be much more visible. The convolution with known kernels on images are at the foundation of effects in image processing softwares such as Photoshop.

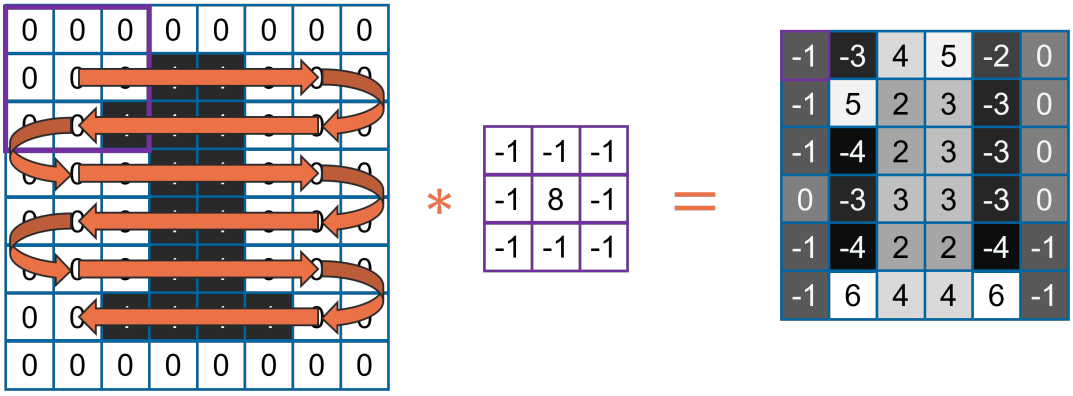


Figure 2.12: Example of the convolution layer on the 8x8 grid using an outline kernel.

In Fig. 2.12, the purple square represents the outline kernel at $x = y = 0$. This kernel is continuously shifted over the grid structure, and at every new position, the value is computed.

The results demonstrate two key points. Firstly, the convolution process reduces the output size compared to the input. This reduction occurs as convolution merges input values. This leads to a more compact model with fewer parameters, reducing memory usage and enhancing statistical efficiency [57]. Secondly, the kernel is applied uniformly across all inputs, unlike the separate weights in ANNs. This means that CNNs have sparse interaction because of this uniform application, and it grants the model a property known as equivariance to translation, implying that transformations in the input lead to analogous changes in the output [57].

The output computed by the convolution operation is linear. An activation function such as the sigmoid is used to introduce nonlinearities.

The second important layer is the pooling layer. Like the convolution operation, it looks at groups of values and then applies some statistical functions, such as average or maximum, to find the outputs; Fig. 2.13 shows an example using maximum pooling.

The pooling layer has two effects. First, it reduces the data dimensions going into the next convolutional layer. Second, it helps the generalisation performance. Because only the most important features are preserved during pooling, a certain translation invariance is provided [58].

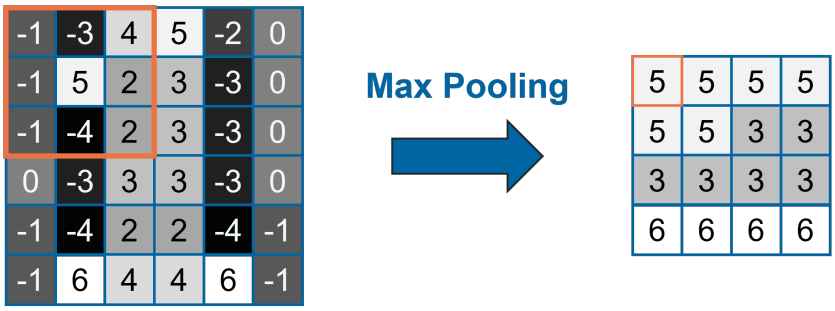


Figure 2.13: Max pooling layer performed on the output of the outline kernel.

The general structure of a CNN is similar. A convolutional layer is stacked together with a pooling layer. This process is repeated a couple of times, and every time, the input is shrunk. After the convolutional and pooling layer pairs, the result includes general embeddings from the input. A fully connected layer is then used to derive the final output from these embeddings (see Fig. 2.11).

Although CNNs are designed to work on grid-like structures, they can also be used for time series predictions by using 1D convolutional layers[6]. [59], for example, uses 1D CNNs to forecast the load consumption data.

[6]The 1D convolutional layer slides the kernel along a single dimension, namely time, and works similarly to find the embeddings and arrive at a prediction. In this section, it was chosen to show 2D convolution as that would provide a more tangible example to understand the basics.

CNNs have several advantages. First, the applied operations are highly parallelisable, meaning CNNs are much faster to train than sequential networks such as RNNs. Second, CNNs can automatically detect (hierarchical) patterns and features in the data. Third, some studies have also shown they can outperform other approaches such as LSTM and MLP [60].

There are also limitations to CNNs. For example, the input should have fixed dimensions because the kernels have a fixed size. This can be problematic for dynamic simulations where the input can be of different lengths. Moreover, the CNN only works well when data points are grid-like, i.e. equally spaced.

2.4. Graph and spatiotemporal neural networks

GNNs are a relatively new type of neural network designed to work on graph-like structures. Stanford defines graphs as *"a general language for describing and analysing entities with relations [or] interactions"* [61]. In graph theory, entities are called nodes or vertices, and relations are edges or links.

Graphs are ubiquitous with applications in many domains. For example, they can model transportation networks such as traffic and transit. Moreover, they can be used to model social and knowledge graphs and create recommender systems that show things of potential interest, such as people you might know [62]. Another excellent example is Google's PageRank algorithm, which determines a web page's importance by investigating how many other pages point to it [63].

The large diversity in graphs results in various GNN methods. All these models can be represented by a general design pipeline [64].

The first step involves finding the graph structure. This is obvious in applications where the graph structure is explicit, such as applications on molecules, transport networks and power systems. Other applications require the building of a graph first.

In the second step, the graph type and scale are specified. Graphs can be directed or undirected, indicating the bi-directionality of the relation between two entities. Moreover, graph components can be homogenous (identical) or heterogeneous (different from each other). Finally, the graph can be static (a traffic network) or dynamic (a social graph). All these choices affect which type of GNN is best suited.

The third step is the design of a loss function, which is task-specific. There are three types of tasks for graph learning: node level, edge level and graph level. Node-level tasks could focus on the classification of entities or regression of a specific value. Edge-level tasks are tasks where the type of a relation is predicted or whether a relation exists between two given entities. Graph-level tasks are performed on the entire network and try to predict holistic properties; classification, regression and matching are all possible. Based on the task level and whether the problem is supervised (labelled) or not, a different loss function is suitable.

The fourth step is to build a model using computational modules. A graph's computational module can generally be represented as in Fig. 2.14. The input is passed through several GNN layers, each learning the optimal parameters that minimise the loss function at the output embedding. Every GNN layer consists of the following general components. A **sampling module** is used to sample the information propagated on the graphs, especially on large graphs. The **propagation module** determines how the aggregated information is shared between nodes. [64] distinguishes between convolution and recurrent operators based on the principles seen in Section 2.3; **skip connections** can be used to propagate certain information directly. Finally, **pooling modules** can be used when performing graph-level tasks. Their goal is similar to that of CNNs, namely to extract high-level information from individual features or nodes.

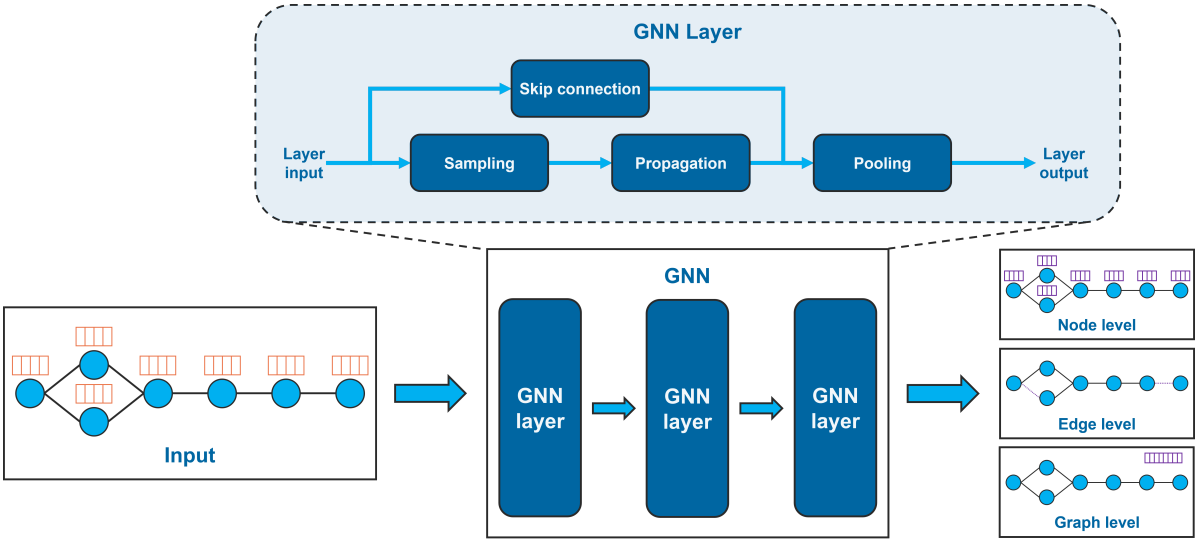


Figure 2.14: Basic design pipeline of a GNN (adapted from [64]).

GNN are also making their way to power systems with possible applications ranging from fault localisation and prediction and time series prediction of power generation or consumption to power flow calculations and (synthetic) data generation [65]. Since each application is distinct, they rely on different network types. A couple of important ones are discussed in more detail below.

Graph Convolutional Networks (GCNs) are most similar to CNNs as their goal is to extract a latent representation of graph-structured data. A distinction can be made between spatial and spectral GCNs. The main distinction is that the former performs convolutions in the graph domain while the latter does this in the spectral domain through the Fourier transform. This means the spatial-based GCNs show more robust generalisation and flexibility than the spectral-based network [65]. [66] detects faults in PV arrays from various measurements by using graph filters, which avoids computing the matrix inverse but rather uses powers of the shift operators (a repetitive operation). In [67], a spectral-based GCN based on Chebyshev polynomials localises faults in power distribution systems. A re-normalised spectral GCN is trained to infer upsampled DN measurements in [68]; state estimation is used to tune the output to be within physical constraints. A knowledge graph is combined with a GCN in [69] to identify the system topology, where the GCN detects conflicting information. Spectral GCN is also used in [70], but a clustering method is used to forecast residential power consumption.

Graph Attention Networks (GATs) are similar to GCNs but can learn the importance of adjacent nodes (or neighbours). Intuitively, this makes sense as some nodes (e.g. the slack bus) are more important in the network than others (e.g. a passive load).

GCNs and GATs alone cannot be used directly to predict power system dynamics. This is because the temporal aspect is not present. It would be possible to modify the input features to include the values at different time instances and do the same for the output. However, the underlying convolutional operation would still (randomly) learn to represent the input-output relation instead of having a structure that fully benefits the temporal sequentiality.

Spatiotemporal Graph Neural Networks (STGNNs) solve this issue by accounting for the temporal element in the data. Many of these approaches apply graph and time learning disjointly, meaning that the spatial part is processed separately and then fed into a temporal model (or vice versa). This type of STGNN is easier to implement and uses readily available components, which presumably is one reason why these models are popular in literature. A drawback is that there is limited theoretical analysis backing this approach and spatiotemporal dependencies are not captured as efficiently [71].

Graph Recurrent Neural Networks (GRNNs) are an example of STGNNs. They are similar to RNNs but instead have their features in the graph domain and can be used on node-level and graph-level problems that require outputting sequences, hence exploiting the memory properties of RNNs. These combined networks are considered disjoint if the spatial processing is an entirely separate step. In

[72], the transient stability is assessed by passing graph-structured measurements of a disturbance through GCN layers and feeding the embeddings through an LSTM. A similar approach is used in [73] where PMU measurements and known network parameters are used to predict the post-disturbance frequency trajectory. When the spatial processing is embedded in an RNN, this is known as a recursive model. This approach is explored for the prediction of solar irradiance where the graph convolution operation is part of the recurrent operation, and the cell and hidden states of the LSTM are updated by considering the influence of neighbours [74]. Moreover, it is possible to use GRUs instead of LSTM cells, as done in [75], to predict the fault location by using fault current and voltage measurements, the calculated short circuit current and the operation of protection devices. [76] takes a different approach by building a graph-convolutional recurrent adversarial network, training a generator to recover and predict synchrophasor measurements by creating data indistinguishable from real PMU data. [77] explores both a disjoint and a recursive model and finds a competitive performance compared to an LSTM and LSTMN-CNN on the Moving-MNIST and Penn Treebank datasets.

Graph Convolutional Networks with Temporal Convolutional Neural Networks (GCN-TCNNs) are another example of STGNNs but represent the temporal aspect with a CNN rather than a GNN. The advantage of this change is that CNNs do not have dependencies on previous steps, allowing for parallelisation and faster training [78]. The GCN-TCNN in [78] consists of a GCN layer sandwiched between two gated CNN layers and is used for predicting future traffic measurements based on past observations. A gated CNN modulates the convoluted output of a layer by a gate regulated by a different convoluted output [79]. A similar approach is used in [80] to forecast the PV power output at different locations by leveraging both the correlation in the output of nearby units and the temporal relation of a single unit. The receptive field of the gated CNN is expanded in [81] using a dilated convolution to estimate the remaining useful lifetime of NASA turbofan engines from sensor measurements. All these methods are examples of disjoint STGNNs since the spatial part is processed separately from the temporal part.

Graph-Time Convolutional Neural Networks (GTCNN) model the spatial and temporal aspects in a single, expanded graph. This is based on product graphs, where the spatial graph (indicating the connection of nodes at one time step) is combined with the time graph (containing information on how a node's features evolve over time) [82]. The result is a bigger product graph that contains both the spatial and temporal aspects. A big advantage of this approach is that different GNN structures can be applied directly since the resulting product graph is just a graph from the model's perspective. The GTCNN performs similarly to existing approaches such as LSTMs and Gated GRNNs for regression and classification tasks [82]. However, the GTCNN consistently performs competitively in the Molene and NOAA datasets, while the NN-based approaches outperform linear models when a lot of data is available and vice versa.

The approaches discussed so far are very similar to neural networks and combine the sampling and propagation steps of Fig. 2.14. However, some methods include a separate sampling module to collect the data. For example, missing data is recovered in [83] by using random walks to learn a latent vector feature representation of spatial and temporal features, which can then be used to train a regression NN that predicts the actual outputs. The interested reader is referred to [64] for a more comprehensive overview.

GNNs are promising for various power system applications since the network follows a graph structure, where buses could be seen as nodes and lines as edges. Using NNs biased with graph structures could result in better accuracy and faster training than a more generic network such as the ANN. From the discussed GNNs, the STGNNs are more suitable for designing dynamic equivalents for transient stability. This is because they include the temporal element in the network architecture, either in a separate model, through a recursive step or by using a product graph. GRNNs could better capture long-term dependencies because they keep track of a hidden (and possibly a cell state). However, these models are sequential because of the recursive computation (as illustrated in 2.7), making them train slower. GCN-TCNNs are faster because the 1D-CNN is a parallel computation, but their disjoint nature will not effectively capture the spatiotemporal dependencies. GTCNNs could be the middle ground, where the spatial and temporal elements are included in the same graph, but no recursive computation is required.

3

Methodology

This chapter describes the method to create the equivalent model. The proposed equivalent model is based on the system structure and includes previous-day conditions. Moreover, the equivalent model is implemented using a GTCNN network, a GNN that relates the available measurements through a product graph structure.

This chapter is structured as follows. First, the problem of equivalent modelling under changing operating conditions is highlighted in section 3.1. Then, section 3.2 explains the working of the GTCNN and its building blocks, the GCN with filter banks and the product graph. Finally, in section 3.3, the proposed method using the GTCNN is described.

3.1. Equivalent modelling under changing conditions

The Dutch high voltage (HV) grid diagram is seen in Figure 3.1. The grid map represents the perspective of the transmission system operator. The TSO models relevant parts of the network (e.g. the high-voltage overhead lines), while for other parts, they use equivalent models. The more detailed DN model must be connected to the more extensive network to derive a new equivalent network, e.g. for the Zeeland 150 kV grid on the bottom left. Events are generated in the HV grid, which yields a voltage V and frequency f disturbance at the PCC with the DN. In response, the DN will change its active P and reactive power Q outputs. V, f, P and Q are vectors of length T containing the time-evolving input and output sequences of the DN under one scenario. By recording the input and output pairs, a dataset can be constructed that shows how the DN behaves under different events. An equivalent model will then learn a function $f(\cdot)$ that relates $(V, f) \mapsto (P, Q)$ of the collected input and output pairs:

$$\hat{P}_{PCC}(t) = f_P(V_{PCC}(t), \theta_{PCC}(t), W_P) \tag{3.1a}$$

$$\hat{Q}_{PCC}(t) = f_Q(V_{PCC}(t), \theta_{PCC}(t), W_Q) \tag{3.1b}$$

In Eq. 3.1, $f(\cdot)$ is a mapping function which depends on the used model, $V_{PCC}(t)$ is the time-varying voltage magnitude $|V|$ at the PCC, $\theta_{PCC}(t)$ is the time-varying voltage angle θ at the PCC. For a grey box model, W contains the parameters of the different components of the equivalent model, making W more interpretable. For a black box model, W are the trainable weights of the mapping function $f(\cdot)$. This mapping function differs based on the used model, e.g., a GTCNN, an LSTM, a CNN or Prony exponentials. When an ML model is used, W represents the ML model weights. Using an ML model introduces less modelling bias than grey box models, as a mapping function $f(\cdot)$ is used that makes fewer assumptions about the underlying components it represents.

DNs are increasingly active, meaning they also produce power, e.g. through WTGs and PVs. The amount of generated power also influences the DN's ability to react in response to an event. For example, more wind and solar irradiance generate more power, meaning the DN can provide more active and reactive power to the TN. Similarly, the amount of consumed power also affects the dynamic response of the DN. Therefore, it is also important to consider events under different operating conditions. These

changes are reflected by changes in the initial power injections P_0 and Q_0 at various points in the DN. This changes the mapping to$(V, f, P_0, Q_0) \mapsto (P, Q)$, where P_0 and Q_0 are scalar values. This changes Eq. 3.1 to:

$$\hat{P}_{PCC}(t) = f_P(V_{PCC}(t), \theta_{PCC}(t), P_0, Q_0, W_P) \tag{3.2a}$$

$$\hat{Q}_{PCC}(t) = f_Q(V_{PCC}(t), \theta_{PCC}(t), P_0, Q_0, W_Q) \tag{3.2b}$$

Simultaneously, the internal structure of a DN is not static but is evolving. Lines and cables will be built to reinforce the network. Inactive components such as redundancy transformers and generators are turned on when extra capacity is needed. Transformers with taps change their positions to keep the voltage within operational limits. These are all examples of topological changes in the network, which also affect the DN dynamic response to a disturbance. Considering the model's performance under different topological conditions is crucial to verify its accuracy.

If a topological change is explicit, such as the addition of generators interfaced through transformers, an extra input $\mathbf{S}$ should be added to the mapping function $(V, f, P_0, Q_0, \mathbf{S}) \mapsto (P, Q)$. However, in this thesis, only two *hidden* topological changes are considered: activating inactive transformers and changing the tap positions of DN transformers. These topological changes are hidden since they are not observable by the TN in real applications. This means the mapping remains $(V, f, P_0, Q_0) \mapsto (P, Q)$.

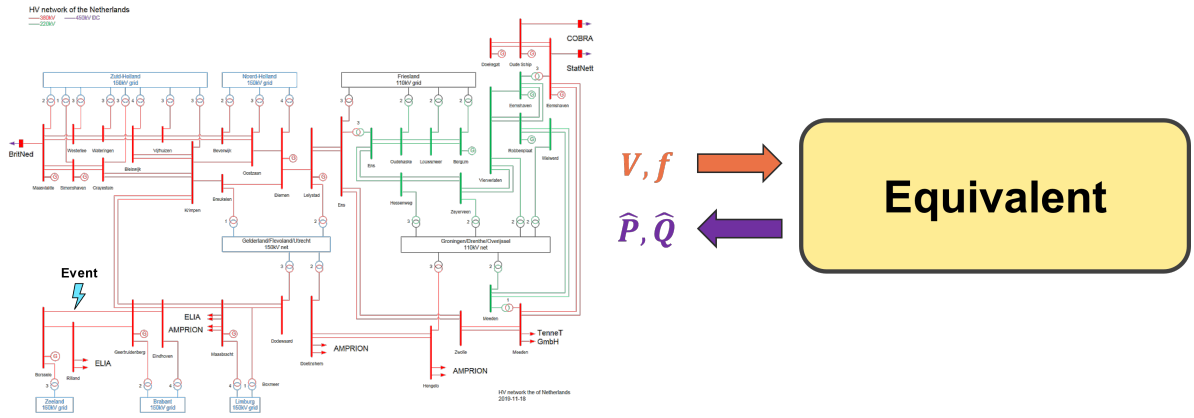


Figure 3.1: Diagram of the Dutch high voltage grid interfaced with an equivalent model (adapted from [84])

3.2. GTCNNs

GTCNNs are a type of GNN specialised in learning spatio-temporal relations. Being a type of GNN means that GTCNNs learn graph-structured data, which is useful for power systems as its network can be modelled by a graph. This section will discuss the working of product-graph-based GTCNNs. First, some fundamental graph notations are discussed. Next, the working of GCNs with filter banks, which form the sampling and propagation operators of the GNN, is described. Finally, the temporal aspect is included by using product graphs.

3.2.1. Graphs and graph signals

Graphs are characterised by nodes (entities) and edges (the connection between nodes). In Fig. 3.2, the circles with a number represent the nodes, and the black lines are the edges. This means that, in total, there are seven nodes and eight edges.

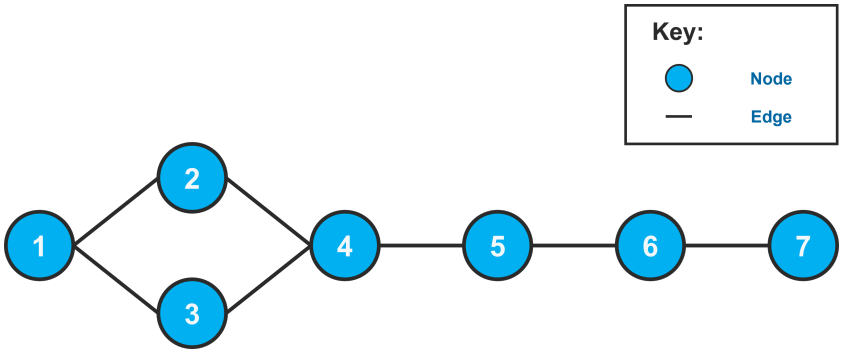


Figure 3.2: A simple graph.

Mathematically, a graph $\mathcal{G}$ is defined by $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes in the graph, and $\mathcal{E}$ is the set consisting of all node pairs connected by an edge. For the graph of Fig. 3.2, $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{E} = \{(1, 2), (1, 3), (2, 4), (3, 4), (4, 5), (5, 6), (6, 7)\}$. The graph in Fig. 3.2 is undirected since all the edges are lines instead of arrows. All nodes in $\mathcal{V}$ are connected to at least one other node, meaning the graph is connected. This also implies that for every node $\mathcal{V}_i$, at least one neighbour exists, implying a non-empty set $\mathcal{N}_i$. As an example, the neighbours of $\mathcal{V}_2$ are $\mathcal{N}_2 = \{1, 4\}$.

Two common ways to represent a graph $\mathcal{G}$ are the adjacency matrix $\mathbf{A}$ and the Laplacian $\mathbf{L}$, both of dimensions $\mathbb{R}^{N \times N}$ where N is the number of nodes $|\mathcal{V}|$. The adjacency and Laplacian matrices of the graph in Figure 3.2 are:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

The adjacency matrix $\mathbf{A}$ is intuitive to understand. Every row (or column) represents a node. For example, row i represents node $\mathcal{V}_i$. If node $\mathcal{V}_i$ is connected to $\mathcal{V}_j$ i.e. $(\mathcal{V}_i, \mathcal{V}_j) \in \mathcal{E}$, the entry A_{ij} will be 1. Because graph $\mathcal{G}$ is undirected, $\mathbf{A}$ is symmetric and the entry $A_{ji} = A_{ij} = 1$. If the graph is directed, $A_{ij} = -1$ will indicate an outward link from $\mathcal{V}_i$ to $\mathcal{V}_j$ while a 1 represents an incoming connection.

The Laplacian $\mathbf{L}$ is computed as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is a diagonal matrix whose entry D_{ii} is the degree of node $\mathcal{V}_i$ or, in other words, the number of neighbours in $\mathcal{N}_i$. This is equivalent to $\mathbf{D} = diag(\mathbf{A1})$ with $\mathbf{1}$ an all ones matrix.

The adjacency $\mathbf{A}$ and Laplacian $\mathbf{L}$ matrices shown before are unweighted. This means that all edges are considered equal, in this case, they all have the value 1 [68], [76]. It is possible to assign a different level of importance by using a different value. This value could represent a distance metric (Euclidean, Mehalanobis etc.) between nodes [66], [67], [70], [80], [81], the line impedance/admittance between two buses (nodes) [72], [73] or a correlation coefficient indicating the correlation between measurements [74]. Whether the adjacency $\mathbf{A}$ and Laplacian $\mathbf{L}$ matrices are weighted or not depends on the application and the available information.

The graph of Fig. 3.2 only shows which nodes are connected to each other. Additional features are needed to make full use of GNNs. Generally, a distinction is made between features associated with nodes (**node features**) and those related by edges (**edge features**).

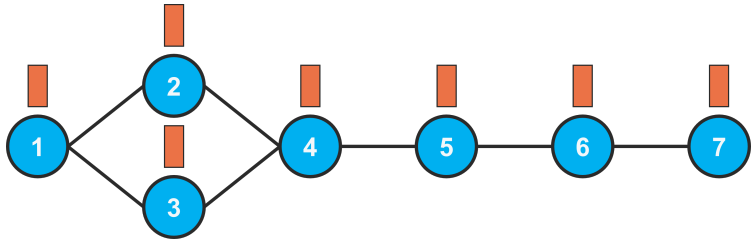


Figure 3.3: A simple graph with scalar node features.

In Fig. 3.3, all the nodes in the graph have a single feature V ( ). If the value of V at node $\mathcal{V}_i$ is given by v_i, we can collect the features into the vector $\mathbf{v} = [v_1, v_2, \dots, v_N]$.

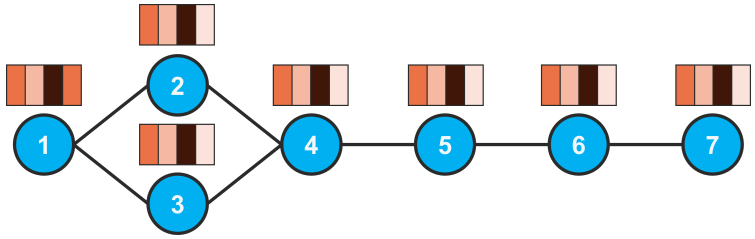


Figure 3.4: A simple graph with multiple scalar node features.

If every node has F features instead of a single scalar, Fig. 3.4 is obtained. If feature i is represented by the vector $\mathbf{v}_i$, then putting all the node features together will result in the matrix $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N] \in \mathbb{R}^{F \times N}$. A similar approach can be used to derive the edge features, but the resulting matrix will be of size $\mathbb{R}^{|\mathcal{E}| \times F}$.

In power systems, node features are quantities that can be observed at buses, such as the voltage magnitude and angle, frequency, and active and reactive power injections. On the other hand, edge features could represent values such as current and power flow through lines, losses in the lines, and loading percentages. Because the values are scalars, they would be a snapshot of the quantities at the time of observation.

3.2.2. Graph Convolutional Filters and GCNs

During training, CNNs learn filter (the kernel) parameters by sliding the kernel over grid-like data structures to capture spatial hierarchies and patterns. When the filter (or kernel) is treated as a fixed point of reference, it becomes evident that the input data is moved or shifted in various amounts relative to this stationary filter to compute the network's output. This operation, where the filter is applied across different regions of the input data to capture local patterns, is known as convolution. GCNs operate on this principle but instead use an operator $\mathbf{S}$ to achieve the shift [85]. Often, these shift operators are the adjacency $\mathbf{A}$ and Laplacian $\mathbf{L}$ matrices or their normalised versions $\mathbf{A}_n$ and $\mathbf{L}_n$. This approach works because the powers of the adjacency matrix $\mathbf{A}$ indicate the existence of k-hop walks[1] between nodes. For example, a nonzero entry in the squared adjacency matrix $\mathbf{A}^2_{ij}$ means that there exists (at least) one walk connecting $\mathcal{V}_i$ to $\mathcal{V}_j$, which traverses precisely two edges. Something similar is observed for the Laplacian.

Suppose that the graph signal is given by $\mathbf{v} = [1, 0, 0, 0, 0, 0, 0]$, an unit pulse originating at node 1. Shifting this signal by powers of the shift operator $\mathbf{S}$ results in an exchange of information from neighbours up to K-hops away. This is illustrated in Fig. 3.5[2].

[1] In graph theory, a walk is a sequence of nodes in a graph where consecutive nodes are connected by edges. This can be interpreted as the route taken from the source to the target node while passing other nodes. The length of the walk is given by the number of traversed edges. A k-hop walk means that exactly three edges were used to reach the target node.

[2] The exact contribution between a pair of nodes will depend on the shift operator and the graph's topology. For example, the multiplication $\mathbf{Av}$ using the adjacency matrix of this simple graph would result in an exchange only to nodes 2 and 3. Nevertheless, a higher power $\mathbf{S}^k$ will be equivalent to aggregating values from a larger neighbourhood.

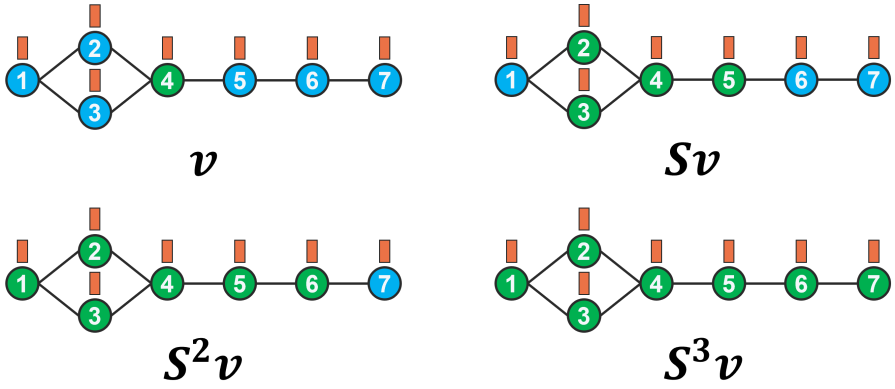


Figure 3.5: Effect of shift operator powers on the propagation of a scalar signal originating in node 1 (inspired by [86]).

The multiplication by powers of the shift operator in Fig. 3.5 is equivalent to shifting the grid-like data structures in the CNN. It is now possible to learn the importance of the various shifts by constructing a graph filter as in Fig. 3.6 yielding Equation 3.3.

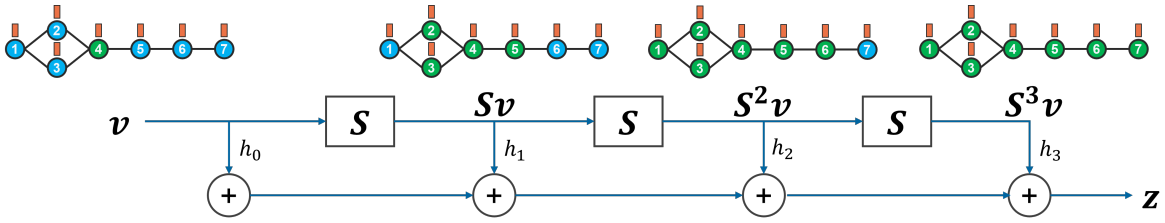


Figure 3.6: The graph convolutional filter outputs a signal consisting of a linear combination of graph-shifted versions of the signal (inspired by [86]).

$$\mathbf{z} = H(\mathbf{S})\mathbf{v} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{v} \tag{3.3}$$

The filter consists of a linear combination of different graph reception fields of increasing neighbourhoods scaled by the learnable (scalar) parameters h_k. The larger the value of K, the bigger the neighbourhoods will be considered by the graph filter. Note how, similar to the CNNs, the parameters h_k are shared between nodes and how the number of parameters does not depend on the dimension of $\mathbf{S}$. Moreover, the output $\mathbf{z}$ can be computed efficiently regardless of the matrix power $\mathbf{S}^k$. This is because the required computation is $\mathbf{S}^k\mathbf{v}$ instead of $\mathbf{S}^k$, which allows for recursive calculation as in Eq. 3.4. Also, $\mathbf{S}$ is usually super sparse, meaning that few actual multiplications are required.

$$\mathbf{v}^{(k)} = \mathbf{S}\mathbf{v}^{(k-1)} = \mathbf{S}(\mathbf{S}\mathbf{v}^{(k-2)}) = \mathbf{S}(\mathbf{S}(\mathbf{S} \dots (\mathbf{S}\mathbf{v}))) \tag{3.4}$$

The filter in Eq. 3.3 is linear, meaning it has limited expressive power [86]. However, remember that the nodes in an ANN also consist of a linear operation and a non-linearity is introduced by adding an activation function $g(\cdot)$. A similar idea can be applied to the linear graph filter to result in the graph perceptron:

$$\phi(\mathbf{v}) = \sigma\left(\sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{v}\right) \tag{3.5}$$

In Eq. 3.5, $\sigma(\cdot)$ represents the element-wise non-linear function, creating a more powerful structure than the linear graph filter. It is possible to stack multiple graph perceptrons together to create a GCN, as seen in Fig. 3.7.

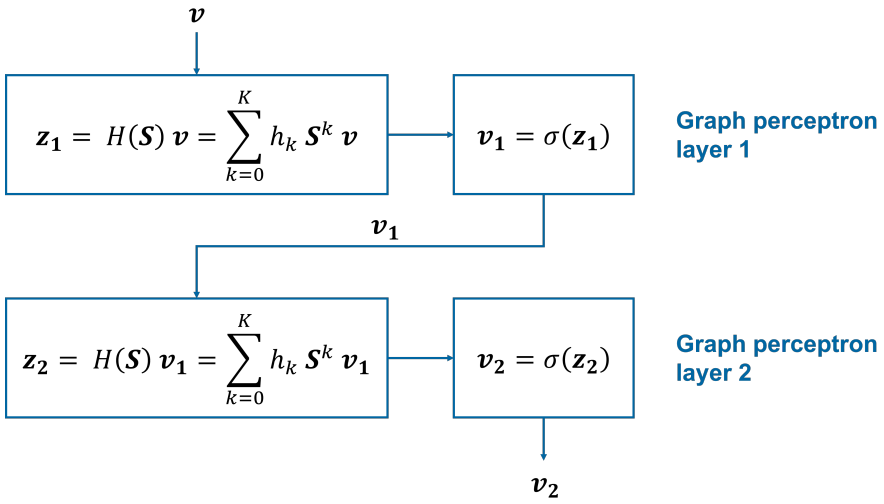


Figure 3.7: A multi-layer graph perceptron (adapted from [86]).

The advantage of this structure is that the number of parameters of the linear graph filter is independent of the dimension of the input $\mathbf{v}$ (unlike the fully connected ANN) but instead depends on the graph filter order K. Moreover, although the ANN should give a smaller cost (or higher accuracy) when training the network as the GNN is a particular case of an ANN, the GNN will pull head since it generalises better to unseen signals and graphs [86].

The graph perceptron of Eq. 3.5 consists of a single set of parameters and, therefore, represents a single filter. The output $\phi(\mathbf{v})$ could be considered the hidden feature of this graph perceptron. Usually, it is desired to learn multiple hidden features to have more expressive power (and possibly better performance). CNNs, in fact, also learn multiple filters to extract different information from the data. The same principle can be applied to the graph perceptron using filter banks instead of a single graph convolutional filter:

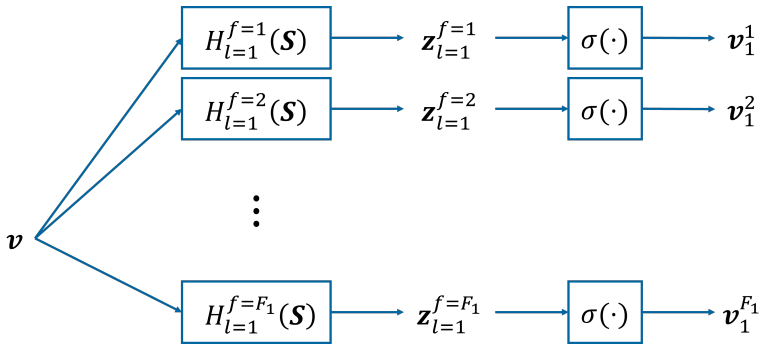


Figure 3.8: A filter bank of F_1 filters is applied on a single-featured input signal $\mathbf{v}$ (adapted from [86]).

In Fig. 3.8, F_1 graph convolutional filters of Eq. 3.3 are applied to the input $\mathbf{v}$ consisting of only one feature:

$$\boldsymbol{z}_1^f = H_1^f(\mathbf{S})\mathbf{v} = \sum_{k=0}^{K} h_{1k}^f \mathbf{S}^k \mathbf{v} \tag{3.6}$$

In Eq. 3.6, f represents the f-th (hidden) feature obtained by the corresponding graph convolutional filter and l the layer number. Passing the input $\mathbf{v}$ through the filter bank results in $\mathbf{Z}_1 = \left[\boldsymbol{z}_1^1, \boldsymbol{z}_1^2, \ldots, \boldsymbol{z}_1^f, \ldots, \boldsymbol{z}_1^{F_1}\right] \in \mathbb{R}^{N \times F_1}$. The resulting outputs $\mathbf{Z}_1$ are passed through the non-linear function $\sigma(\cdot)$, resulting in $\mathbf{V}_1 \in \mathbb{R}^{N \times F_1}$. Note how using F_1 filter banks allows for controlling the number of features going out of each graph perceptron with filter banks layer.

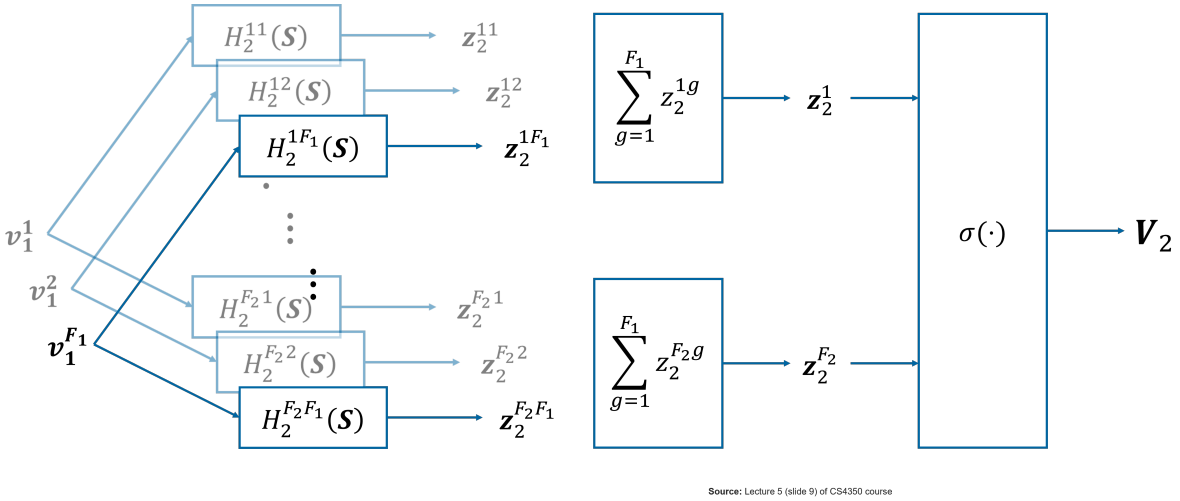


Figure 3.9: A GCN consisting of F_2 graph filters is applied on a multi-featured input signal $\mathbf{V}_1$ (adapted from [86]).

The input going into the second layer ($l = 2$) is given by $\mathbf{V}_1 = \left[\mathbf{v}_1^1, \mathbf{v}_1^2, \dots, \mathbf{v}_1^g, \dots, \mathbf{v}_1^{F_1}\right]$. The processing in the second layer will be similar, except that there are F_2 filter banks, and the input is now a matrix $\mathbf{V}_1$ instead of a vector $\mathbf{v}_1$, as seen in Fig. 3.9[3]. However, it is possible to apply Eq. 3.6 on each column of $\mathbf{V}_1$ since column $\mathbf{v}_1^g$ is identical in shape to the single-featured input signal:

$$\mathbf{z}_2^{fg} = H_2^{fg}(\mathbf{S})\mathbf{v}_1^g = \sum_{k=0}^{K} h_{2k}^{fg}\mathbf{S}^k\mathbf{v}_1^g \tag{3.7}$$

In Eq. 3.7, g stands for the input features, and f stands for the (hidden) features of the current layer. Equation 3.7 is repeated on all columns of $\mathbf{V}_1$, resulting in $\mathbf{Z}_2^f = \left[\mathbf{z}_2^{f1}, \mathbf{z}_2^{f2}, \dots, \mathbf{z}_2^{fF_1}\right] \in \mathbb{R}^{N \times F_1}$ for feature f of the second GCN layer. Because there are F_2 filter banks in the second layer, the output of this layer would become three-dimensional. Therefore, the columns of $\mathbf{Z}_2^f$ are summed to give $\mathbf{z}_2^f \in \mathbb{R}^N$. Collecting the outputs of all F_2 filter banks gives $\mathbf{Z}_2 = \left[\mathbf{z}_2^1, \mathbf{z}_2^2, \dots, \mathbf{z}_1^f, \dots, \mathbf{z}_2^{F_2}\right] \in \mathbb{R}^{N \times F_2}$. This process can be written as:

$$\mathbf{Z}_l = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{V}_{l-1} \mathbf{H}_{lk} \tag{3.8}$$

In Eq. 3.8, l represents the layer number. Moreover, $\mathbf{H}_{lk}$ is a matrix containing all $F_{l-1} \times F_l$ filter parameters of the k[th]-hop neighbourhood. This means that for every incoming feature, a new set of F_l linear graph convolutional filters of Eq. 3.3 is applied, and thus, independent parameters are learnt per feature. Different features convey different information, so different parameters are needed to include the information effectively. Again, the non-linear function $\sigma(\cdot)$ is applied to the point-wise non-linear function to obtain $\mathbf{V}_1$.

The GCN with filter banks described here keeps the advantages of the linear graph convolutional filter of Eq. 3.3. This means that the number of trainable parameters of the GCN with filter banks does not depend on the size of the shift operator $\mathbf{S}$ (and therefore the number of nodes) but instead relies on the number of features at the input and in the hidden layers and the value of K [86]. The computational complexity also depends on the number of edges in $\mathbf{S}$.

3.2.3. Product Graphs

The GCN created from the graph convolutional filter does not consider time. This makes GCNs not useful for equivalent models for transient stability. It would be possible to consider time by creating

[3]The multi-featured input signal $\mathbf{V}_1$ was obtained by applying F_1 filters on a single-featured input in the previous layer. However, it is also possible that multiple features are considered, starting with an already multi-featured input. Regardless of the origin of the multiple features of $\mathbf{V}_1$, Eq. 3.8 can be applied, resulting in a GCN that can work with multiple features.

separate features for different time instants. For example, if $\mathbf{v}(t)$ contains the values of the voltage at time t at all N nodes, it would be possible to create a matrix $\mathbf{V} = [\mathbf{v}(0), \mathbf{v}(1), \dots, \mathbf{v}(t), \dots, \mathbf{v}(T)] \in \mathbb{R}^{N \times T}$ where the rows would give the voltage $v(t)$ at node i from $t = 0$ until $t = T$. However, the GCN layer of Eq. 3.8, and any GNNs and ML models that do not explicitly account for time, will learn weights for the input features without recognising their temporal relationships. This means these models treat each feature independently, disregarding the sequential nature of the data. Since the measurements are a temporal sequence, models that include this property explicitly are expected to perform better. This is reflected by the superiority of RNNs on regression tasks that predict temporal sequences.

Product graphs are an alternative approach to recurrent models and merge the temporal and spatial dependencies into one graph [82]. Suppose a single feature on the simple graph is time evolving, e.g., the voltage $v(t)$. Sampling this feature at time t for all nodes gives $\mathbf{v}(t)$ at each sample $t \in [0, 1, \dots, T]$. It is possible to replicate the graph $\mathcal{G}$ for every time instance t, yielding $\mathcal{G}(t)$. Each replica $\mathcal{G}(t)$ contains a snapshot of the values $\mathbf{v}(t)$:

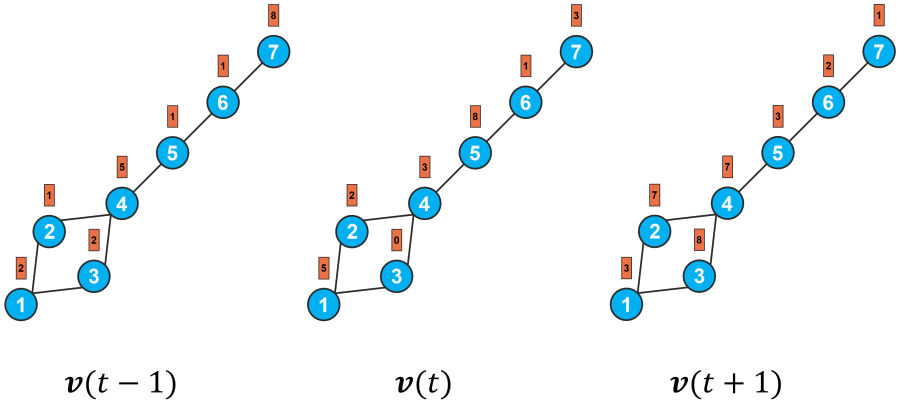


Figure 3.10: Three snapshots of a time-dependent feature were taken at different times, resulting in three graph replicas.

The different replicas in Fig. 3.10 are now standalone. However, the values of a single node are connected in time by the temporal graph in Fig. 3.11. This is known as the temporal graph $\mathcal{G}_T$, consisting of the nodes $\mathcal{V}_T$ and the edges $\mathcal{E}_T$. Note how the edges are now directed, as indicated by the arrow.

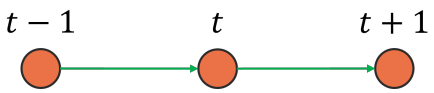


Figure 3.11: The temporal graph of feature v at node $\mathcal{V}_i$.

The graph replicas of Fig. 3.10 should be connected to include the dependencies given by the temporal graph of Fig. 3.11. This is done by combining the time graph $\mathcal{G}_T$ with the spatial graph $\mathcal{G}$ according to some rule $\diamond$ into a bigger product graph $\mathcal{G}_\diamond$:

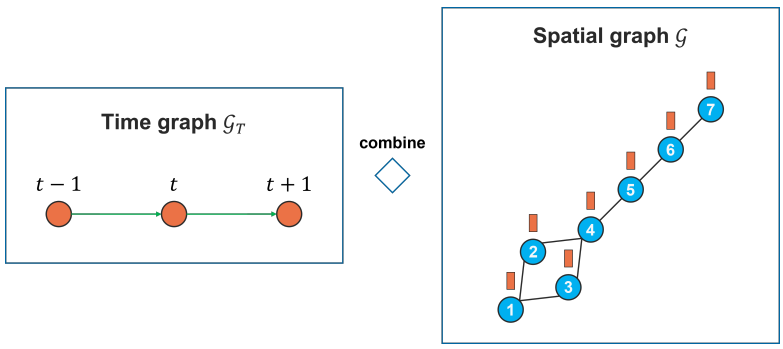


Figure 3.12: The product graph is created by combining the temporal and spatial graphs into a larger spatiotemporal graph.

There are different combination rules $\diamond$ which all lead to different product graphs $\mathcal{G}_\diamond$ [82]. Three types will be discussed: the Cartesian, Kronecker and strong product graphs. They will be compared on three points: spatial dependencies at current time t, temporal dependency of nodes on themselves and temporal dependencies from neighbours.

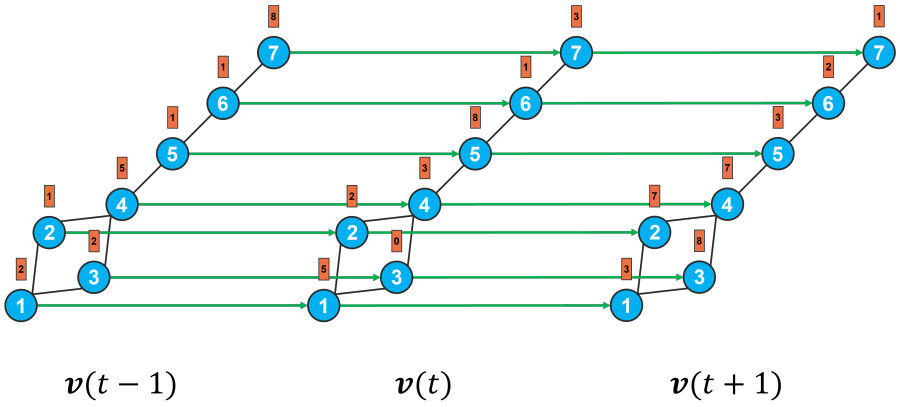


Figure 3.13: The Cartesian product graph is created by considering the spatial relations between nodes at the current time instance and also connecting a node to itself at the previous graph time replica.

The Cartesian product graph (Fig. 3.13) considers the spatial dependencies at the current time and the temporal dependency of nodes on themselves. This type is also the most interpretable since it provides a clear and separate representation of the spatial and temporal aspects. The Cartesian product graph shift operator is given by:

$$\mathbf{S}_\times = \mathbf{S}_T \otimes \mathbf{I}_{|\mathcal{V}|} + \mathbf{I}_{|\mathcal{V}_T|} \otimes \mathbf{S} = \begin{bmatrix} \mathbf{S} & \mathbf{0} & \mathbf{0} \\ \mathbf{I}_{|\mathcal{V}|} & \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{|\mathcal{V}|} & \mathbf{S} \end{bmatrix} \in \mathbb{R}^{NT \times NT} \tag{3.9}$$

In Eq. 3.9, $\mathbf{S}$ is the shift operator (e.g. the adjacency matrix $\mathbf{A}$), $\mathbf{S}_T = \left[\begin{smallmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{smallmatrix}\right]$ is a matrix containing the temporal coupling between different graph replicas, $\mathbf{I}_{|\mathcal{V}|} \in \mathbb{R}^{N \times N}$ and $\mathbf{I}_{|\mathcal{V}_T|} \in \mathbb{R}^{T \times T}$ are identity matrices, $\mathbf{0} \in \mathbb{R}^{N \times N}$ is an all-zeroes matrix and $\otimes$ is the Kronecker product between two matrices.

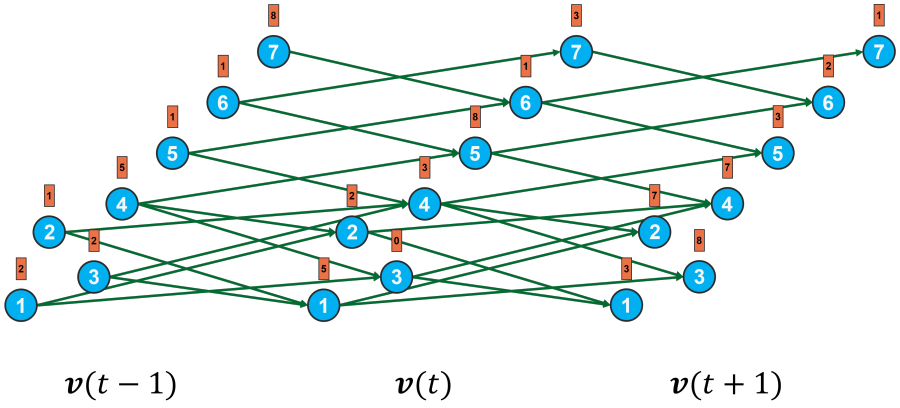


Figure 3.14: The Kronecker product graph is created by connecting a node at the current time to its neighbours at the previous time replica.

The Kronecker product graph (Fig. 3.14) considers only the temporal dependencies from neighbours. This type of product graph would be useful in a situation where the value node $\mathcal{V}_i$ at time t depends only on the previous values of its neighbours. The Kronecker product shift operator is given by:

$$\mathbf{S}_\otimes = \mathbf{S}_T \otimes \mathbf{S} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{S} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{NT \times NT} \tag{3.10}$$

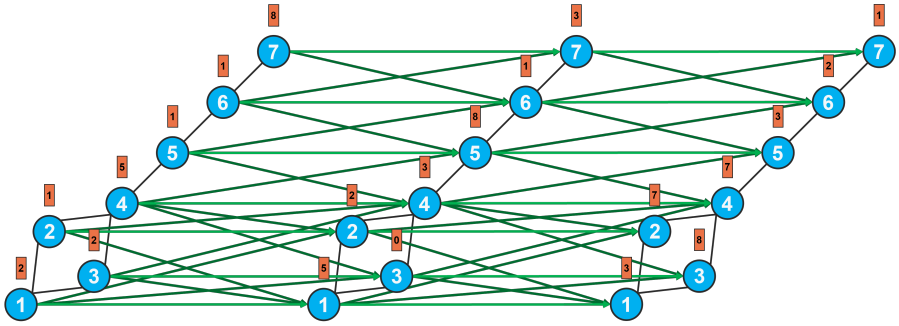


Figure 3.15: The strong product graph is created by combining the Kronecker and Cartesian product graphs.

The strong product graph (Fig. 3.15) is the sum of the Cartesian and Kronecker product graphs and therefore considers spatial dependencies at current time t, temporal dependency of nodes on themselves and temporal dependencies from neighbours. This is the most powerful network since it considers all dependencies, but it is also the hardest to train since it has many more interactions than either the Cartesian or Kronecker product graphs. The strong shift operator is obtained by summing Eq. 3.9 and Eq. 3.10:

$$\mathbf{S}_{\boxtimes} = \mathbf{S}_T \otimes \mathbf{I}_{|\mathcal{V}|} + \mathbf{I}_{|\mathcal{V}_T|} \otimes \mathbf{S} + \mathbf{S}_T \otimes \mathbf{S} = \begin{bmatrix} \mathbf{S} & \mathbf{0} & \mathbf{0} \\ \mathbf{S} + \mathbf{I}_{|\mathcal{V}|} & \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} + \mathbf{I}_{|\mathcal{V}|} & \mathbf{S} \end{bmatrix} \in \mathbb{R}^{NT \times NT} \tag{3.11}$$

It is possible to write the shift operators $\mathbf{S}_{\times}, \mathbf{S}_{\otimes}$ and $\mathbf{S}_{\boxtimes}$ of the Cartesian, Kronecker and strong product graphs respectively as a general shift operator $\mathbf{S}_{\diamond}$:

$$\mathbf{S}_{\diamond} = s_{00}(\mathbf{I}_{|\mathcal{V}_T|} \otimes \mathbf{I}_{|\mathcal{V}|}) + s_{10}(\mathbf{S}_T \otimes \mathbf{I}_{|\mathcal{V}|}) + s_{01}(\mathbf{I}_{|\mathcal{V}_T|} \otimes \mathbf{S}) + s_{11}(\mathbf{S}_T \otimes \mathbf{S}) \tag{3.12}$$

In Eq. 3.12, the scalars s_{00}, s_{10}, s_{01} and s_{11} control which product graph is created. Setting s_{10} and s_{01} to 1 gives $\mathbf{S}_{\diamond} = S_{\times}$, $s_{11} = 1$ gives $\mathbf{S}_{\diamond} = \mathbf{S}_{\otimes}$, and s_{10}, s_{01} and s_{11} equal to 1 gives $\mathbf{S}_{\diamond} = \mathbf{S}_{\boxtimes}$. s_{00} can be used to add self-loops, meaning that a node's current feature values will also be considered.

Considering spatiotemporal dependencies with product graphs has one major advantage: they can be used with existing GNN methods. This is because the temporal element is embedded with the spatial element in the new product graph $\mathcal{G}_{\diamond}$; the GNNs see a bigger shift operator $\mathbf{S}_{\diamond}$. This means that Eq. 3.8 can be used to create a Graph-Time Convolutional Neural Network (GTCNN):

$$\mathbf{Z}_{\diamond,l} = \sum_{k=0}^{K} \mathbf{S}_{\diamond}^{k} \mathbf{V}_{\diamond,l-1} \mathbf{H}_{lk} \tag{3.13}$$

In Eq. 3.13, $\mathbf{V}_{\diamond,l-1}$ is given by concatenating $\mathbf{V}_{l-1}(t)$ at time steps $t \in [0, 1, \ldots, T]$. This results in:

$$\mathbf{V}_{\diamond,l-1} = \begin{bmatrix} \mathbf{V}_{l-1}(0) \\ \vdots \\ \mathbf{V}_{l-1}(t) \\ \vdots \\ \mathbf{V}_{l-1}(T) \end{bmatrix} \in \mathbb{R}^{NT \times F} \tag{3.14}$$

Applying the non-linear function gives:

$$\mathbf{V}_{\diamond,l} = \sigma\left(\sum_{k=0}^{K} \mathbf{S}_{\diamond}^{k} \mathbf{V}_{\diamond,l-1} \mathbf{H}_{lk} \right) \in \mathbb{R}^{NT \times F_l} \tag{3.15}$$

The number of parameters of the GTCNN depends only on K, the number of features at the input and hidden layers and the number of layers, plus the parameters of the non-linear activation functions $\sigma(\cdot)$.

This independence on the size of $\mathbf{S}_\diamond$ is essential since it allows the GTCNN to remain trainable even when there are many nodes, a characteristic of distribution networks. The computational complexity, however, scales with the number of edges $|\mathcal{E}_\diamond|$ in $\mathbf{S}_\diamond$.

3.3. GTCNN-based equivalent model

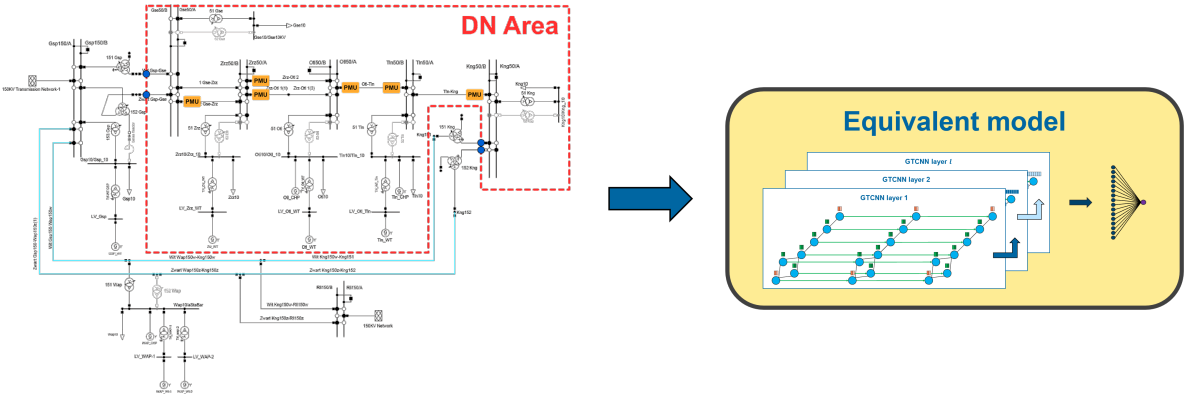


Figure 3.16: The DN is represented by a GTCNN-based equivalent model.

The GTCNN based on product graphs is chosen for the implementation of the equivalent network. This preference has several reasons. First, it is a type of GNN, which, as a type of NN, means that it can approximate any arbitrary function. Also, power systems are a network. This means that they can be modelled by graphs, making GNNs a specialised NN structure suitable for efficiently modelling power systems while keeping the model interpretable. Finally, the number of parameters in the GTCNN depends neither on the number of nodes nor the number of time steps. This property is useful as the number of nodes in DN networks is large, meaning the model's training complexity does not grow with bigger networks. In contrast, RNNs model different nodes as separate input features, which makes the RNN harder to train as it has more parameters.

The implementation of the GTCNN for equivalent modelling is discussed in the remainder of this section. First, the graph representation of the DN is addressed in section 3.3.1. Next, the GTCNN structure is presented in section 3.3.2. Section 3.3.3 discusses the training of the GTCNN-based equivalent model. Finally, section 3.3.4 shows the proposed workflow of training and sharing the equivalent between DSO and TSO.

3.3.1. Graph representation of the DN

The GTCNN needs a graph representation of the DN to work. There are two options for the nodes and the edges. The first option is to represent every bus as a node and every line, cable or transformer as an edge. This approach is more interpretable as the graph explicitly represents the network elements. However, this requires having measurements available at all buses in the DN, which is hard under the limited observability of DNs. The second option is to model measurement locations as nodes. The edges will then indicate the connection between the different measurement locations. The advantage of this approach is that it will use the already available measurements. However, this makes the edge definition more abstract.

Having the nodes $\mathcal{V}_{GTCNN}$ represent the measurement locations is more feasible in real applications because the information is already available at all the nodes. Although the PMUs are close to one system bus, they measure on the transmission lines or cables (see Figure 3.17a). Therefore, the nodes $\mathcal{V}_{GTCNN}$ represent the lines on which measurements are available (i.e., the measurement points), while the edges $\mathcal{E}_{GTCNN}$ indicate how two PMUs are electrically connected.

To derive the GTCNN graph $\mathcal{G}_{GTCNN}$, the line graph of the power system graph will be adapted. Suppose that $\mathcal{V}$ gives all the buses in the DN, and $\mathcal{E}$ contains all the connections between two buses (see Fig. 3.17b). The line graph $L(\mathcal{G})$ replaces every edge e_k of graph $\mathcal{G}$ with a node. Moreover, if two edges e_{k1} and e_{k2} share the same node in graph $\mathcal{G}$, that common node will be an edge in $L(\mathcal{G})$. The nodes $\mathcal{V}_{L(\mathcal{G})}$

and edges $\mathcal{E}_{L(\mathcal{G})}$ are given by Eq. 3.16, and the resulting line graph $L(\mathcal{G})$ can be seen in Figure 3.17c.

$$\mathcal{V}_{L(\mathcal{G})} = \{e_k | \ \forall e_k \in \mathcal{E}\} \tag{3.16a}$$

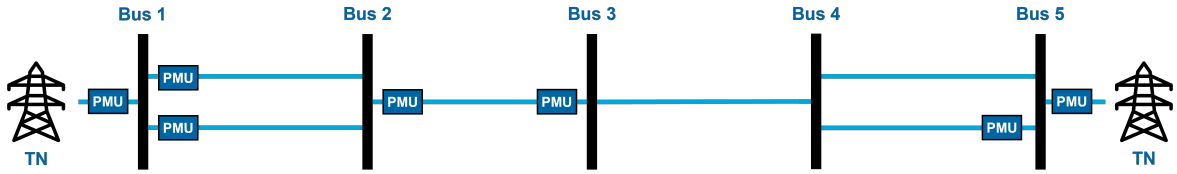
$$\mathcal{E}_{L(\mathcal{G})} = \{(e_{k1}, e_{k2}) | \text{ if } e_{k1} \text{ and } e_{k2} \text{ share a common node in } \mathcal{G}\} \tag{3.16b}$$

The line graph can be used to construct the GTCNN graph $\mathcal{G}_{GTCNN} = (\mathcal{V}_{GTCNN}, \mathcal{E}_{GTCNN})$ that relates the measurements, but it requires three adaptations. First, only the observable elements $\mathcal{G}_{observable} = (\mathcal{V}_{observable}, \mathcal{E}_{observable})$ should be used to find the nodes $\mathcal{V}_{L(\mathcal{G})}$ in Eq. 3.16a. Using the observable graph ensures that measurements are available at all nodes. As there is no PMU on one of the double lines between buses 4 and 5, the node in $L(\mathcal{G})$ that corresponds to that edge should be removed. Second, there may be two PMUs on the same line. There will be a slight difference in the measurements in the power system because of losses in the line. To keep both measurement locations, a fictitious midpoint node is added to graph $\mathcal{G}$ on every edge with a PMU at every end, giving graph $\mathcal{G}_{modified}$ This corresponds to the edge between bus 2 and 3 in Fig. 3.17a, and the fictitious node is shown in orange in Fig. 3.17d. These two modifications result in the graph $L(\mathcal{G}_{modified})$, whose nodes $\mathcal{V}_{GTCNN}$ and edges $\mathcal{E}_{L(\mathcal{G}_{modified})}$ are given by Equations 3.17a and 3.17b. The third modification prevents creating multiple disjoint subgraphs when computing the modified line graph $\mathcal{G}_{GTCNN}$. This issue occurs when there is no PMU on any line between two buses, as is the case between buses 3 and 4 (see Fig. 3.17a). After obtaining the modified line graph $L(\mathcal{G}_{modified})$, an edge is added between two nodes if the PMUs are electrically connected[4] in graph $\mathcal{G}$. The graph $\mathcal{G}_{GTCNN} = (\mathcal{V}_{GTCNN}, \mathcal{E}_{GTCNN})$ can be seen in Figure 3.17e.

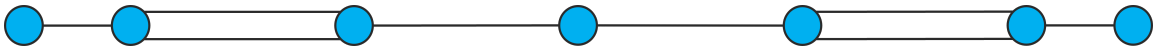
$$\mathcal{V}_{GTCNN} = \{e_k | \ \forall e_k \in \mathcal{E}_{observable}, \mathcal{E}_{observable} \subseteq \mathcal{E}_{modified}\} \tag{3.17a}$$

$$\mathcal{E}_{L(\mathcal{G})} = \{(e_{k1}, e_{k2}) | \text{ if } e_{k1} \text{ and } e_{k2} \text{ share a common node in } \mathcal{G}_{modified}\} \tag{3.17b}$$

$$\mathcal{E}_{GTCNN} = \{(e_{k1}, e_{k2}) | (e_{k1}, e_{k2}) \in \mathcal{E}_{L(\mathcal{G})} \text{ or } e_{k1} \text{ and } e_{k2} \text{ are electrically connected by } \mathcal{G}\} \tag{3.17c}$$

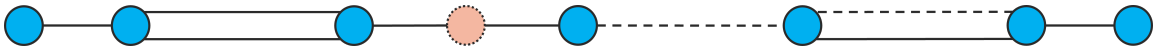


(a) A radial DN connected to the TN at two PCCs. Moreover, various PMU measurement locations are indicated.

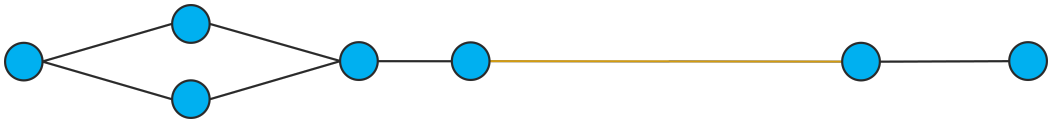


(b) Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of the radial DN.

(c) Line graph $L(\mathcal{G})$ of the radial DN.



(d) The graph $\mathcal{G}_{modified}$ of the radial DN modified according to the two rules. Edges that are not observable (- - -) are not considered when computing the line graph $L(\mathcal{G}_{modified})$. Also, a fictitious midpoint node ( ) is added on edges in $\mathcal{G}$ with two PMUs. This gives two nodes in $\mathcal{G}_{GTCNN}$, allowing the mapping of both PMUs to a separate node.



(e) Modified line graph $\mathcal{G}_{GTCNN} = (\mathcal{V}_{GTCNN}, \mathcal{E}_{GTCNN})$ used by the GTCNN. The line ( ) was added by the third rule to prevent the creation of a two disjoint graphs from $\mathcal{G}_{modified}$ as there is no PMU between buses 3 and 4.

Figure 3.17: An example DN and overview of the different possible graphs

[4] In this context, electrically connected means that the power can flow from one PMU to another without encountering a different PMU.

To include the temporal element, the graph $\mathcal{G}_{GTCNN}$ is transformed into a product graph by including the time graph using the Cartesian product graph. The stability and functioning of power systems depend on having instantaneous power balance. Hence, it is crucial to consider the exchange between different points at the same time instance. The Kronecker product graph does not contain any spatial dependencies at the current time instant, as seen in Eq. 3.11, and is therefore not an option for the graph $\mathcal{G}_{GTCNN}$. This leaves the Cartesian and strong product graphs as candidates. Both types include the temporal dependency of nodes on themselves, which is a desired property as the current value of any temporal sequence will depend on its previous values. The difference between the Cartesian and strong product graphs is the inclusion of the temporal dependencies from neighbours in the strong product graph. Although these temporal dependencies might exist in the measurement data, no clear link exists in the physical world. Hence, the Cartesian product graph is chosen as it yields the most realistic representation of the network. Note that this omission does not necessarily decrease the performance w.r.t. the strong product graph, as the Cartesian product graph will have fewer parameters to learn. Fewer parameters can prevent overfitting and thus improve the generalisation performance.

3.3.2. GTCNN structure

The inputs for the equivalent model are chosen based on the available information. TSOs (sometimes even DSOs) have limited information about a DN, making the modelling much harder. The first two inputs are the controlled quantities of the power system, the voltage V and frequency f, as a disturbance in one of these quantities will trigger a dynamic response from the DN. [28], [31], [34] include both the voltage V and frequency f while newer works [29], [38], [41], [42], [44] only include the voltage V. In this thesis, only the system voltage will be considered, namely the voltage magnitude $|V|$ and angle θ. Therefore, the developed equivalent will be accurate for voltage disturbances.

The equivalent model should also perform well under different operating conditions and hidden topological changes. A change in the operating conditions of various components in the network leads to a different amount of active and reactive power being produced or consumed. This will affect the produced and consumed active and reactive power at the measurement nodes in the system. Therefore, the initial injections P_0 and Q_0 are also chosen as inputs to the GTCNN. These two quantities can be found online on the ENTSO-E transparency platform [87], measured at the PCC, or shared by the DSO. The considered hidden topological changes affect the voltage levels in the system. The flow of reactive power is governed by the voltage difference between buses, and a more optimal distribution of voltages can reduce the active and reactive power losses in the network [50]. This means that the hidden topological changes can also be observed in the initial injections P_0 and Q_0.

The four inputs $|V|, \theta, P_0$ and Q_0 will be used to learn a mapping function $f(\cdot)$ that predicts the outputs P and Q, as seen in Eq. 3.2. The GTCNN is chosen as the mapping function since it models the spatial relations of the power system network and can also include the temporal aspect without requiring the recursive computation of LSTMs. To use the GCN with filter banks of Eq. 3.15, the inputs $|V|, \theta, P_0$ and Q_0 should be mapped to the graph $\mathcal{G}_{GTCNN}$. This is done as seen in Figure 3.18. A distinction should be made between the PCC nodes (🟠), which are connected to the TN, and the internal nodes (🟢). The TSO can observe the PCC nodes, while the internal nodes cannot be observed. This means the voltage magnitude $|V|$ and angle θ can only be measured at the PCC nodes. Moreover, the active P and reactive power Q will also be observed here, so this is where these two quantities will be estimated ($\hat{P}$ and $\hat{Q}$). Note that these four quantities are dynamic, meaning they vary throughout a measurement. The initial setpoints P_0 and Q_0 are inputs to the internal nodes and represent the active and reactive power injections at the buses in the system. These two quantities are static, meaning their value does not change with time. Using this configuration, the TSO can input the voltage obtained from their simulations and get the dynamic response of the equivalent model at the connection point with the TN (the PCCs):

$$\hat{P}_{PCCi} = f_{GTCNN}(V_{PCCi}(t), \theta_{PCCi}(t), P_{0,j}, Q_{0,j}, \mathcal{G}_{GTCNN}, W_P),\ \forall \mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}},\ \forall \mathcal{V}_j \in \Omega_{\mathcal{V}_{internal}} \tag{3.18a}$$

$$\hat{Q}_{PCCi} = f_{GTCNN}(V_{PCCi}(t), \theta_{PCCi}(t), P_{0,j}, Q_{0,j}, \mathcal{G}_{GTCNN}, W_Q),\ \forall \mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}},\ \forall \mathcal{V}_j \in \Omega_{\mathcal{V}_{internal}} \tag{3.18b}$$

In Eq. 3.18, $f_{GTCNN}(\cdot)$ is the mapping function of the GTCNN network, $V_{PCCi}(t)$ and $\theta_{PCCi}(t)$ are the time-varying voltage magnitude $|V|$ and angle θ at the PCC node $\mathcal{V}_i$, $P_{0,j}$ and $Q_{0,j}$ are the initial active and reactive power injections at the internal node $\mathcal{V}_j$, $\mathcal{G}_{GTCNN}$ is the modified line graph of the ADN and W_P and W_Q are the weights of the GTCNN network for the active and reactive power models respectively. $\Omega_{\mathcal{V}_{PCC}}$ and $\Omega_{\mathcal{V}_{internal}}$ are two sets containing the PCC and internal nodes, respectively, and are given by $\Omega_{\mathcal{V}_{PCC}} = \{\mathcal{V}_i|$ if $\mathcal{V}_i$ is a PCC node$\}$ $\Omega_{\mathcal{V}_{internal}} = \{\mathcal{V}_j|$if $\mathcal{V}_j$ is an internal node$\}$ and $\Omega_{\mathcal{V}_{PCC}}, \Omega_{\mathcal{V}_{internal}} \subset \mathcal{V}_{GTCNN}$.

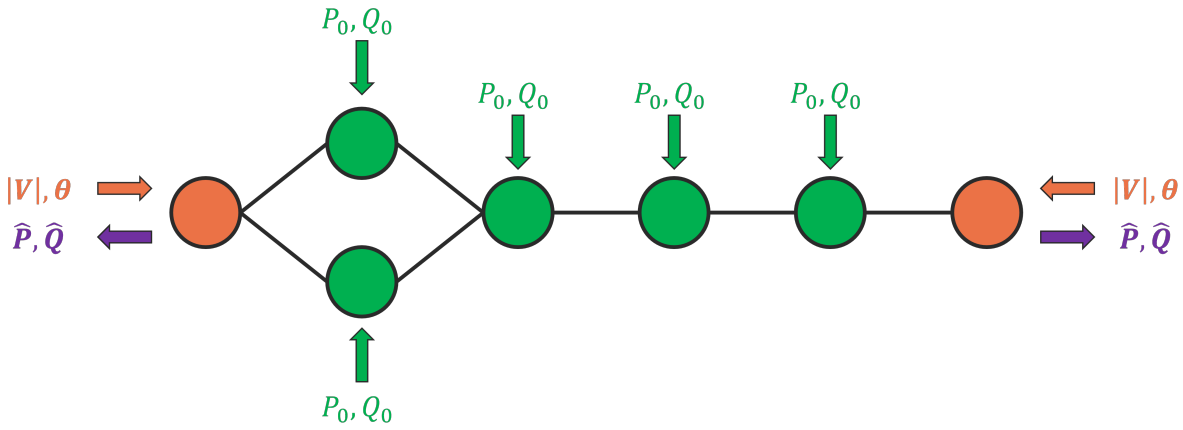


Figure 3.18: The inputs of the equivalent applied to the graph $\mathcal{G}_{GTCNN}$. The nodes at the PCC (●) receive the dynamic inputs $|V|$ and θ and produce the outputs $\hat{P}$ and $\hat{Q}$. The internal nodes (●) only have the static initial setpoints P_0 and Q_0.

The graph $\mathcal{G}_{GTCNN}$ of Fig. 3.18 is heterogeneous because there are two types of nodes, internal and external. The GTCNN of section 3.2 is homogenous, meaning the GNN expects nodes of the same type. Because the number of inputs per node is the same (V and θ for the PCC nodes and P_0 and Q_0 for the internal nodes), it is possible to use the GTCNN without modifications. The input matrix $\mathbf{V}_0$ at t becomes:

$$\mathbf{V}_0(t) = [\mathbf{v}_0^1(t), \mathbf{v}_0^2(t)] = \begin{bmatrix} V_{PCC0}(t) & \theta_{PCC0}(t) \\ \vdots & \vdots \\ V_{PCCi}(t) & \theta_{PCCi}(t) \\ P_{0,0} & Q_{0,0} \\ \vdots & \vdots \\ P_{0,j} & Q_{0,j} \end{bmatrix} \in \mathbb{R}^{N\times 2} \tag{3.19}$$

In Eq. 3.19, $\mathbf{v}_0^1(t)$ is the input vector at the 0-th layer (input) and the first node feature, which contains the voltage magnitude measurements $V_{PCCi}(t)$ at the PCC nodes and the initial active power injections $P_{0,j}$ at the internal nodes at time t. Similarly, $\mathbf{v}_0^2(0)$ has the second node feature, the voltage angle $\theta_{PCCi}(t)$ at the PCC nodes and the initial reactive power injections Q_0, j at the internal nodes at time t. The downside of this approach is that the sampling and propagation function is shared between the different node types, losing some of the interpretability and expressive power. However, the GTCNN can learn multiple hidden features that can improve performance.

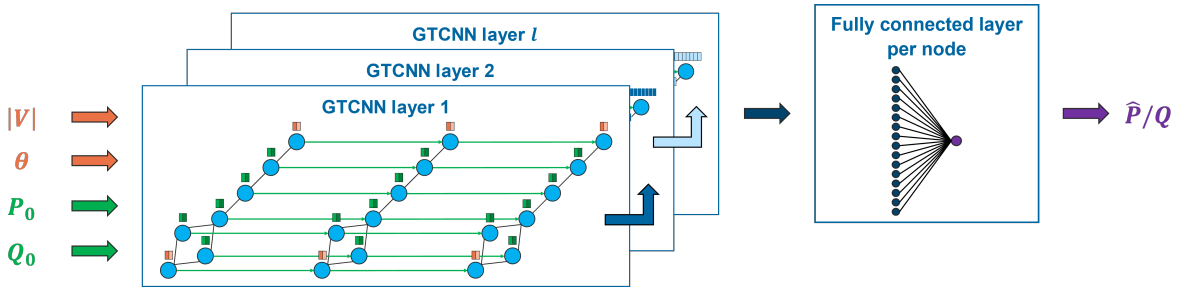


Figure 3.19: Overview of the GTCNN structure.

The graph $\mathcal{G}_{GTCNN}$ is transformed into the spatiotemporal graph by using the Cartesian product of the shift operator of $\mathcal{G}_{GTCNN}$ and the temporal coupling graph $\mathcal{G}_T$. This gives a single GTCNN layer, as depicted in Fig. 3.19.

The input at the first GTCNN layer will be the matrix $\mathbf{V}_{\diamond,0} \in \mathbb{R}^{NT\times F}$, where N is the number of nodes in graph $\mathcal{G}_{GTCNN}$, T is the measurement duration, and F is the number of input features per node[5]. In this case, $F = 2$, where the PCC nodes have as inputs $|V|$ and θ, and the internal nodes have P_0 and Q_0:

$$\mathbf{V}_{\diamond,0}(t) = \begin{bmatrix} \mathbf{V}_0(t-T) \\ \vdots \\ \mathbf{V}_0(t-2) \\ \mathbf{V}_0(t-1) \\ \mathbf{V}_0(t) \end{bmatrix} = \begin{bmatrix} V_{PCC0}(t-T) & \theta_{PCC0}(t-T) \\ \vdots & \vdots \\ V_{PCCi}(t-T) & \theta_{PCCi}(t-T) \\ P_{0,0} & Q_{0,0} \\ \vdots & \vdots \\ P_{0,j} & Q_{0,j} \\ \vdots & \vdots \\ V_{PCC0}(t) & \theta_{PCC0}(t) \\ \vdots & \vdots \\ V_{PCCi}(t) & \theta_{PCCi}(t) \\ P_{0,0} & Q_{0,0} \\ \vdots & \vdots \\ P_{0,j} & Q_{0,j} \end{bmatrix} \in \mathbb{R}^{NT\times 2} \tag{3.20}$$

In Eq. 3.20, the input matrices $\mathbf{V}_0(t)$, given by Eq. 3.19, are concatenated for all time instances $t \in [0, T]$. The output of this layer $\mathbf{V}_{\diamond,1} \in \mathbb{R}^{NT\times F_1}$ will be F_1 new features for the NT nodes, given by Eq. 3.15:

$$\mathbf{V}_{\diamond,1}(t) = \sigma\left(\sum_{k=0}^{K_1} \mathbf{S}_\diamond^k \mathbf{V}_{\diamond,0}(t)\mathbf{H}_{1k}\right) \in \mathbb{R}^{NT\times F_1} \tag{3.21}$$

These features are computed using the multi-feature GCN with filter banks of Eq. 3.8 for a graph filter order K_1. The non-linear activation function is the rectified linear unit (ReLU) given by $\sigma(\mathbf{Z}_{\diamond,0}(t)) = max(0, \mathbf{Z}_{\diamond,0}(t))$, and is applied element-wise for every entry in $\mathbf{Z}_{\diamond,0}(t)$. $\mathbf{V}_{\diamond,1}(t)$ is the input to the second layer, which again uses Eq. 3.15 and a graph filter order K_2 to get the output $\mathbf{V}_{\diamond,2}(t) \in \mathbb{R}^{NT\times F_2}$:

$$\mathbf{V}_{\diamond,2}(t) = \sigma\left(\sum_{k=0}^{K_2} \mathbf{S}_\diamond^k\, \sigma\left(\sum_{k=0}^{K_1} \mathbf{S}_\diamond^k \mathbf{V}_{\diamond,0}(t)\mathbf{H}_{1k}\right)\mathbf{H}_{2k}\right) \in \mathbb{R}^{NT\times F_2} \tag{3.22}$$

This computation is repeated for all the l layers of the GTCNN until the output $\mathbf{V}_{\diamond,l}(t) \in \mathbb{R}^{NT\times F_l}$ is obtained:

$$\mathbf{V}_{\diamond,l}(t) = \sigma\left(\sum_{k=0}^{K_l} \mathbf{S}_\diamond^k\, \sigma\left(\dots\sigma\left(\sum_{k=0}^{K_1} \mathbf{S}_\diamond^k \mathbf{V}_{\diamond,0}(t)\mathbf{H}_{1k}\right)\dots\right)\mathbf{H}_{lk}\right) \in \mathbb{R}^{NT\times F_l} \tag{3.23}$$

The output at the last GTCNN layer $\mathbf{V}_{\diamond,l}(t) \in \mathbb{R}^{NT\times F_l}$ is multi-dimensional, while a single scalar value is desired at the output: the (predicted) active power $\hat{P}(t)$ or reactive power $\hat{Q}(t)$ at every PCC node. Therefore, the outputs are aggregated using a fully connected linear layer. The aggregation is done per node, meaning the output is learnt per node from the input $\mathbf{V}_{\diamond,l}^{\mathcal{V}_i}(t) \in \mathbb{R}^{TF_l}$, which contains the F_l hidden features of node $\mathcal{V}_i$ across all T time values:

$$f_{GTCNN}^{(\mathcal{V}_i)}(t) = \mathbf{W}_l^{(\mathcal{V}_i)}\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) + \mathbf{b}_l^{(\mathcal{V}_i)},\ \forall \mathcal{V}_i \text{ in } \mathcal{V}_{GTCNN} \tag{3.24}$$

In Eq. 3.24, $\mathbf{W}_l^{(\mathcal{V}_i)}$ and $\mathbf{b}_l^{(\mathcal{V}_i)}$ are the weights and biases of the linear layer at node $\mathcal{V}_i$, and $\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t)$ is obtained by reshaping and rearranging $\mathbf{V}_{\diamond,l}(t)$ to get $\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) \in \mathbb{R}^{F_l T}$ (the hidden features at the last layer across all time copies of node $\mathcal{V}_i$):

[5]The time dimension is included in $\mathbf{V}_0$ by the use of *time-nodes*. The product graph allowed to start with the shift operator $\mathbf{S}$ of a single graph (e.g. $\mathcal{G}_{GTCNN}$) and include the time graph $\mathcal{G}_T$ by creating a new shift operator $\mathbf{S}_\diamond$. The result of this was a bigger product graph $\mathcal{G}_\diamond$, which contains T times the number of nodes N. This is because the product graph operation is equivalent to replicating the graph $\mathcal{G}_{GTCNN}$ for every time instant. Therefore, every node in graph $\mathcal{G}_\diamond$ can be called a *time-node*, and will contain the value of its features at one of the time instances. This approach allows to merge the time dimension into the node dimension and enables the use of existing GNNs.

$$\mathbf{V}_{\diamond,l}(t) \in \mathbb{R}^{NT\times F_l} \xrightarrow{\text{reshape}} \mathbb{R}^{N\times T\times F_l} \xrightarrow{\text{extract } \mathcal{V}_i} \mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) \in \mathbb{R}^{TF_l} \tag{3.25}$$

This approach of node-specific predictions motivates the GTCNN to identify hidden features in the final layer that accurately reflect each node's output rather than relying on arbitrary combinations of weights across all features and nodes. This approach ensures that the model's learning is more focused and interpretable.

Using Eqs. 3.23 and 3.24, the mapping functions $f_{GTCNN}(t)$ of the GTCNN w.r.t. the input $\mathbf{V}_{\diamond,0}(t)$become:

$$\hat{P}^{(\mathcal{V}_i)}(t) = f_{GTCNN,P}^{(\mathcal{V}_i)}(t) = \mathbf{W}_{l,P}^{(\mathcal{V}_i)}\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) + \mathbf{b}_{l,P}^{(\mathcal{V}_i)} \quad \forall \mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}} \tag{3.26a}$$

$$\hat{Q}^{(\mathcal{V}_i)} = f_{GTCNN,Q}^{(\mathcal{V}_i)}(t) = \mathbf{W}_{l,Q}^{(\mathcal{V}_i)}\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) + \mathbf{b}_{l,Q}^{(\mathcal{V}_i)} \quad \forall \mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}} \tag{3.26b}$$

$$\mathbf{V}_{\diamond,l}^{(\mathcal{V}_i)}(t) = \text{reshape}\left(\mathbf{V}_{\diamond,l}(t), F_l T\right)$$

$$\mathbf{V}_{\diamond,l}(t) = \sigma\left(\sum_{k=0}^{K_l} \mathbf{S}_{\diamond}^{k}\, \sigma\left(\dots \sigma\left(\sum_{k=0}^{K_1} \mathbf{S}_{\diamond}^{k}\mathbf{V}_{\diamond,0}(t)\mathbf{H}_{1k,P/Q}\right)\dots\right)\mathbf{H}_{lk,P/Q}\right) \in \mathbb{R}^{NT\times F_l}$$

IThe three hyperparameters of the GTCNN structure are the number of features in each layer F_l, the graph filter order K_l and the product graph rule: $\mathcal{G}_T \diamond \mathcal{G}_{GTCNN}$. During training, the graph filter coefficients $\mathbf{H}_{1k,P/Q}, \mathbf{H}_{2k,P/Q}, \dots, \mathbf{H}_{lk,P/Q}$ are learned for every k-hop neighbourhood together with the weights $\mathbf{W}_{l,P/Q}^{(\mathcal{V}_i)}$ and biases $\mathbf{b}_{l,P/Q}^{(\mathcal{V}_i)}$ of the fully connected layer at the output for every node $\mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}}$. The active and reactive power equivalent models learn a different set of weights.

3.3.3. Training the GTCNN

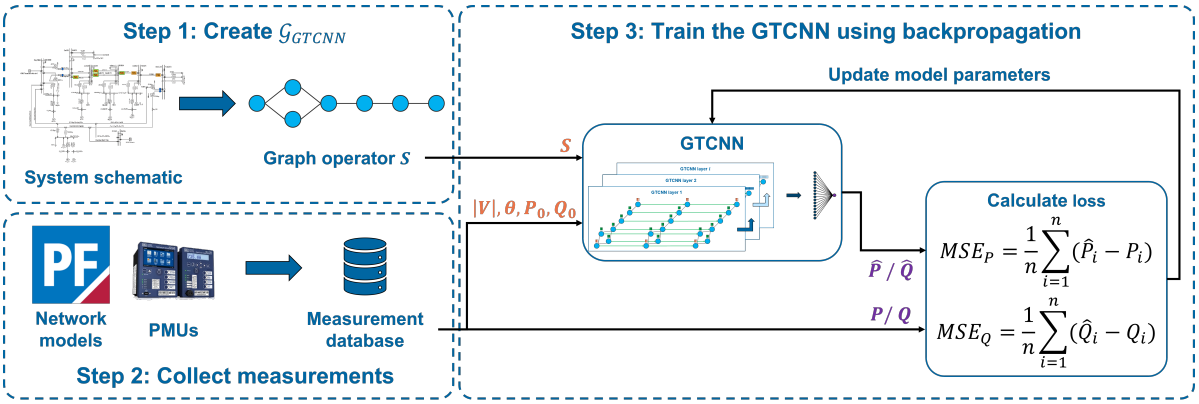


Figure 3.20: The training workflow for the equivalent model using the proposed GTCNN structure. The PowerFactory and entso-e logos are trademarks of DIgSILENT GmbH and EN TSO-E respectively. The PMU image is from Schweitzer Engineering Laboratories Model SEL-735 (not affiliated).

The training workflow for the proposed GTCNN structure is seen in Fig. 3.20. The workflow consists of three steps, which will be explained in more detail below.

Step 1: Create $\mathcal{G}_{GTCNN}$
The first step is to create the graph $\mathcal{G}_{GTCNN}$. This is done by using the modified line graph $L(\mathcal{G}_{modified})$ using Eq. 3.17 (as explained in section 3.3.1). The graph $\mathcal{G}$ can be found by inspecting a system schematic that shows the different buses, their interconnections and the location of the PMU devices. If graph $\mathcal{G}$ is unknown, it is possible to get the graph $\mathcal{G}_{GTCNN}$ by estimating how the PMUs are connected by the DN infrastructure. After the graph $\mathcal{G}_{GTCNN}$ is obtained, the graph operator $\mathbf{S}$ can be computed using either the adjacency matrix $\mathbf{A}$, the Laplacian $\mathbf{L}$, or one of their normalised versions, $\mathbf{A}_n$ or $\mathbf{L}_n$.

Step 2: Collect measurements
The second step is to create the measurement database to train the model. The measurement includes

the inputs $|V|, \theta, P_0$ and Q_0 and the outputs P and Q. These measurements can be collected by real PMU devices in the DN or are generated by a network model in power system simulation software such as PowerFactory, PSCAD, Simulink etc. The measurements collected should cover the faults, operating conditions, and topological changes of interest for transient stability studies, also referred to as the scenarios.

Step 3: Train the GTCNN using backpropagation
The third step is to train the GTCNN with the collected data. The training is done on the entire database, with a portion being held back for validation and testing of the model. The training of the equivalent model will be done using a supervised learning approach. Supervised learning is a type of machine learning where an algorithm is trained on a labelled dataset, which means that each training sample is paired with an output label. The inputs for the GTCNN-based equivalent model are the voltage magnitude $|V|$ and angle θ together with the initial power injections P_0 and Q_0; the output labels are the true active P and reactive power Q response. This approach enables the model to learn the functions $f^{(\mathcal{V}_i)}_{GTCNN,P}(\cdot)$ and $f^{(\mathcal{V}_i)}_{GTCNN,Q}(\cdot)$ of Eq. 3.26 that map the inputs $|V|, \theta, P_0$ and Q_0 to the predicted outputs $\hat{P}$ and $\hat{Q}$. In the context of dynamic equivalent models, supervised learning is the best approach as it allows for the precise modelling of power system dynamics based on historical or simulated data, ensuring that the model learns to reproduce the true dynamics.

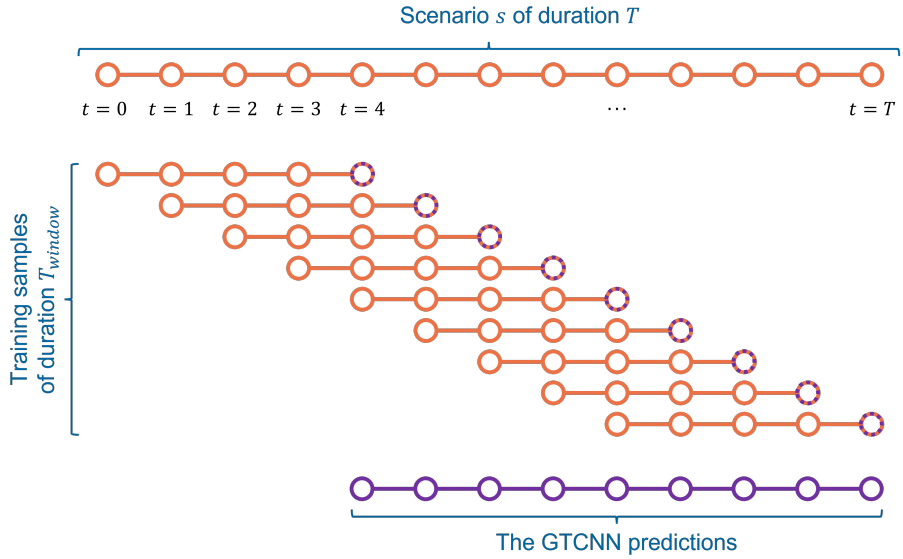


Figure 3.21: A scenario s is divided into multiple samples of duration T_{window}.

The GTCNN is not trained per scenario of duration T. Instead, a sliding window approach is used to divide scenario s into samples of duration T_{window}; T_{window} is also known as the observation window. The sliding window gives $N_{samples} = T - T_{window} + 1$ samples per scenario s (see Fig. 3.21). There are two advantages to using this approach. First, the size of the Cartesian product graph will be reduced from NT (time) nodes to NT_{window}, which will also reduce the number of computations for the GTCNN. In power systems, and especially DNs, the number of nodes N is high. With this approach, it is possible to have T/T_{window} times the number of nodes N before the same computational complexity is reached. Second, the GTCNN becomes insensitive to the moment of the event that causes the dynamic response. As the GTCNN is trained on smaller training samples, it learns how the input changes leading up to an event within the window. The moment this occurs could be at a different time in the scenario, while the input the model sees remains the same, therefore making the model more robust. Similarly, the windowing approach also ensures that the model considers training samples where there are no dynamics, ensuring a more balanced dataset with both training samples with and without dynamics.

The GTCNN predicts $\hat{P}$ or $\hat{Q}$ at the end of each training sample (see Fig. 3.21). This way, every time instance t is predicted only once. The output of the GTCNN to a single scenario can be obtained by giving the training samples of a scenario s in chronological order, making the predictions with the

GTCNN, and putting all predictions together[6]. The drawback of this approach is that the model will not be able to output a dynamic response before T_{window}. This limitation can be overcome by inputting a longer measurement before the event.

The proposed GTCNN structure has four hyperparameters, three of which come from the GTCNN (see section 3.3.2). The first hyperparameter is the GTCNN structure. This parameter indicates how many hidden layers the GTCNN has and the number of hidden features F_l per layer. More hidden features and layers can improve the expressive power of the network but also make it more prone to overfitting the data. The second hyperparameter is the graph filter order K_l per layer (see Eq. 3.23). A larger value of K_l results in filters that consider the signal at further-away nodes (larger neighbourhood) when computing the embeddings $\mathbf{V}_{\diamond,l}$ (as seen in Fig. 3.5). A larger K_l can result in hidden features that focus more on the entire graph, but it will also increase the number of trainable parameters and, thus, the training time. The third hyperparameter is the type of product graph to use (Cartesian, Kronecker or strong). As section 3.3.1 explains, the Cartesian product graph is preferred for its interpretability. A strong product graph could yield better results as it also considers the connection of a node's value at the current time with respect to the value of its neighbours at a previous time instance. Adding this relation will also increase the runtime as there are more edges on which the graph signal should be propagated. The fourth hyperparameter comes from the training approach and is the observation window T_{window}. The observation window gives the number of measurement points in a single training sample. More measurement points mean that a larger segment of the inputs is considered, which makes the model more accurate for a particular scenario because the GTCNN model sees a larger part of the input. However, having more measurement points per training sample will likely lead to overfitting.

The output $\hat{P}$ and $\hat{Q}$ from every training sample is compared with the actual output P and Q by the loss function. The prediction of the dynamic response is a supervised regression task. Therefore, a loss function compares the predictions to the true targets. The MSE loss is chosen as the loss function for the GTCNN, which is applied per node as the predictions are made per node. The MSE loss will only be computed for the nodes where the targets P and Q are available. In the GTCNN structure, this is at the PCC nodes (see also Fig. 3.18), which means that the hidden features $F_1, F_2, \ldots F_{l-1}$ is updated w.r.t. the loss at the PCC nodes instead of all nodes. After computing the loss, the optimiser updates the graph filter coefficients $\mathbf{H}_{1k,P/Q}, \mathbf{H}_{2k,P/Q}, \ldots, \mathbf{H}_{lk,P/Q}$ and the weights $\mathbf{W}_{l,P/Q}^{(\mathcal{V}_i)}$ and biases $\mathbf{b}_{l,P/Q}^{(\mathcal{V}_i)}$ of the fully connected layer at the output for every node $\mathcal{V}_i \in \Omega_{\mathcal{V}_{PCC}}$. This process is repeated for a predetermined number of epochs, the maximum allowable runtime or until the model no longer improves.

3.3.4. Exchange with the TSO

The workflow using the GTCNN equivalent model is shown in Fig. 3.22. The TSO has their TN model with some scenarios of interest, for which they want to know the dynamic response of the DN. The outcome of these studies can show TSOs the severity of the active and reactive power produced by various connected DNs and can help them reinforce the network where necessary. The TSO shares these scenarios with the DSO (e.g., once per month), which can use its network model to collect the measurements $|V|, \theta, P$ and Q in a measurement database. The DSO can also use the grid code requirements to generate scenarios and measurements that show the DN's dynamic behaviour. Alternatively, the measurement database can be constructed with measurements directly taken from PMUs throughout the system (or both). The DSO uses the measurement database to train the equivalent model[7].

The TSO receives the equivalent model from the DSO[8], which it will use in its TN network model to predict the dynamic responses of the DNs in response to an event. The current active and reactive power injections, P_0 and Q_0, are obtained from the recent load, PV and WTG profiles from the ENTSO-E transparency platform. Using these profiles to find the current injections, the GTCNN will give an output

[6] The predictions $\hat{P}$ and $\hat{Q}$ will be in a correct chronological order since the training samples were also inputted in the correct order.

[7] The measurement database is constructed in this thesis using the DN network model; this process is described in more details in section 4.2.

[8] The proposed methodology here suggests that DSOs make the equivalent model as they have the most information available about their DN. However, it is also possible for the TSO to create an equivalent model using this methodology by using recorded measurements to construct the training database.

that is more representative of the current operating conditions and, thus, more accurate. Because the GTCNN is trained to be robust under changing operating conditions and topological changes, the equivalent model has to be updated less frequently, for example, once per month. A possible way to connect the equivalent model to the TN network is through a co-simulation interface. Instead of inputting the entire scenario duration T, the equivalent model will receive only T_{window}, containing the current and previous $T_{window} - 1$ values of the input. It will then compute the output at the current solver time t: $P(t)$ and $Q(t)$. The solver can use these values to find the inputs at the next time step, and the process is repeated for the entire simulation duration.

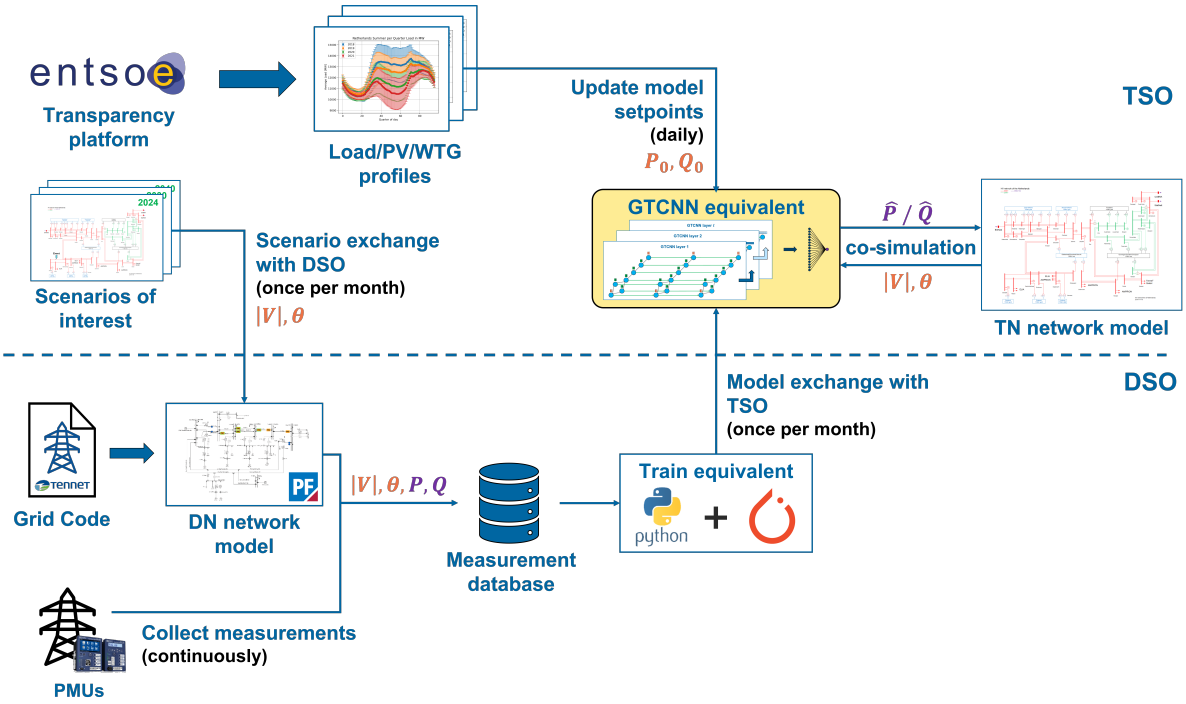


Figure 3.22: Workflow for the GTCNN equivalent when used by the TSO for dynamic studies. The PowerFactory, TenneT, Python, PyTorch and entso-e logos are trademarks of DIgSILENT GmbH, TenneT TSO B.V., Python Software Foundation, PyTorch Foundation and ENTSO-E, respectively. The PMU image is from Schweitzer Engineering Laboratories Model SEL-735 (not affiliated)

The possible application presented here should not be hard to implement, as much of the information used by the GTCNN equivalent is already available. For example, many European grid operators already share the actual power production and consumption on the ENTSO-E transparency platform [87]. If these profiles are unavailable, the TSO can use their measurements at the PCC to determine the injections at the DN. Similarly, the graph G_{GTCNN} can be found relatively easily if a top-level grid map is available and the measurement locations are known. If the DSO uses a network model, this is no problem. If the TSO collects its measurements, it will know the PMU locations and can connect them accordingly to the underlying top-level infrastructure, as no network-specific parameters, such as line admittances, are needed.

The biggest challenge will be the collection of the various measurements under the different scenarios. It is expected that when DN dynamics start playing a crucial role in the power system, the DSOs will have a dynamic network model of their system. This dynamic network model can be used to generate the measurement database. It is also possible to use collected PMU measurements from various points in the system. The downside of this approach is that faults or other events rarely occur in the power system, which means that these events will hardly be present in the database. This scarcity will make the equivalent model less accurate and drastically reduce its performance for unseen scenarios. Therefore, it is crucial to have a representative measurement database when training the equivalent model.

4

Case Study

This chapter will validate the GTCNN-based equivalent model proposed in Chapter 3. As the equivalent is black-box-based, it does not require having a full system model, but collecting only PMU measurements and the network graph $\mathcal{G}_{GTCNN}$ is sufficient. Generally, this information is not directly available unless specific agreements are made with grid operators, as in [40]. Therefore, the required data is generated using the PowerFactory model of a real distribution ring in Zeeland, The Netherlands, called the Zeeland (Delta) 50 kV distribution ring [88]. This network will be referred to as the Zeeland 50 kV ring in the remainder of this thesis.

This chapter is structured as follows. First, an introduction to the Zeeland 50 kV ring is given in section 4.1. Second, in section 4.2, the data generation process with this network is elaborated. Third, section 4.3 discusses two baseline models. Next, the choice for the GTCNN hyperparameters is elaborated in section 4.4. Sections 4.5, 4.6 and 4.7 compare the results of the GTCNN with the baseline models under different datasets. In Section 4.8, four additional case studies are included supporting the proposed methodology. Finally, in section 4.9, the scaling behaviour of the GTCNN is discussed.

4.1. Introduction to the Zeeland 50 kV ring

The Zeeland 50 kV ring represents a distribution network near the North Sea in the southwest of the Netherlands. It consists of five 50 kV substations, Goes Evertsenstraat (Gse), Zierikzee (Zrz), Oosterland (Otl), Tholen, (Tln) and Kruiningen (Kng), connected in a ring-like structure and coupled to the 150 kV transmission network at two locations, Kng and Goes de Poel (Gsp). Historically, the distribution network operator was Delta Network Group, which is why the network is also known as the Delta 50 kV ring. In 2006, 200 kilometres of 150 kV and 380 kV network parts were sold to the TSO TenneT as part of the passed law for independent network operation (Wet Onafhankelijk Netbeheer) [89]. Moreover, the network operation had to be done by an independent party that was not supplying energy since that would create monopolies. This led to the Delta network operation division (DELTA Netwerkbedrijf or DNWB) changing its name to Enduris in 2016 [90]. In 2017, DNWB was acquired by the network operator Stedin, making the Zeeland 50 kV ring now part of Stedin's assets. To avoid any confusion or incorrectness about who owns the network, the network is called the Zeeland 50 kV ring.

4.1.1. Network description

The Zeeland 50 kV ring is depicted in Figure 4.1. The five 50 kV substations (Gse, Zrz, Otl, Tln and Kng) are connected using underground cables [91]. Although Figure 4.1 represents the substations linearly, their geographical location is in a ring-like configuration (see Figures 1 and 2 of [91]). Each 50 kV substation is connected to local 10 kV distribution grids through two 50/10 kV transformers. Loads are aggregated into a single load at each 10 kV bus. Moreover, there are wind turbine generators (WTGs) at three locations (Zrz, Otl and Tln) and combined heat and power plants (CHPs) at Otl and Tln. This makes the Zeeland 50 kV ring an active distribution network [13], with an overall system generation that is roughly double the system load [88].

The Zeeland 50 kV ring is connected to the 150 kV transmission network in the south at two locations

(Gsp and Kng) through four three-winding (150/50/10 kV) transformers. The presence of two identical transformers at each point of common coupling (PCC) signifies an n-1 redundancy.

The network is monitored by Phasor Measurement Units (PMUs) installed at each 50 kV substation. This was done to have maximum observability with information on the voltage phasors at all five substations [88]. Zrz has two PMUs since the two lines connecting it to Otl have different parameters, whereas an equal impedance eliminates this need between Gse and Zrz [88].

The used model is the one implemented by the authors of [91] and is based on the real Zeeland 50 kV distribution ring as described by DNWG in [88]. Interested readers can study [91] for more information about the modelling and validation of the Zeeland 50 kV distribution ring.

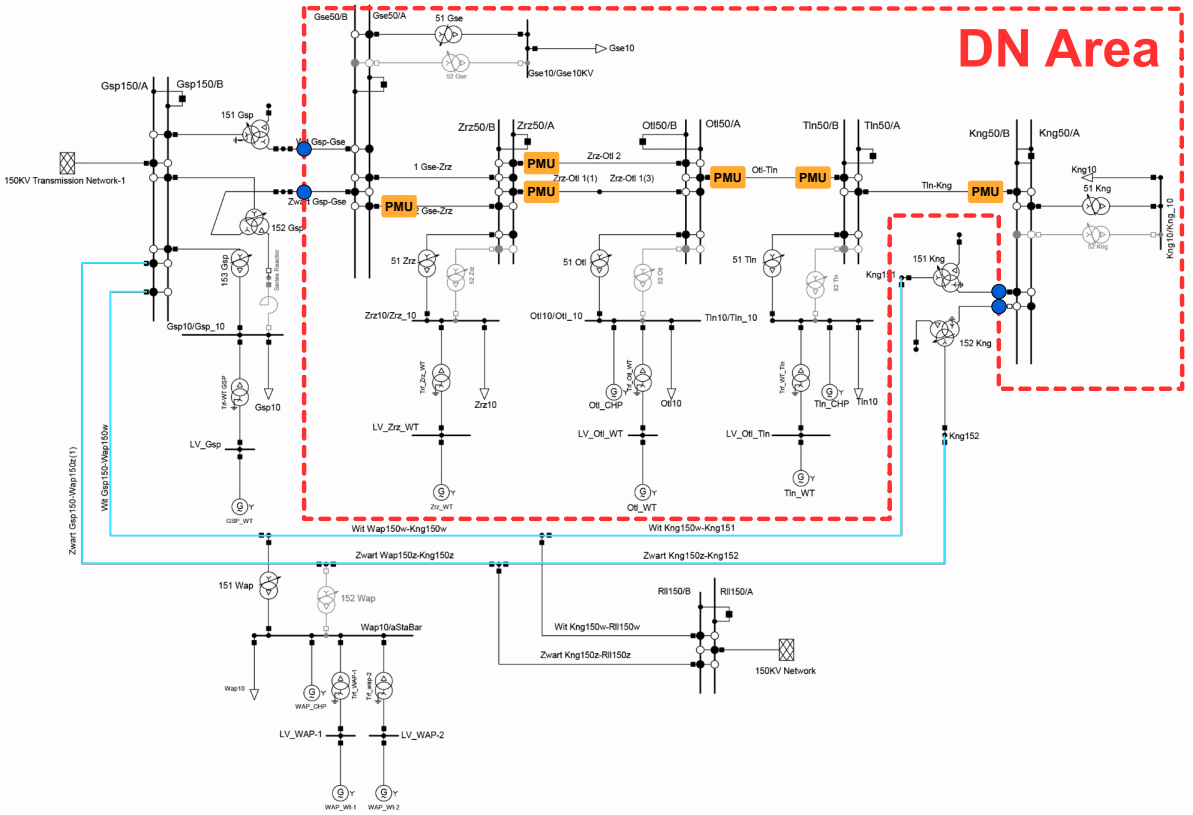


Figure 4.1: The Zeeland (Delta) 50 kV distribution ring is implemented in PowerFactory. Only the 50 kV area of the network is considered to be the DN ( ). The DN has six PMUs ( ), at least one near each 50 kV bus. The DN is connected to the 150 kV TN at two points by four PCC terminals ( ). The fault events are generated on the 150 kV transmission lines between Gsp and Kng ( ).

4.1.2. Comparison to other networks in the literature

The Zeeland 50 kV ring was chosen for three reasons. First, its size and voltage level are comparable to existing literature, which are in the range of tens of kVs. A 22 kV microgrid is used in [46]. [34] uses the Cigre Task Force C6.04.02 Benchmark Medium Voltage Distribution Network. The real PMU measurements in [41] are collected from ADNs ranging between 10 and 20 kV. In [11], [35], [36], [39], a 75-bus 11 kV network is deployed; [29] uses a different 11 kV network. A 400 V laboratory scale network is utilised by [42]. Second, the Zeeland 50 kV ring is modelled after a real ADN [91]. The realistic nature of this network will make the generated data more representative. Third, this ADN is connected to the TN at two PCCs, while existing works consider a single PCC [28], [29], [35], [36], [38], [39], [41], [42], [44], [46]. Using a meshed ADN shows the performance of the proposed GTCNN-based equivalent model under a more complex and futuristic DN.

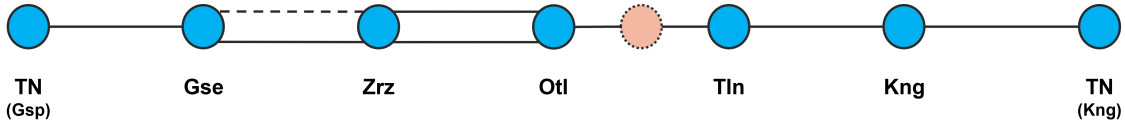
This network is connected to the TN at the 150 kV terminals at Gsp and Rll, as seen in Fig. 3.1. Unfortunately, the PowerFactory model of the Dutch high-voltage grid is not available to generate the

events that cause a response from the Zeeland 50 kV network. There are still several ways to generate events when the TN model is unavailable. It is possible to connect a different TN to the DN [44], [92]. Another option is to create a (voltage) disturbance at the PCC [29], [38], [42]. Additionally, it's feasible to split the entire ADN into two parts: one dedicated to constructing an equivalent model and the other focused on generating events [28], [34], [46].

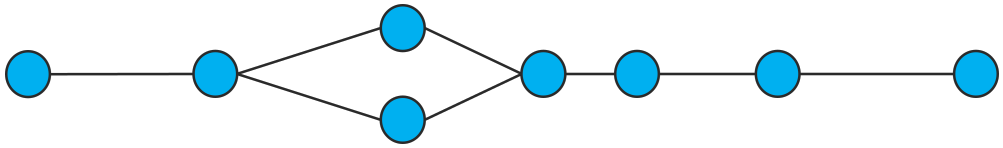
Splitting the ADN into two parts is the best choice for this PowerFactory model. This is because the 150 kV parts of the network should be part of the TN [89]. Moreover, the data remains representative as the events are generated within the same previously validated network.

4.1.3. Graph representation

The modified graph $\mathcal{G}_{modified}$ of the Zeeland 50 kV ring is depicted in Fig. 4.2a. This graph is constructed using the DN area of Fig. 4.1 and the rules from section 3.3.1. The cable *1 Gse-Zrz* between *Gse* and *Zrz* is not observable and is, therefore, not considered in $\mathcal{G}_{modified}$. Moreover, a fictitious node is added between the buses *Otl* and *Tln* as there are two PMUs on the cable *Otl-Tln*; note how this results in two nodes in the modified line graph $\mathcal{G}_{GTCNN}$.



(a) The graph $\mathcal{G}_{modified}$ of the Zeeland 50 kV ring.



(b) Modified line graph $\mathcal{G}_{GTCNN} = L(\mathcal{G}_{modified})$ of the Zeeland 50 kV ring.

Figure 4.2: The modified graph $\mathcal{G}_{modified}$ and the corresponding GTCNN graph $\mathcal{G}_{GTCNN}$.

4.2. Preparing the data

The proposed GTCNN equivalent model requires simulation data as it is a black box model. This data will be generated from the Zeeland 50 kV ring using the PowerFactory model discussed in section 4.1. Two stages are needed to generate the required data for the GTCNN (see Fig. 4.3): data generation (**stage 1**) and data preprocessing (**stage 2**). In the first stage, the necessary data is gathered through a Python script that simulates different scenarios in a PowerFactory model. In the second stage, the generated data is preprocessed into the appropriate form for the model. Finally, the GTCNN can be trained using the training workflow discussed in section 3.3.3. The full code is available upon request.

This section will briefly describe the two steps needed to generate the data for the GTCNN-based equivalent model. First, the data generation is discussed in section 4.2.1. Second, section 4.2.2 elaborates on the preprocessing of the generated data.

Figure 4.3: The general pipeline for equivalent modelling using GNNs.

4.2.1. Data generation

The derivation of any grey- or black-box equivalent model requires creating a dataset with various scenarios s_i of interest. In this thesis, a scenario refers to a unique combination of an event f, operating condition oc and topological state tc of the network. Algorithm 1 incorporates the three elements of the scenario definition into the network.

Algorithm 1 works as follows. First, the class `PowerFactory` is created. This class communicates with PowerFactory through its Python interface and can access all the network elements and their parameters, allowing it to run repetitive simulations without manual intervention. The class first reads a JSON file, which tells it how to simulate the scenarios and from what components to collect the measurements. After that, the class method `run_topology_simulations` is called, which creates the scenarios according to the provided settings, implements them into the network and runs the simulation. This is done with several for-loops: one iterates over the faults f (i.e. the events), three over the load, WTG and PV profiles to generate different operating conditions oc and one over the topological changes tc.

The outermost loop iterates over **fault events** that excite a dynamic response from the DN. Faults are sampled from a set of events in the study case of the PowerFactory model together with their clear event. Each of these events represents a different fault location, i.e. the component in which the fault occurs, such as a line. The fault events will be generated on one of the eight 150 kV transmission lines outside the DN (see Fig. 4.1)[1]. After choosing a line, the fault impedance and location percentage in the line are sampled. The sampled values are then assigned to the selected event, resulting in a new fault f.

The next three for-loops iterate over the load, WTG and PV profiles obtained from ENTSO-E to sample different **operating conditions** oc at different points on the profiles. The sampling of different oc will be illustrated using the WTGs; the process is identical for the loads and PVs. It is possible to find the active and reactive power production per day, week, season, etc., on the transparency platform. This allows for the computation of the daily averages and deviation per season. As TenneT, the Dutch TN operator, makes this information available in intervals of a quarter of an hour, the mean and standard deviations are stored in day quarters (96 in a day) for the four seasons. Selecting the mean and standard deviation in a different season ($season$) and at a different time of the day (day-quarter t), it is possible to evaluate the response of the DN under different conditions (e.g. low or high generation). The iteration of the seasons is done in the second and the day quarters in the third for-loop of algorithm 1.

Once the average and the spread of the WTGs are determined at a particular time of the year, new values are sampled. This can be done using the normal distribution, where its parameters are found from the computed values at this time interval. However, there is one problem: all samples are considered independent and indentically distributed. This means that scenarios would occur where some wind farms have a high production (at one tail of the normal distribution) while others do not produce at all (at the other tail). In reality, the different WTGs are geometrically close to each other, meaning that their production should be correlated because the weather conditions are shared. This is solved using Monte Carlo sampling with correlated samples that follow a Kumaraswamy distribution $\mathcal{K}(a, b)$. This type of sampling includes the correlation between the different WTGs in the DN. The probability density function $f(x; a, b) = abx^{a-1}(1-x^a)^{b-1}$ of the Kumaraswamy distribution $\mathcal{K}(2, 2)$ resembles the normal distribution $\mathcal{N}(\frac{1}{2}, \sigma^2)$ but is a bit skewed to the right and bounded between 0 and 1. The original support $x \in (0, 1)$ is scaled to $x \in (\mu - 3\sigma, \mu + 3\sigma)$, where μ is the normalised average production/consumption at the specified $season$ and day quarter t and σ is the first standard deviation at the same point on the profile. These bounds are more realistic as they prevent extreme outliers in components that operate with finite limits.

The sampling of new operating conditions also requires checking the power flow convergence before running the dynamic (RMS) simulation[2]. This is to ensure that no limits are violated prior to the fault, as that would result in protection equipment stepping in or equipment damage if the protection fails. The new operating condition oc is considered to be converging if all bus voltages are between 0.95

[1] This approach of generating fault events within the same model but outside the DN part is similar to [34].

[2] This step was not required when implementing the fault event since the network in its default values is already stable and converging.

and 1.05 pu and when the loading percentage of the transformers, lines and cables does not exceed 100% (their maximum capacity). If one of these limits is violated, new values are sampled for the loads, WTGs, and PVs until convergence (or surpassing the maximum tries). If no limits are violated, a new operating condition oc is obtained.

Algorithm 1 Run Topology Simulations

```
powerFct ← new PowerFactory(JSON, project, studycase)

procedure RunTopologySimulations(savedir)          # part of PowerFactory class
    Check inputs
    Optional: Implement custom start scenario
    Save network state to adaptedState
    iteration = 0
    for f ← 0 to numFaults do
        Cancel all events
        Update seed
        Sample fault f
        Save network state to ocState
        for all season ∈ seasons do
            for all t ∈ dayQuarters do
                for ti ← 0 to numTi do
                    Sample operating condition oc
                    Check scenario feasibility
                    Save network state to tcState
                    for tc ← 0 to numTopologies do
                        Sample topological change tc
                        Check scenario feasibility
                        if convergence conditions are met then
                            Store current scenario s_i = {f, oc, tc}
                            Setup and run simulation
                            Save simulation results
                        end if
                        if no more topological changes to sample then
                            Restore network state tcState
                            break loop
                        end if
                        iteration += 1
                        Restore network to tcState
                    end for
                    Restore network to ocState
                end for
            end for
        end for
        Restore network to adaptedState
    end for
    Restore network to initial state
    Save the list of collected scenarios s
end procedure
```

The innermost for-loop samples **topological changes**. Topological changes change the electrical network, resulting in a different dynamic behaviour. Some examples are the disconnection of network parts because of damage or planned maintenance, the connection of inactive (redundancy) components, the addition of lines and cables, the connection of existing feeders through interconnectors, the addition of DGs through transformers, etc[3]. The sampling of topological changes in algorithm 1 has

[3]The connection of an extra load is not considered a topological change since it is equivalent to changing the setpoint of existing

two stages. First, it chooses which component to alter from a list of available components. Second, it implements a new topological change based on the component type. The value is flipped for binary variables such as the (dis)connection of components. For the tap changeable transformers, a new position is sampled between its minimum and maximum positions. After a topological change has been sampled, the convergence of the power flows is checked before running the RMS simulation. The topological change tc is stored to prevent resampling of the same action[4].

Once the sampling is done and a convergent scenario $s_i = \{f, oc, tc\}$ is obtained, the RMS simulation is run. The measurements of the observable elements, as specified in the JSON file, are logged during the simulation and stored in Parquet files[5]. For the Zeeland 50 kV ring, the observable elements are the cables and terminal of the PMU devices and the four PCC terminals (see Fig. 4.1). A 2x decrease in disk size is obtained using Parquet files, which are chosen as the preferred format for storing the simulations. A separate file containing the converging scenarios s_i is created at the end of the `run_topology_simulations` function. After all the scenarios are generated, the network is restored to its initial state.

4.2.2. Data preprocessing

All the scenarios generated by Algorithm 1 are represented by a separate Parquet file. The scenario files of the training, validation or test sets are saved in different folders, while the scenarios of a single set are in the same folder together with the list of collected scenarios. Each scenario file is formatted as a table where every column represents a different measurement variable (e.g. P at the 50 kV substation Zrz), and the rows contain the values at different time instances. This format is not suitable for training the GTCNN-based equivalent model without preprocessing.

Algorithm 2 constructs a single dataset from all the Parquet files in a specific folder in the expected format by the ML model. The procedure `prepare_data` is used to transform scenarios from the format of the PowerFactory simulations $T \times M$ (where M is the number of collected measurement values) into the expected format of the ML model. In the GTCNN implementation of [82], whose codebase is used in this thesis, the expected format is $N_{samples} \times F \times T_{window} N$. The procedure `prepare_data` iterates over all files and applies the same steps. First, it will skip the scenario file s_i if it is a duplicate scenario[6]. Next, it will only read the data of scenario s_i from the Parquet file if it is not a duplicate. The next two steps are optional: merging PCC terminals[7] and interpolating the data to have a constant sampling rate[8].

The next step is to transform the data from the PowerFactory format ($T \times M$) into the desired format. This is done by the function `transform_data`, whose working is visualised in Fig. 4.4. First, the

loads and is thus an operating condition.

[4] This step is needed to efficiently sample the topological changes since the changes are discrete, increasing the likelihood of picking the same event. The sampling of the operating conditions did not have this problem since its distribution was continuous between the upper and lower bounds, making it less likely to have exactly the same values at all the loads, WTGs and PVs between two given sampled scenarios.

[5] *"Apache Parquet is an open source, column-oriented data format designed for efficient data storage and retrieval."* [93] Comma Separated Values (CSV) files, on the other hand, are row-based. In both cases, the rows represent the different elements, and the columns are usually attributes. Because CSV files are row-based, they require reading in the entire dataset when accessing a single attribute, while parquet files can import only specific columns. Moreover, Parquet files also apply some compression, which reduces the disk size. For these reasons, Parquet files can drastically reduce costs of storage and compute power [94].

[6] The list of collected scenarios is used for this purpose, containing a different scenario $s_i = \{f, oc, tc\}$ per row. This step is intended as a last resort in case Algorithm 1 fails to prevent duplicate scenarios. However, when the settings in the JSON file are chosen correctly, no duplicates should exist. The fault events and operating conditions are sampled from continuous distributions, making it highly unlikely that the same values are implemented. Moreover, although the topological changes are sampled from a discrete set, Algorithm 1 keeps track of the sampled tc to prevent resampling the same actions.

[7] The merging of the PCC terminals can be used when a single PCC has two single terminals (as is the case for the Zeeland 50 kV ring of Fig. 4.1) to create a single PCC. The merging will improve the GTCNN performance when both terminals have a very distinctive behaviour. The measurements from both terminals are combined into a single measurement according to the following rules. If the measured quantities are the voltage magnitude $|V|$, the phase angle θ or loading percentage, take the first value as these should be identtical. If the measured quantities are the active power P, the reactive power Q or the current I, add the two values together.

[8] This processing step should be used when the steps between different measurement sample points is not constant. This difference is mostly a problem in simulation software such as PowerFactory, where the solver might use a different step size when needed to be able to solve the differential equations properly. It is recommended to solve this issue inside the simulation software before generating the data with Algorithm 1 as the interpolator will approximate the measurement data, thus reducing its accuracy and increasing the processing runtime.

table obtained from the PowerFactory simulations is transformed into the standard format of PyTorch Geometric[9], which is N (nodes) $\times F$ (node features); a third dimension is added to include the temporal aspect. This transformation is realised by splitting the different columns M into nodes N and features F. The resulting 3D array is windowed using the approach described in section 3.3.3 to create $N_{samples}$ of length T_{window} from the original measurements with duration T. This gives a 4D array of dimensions $N_{samples} \times N \times F \times T_{window}$. This 4D array can then be manipulated using permutations and reshaping to arrive at the expected input format of the model[10]; in the case of the GTCNN, this is $N_{samples} \times F \times T_{window} N$. These data transformations are repeated for all scenario files s_i in the current folder, resulting in $N_{scenarios} \cdot N_{samples}$ total training samples for the model. The training samples are shuffled and saved with the targets inside a PyToch Geometric `DataLoader` of a specific batch size, which can be inputted into the different models. A wrapper function `construct_datasets` is written to create a single Python dictionary with three data loaders: one for the training, one for the validation and one for the test set. If used, the scalers are fit on only the training data.

Algorithm 2 Prepare Data for Training

function TransformData($pf\_dataset$)
  $gnn\_data \leftarrow$ create_3d_dataset($pf\_dataset$) # 3D tensor $N \times F \times T$
  $data \leftarrow$ window_dataset(gnn_data) # 4D tensor $N_{samples} \times N \times F \times T_{window}$
  $targets \leftarrow$ find_targets(gnn_data)
  Permute $data$, $targets$ for model # specific to the ML model
  return ($data$, $targets$)
end function

procedure PrepareData($path$)
  Check inputs
  if $path$ is a folder **then**
    $files \leftarrow$ sorted list of scenario results s_i in $path$
  else if $path$ is a file **then**
    $files \leftarrow$ list containing the file at $path$
  end if
  Optional: Create training/validation/test splits from $files$
  for scenario file s_i in $files$ **do**
    Optional: Skip s_i if duplicate
    $pf\_dataset \leftarrow$ read_parquet(s_i)
    Optional: Merge PCC terminals
    Optional: Interpolate data
    $data$, $targets \leftarrow$TransformData($pf\_dataset$)
    Add samples to $data$, $targets$
    Optional: Compute time vector
  end for
  Optional: Fit scalers
  Create data points from $data$, $targets$
  Optional: Shuffle datapoints
  Create DataLoaders of specified batch size
end procedure

The procedure `prepare_data` can also be used on a single scenario file s_i. This mode is useful when finding the accuracy of the equivalent model for a single scenario s_i. The main difference in steps is that the time vector will be computed for plotting and that the data is not shuffled before returning it, as the temporal sequence of the samples should be preserved.

[9] PyTorch Geometric is a Python for GNNs built on to the popular ML library PyTorch. Many papers on GNNs publish their models to the package. As this library is actively maintained and a central hub for GNN implementations in Python, adopting their data format will simplify future work on GNNs for equivalent modelling.

[10] The basic skeleton of the function `transform_data` can be used to format the data for any ML model. This makes the implementation extremely powerful, as it allows for easily adding different ML models and comparing their performance to the proposed GTCNN model.

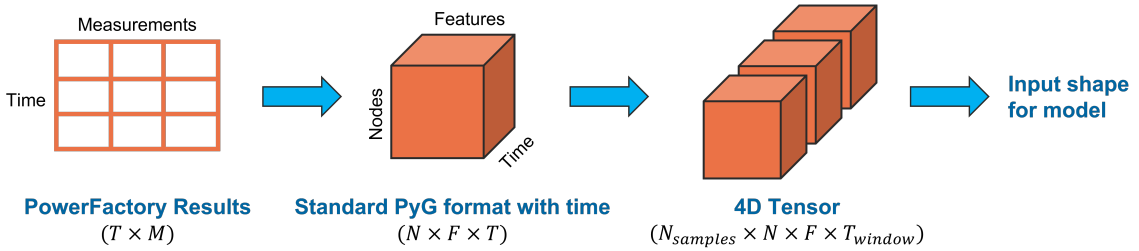


Figure 4.4: A single simulation scenario s_i is transformed from its tabular format into a 3D array in the PyTorch Geometric format, including time. The resulting array is then windowed across the temporal dimension to create the different training samples. This output can then be permuted into the expected input format of the different models.

4.3. Baseline models

The proposed GTCNN structure will be compared against a state-of-the-art black box model based on an LSTM network, which showed promising performance because of its ability to learn the differential equations of the power system with LSTM networks and the algebraic equations with the fully connected layers [44]. The network can be seen in Fig. 4.5. The input sequence, in this case $V(t)$, is used as input to train N_{LSTM} networks of L_{LSTM} layers. The hidden features output F_{LSTM} of each LSTM network is used as input to the fully connected layer of F_1 hidden features. The F_1 features from the first fully connected layer are passed through a second fully connected layer, giving F_2 features. These features are passed through an output layer to give the output P or Q. The LSTM-based equivalent model follows the same prediction approach as the GTCNN, as seen in Fig. 3.21. This means that the network uses T_{window} historical measurements to predict the active P or reactive Q power output at the end of the current window, meaning a single value is obtained. The following values were used in the case studies: $N_{LSTM} = 100, F_{LSTM} = 64, L_{LSTM} = 2, F_1 = 150$ and $F_2 = 80$. Moreover, the observation window T_{window} is set to 5 (as in [44]) and 100[11]. These models are referred to as LSTM5 and LSTM100, respectively. The number of trainable parameters of these two models is 6,399,592[12].

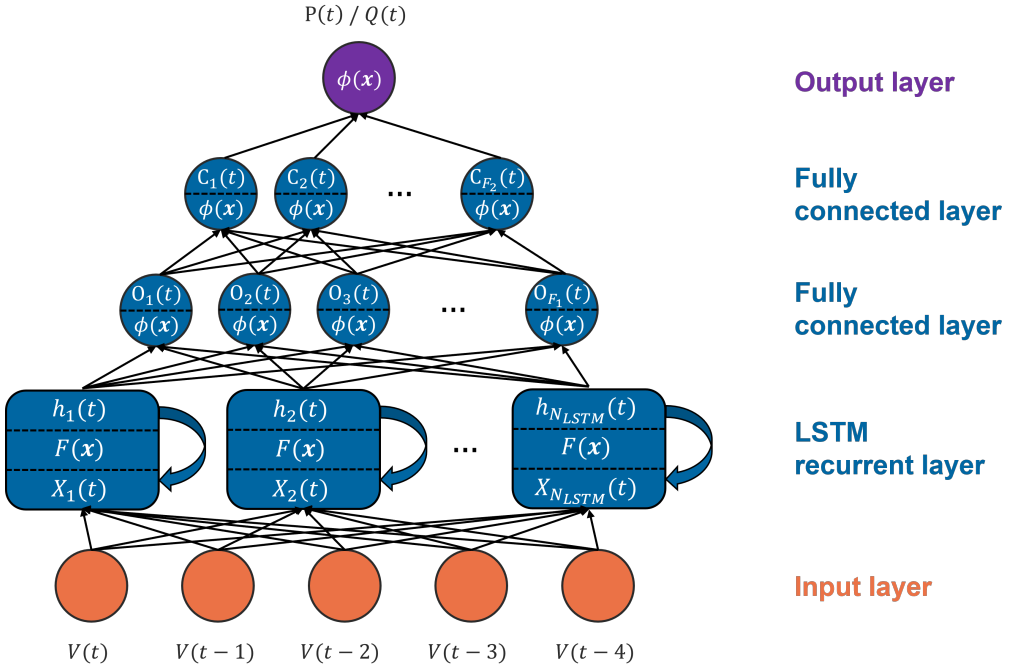


Figure 4.5: The neural network architecture of the LSTM-based equivalent model from [44].

[11] This is done to match the observation window of the GTCNN, ensuring that the same data is inputted into the LSTM model.

[12] The number of parameters is the same for both models as the number of trainable parameters does not depend on the length of the input sequence T_{window}. However, it is expected that the LSTM100 will be slower to train than the LSTM5 as more data is passed through the former network.

A hybrid model has been created that combines the GTCNN with the LSTM network; this network is referred to as the GTCNN-LSTM network. The network can be seen in Figure 4.6. The GTCNN-LSTM uses the same layers as the GTCNN network, but instead of using the fully connected layers per node to find the outputs P and Q, it uses the LSTM network. The F_l hidden features at the last GTCNN layer are inputted per temporal graph replicas of $\mathcal{G}_{GTCNN}$, meaning the inputs at each time instant will have the dimensions $N \cdot F_l$. This approach of transferring the outputs from the GTCNN to the LSTM is selected because the temporal graph $\mathcal{G}_T$ ensures that the hidden features within the GTCNN at a specific graph replica provide insights into the inputs at that particular time and prior instances. This approach requires that T_{window} is the same for the GTCNN and the LSTM. This approach is classified as a disjoint model as two different models are combined, and it will evaluate if the LSTM model using temporally structured hidden graph features yields better performance than either individual model. The hybrid GTCNN-LSTM model has an observation window of $T_{window} = 100$, two layers of each 20 hidden features in the GTCNN, a graph filter order $K = 2$ and uses the Cartesian product graph. The LSTM part has the same parameters as the LSTM100 network: $N_{LSTM} = 100, F_{LSTM} = 64, L_{LSTM} = 2, F_1 = 150$ and $F_2 = 80$. The input length of the LSTM part is 100 to match the number of temporal graph replicas of the GTCNN. These hyperparameters give the number of trainable parameters of 10,086,912.

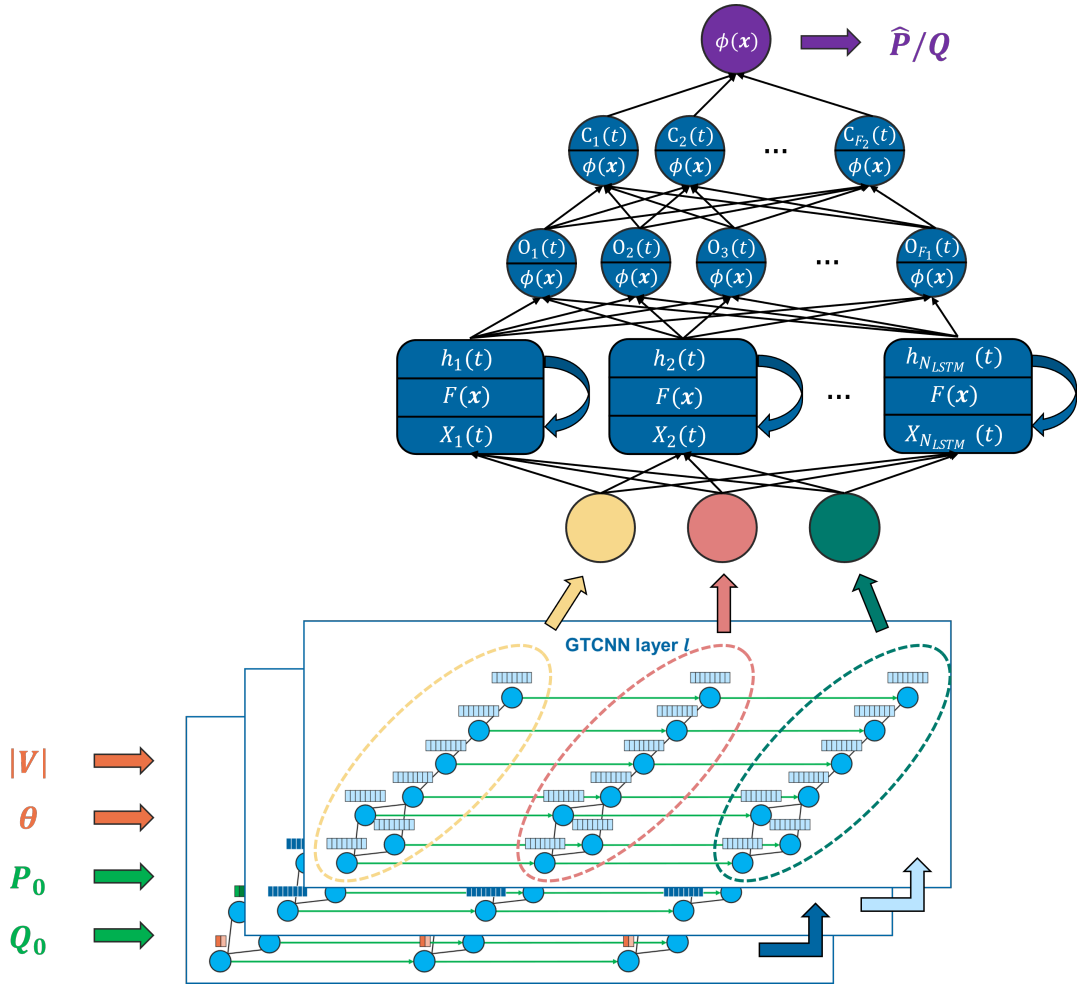


Figure 4.6: The neural network architecture of a disjoint model that combines the proposed GTCNN structure with the LSTM-based equivalent model from [44].

4.4. Hyperparameter tuning

The proposed GTCNN-based equivalent model has four hyperparameters that need to be tuned: the observation window T_{window}, the GTCNN structure and the filter order K. The GTCNN is tuned independently on each hyperparameter by evaluating the average MSE and the standard deviation of several candidate values. The default parameter is changed with the best-performing one before tuning the next hyperparameter.

First, the observation window T_{window} is tuned. T_{window} indicates the number of measurement points per training sample, where a larger value corresponds to a longer input sequence. A separate model is trained to predict the active power P and reactive power Q. The GTCNN is trained for 100 epochs, but the training is stopped early if the validation accuracy does not improve over 25 epochs. A batch size of 64 is used. The evaluated values are $T_{window} = \{25, 50, 100\}$.

The results are seen in Table 4.1, with the best results marked in green. For the model predicting the active power, $T_{window} = 50$ gives the best results, while a lower or higher value slightly increases the average error. The results obtained for $T_{window} = 25$ or $T_{window} = 100$ are nearly identical, but considering more measurement points per training sample more than doubles the training time of $T_{window} = 25$. $T_{window} = 100$ gives the best results in the model for the reactive power, while $T_{window} = 25$ and $T_{window} = 50$ give similar results. Again, a larger observation window increases the training time except for $T_{window} = 50$, where the model was stopped early as it was not improving anymore. The optimal observation window is $T_{window} = 50$ for the equivalent of the active power and $T_{window} = 100$ for the reactive power equivalent.

Table 4.1: Effect of different T_{window} on the model performance.

Target	Twindow	Training	Validation	Test	Training time
P	100	0.013 (± 0.009)	0.011 (± 0.008)	0.010 (± 0.005)	00:28:14
	25	0.012 (± 0.009)	0.009 (± 0.007)	0.010 (± 0.004)	00:10:43
	50	0.006 (± 0.003)	0.005 (± 0.002)	0.006 (± 0.002)	00:12:25
Q	100	0.030 (± 0.031)	0.027 (± 0.022)	0.026 (± 0.013)	00:27:28
	25	0.051 (± 0.034)	0.043 (± 0.024)	0.041 (± 0.017)	00:10:40
	50	0.054 (± 0.043)	0.043 (± 0.031)	0.034 (± 0.019)	00:08:52*

*The training was stopped early as there was no improvement over 25 epochs.

Second, six different GTCNN structures are evaluated: two layers of each ten hidden features (*10-10*), three layers of each of 10 hidden features (*10-10-10*), two layers with more features in the second layer (*10-20*), two layers with fewer features in the second layer (*20-10*), three layers with half the number of features in the next layer (*20-10-5*) and two layers with more hidden features (*20-20*). These structures are evaluated using the previously found optimal observation windows: $T_{window} = 50$ (active power model) and $T_{window} = 100$ (reactive power model). The GTCNN is trained for 100 epochs, and a batch size of 64 is used.

The results are seen in Table 4.2, with the best results marked in green. Increasing the number of hidden features improves the MSE for both equivalents while slightly increasing the training time. Adding an extra layer results in a higher MSE for both equivalents while significantly increasing the training time as more hidden features are added. The best results are obtained for the GTCNN structure of two layers with 20 hidden features for the active power model. The *10-20* and *20-20* GTCNN structures give the best results for the reactive power model, with the *20-20* structure giving a slightly lower standard deviation. Two layers with 20 hidden features are selected as the optimal GTCNN structure for both models.

Table 4.2: Effect of different GTCNN structures on the performance. The numbers indicate the number of hidden features in a layer, and layers are separated by a hyphen (e.g. *10-10* indicates two layers of 10 hidden features each). These results are for the optimal values of T_{window}, i.e. 50 for the active power model and 100 for the reactive power model.

Target	Structure	Training	Validation	Test	Training time
P	10-10	0.011 (± 0.006)	0.010 (± 0.005)	0.011 (± 0.003)	00:11:52
	10-10-10	0.008 (± 0.007)	0.009 (± 0.005)	0.009 (± 0.002)	00:17:25*
	10-20	0.007 (± 0.005)	0.006 (± 0.002)	0.006 (± 0.002)	00:12:33
	20-10	0.007 (± 0.006)	0.006 (± 0.004)	0.004 (± 0.002)	00:12:14
	20-10-5	0.031 (± 0.019)	0.055 (± 0.015)	0.041 (± 0.007)	00:14:17
	20-20	0.007 (± 0.004)	0.005 (± 0.002)	0.007 (± 0.003)	00:12:40
Q	10-10	0.046 (± 0.032)	0.036 (± 0.023)	0.034 (± 0.015)	00:26:43
	10-10-10	0.072 (± 0.046)	0.061 (± 0.038)	0.042 (± 0.017)	00:48:47
	10-20	0.036 (± 0.041)	0.032 (± 0.032)	0.030 (± 0.017)	00:27:45
	20-10	0.043 (± 0.031)	0.033 (± 0.018)	0.032 (± 0.014)	00:28:20
	20-10-5	0.065 (± 0.041)	0.054 (± 0.030)	0.046 (± 0.015)	00:34:30
	20-20	0.027 (± 0.028)	0.032 (± 0.027)	0.022 (± 0.011)	00:28:20

*The training was stopped early as there was no improvement over 25 epochs.

Next, five possible values for the graph filter order K are considered. Three of these consider the same order in both layers (*2-2*, *4-4* and *6-6*). The other two options, *2-4* and *4-2*, show the impact of increasing or decreasing the reach of the graph filter on the second layer. These structures are evaluated using the previously found optimal observation windows and GTCNN structure. The GTCNN is trained for 100 epochs, and a batch size of 64 is used.

The results are seen in Table 4.3, with the best results marked in green. Increasing the graph filter order K results in a larger MSE for both the active and reactive power models while increasing the training time. Moreover, the MSE on the test set is higher for larger K values, indicating that the equivalent model overfits the training and validation data. This is most noticeable for the active power model and the hyperparameter values *4-4* and *6-6* of the reactive power model. The overfitting problem is also shown by the training time, where three active power models with a larger graph filter order are stopped early; the same is observed for the reactive power model with the largest graph filter order (*6-6*). The best results are obtained with two taps in each layer (*2-2*).

Table 4.3: Effect of having different graph filter orders K per layer. These results are for the optimal GTCNN structure, two layers of 20 hidden features, and T_{window} of 50 for the active power model and 100 for the reactive power model.

Target	Filter order K	Training	Validation	Test	Training time
P	2-2	0.008 (± 0.004)	0.006 (± 0.004)	0.007 (± 0.003)	00:10:03
	2-4	0.014 (± 0.010)	0.010 (± 0.008)	0.008 (± 0.005)	00:19:19
	4-2	0.020 (± 0.005)	0.009 (± 0.005)	0.022 (± 0.008)	00:09:48*
	4-4	0.034 (± 0.021)	0.026 (± 0.021)	0.028 (± 0.015)	00:10:54*
	6-6	0.067 (± 0.034)	0.047 (± 0.023)	0.076 (± 0.030)	00:18:52*
Q	2-2	0.034 (± 0.031)	0.032 (± 0.025)	0.026 (± 0.017)	00:19:26
	2-4	0.053 (± 0.040)	0.048 (± 0.027)	0.035 (± 0.019)	00:48:34
	4-2	0.034 (± 0.028)	0.034 (± 0.032)	0.031 (± 0.017)	00:22:36
	4-4	0.123 (± 0.055)	0.085 (± 0.051)	0.086 (± 0.037)	00:46:52
	6-6	1.816 (± 0.317)	1.903 (± 0.303)	1.978 (± 0.287)	00:36:02*

*The training was stopped early as there was no improvement over 25 epochs.

The optimal values found with the hyperparameter tuning are:

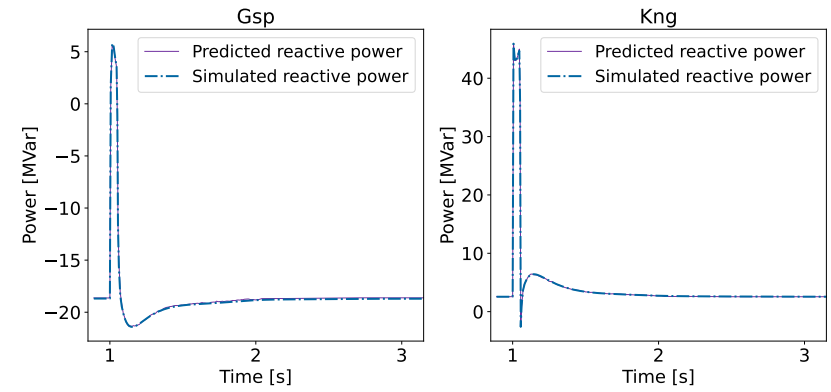
- $T_{window} = 50$ for the active power model and $T_{window} = 100$ for the reactive power model
- two layers of each 20 hidden features
- $K = 2$ for both models

The larger observation windows mean that the GTCNN performs better when there is more data at its input. However, a too-large observation window reduces the performance, likely because of overfitting. Moreover, a simpler GTCNN structure performs better, suggesting overfitting for the more complex structure. These hyperparameters result in 1,921 trainable parameters for the active power model and 2,921 for the reactive power model.

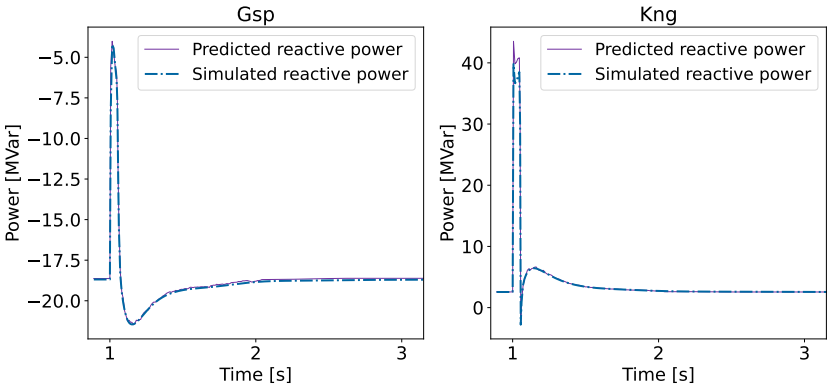
4.5. Performance on different fault locations

An equivalent model for transient stability should accurately represent the dynamic response for different events. In this case study, the performance of the GTCNN is compared to the two baseline LSTM models (*LSTM5* and *LSTM100*) and the hybrid GTCNN-LSTM model. All models receive the same inputs $|V|, \theta, P_0$ and Q_0. The fault events are three-phase short circuits, represented by a fault impedance $Z = 0$ [50]. The fault occurs 1 second into the simulation, with a clearing time of 50 ms (two and a half cycles), corresponding with current protection devices [95]. The Zeeland 50 kV ring is stable, meaning its control systems can reach a new steady state after the fault event has been cleared. The network is simulated for 7 seconds, allowing the Zeeland 50 kV ring to return to a steady state; this duration is similar to previous works on equivalent modelling [17], [34], [41], [44] and focuses on the short-term dynamics. The PMU measurements are recorded from PowerFactory simulations using a 100 Hz sampling rate, corresponding to the highest standard PMU reporting rate for a 50 Hz system [96]. The measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. Different fault events are obtained by choosing a different fault location (i.e. the cable where the fault occurs) and the fault percentage (the location in the selected cable). 100 faults are generated for the training set with the fault percentages sampled from the uniform distribution $f_{\%} \sim U(10, 90)$. The validation set consists of 30 faults with fault percentages sampled from the same distribution. The test set consists of 10 faults, with the faults percentages sampled from $f_{\%} \sim L_{F_{\%}} = \{2, 5, 30, 60, 95, 99\}$, meaning the test set also includes percentages outside the training and validation sets. The models are trained on the DelftBlue Supercomputer on the GPU type-a node using a single CPU core and 12 GB of RAM [97].

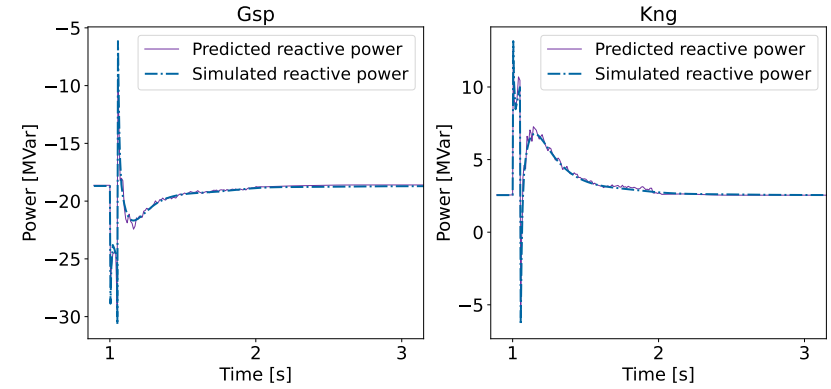
The results are seen in Table 4.4, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. Four scenarios with different MSEs are plotted in Fig. 4.7 to visualise how different MSEs impact the prediction accuracy. The prediction with an MSE of 0.004 can accurately capture the dynamic behaviour, including around the extremes (see Fig. 4.7a). An MSE of 0.049 follows the majority of the simulated output accurately, but there are bigger deviations of the predicted response near the fault event (see Fig. 4.7b). An even higher MSE of 0.136 shows differences in the predicted and simulated output throughout larger portions of the scenario (see Fig. 4.7c). An MSE of 3.632 does not capture the dynamics accurately as it does not reach the correct peak values during the response (see Fig. 4.7d).



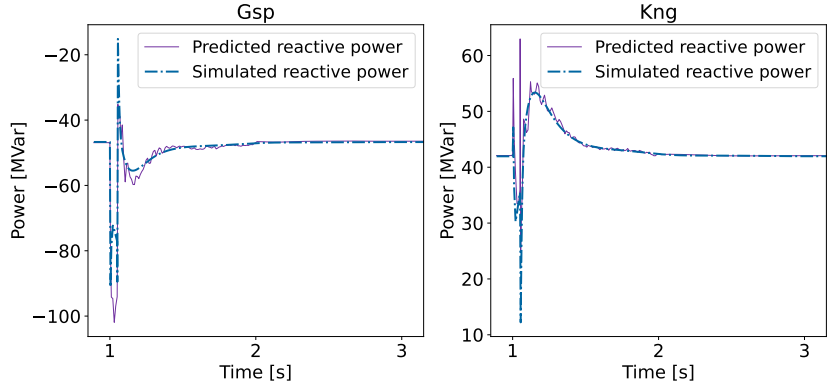
(a) MSE of 0.004



(b) MSE of 0.049



(c) MSE of 0.136



(d) MSE of 3.632

Figure 4.7: Impact of the MSE error on the prediction accuracy of four scenarios.

Table 4.4: Performance of the four models for different fault locations

Target	Model	Training	Validation	Test	Training time
P	GTCNN	0.002 (± 0.002)	0.002 (± 0.002)	0.003 (± 0.002)	00:13:06
	GTCNN-LSTM	0.008 (± 0.021)	0.006 (± 0.014)	0.006 (± 0.006)	02:14:07
	LSTM100	0.010 (± 0.023)	0.009 (± 0.018)	0.009 (± 0.009)	02:13:47
	LSTM5	0.008 (± 0.010)	0.007 (± 0.011)	0.006 (± 0.005)	02:05:40
Q	GTCNN	0.016 (± 0.018)	0.015 (± 0.013)	0.011 (± 0.004)	00:13:27
	GTCNN-LSTM	0.011 (± 0.028)	0.011 (± 0.021)	0.009 (± 0.006)	02:14:41
	LSTM100	0.014 (± 0.033)	0.014 (± 0.028)	0.007 (± 0.005)	02:13:48
	LSTM5	0.017 (± 0.014)	0.017 (± 0.013)	0.016 (± 0.012)	02:05:38

From Tab. 4.4, several things can be observed. The GTCNN performs the best for the active power prediction, while the LSTM100 performs the best for the reactive power model. However, all the models are performing fairly similarly. The GTCNN has a smaller deviation than the other models and is trained 8x faster than the equivalents with an LSTM model. These results show that the GTCNN can predict the active power response of the ADN to a three-phase short circuit with different fault locations most accurately, while its performance is slightly worse for the reactive power. However, a model with an MSE of 0.011 still gives an accurate prediction, as seen in Fig. 4.7. The lower spread of the GTCNN suggests a more consistent performance for the different fault locations and, therefore, a performance closer to the mean MSE. Moreover, the MSE error does not increase between the training, validation and test sets, indicating a good generalisation performance of the model and no overfitting in any of the models. This consistency in performance showcases that the models can accurately predict the dynamic response to unseen faults.

4.6. Performance on different operating conditions

The equivalent model is also studied under changing operating conditions to determine if the equivalent model is robust when the conditions in the ADN change. This is done with a different dataset that contains different setpoints for the six loads and three WTGs in the Zeeland 50 kV ring. These setpoints are sampled from the Kumaraswamy distribution $oc \sim \mathcal{K}(2,2)$ with the lower and upper bounds scaled to $x \in (\mu - 3\sigma, \mu + 3\sigma)$. The loads are correlated by the correlation coefficient $c_{load} = 0.6$, while the WTGs use $c_{WTG} = 0.75$. A stronger correlation is assumed for the WTGs since they share the same meteorological conditions because of their geographical proximity, while the inhabitants of different cities will behave slightly differently from each other. To account for varying levels of production and consumption throughout the year, two seasons (summer and winter) and three day-quarters (15, 27 and 70) are considered; the day-quarters correspond to low (15), base (28) and peak consumption (70). Multiple samples are drawn from the distribution K(2,2) for every season and day quarter combination: 30 for the training set, 20 for the validation set and 15 for the test set. The fault event is a three-phase short-circuit fault in the transmission line *Zwart Kng150z-Kng152* at three different percentages: $f_{\%} = \{25, 50, 75\}$[13]. These settings yield 540 training, 360 validation and 270 test scenarios. Each scenario is simulated for 7 seconds, with the fault event occurring at 1 s and being cleared in 50 ms. The measurements are collected at a 100 Hz sampling rate, and the measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. The models are trained on the DelftBlue Supercomputer on the GPU type-a node using a single CPU core and 12 GB of RAM [97].

The results are seen in Table 4.5, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. The GTCNN and LSTM5 models perform the best for the active power equivalent, with the GTCNN having a slightly lower standard deviation. The LSTM100 model is not far behind. The hybrid GTCNN-LSTM model has a poorer performance with a 6x loss increase. For the reactive power equivalent, the LSTM-based models perform similarly and perform the best out of all models. The GTCNN model has a 2x higher loss than the other equivalents, but its training time is 4x shorter. The MSE is very low for all models, meaning they can reproduce the ADN's dynamic response accurately under varying network operating

[13]The fault percentages are not sampled, but chosen sequentially. This leads to three fault events.

Table 4.5: Performance of the four models for different operating conditions.

Target	Model	Training	Validation	Test	Training time
P	GTCNN	0.004 (± 0.004)	0.005 (± 0.006)	0.005 (± 0.005)	01:27:08
	GTCNN-LSTM	0.017 (± 0.014)	0.038 (± 0.071)	0.032 (± 0.041)	05:52:52**
	LSTM100	0.005 (± 0.004)	0.008 (± 0.014)	0.008 (± 0.009)	05:43:47**
	LSTM5	0.004 (± 0.003)	0.006 (± 0.009)	0.005 (± 0.005)	05:52:22**
Q	GTCNN	0.022 (± 0.013)	0.023 (± 0.019)	0.026 (± 0.025)	01:37:23*
	GTCNN-LSTM	0.009 (± 0.006)	0.014 (± 0.033)	0.012 (± 0.013)	05:52:33**
	LSTM100	0.008 (± 0.006)	0.010 (± 0.012)	0.010 (± 0.010)	05:46:16**
	LSTM5	0.008 (± 0.006)	0.011 (± 0.013)	0.011 (± 0.009)	05:55:20**

*The training was stopped early not to exceed the requested runtime.
**The training was stopped early not to exceed the maximum training time of 6 hours.

conditions using a single model instead of requiring different models per operating condition. The performance for the reactive power response is slightly worse, but it still yields an accurate response when looking at Fig. 4.7. This is also seen by the prediction of the reactive power model of test scenario 88 in Fig. 4.8 with an MSE of 0.09. The MSE does not vary significantly between the training, validation, and test sets, indicating that the equivalents can generalise well to unseen operating conditions and that a single equivalent can be used for different operating conditions.

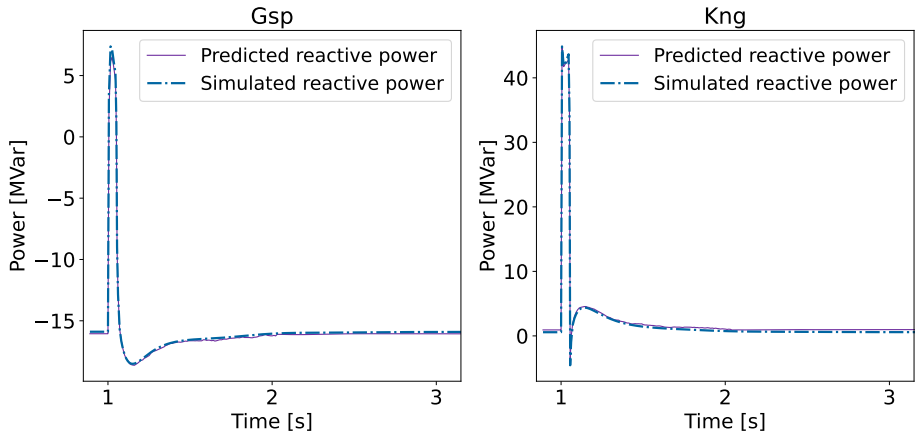


Figure 4.8: The predicted reactive power dynamic response of test scenario 88 with an MSE of 0.09 by the best GTCNN-based equivalent model.

4.7. Performance on different topological changes

The equivalent model should also work accurately when the DN is changing. To evaluate the robustness, a different dataset is used focusing on hidden topological changes, i.e. topological changes that are not directly observed by the TSO and, therefore, are not an input to the equivalent model. The two types considered in this case study are activating one inactive 50 kV transformer inside the Zeeland 50 kV ring or changing the tap positions of one active 50 kV transformer. There are five active 50/10 kV transformers in the Zeeland 50 kV ring: *51 Gse, 51 Zrz, 51 Otl, 51 Tln* and *51 Kng*. Moreover, five inactive transformers function as redundancy transformers to their active counterpart: 52 Gse, 52 Zrz, 52 Otl, 52 Tln, and *52 Kng*. A single component is sampled randomly, and the corresponding action is implemented. To create the training, validation and test sets, a component is uniquely defined in one of the sets. The training set has the tap-changeable transformers *51 Gse, 51 Zrz* and *51 Kng* and the inactive transformers *52 Otl, 52 Tln* and *52 Kng*. The validation set contains the tap-changeable transformer *51 Otl* and inactive transformer *52 Gse*. The test set has the tap-changeable transformer *51 Tln* and the inactive transformer *52 Zrz*. The fault event is a three-phase short-circuit fault in the

transmission line *Zwart Kng150z-Kng152* at three different percentages: $f_{\%} = \{25, 50, 75\}$[14]. These settings yield 48 training, 18 validation and 18 test scenarios[15]. Each scenario is simulated for 7 seconds, with the fault event occurring at 1 s and being cleared in 50 ms. The measurements are collected at a 100 Hz sampling rate, and the measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. The models are trained on the DelftBlue Supercomputer on the GPU type-a node using a single CPU core and 12 GB of RAM [97].

Table 4.6: Performance of the four models for different topological changes.

Target	Model	Training	Validation	Test	Training time
P	GTCNN	0.002 (± 0.002)	0.002 (± 0.001)	0.002 (± 0.001)	00:07:02*
	GTCNN-LSTM	0.004 (± 0.003)	0.004 (± 0.003)	0.005 (± 0.003)	01:09:12
	LSTM100	0.005 (± 0.002)	0.006 (± 0.003)	0.005 (± 0.003)	00:31:12**
	LSTM5	0.003 (± 0.001)	0.004 (± 0.002)	0.004 (± 0.003)	01:03:36
Q	GTCNN	0.012 (± 0.011)	0.014 (± 0.010)	0.016 (± 0.010)	00:06:00*
	GTCNN-LSTM	0.025 (± 0.025)	0.021 (± 0.009)	0.030 (± 0.027)	00:30:39**
	LSTM100	0.029 (± 0.027)	0.025 (± 0.011)	0.029 (± 0.017)	00:34:05**
	LSTM5	0.015 (± 0.017)	0.014 (± 0.007)	0.015 (± 0.012)	01:04:03

*The training was stopped early as there was no improvement over 100 epochs.
**The training was stopped early as there was no improvement over 20 epochs.

The results are seen in Table 4.6, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. The GTCNN is the best-performing equivalent for the active power model and has the best performance together with the LSTM5 model for the reactive power equivalent. This means that the GTCNN equivalent can reproduce the dynamic response of the DN accurately under the different hidden topological changes. Moreover, there is no significant increase in MSE between the training, validation and test set, indicating that a single model can capture a broad range of hidden topological changes. The MSE of the reactive power equivalents is slightly higher but still produces accurate results when comparing it with Fig. 4.7. This is also seen by the prediction of test scenario 11 in Fig. 4.9, which has an MSE of 0.01. In both cases, the GTCNN trained much faster than the LSTM-based models. Two models (GTCNN and LSTM100) were stopped early for the active power equivalent, while three models (GTCNN, LSTM100 and GTCNN-LSTM) were stopped early for the reactive power model; all these models share an observation window T_{window} of 100 measurement points.

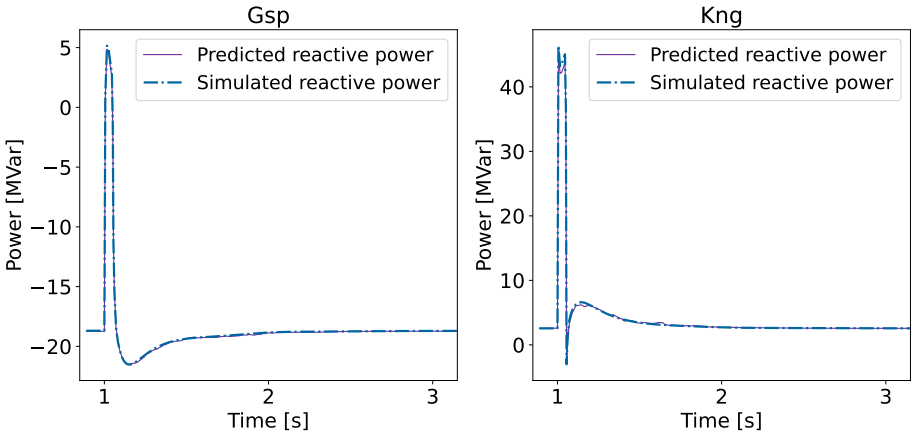


Figure 4.9: The predicted reactive power dynamic response of test scenario 11 with an MSE of 0.01 by the best GTCNN-based equivalent model. Test scenario 11 has a three-phase short-circuit fault event in *Zwart Kng150z-Kng152* at a fault percentage of 50% and the tap position of *51 Tln* has been changed to -4.

[14]The fault percentages are not sampled, but chosen sequentially. This leads to three fault events.
[15]There are fewer scenarios than all possible options since not all topological changes result in convergent power flows.

4.8. Design considerations

This section contains four case studies validating the following design choices: including initial power injections P_0 and Q_0 in the GTCNN-based equivalent model, creating a heterogenous GTCNN, choosing the Cartesian product graph and predicting the output at individual PCC terminals.

4.8.1. Effect of including the initial power injections

To evaluate the effect of adding the initial power injections, the datasets of sections 4.5 (faults), 4.6 (operating conditions) and 4.7 (*hidden* topological changes) are used. The GTCNN with initial power injections is compared against the LSTM5 model without the initial power injections[16]. Both models have the same hyperparameters as described in sections 4.3 and 4.4, except that the GTCNN-based equivalent model for active power has an observation window of $T_{window} = 100$.

The results for different fault locations are seen in Table 4.7, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. From Tab. 4.7, it can be seen that the GTCNN performs the best for the active power equivalent and the LSTM5 for the reactive power equivalent. In both cases, the GTCNN has a smaller spread of the MSE for the different scenarios. The GTCNN and LSTM models have an MSE below 0.011, meaning both models can capture the dynamic response accurately. This comparable performance suggests that including the initial power injections P_0 and Q_0 has a negligible effect when only considering different fault events.

Table 4.7: Performance of the GTCNN with initial power injections P_0 and Q_0 and the LSTM-based model without initial power injections for different fault locations.

Target	Model Type	Training	Validation	Test
P	GTCNN	0.002 (± 0.002)	0.002 (± 0.002)	0.003 (± 0.002)
	LSTM5	0.004 (± 0.009)	0.005 (± 0.009)	0.004 (± 0.005)
Q	GTCNN	0.016 (± 0.018)	0.015 (± 0.013)	0.011 (± 0.004)
	LSTM5	0.009 (± 0.017)	0.009 (± 0.012)	0.007 (± 0.006)

The results for different operating conditions are seen in Table 4.8, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. The GTCNN and LSTM5 models perform similarly for the active power, but the LSTM5 model performs poorer than the GTCNN for the reactive power equivalent. Therefore, including the initial setpoints P_0 and Q_0 helps the GTCNN to better learn the difference in operating conditions.

Table 4.8: Performance of the GTCNN with initial setpoints P_0 and Q_0 and the LSTM-based model without initial setpoints for different operating conditions.

Target	Model Type	Training	Validation	Test
P	GTCNN	0.004 (± 0.004)	0.005 (± 0.006)	0.005 (± 0.005)
	LSTM5	0.006 (± 0.005)	0.005 (± 0.005)	0.005 (± 0.004)
Q	GTCNN	0.022 (± 0.013)	0.023 (± 0.019)	0.026 (± 0.025)
	LSTM5	0.286 (± 0.249)	0.341 (± 0.341)	0.352 (± 0.306)

The results for different *hidden* topological changes are seen in Table 4.9, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. The GTCNN is better than the LSTM5 model for both the active and reactive power equivalents. However, the performance of the LSTM5 model is still acceptable as an MSE below 0.03 can still reproduce the dynamics accurately (see Fig. 4.7). The better performance of the GTCNN shows that including the initial power injections P_0 and Q_0 results in better performance under different *hidden* topological changes.

[16] This model is the closest to the one in [44].

Table 4.9: Performance of the GTCNN with initial power injections P_0 and Q_0 and the LSTM-based model without initial power injections for different topological changes.

Target	Model Type	Training	Validation	Test
P	GTCNN	0.002 (± 0.002)	0.002 (± 0.001)	0.002 (± 0.001)
	LSTM5	0.003 (± 0.002)	0.003 (± 0.002)	0.003 (± 0.001)
Q	GTCNN	0.012 (± 0.011)	0.014 (± 0.010)	0.016 (± 0.010)
	LSTM5	0.032 (± 0.028)	0.026 (± 0.006)	0.030 (± 0.013)

4.8.2. Performance of the heterogenous GTCNN

The proposed GTCNN-based equivalent model is a heterogeneous GNN, as it has two types of nodes, each with a different input. The PCC nodes have the inputs $|V|$ and θ, which vary over time, while the internal nodes receive the static initial power injections P_0 and Q_0. This network is known as the heterogeneous GTCNN. By design, the GCN with filter banks is homogeneous, meaning it expects the same features at every node. Therefore, the initial configuration that was evaluated consisted of a single node type, which received all available measurements: the voltage magnitude $|V|$, the voltage angle θ, the current I, the line loading L_{line} and the initial active and reactive power injections P_0 and Q_0 (see Fig. 4.10). The performance of these two GTCNN types is evaluated using the optimal hyperparameters of section 4.4 and the dataset with fault events of section 4.5.

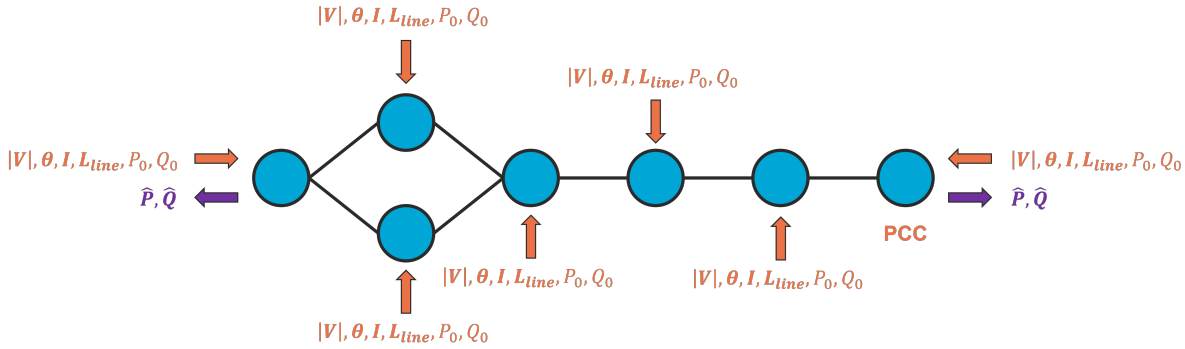


Figure 4.10: The homogenous GTCNN with the dynamic features $|V|, \theta, I$ and L_{line} and the static initial power injections P_0 and Q_0 at every node.

The results are seen in Table 4.10, with the best results marked in green. The heterogeneous GTCNN performs better for the active power model than the homogeneous GTCNN. This difference could be caused by the homogeneous model slightly overfitting to the training and validation splits, which is also observed by the slight increase in average MSE on the test split from 0.001 to 0.004. Nevertheless, the MSE of both models is similar, and both are suitable as equivalent models. For the reactive power model, the homogeneous GTCNN performs better than the heterogeneous GTCNN (0.003 instead of 0.02). The MSE of the homogeneous GTCNN is similar for both the active and reactive power models, while that of the heterogeneous GTCNN increases by a factor of 10. This difference suggests that having less information in heterogenous GTCNN will reduce the model performance. However, an MSE of 0.02 still yields a good prediction of the dynamic response (see Fig. 4.7). Moreover, having these measurements as inputs to all nodes is unrealistic for an equivalent model, where the inputs will come from the TN model. These measurements would only be available at all nodes if the TN is connected to the DN at all points, but then there will be no point in having an equivalent model as the TSO could directly observe the outputs P and Q. Therefore, heterogeneous GTCNN is preferred over homogeneous GTCNN.

Table 4.10: Performance comparison of the heterogeneous GTCNN (current implementation) and the homogeneous GTCNN, which includes the dynamic features $|V|, \theta, I$ and L_{line} and the static initial power injections P_0 and Q_0 at every node.

Target	GTCNN Type	Training	Validation	Test
P	Heterogeneous	0.002 (± 0.002)	0.002 (± 0.002)	0.002 (± 0.001)
	Homogeneous	0.001 (± 0.001)	0.001 (± 0.000)	0.004 (± 0.003)
Q	Heterogeneous	0.023 (± 0.023)	0.024 (± 0.027)	0.020 (± 0.018)
	Homogeneous	0.004 (± 0.003)	0.003 (± 0.002)	0.003 (± 0.002)

4.8.3. Impact of different product graphs

In section 3.3.1, the Cartesian product graph was selected as it has a more realistic representation of the power system. However, it is possible that considering the extra relations (temporal dependency on past neighbours) of the strong product graph yields a better performance. These two product graph types are compared using the previously found optimal observation windows, GTCNN structure and graph filter order K. The GTCNN is trained on different fault events for a maximum of 250 epochs, and a batch size of 64 is used.

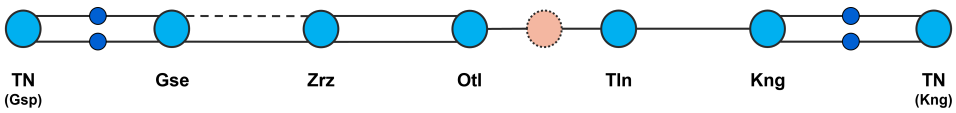
The results are seen in Table 4.11, with the best results marked in green. The Cartesian product graph gives a significantly lower MSE for both the active and reactive power equivalents, as it can train much lower before the model stops improving. Therefore, the Cartesian product graph is kept as the product graph type.

Table 4.11: Effect of using a parametric or strong product graph. These results are for the optimal GTCNN structure, 2 layers of 20 hidden features with a filter order of 2, and T_{window} of 50 for the active power model and 100 for the reactive power model.

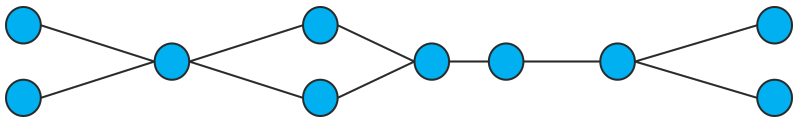
Target	Product type	Training	Validation	Test	Epochs trained
P	Cartesian	0.004 (± 0.002)	0.004 (± 0.003)	0.008 (± 0.006)	172
	Strong	0.042 (± 0.029)	0.036 (± 0.018)	0.052 (± 0.030)	52
Q	Cartesian	0.010 (± 0.005)	0.010 (± 0.004)	0.008 (± 0.004)	237
	Strong	0.016 (± 0.013)	0.015 (± 0.010)	0.015 (± 0.006)	140

4.8.4. Performance in predicting individual terminals

The Zeeland 50 kV ring has two terminals at each PCC to increase its reliability (as seen in Fig. 4.1). Throughout this thesis, the GTCNN is predicting the outputs of the two PCCs, meaning that the two terminals at each PCC are merged into one by taking the first value of the voltage magnitude $|V|$ and the phase angle θ, or summing the active power P and reactive power Q. The performance of the GTCNN in predicting the dynamic response at each terminal is investigated using the optimal hyperparameters of section 4.4 and the dataset with fault events of section 4.5. The corresponding graphs $\mathcal{G}_{modified}$ and $\mathcal{G}_{GTCNN}$ are shown in Fig. 4.11.



(a) The graph $\mathcal{G}_{modified}$ of the Zeeland 50 kV ring with the four terminals ( ).



(b) Modified line graph $\mathcal{G}_{GTCNN} = L(\mathcal{G}_{modified})$ of the Zeeland 50 kV ring.

Figure 4.11: The modified graph $\mathcal{G}_{modified}$ and the corresponding GTCNN graph $\mathcal{G}_{GTCNN}$ of the Zeeland 50 kV ring with four terminals.

The results are seen in Table 4.9, with the best results marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. For the active power model, the GTCNN is equally good at predicting the dynamic response at the individual terminals and the PCCs. The average MSE at individual terminals is slightly lower, but the spread is higher, meaning the average MSEs are similar. Both models yield accurate responses since their MSE is low (as seen in Fig. 4.7). However, the reactive power model predicting at the terminals has a very high average MSE of 11.93. The GTCNN-based equivalent model, in that case, is not usable, as it cannot capture the dynamics accurately.

Table 4.12: Performance of the GTCNN in predicting the dynamic response at each terminal or at the PCCs.

Target	Prediction at	Training	Validation	Test	Training time
P	PCC	0.002 (± 0.002)	0.002 (± 0.002)	0.002 (± 0.001)	00:21:32
	terminal	0.018 (± 0.010)	0.021 (± 0.011)	0.018 (± 0.011)	00:27:55
Q	PCC	0.023 (± 0.023)	0.024 (± 0.027)	0.020 (± 0.018)	00:49:15
	terminal	10.088 (± 5.884)	12.801 (± 5.344)	11.927 (± 5.536)	01:08:44

In Figs. 4.12 and 4.13, the reactive power predictions are shown at the two terminals of the PCCs *Gsp-Gse* and *Kng*, respectively. The prediction of the terminals *Wit Gsp-Gse* and *Zwart Gsp-Gse* follows the simulated dynamic response closely (see Fig. 4.12). In contrast, in Fig. 4.13, the prediction of the 151 Kng and 152 Kng terminals is inaccurate (*151 Kng* has the overshoot in the wrong direction). The difference between these two PCCs (*Gsp* and *Kng*) is that the terminals at *Gsp* have a similar response, while the terminals at *Kng* have opposite responses[17]. The GTCNN predicts a similar dynamic response at each terminal, which is in the middle of the two overshoots. Therefore, the GTCNN tries to learn a single response for the two terminals. As two measurement nodes that correspond to the terminals *151 Kng* and *152 Kng* are close to each other in the graph $\mathcal{G}_{GTCNN}$ (a two-hop distance), these results suggest that the GTCNN is unable to learn different dynamic responses at nodes that are in each other's proximity. Therefore, merging the PCCs yields a better performance.

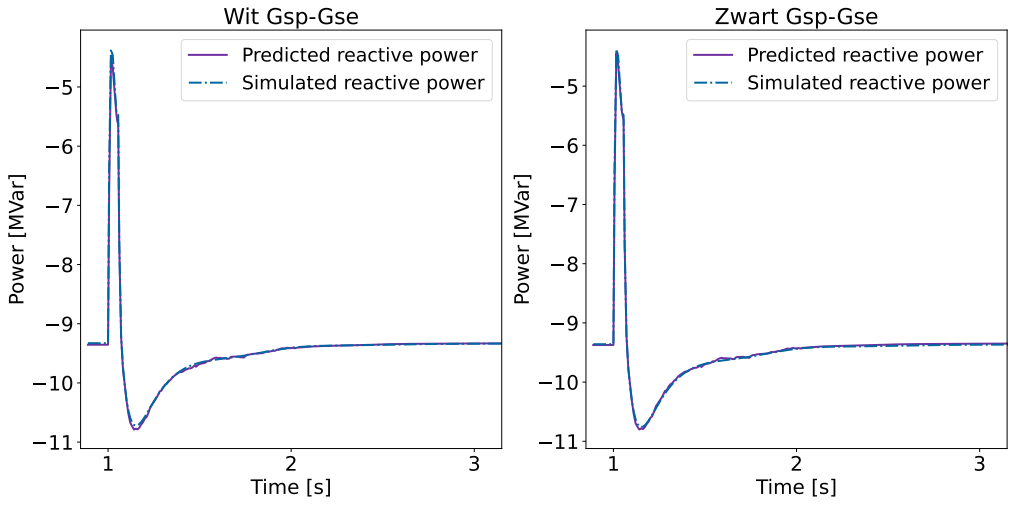


Figure 4.12: Predicted sequence by the reactive power equivalent at each terminal of PCC *Gsp-Gse*.

[17]The different sign in reactive power at *Kng* indicates an opposite flow of reactive power at each terminal.

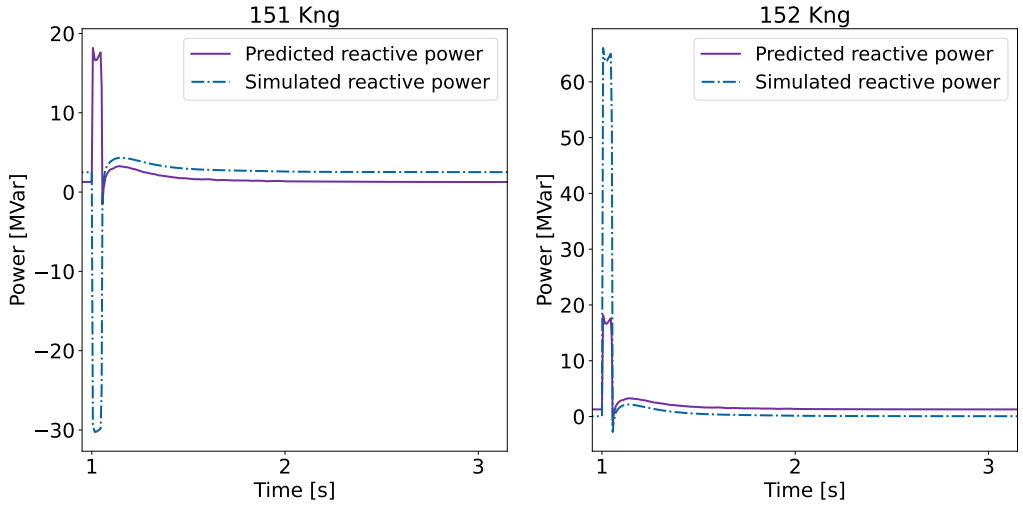


Figure 4.13: Predicted sequence by the reactive power equivalent at each terminal of PCC *Kng*.

4.9. Scaling to larger systems

The scaling performance of the GTCNN against the different model types is evaluated by extending the Zeeland 50 kV ring with more nodes. This extension is realised by copying the 50 kV buses *Zrz*, *Otl* and *Tln*. Moreover, two interconnectors are added between the original 50 kV ring and the copy to create a more meshed network; each interconnector also has a PMU device. This results in an additional eight nodes per copy of the Zeeland 50 kV ring. The Zeeland triple 50 kV ring has been created for evaluating the scaling behaviour by copying the 50 kV buses *Zrz*, *Otl* and *Tln* twice (see Fig. A.3). The modified graph is seen in Fig. A.1, together with its modified line graph $\mathcal{G}_{GTCNN}$ (see Fig. A.2). The number of nodes in $\mathcal{G}_{GTCNN}$ is increased from 8 in the Zeeland 50 kV ring to 24 in the Zeeland triple 50 kV ring. The network is still stable because of the various control systems, but their parameters are not optimal, as these were tuned for the original Zeeland 50 kV ring. The 150/50/10 kV transformer tap positions are changed to 20 to yield an internal pre-fault voltage level of 1 pu. Moreover, the WTG transformer ratings are doubled. Three datasets were created to evaluate the scaling performance of the proposed GTCNN-based equivalent model. Dataset 1 has only faults. Dataset 2 has predominantly operating conditions. Dataset 3 focuses on topological changes. The models are trained on the DelftBlue Supercomputer on the GPU type-a node using a single CPU core and 12 GB of RAM [97].

The **first dataset** contains only three-phase short-circuit fault events at different fault locations and percentages. 100 faults are generated for the training set with the fault percentages sampled from the uniform distribution $f_{\%} \sim U(10, 90)$. The validation set consists of 30 faults with fault percentages sampled from the same distribution. The test set consists of 10 faults, with the faults percentages sampled from $f_{\%} \sim L_{F_{\%}} = \{2, 5, 30, 60, 95, 99\}$, meaning the test set also includes percentages outside the training and validation sets. Each scenario is simulated for 7 seconds, with the fault event occurring at 1 s and being cleared in 50 ms. The measurements are collected at a 100 Hz sampling rate, and the measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. The detailed results for this dataset are found in Tab. A.1.

The **second dataset** contains different operating conditions under three fault events. The fault event is a three-phase short-circuit fault in the transmission line *Zwart Kng150z-Kng152* at three different percentages: $f_{\%} = \{25, 50, 75\}$. To account for varying levels of production and consumption throughout the year, two seasons (summer and winter) and three day-quarters (15, 27 and 70) are considered; the day-quarters correspond to low (15), base (28) and peak consumption (70). Multiple samples are drawn from the distributions $\mathcal{K}(2, 2)$ for every season and day quarter combination: 30 for the training set, 20 for the validation set and 15 for the test set. The loads are correlated by the correlation coefficient $c_{load} = 0.6$, while the WTGs use $c_{WTG} = 0.75$. The lower and upper bounds are scaled to $x_{load} \in (\mu - \sigma, \mu + \sigma)$ and $x_{WTG} \in (\mu - 0.25\sigma, \mu + 0.25\sigma)$[18]. These settings yield 318 training, 209

[18]The support x is smaller for the Zeeland triple 50 kV ring to give more converging scenarios. Because the control systems are

validation and 160 test scenarios[19]. Each scenario is simulated for 7 seconds, with the fault event occurring at 1 s and being cleared in 50 ms. The measurements are collected at a 100 Hz sampling rate, and the measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. The detailed results for this dataset are found in Tab. A.2.

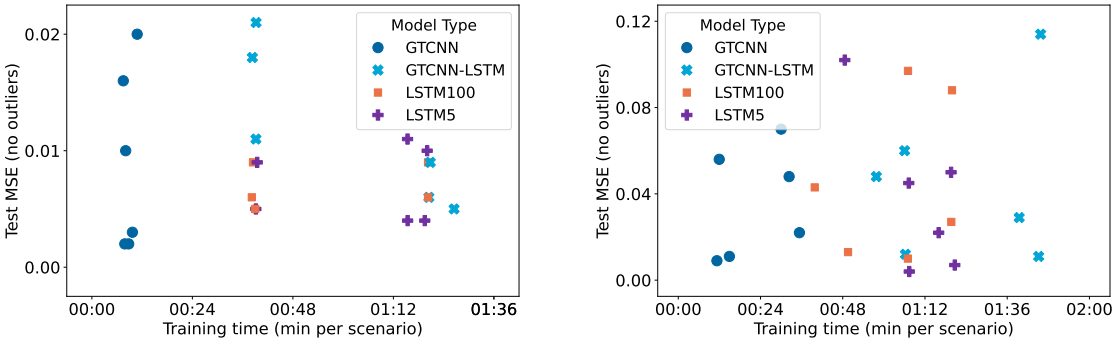
The **third dataset** contains different topological changes under three fault events. The fault event is a three-phase short-circuit fault in the transmission line *Zwart Kng150z-Kng152* at three different percentages: $f_{\%} = \{25, 50, 75\}$. The training set has the tap-changeable transformers *51 Gse*, *51 Tln*, *51 Kng*, *2nd 51 Zrz*, *2nd 51 Tln*, *3rd 51 Otl*, and *3rd 51 Tln*, and the inactive transformers *52 Zrz*, *52 Tln*, *52 Kng*, *2nd 52 Otl*, *2nd 52 Tln*, *3rd 52 Zrz*, and *3rd 52 Otl*. The validation set contains tap-changeable transformers *51 Otl* and *3rd 51 Zrz*, and the inactive transformers *52 Gse* and *2nd 52 Zrz*. The test set has the tap-changeable transformers *51 Zrz* and *2nd 51 Otl*, and the inactive transformers *52 Otl* and *3rd 52 Tln*. These settings yield 108 training, 30 validation and 30 test scenarios. Each scenario is simulated for 7 seconds, with the fault event occurring at 1 s and being cleared in 50 ms. The measurements are collected at a 100 Hz sampling rate, and the measurements at the individual PCC terminals are merged for the two connection points *Gsp* and *Kng*. The detailed results for this dataset are found in Tab. A.3.

The comparison of the average test MSE (without outliers) from each dataset and each output (P and Q) of the original network and the Zeeland triple 50 kV ring can be seen in Fig. 4.14. The training time is normalised against the total number of training scenarios in the dataset[20] to account for the reduced number of scenarios obtained from the Zeeland triple 50 kV ring. The GTCNN has the fastest training time for both networks, while the LSTM-based networks are slower. This difference in speed has two implications. First, the GTCNN-based equivalent can be trained faster for the same number of scenarios, which means that the equivalent model can be exchanged more often between the TSO and DSO, ensuring a more actual model. Second, more scenarios can be included in the training for the same training time, resulting in an equivalent model that has seen more scenarios. The performance of the GTCNN is in the range of the other models for both networks, which implies a similar performance of the GTCNN against existing approaches [44]. The MSE of all models is below 0.02 for the Zeeland 50 kV ring (see Fig. 4.14a), while the MSE is below 0.08 for the Zeeland triple 50 kV ring (see Fig. 4.14b). Both these MSEs give an equivalent output that follows the simulated output pretty closely (see Fig. 4.7). This makes the proposed GTCNN-based equivalent suitable for usage as a dynamic equivalent model.

not optimally tuned, the system is not able to fully recover from a fault when the operating conditions are too far away from their original values. Having a smaller support still allows for the generation of different operating conditions while keeping the system stable. The WTGs have a smaller support since the generated reactive power caused frequent overloading of the various cables and transformers.

[19] There are less scenarios for the Zeeland triple 50 kV ring than the original Zeeland 50 kV ring since not all scenarios resulted in convergent pre-fault power flows.

[20] Training a machine learning model has two steps: training and inference. First, the training data is used to calculate the loss and update the network weights through backpropagation (training). Second, the validation and test data are passed through the network to find its performance on unseen data (inference). The training step takes the longest to run since all weight gradients have to be computed during the backpropagation to update the weights. As the exact difference in computation time between the training and inference steps is unknown, it is chosen to normalise only against the step that takes the longest, i.e. the training step.

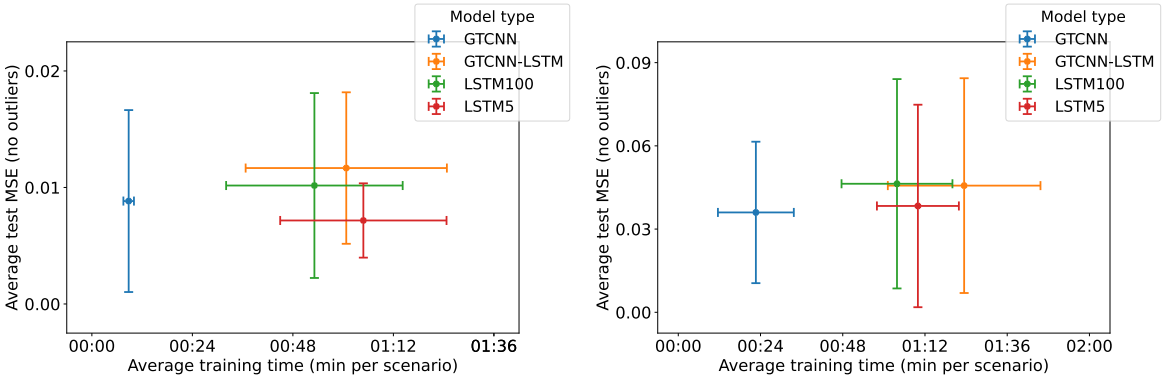


(a) Zeeland 50 kV ring

(b) Zeeland triple 50 kV ring

Figure 4.14: The training time vs the average MSE per dataset of the different equivalent models.

The average performance across all datasets of each model is seen in Fig. 4.15. The LSTM5 model has the lowest average MSE and standard deviation for the Zeeland 50 kV ring (see Fig. 4.15a), which means that this model has the most consistent performance across the different datasets and, therefore, the different type of changing system conditions. The GTCNN is slightly better than the LSTM100 network, while the hybrid GTCNN-LSTM has the highest average MSE. These models have a higher spread, meaning their output varies more between the different system conditions. However, the highest MSE is slightly above 0.02. Therefore, the higher output variability does not significantly impact the accuracy of the equivalent models for the Zeeland 50 kV ring. The GTCNN performs better for the Zeeland triple 50 kV with a slightly lower average MSE than the LSTM5 model and a lower spread than the LSTM-based equivalents. These results mean that the GTCNN-based equivalent produces an output that is slightly closer to the simulated output than those of the GTCNN-based models, although all of them produce fairly accurate responses when considering the impact of the MSE on the accuracy of the models (Fig. 4.7). The GTCNN also has a lower average training time and shows less variability in the average training time per scenario than the LSTM-based models, meaning that the GTCNN model can be trained quicker, and it is more certain how long it will take to train the model.



(a) Zeeland 50 kV ring

(b) Zeeland triple 50 kV ring

Figure 4.15: The average training time vs the average model performance (MSE) across all datasets. The whiskers indicate the first standard deviation of each model.

Fig. 4.16 shows how the MSE of the different models changes with the number of measurement nodes. This figure is created with the results of the different datasets and outputs of each model for the two networks (the datapoints of Figures 4.14a and 4.14b). The average MSE of the models is initially 0.01 and increases above 0.03 (which was also seen in Fig. 4.15), which means that the accuracy of all models decreases slightly as the number of nodes increases. The GTCNN and LSTM5 have a lower average MSE than the other models. The GTCNN has the smallest and lowest 95% confidence interval, suggesting the GTCNN performs the best on the bigger network, although this cannot be said with certainty as the confidence intervals of the models overlap.

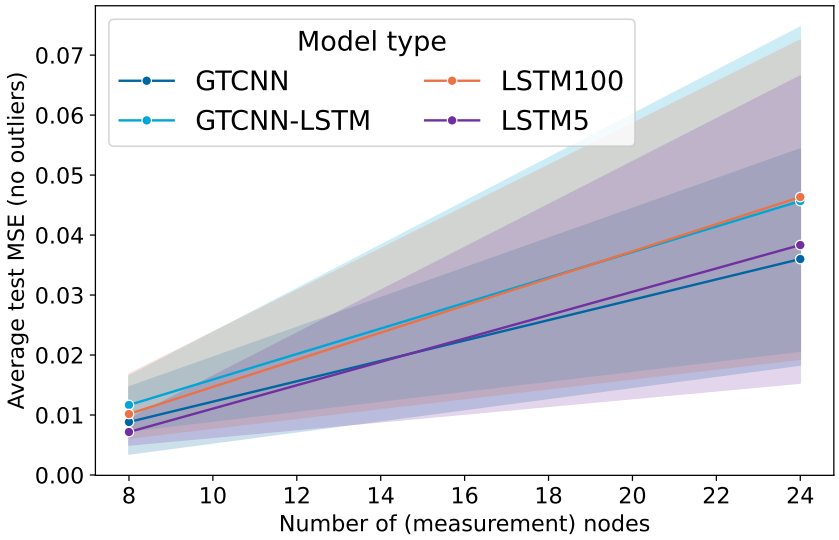


Figure 4.16: The increase in MSE with the number of measurement nodes for each model type. The solid line is the central tendency (average) of the different datasets and model targets (P or Q), while the shaded areas show the confidence interval.

The models have a different rate of MSE increase per number of measurement nodes, as seen in Fig. 4.17. The MSE of the GTCNN increases the slowest, followed by the LSTM5 and LSTM100 networks, with the hybrid GTCNN-LSTM network having the highest slope. This suggests that the GTCNN retains better accuracy than the other models as the number of measurement nodes grows.

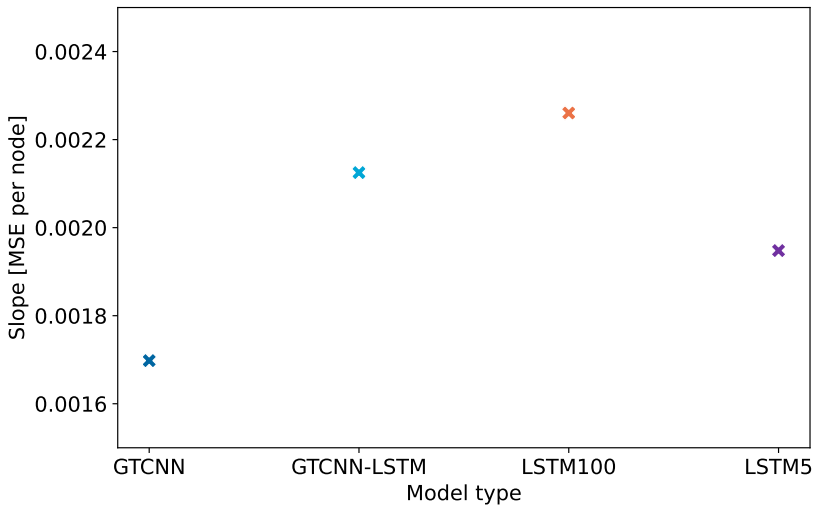


Figure 4.17: The slopes of the increase in MSE with the number of measurement nodes for each model type.

Fig. 4.18 shows how the training time of the different models scales with the number of measurement nodes. The GTCNN has a much lower training time, while the LSTM-based models take 4x longer to train. Also, the 95% confidence intervals show that the training time varies a lot for the LSTM-based models, but their runtime is higher than the GTCNN since their confidence intervals do not overlap.

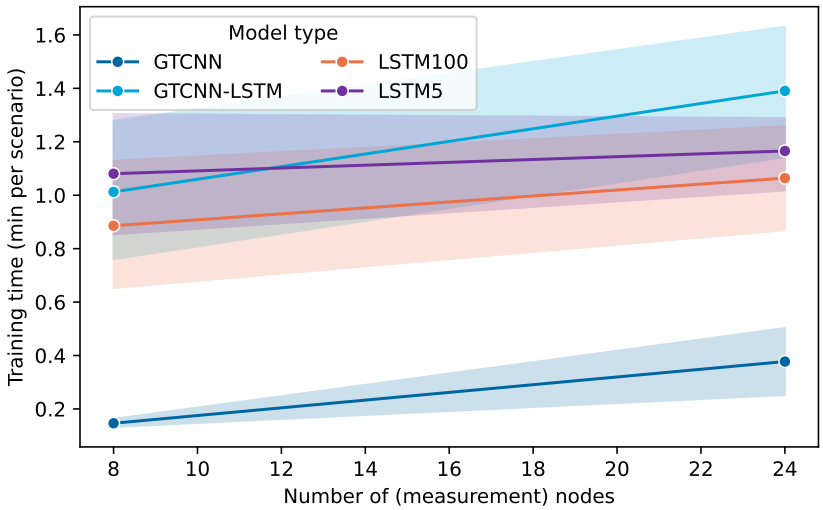


Figure 4.18: The increase in training time with the number of measurement nodes for each model type. The solid line is the central tendency (average) of the different datasets and model targets (P or Q), while the shaded areas show the confidence interval.

The rate of increase in training time per node is different per model (see Fig. 4.19). The training time of the GTCNN increases slightly faster than the LSTM100 model and almost 3x faster than the LSTM5 model; the training time of the GTCNN-LSTM model increases the quickest. The higher slope of the GTCNN-based equivalent implies that the LSTM-based models will train faster after a certain number of measurement nodes is reached. This makes the GTCNN model faster on ADNs with fewer measurement nodes, while the LSTM models will be faster for ADNs with more measurement nodes.

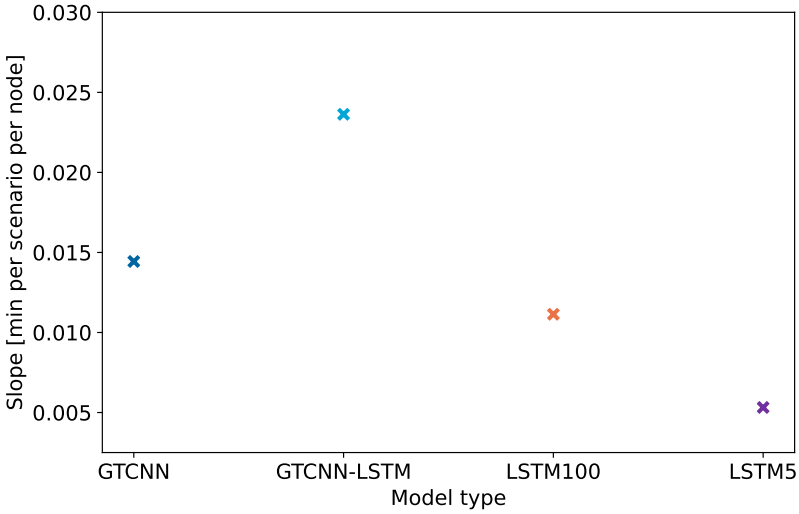


Figure 4.19: The slopes of the increase in training time with the number of measurement nodes for each model type.

5

Discussion and Conclusion

This chapter discusses the results from the case studies in sections 4.5, 4.6, 4.7, 4.8 and 4.9, and how these results relate to the objectives and research on equivalent models in general. This chapter concludes with several recommendations for future research.

5.1. Interpretation of the results

The proposed GTCNN-based equivalent model was evaluated on three case studies, each focussing on a different system condition: different fault events, different operating conditions and different *hidden* topological changes. The model is evaluated against an LSTM-based equivalent [44] with two observation windows. Moreover, a hybrid model that combines the GTCNN and LSTM models is also evaluated. The key findings are discussed below.

A simpler GTCNN yields better performance

The hyperparameter tuning of the GTCNN resulted in an active power model with an observation window of $T_{window} = 50$, two layers of each 20 hidden features in the GTCNN, a graph filter order $K = 2$ and uses the Cartesian product graph; the reactive power model has an observation window of $T_{window} = 100$ with the other parameters remaining the same. Of all the considered hyperparameters, the ones that give a simpler GTCNN resulted in a better performance. One of the reasons is that a longer training time is needed to achieve a similar performance since more parameters have to be trained for a more complex network. However, this is no guarantee that a more complex network will perform better, especially as there will be a higher risk of overfitting. This issue was reflected in the tuning, where the simplest GTCNNs and the most complex GTCNNs did not perform optimally. The small number of parameters of the GTCNN could be why the model shows excellent generalisation performance between the training, validation and test sets.

Faster training speed

The GTCNN model is faster than the LSTM5, LSTM100 and GTCNN-LSTM networks for all case studies. The number of trainable parameters in each network can explain the training speed. The GTCNN has 1,921 and 2,921 parameters, while the LSTM-based models have a number of parameters in the order of millions. Although a low training time is not as critical for equivalent models as for real-time applications, it can make a difference in two ways. The first way is that when an equivalent model can be trained faster, the model can be exchanged more often between the DSO and TSO (see Fig. 3.22), which ensures that the TSO uses a more accurate model. The reduced computational barrier will also facilitate this more frequent exchange, as it requires fewer resources (both human resources and computing power), which can eliminate the need for special allocation of time and resources to compute the equivalent model. The second way the lower training time can make a difference is that the GTCNN-based equivalent model can be trained more extensively than the LSTM-based models within the same time frame. This means the GTCNN-based equivalent model can be trained with many more scenarios, yielding a more accurate model. If the DSO includes fault locations, operating conditions and topological changes in a single measurement database, there will easily be tens or hundreds of

thousands of scenarios[1]. In the case studies of chapter 4, the runtime of the LSTM-based equivalents was already limited to six hours of training per dataset to keep the training time manageable on shared resources such as the DelftBlue Supercomputer. On more extensive datasets, the training time of the LSTM-based models might increase so much that it will become infeasible, even within the monthly time frame. In contrast, the faster training GTCNN would be able to incorporate more scenarios before reaching the same limit.

Accuracy comparable to other models

The accuracy is measured by the average MSE of all training samples. The impact of the MSE error on the prediction accuracy is shown in Fig. 4.7, with MSEs of 0.05 and below providing accurate predictions of the dynamic response and an MSE of 0.136 producing more significant deviations but still giving reasonably accurate performance. The GTCNN slightly outperforms the other models for the active power model, while its performance is somewhat worse for the reactive power model. However, when considering the impact of the MSE on the predicted dynamic response, the performance of the GTCNN (and the other equivalents) is exceptional, with an MSE that is consistently below 0.05. Moreover, the MSE of the GTCNN increased less per number of measurement nodes than its competitors, suggesting that the GTCNN retains a better accuracy on bigger networks. This means that the proposed GTCNN-based model is suitable for equivalent modelling of the transient stability of ADNs. This accurate performance also showcases the importance of the induction bias, as the assumption that the data follows a graph structure allows the GTCNN to obtain a similar performance with 1,921 and 2,921 parameters instead of 6,399, 592 for the LSTM model.

In the results on the Zeeland 50 kV ring, the GTCNN never outperformed the other models when predicting the reactive power response. Similarly, the GTCNN was not always the best-performing equivalent model for the active power on the Zeeland triple 50 kV ring. One reason for this performance could be the locality of the GTCNN. The sampling and propagation steps of the GNN are a local operation since the outputs of the graph convolutional filters, the fundamental building block of the GTCNN, are composed of a sum of graph-shifted versions of the input signal. The best-performing filter order K found in the hyperparameter tuning was 2, meaning the exchange is taking place at a maximum distance of two hops. This implies that the two PCCs never directly exchange their inputs with each other but rather slowly propagate the input through the other nodes, while the LSTM-based networks, LSTM5 and LSTM100, treat all the inputs equally. The PCC inputs are the dynamic input sequences of the voltage $|V|$ and θ, which are expected to contain the most information. Indeed, the homogeneous GTCNN, which received dynamic inputs at all nodes, performed much better on the reactive power model. When both PCC inputs are needed to find the dynamic response, the heterogeneous GTCNN could be at a disadvantage.

Another reason the GTCNN did not outperform the other models could be that its structure is not fully utilised. The GTCNN based on graph convolutional filters and product graphs is a homogenous GNN, assuming every node has the same features. However, the internal nodes have the static initial power injections P_0 and Q_0, which differ from the dynamic voltage inputs $|V|$ and θ at the PCC nodes. A heterogeneous graph convolutional operator will learn different message functions per relational type (a source node - edge type - target node triplet)[98], and can learn more accurately the relation between the inputs and the output. Moreover, the GTCNN network would benefit if it had more dynamic inputs at its nodes, as these contain more information about the dynamic response than the static initial power injections. A difference in performance was also observed when comparing the heterogeneous GTCNN with the homogenous GTCNN with dynamic inputs at all nodes. The latter showed a much better MSE for both the active and reactive power models. Unfortunately, this information is not available for equivalent modelling, where only the terminals of the model are connected and, thus, observed.

A different argument for the slight difference in performance is that the LSTM-based models are excellent in handling time-series data; this is what they were designed for. As all the models received the same inputs to create a fair comparison, the effect of explicitly considering the initial power injections has been hidden. Therefore, two different ML models designed to handle time-series data will perform

[1] The three datasets of the Zeeland 50 kV ring had 100 different fault locations and percentages, 180 different operating conditions and 16 topological changes per fault percentage. Combining all these events in a single database would result in 288,000 scenarios. Also, note how this dataset excludes only operating conditions and only topological changes, which would make the database even larger.

more or less similarly. This was also seen when the GTCNN was compared against the LSTM5 model without initial setpoints: the GTCNN did (slightly) outperform the LSTM5 model for the datasets with changing operating conditions and *hidden* topological changes, but a comparable performance was reached when only considering the fault events (i.e. when only learning time-series data under the same system condition).

Another cause for the similar performance of the GTCNN is that the variation in the system conditions (operating conditions and topological changes) might not be that pronounced. Historically, DNs were passive and thus responding to events in the TN. The behaviour of DNs had to be well-defined (partially by requirements such as the grid code) to keep the power system stable. As a result, the behaviour of DNs might not deviate much before other limits are violated or protection equipment steps in. The effect of including the initial power injections P_0 and Q_0 might be too small to produce a significant difference in performance. This was observed when evaluating the effect of including the initial power injections. Adding the initial power injections resulted in a slightly better performance of the GTCNN, but the LSTM5 model still had an acceptable performance. In the future, when the grid is much more decentralised, the dynamic behaviour of ADNs might vary significantly depending on the role they play at that moment (flexibility or other grid-support services). Then, the GTCNN-based equivalent could show a more significant improvement over existing methods.

Good generalisation performance

All models show a robust performance under the different training, validation and test splits. This is because the MSE does not change significantly from one split to another. In some cases, the average validation and test MSE is lower than the training split's. This suggests that the scenarios of the test split are drawn from the same distribution as the training (and validation) split(s). This is the case because the variations in the data from the different splits are created mainly by sampling from continuous random distributions[2]. It also means that the models do not overfit the training data, which means they can give reliable dynamic responses to unseen data. This property is essential as the equivalent model should be accurate under various changing system conditions.

It is essential to include extreme scenarios in the training dataset

One exception was observed to the excellent generalisation performance: the fault events dataset of the Zeeland triple 50 kV ring. In that dataset, a fault at 99% fault percentage causes an extreme outlier of MSE 3.632 for the reactive power prediction. This outlier affects the performance of all models (as seen in Tab. A.1), highlighting the importance of including extremes in the training dataset. This issue became more transparent in the Zeeland triple 50 kV ring because it is closer to its stability boundary because of its increased number of active components and sub-optimal control parameters. The GTCNN is the most impacted because of its already lower performance on the reactive power.

The GTCNN is better at predicting PCCs than individual terminals

The GTCNN had trouble predicting the dynamic response at individual terminals when their response differs significantly from each other. Although multiple graph filters are computed per input, the filter parameters are shared between all nodes. Therefore, the embeddings that are computed at every node will use similar parameters, and two nearby nodes will get similar embeddings by the GTCNN layers. The fully connected layer per node then needs to find a different dynamic response from the same embeddings, which did not work well. Therefore, the GTCNN should be used to predict the response at PCCs and at not single terminals when their dynamic response is very different.

The hybrid GTCNN-LSTM model is not better

The hybrid GTCNN-LSTM model did not outperform the GTCNN or the LSTM-based equivalent models. This performance can be attributed to the GTCNN-LSTM being a disjoint model: the LSTM part represents the temporal element, while the GTCNN is doing the spatial part. The result is that the input of the LSTM model is the hidden graph features of the GTCNN across different temporal graph replicas. There is no theoretical advantage to why these features would perform better than the inputs $|V|, \theta, P_0$ and Q_0; in fact, the added GNN at the input makes the training much slower as there are

[2]The only exception to this is the test set of the fault events, where extreme fault location percentages (2, 5, 95 and 99%) are included.

approximately 1.5 times the number of parameters of the LSTM-based equivalent models.

5.2. Addressing the research questions

As the power system grows more complex and active, equivalent models have become a solution for modelling parts of the network that have limited observability or are confidential or too complex to simulate otherwise. In the past decade, this topic has also made its way to DN because of their transition towards an active network, something which was not the case before. The scientific literature is mainly focused on grey box modelling because of its explainability and easy integration into existing simulation software. These models typically generalise the dynamic behaviour rather than adapt to specific changes in operating conditions or network topologies. Moreover, the accuracy of grey-box models is limited, as it depends on the components comprising the equivalent model or the selected mathematical model. In addition, there is no consensus on which model type is optimal. In response to these limitations, there has been a growing interest in black box models, particularly machine learning, for their potential to model complex relationships. These models promise greater adaptability and accuracy, but they lack interpretability and require a lot of data for training across diverse scenarios, resulting in limited validation of the models under different system conditions. At the same time, this is a necessity for accurate equivalent models.

To overcome these limitations, the following three research questions were answered:

1. How can an equivalent model be created that can estimate the dynamic response of an ADN with multiple PCCs under diverse operating conditions?
2. How can an equivalent model be created that can estimate the dynamic response of an ADN with multiple PCCs under topological changes?
3. How does the method scale to larger systems?

How can an equivalent model be created that can estimate the dynamic response of an ADN with multiple PCCs under diverse operating conditions?

An equivalent that can estimate the dynamic response of an ADN with multiple PCCs under diverse operating conditions can be created by training a GTCNN-based model using measurement data that contains different operating conditions. The GTCNN relates the measurement data from PMUs in the network to its graph structure, and it directly considers system conditions through the initial power injections P_0 and Q_0, which help the equivalent model to learn how different operating conditions impact the dynamic response. Training and validating the GTCNN model on a dataset with various operating conditions for the same faults indicated that the GTCNN could predict the dynamic response accurately with an average MSE of 0.005 for the active power P and 0.026 for the reactive power Q. This performance is within the range of existing LSTM-based equivalents, but the results were obtained much faster. This difference in speed makes the GTCNN a promising alternative which could enable training on more comprehensive datasets.

How can an equivalent model be created that can estimate the dynamic response of an ADN with multiple PCCs under topological changes?

The proposed GTCNN-based equivalent model can also be used to estimate the dynamic response of an ADN with multiple PCCs under topological changes by training the equivalent using measurement data containing different topological changes, something existing works did not do. Two topological changes were considered: including 50/10 kV redundancy transformers and changing tap positions of already connected 50/10 kV transformers. These topological changes are hidden since they are not directly observed as input to the GTCNN. The GTCNN implicitly includes the impact of the hidden topological changes through changes in the initial power injections P_0 and Q_0. Training and validating the GTCNN model on a dataset with various hidden topological changes demonstrated that the GTCNN could predict the dynamic response accurately with an average MSE of 0.002 for the active power P and 0.016 for the reactive power Q. This performance was the lowest out of the considered LSTM-based equivalents and obtained much faster. However, it was still within range of the existing approaches.

How does the method scale to larger systems?

The proposed GTCNN-based equivalent model showed promising scaling performance for ADNs with fewer measurement points as it boasts similar performance but trains faster than LSTM-based approaches. The scaling performance was evaluated by replicating the 50 kV nodes *Zrz*, *Otl* and *Tln* to create the bigger Zeeland triple 50 kV network, which contains 24 instead of 8 measurement nodes. The models were trained with three datasets: one containing faults, one predominantly containing operating conditions and one predominantly containing topological changes. The GTCNN-based equivalent model requires less training time to reach comparable performance to the LSTM-based models, but its runtime increases faster when going from 8 to 24 measurement points. At the same time, the MSE of the GTCNN increases slower than the LSTM-based when going from 8 to 24 measurement points. These differences in slope suggest that the GTCNN network can be trained faster for ADNs with fewer measurement points, while it will be more accurate with more measurement points.

5.3. Limitations and recommendations

The proposed GTCNN-based equivalent model has demonstrated comparable performance to the LSTM-based approaches but at a smaller runtime. The current research has some limitations, which can be addressed by future research.

To test the robustness of the GTCNN on different system conditions, three datasets have been created with a different focus: the first one contained faults, the second had operating conditions and the third focussed on topological changes. The performance was thus evaluated on a specific portion of the system conditions and showed that the proposed method could generalise over one of these different conditions, thus answering the first two research questions. A more comprehensive dataset should be studied in future work to evaluate the model's performance when all these events are combined into a bigger, more comprehensive dataset. As this dataset will likely contain tens of thousands of scenarios, different challenges that need to be solved will arise, including how to deal with long training times. Future research should investigate how much and which training data is required for an accurate equivalent. Perhaps having an equivalent per fault type is better to keep the training manageable. Moreover, future work should involve TSOs and DSOs to develop more tailored equivalents that solve more pressing issues.

The scaling performance was compared only between two networks, resulting in two data points. This only gives partial information about how the MSE and training time change between 8 and 24 measurement nodes; a third data point could reveal a non-linear increase in one of the models and yield a more complete picture of how the training time increases for each model. Also, the LSTM models are implemented using the models from the PyTorch library, which have been coded efficiently and highly optimised; the GTCNN implementation is from [82]. The LSTM and the GTCNN depend on the input dimension through their non-linear activation functions $\sigma(\cdot)$, but the LSTM has four activation functions connecting the input while the GTCNN has only one. This difference suggests that the runtime of the LSTM should increase faster than that of the GTCNN, but the opposite was observed. The faster increase in the runtime of the GTCNN could be caused by the current code implementation that does not exploit the sparsity of $\mathbf{S}_\diamond$. As the shift operator $\mathbf{S}_\diamond$ grows exponentially with the number of nodes ($\mathbf{S}_\diamond \in \mathbb{R}^{NT \times NT}$), the inefficient computation could drastically affect the performance. Future work should consider a more efficient GTCNN implementation to yield a more accurate picture of the MSE and runtime increase with the number of measurement nodes. Luckily, these limitations do not change the fact that the GTCNN is faster for networks with 24 measurement nodes. Moreover, it is unlikely that ADNs will have hundreds of PMU locations as DSOs are still incorporating PMUs in their network, and it does not make sense to have PMUs at every bus in radial feeders.

The developed equivalent only considers the voltage $|V|$ and θ as an input, which makes it mainly accurate under voltage disturbances. The power grid has another controllable quantity: the frequency f. Future works should investigate the impact of including both the voltage and frequency. Including the frequency could result in an equivalent model that produces a more accurate dynamic response since it can better distinguish between slightly different scenarios. Adding the frequency could also result in an equivalent model that is accurate under a broader range of events and could make the equivalent model the frequency controls as well.

The proposed GTCNN-based equivalent model assumes the ADN is modelled by a heterogeneous graph with two types of nodes. The PCC nodes receive the voltage magnitude $|V|$ and angle θ, which are dynamic inputs, while the internal nodes get the (static) initial active and reactive power injections P_0 and Q_0. These inputs can be implemented easily as they represent available information. However, the current heterogeneous GTCNN does not fully utilise the available information. One of the reasons is that the same graph convolutional operator is used for the voltage magnitude $|V|$ and initial active power injection P_0, and another one is shared between the voltage angle θ and the initial reactive power injection Q_0. In future research, this GCN should be replaced by a heterogeneous operator, which works per relational type (source node - edge type -target node triplet). This approach would ensure individual weights are learned per relation, possibly ensuring better performance.

The equivalent model has been trained as a black box model to reproduce the outputs $\hat{P}$ and $\hat{Q}$ from the inputs $|V|$ and θ. In the workflow of Fig. 3.22, it is suggested to use the GTCNN in co-simulation with the TN model. The model is expected to work in co-simulation, but it has not been validated, and the exact implementation of the model is unknown. Perhaps it is possible to use the Python interface of simulation packages such as PowerFactory and PSCAD to pass the inputs to the GTCNN equivalent and feedback its outputs in the TN. Or the better option could be to use a third-party co-simulation tool to exchange data between the two models. Future research can focus on integrating the black-box dynamic equivalents and popular simulation tools.

The dataset for the topological changes included only hidden topological changes, such as the connection of redundancy transformers and changing transformer tap positions. Therefore, the information on topological changes has been implicitly encoded in the dataset. One of the powers of GNNs is that they operate on graphs, which can also be reinterpreted as inputs to enable transference across graphs. What this could mean for equivalent models is that power system upgrades, such as the addition or disconnection of lines, when passed as an input to the GTCNN, will still produce an accurate dynamic response without requiring retraining of the model. Moreover, if two DNs are similar, and if their dynamic behaviour is bounded similarly by the grid code, the transference property could mean that the same equivalent can be reused for the different DNs. Future research can evaluate the impact of treating the graph as an explicit input and what this means for the accuracy of the GTCNN under different graph topologies.

The proposed GTCNN-based equivalent model only receives dynamic inputs at the PCC nodes. It was observed that the GTCNN performs better when more dynamic input features (e.g. $|V|$, θ, I, L_{line}, P_0 and Q_0) are included at all nodes. Future projects should investigate in which circumstances or applications this information would be available, as that would fully exploit the proposed GTCNN structure. One possible application would be to predict the future outputs of the ADN based on historical measurements, i.e. a trajectory prediction of the DN's outputs.

A

Zeeland triple 50 kV ring

This appendix contains supplementary figures and Zeeland triple 50 kV ring results. In section A.1, the network diagram and the corresponding graph representation are shown. Section A.2 contains the full results of the different models for the Zeeland triple 50 kV ring.

A.1. Network diagram and graph representation

The modified graph $\mathcal{G}_{modified}$ and the corresponding modified line graph $\mathcal{G}_{GTCNN}$ are seen in Figures A.1 and A.2 respectively.

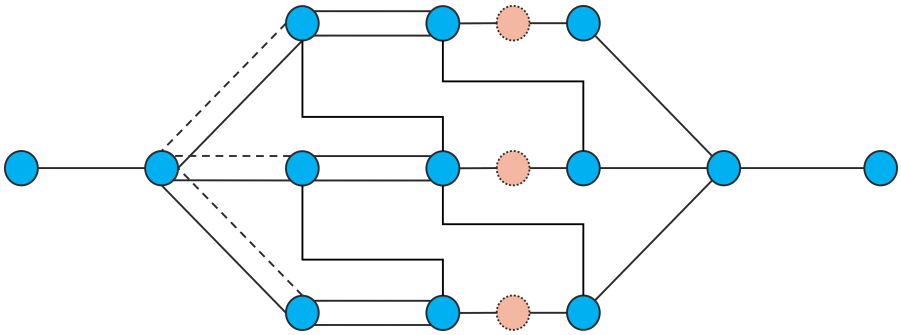


Figure A.1: The graph $\mathcal{G}_{modified}$ of the Zeeland triple 50 kV ring.

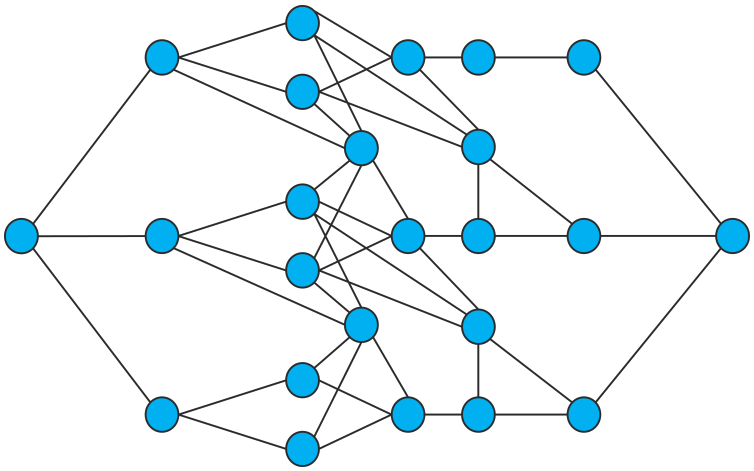


Figure A.2: Modified line graph $\mathcal{G}_{GTCNN} = L(\mathcal{G}_{modified})$ of the Zeeland triple 50 kV ring.

In Fig. A.3, the full PowerFactory network diagram of the Zeeland triple 50 kV network is found.

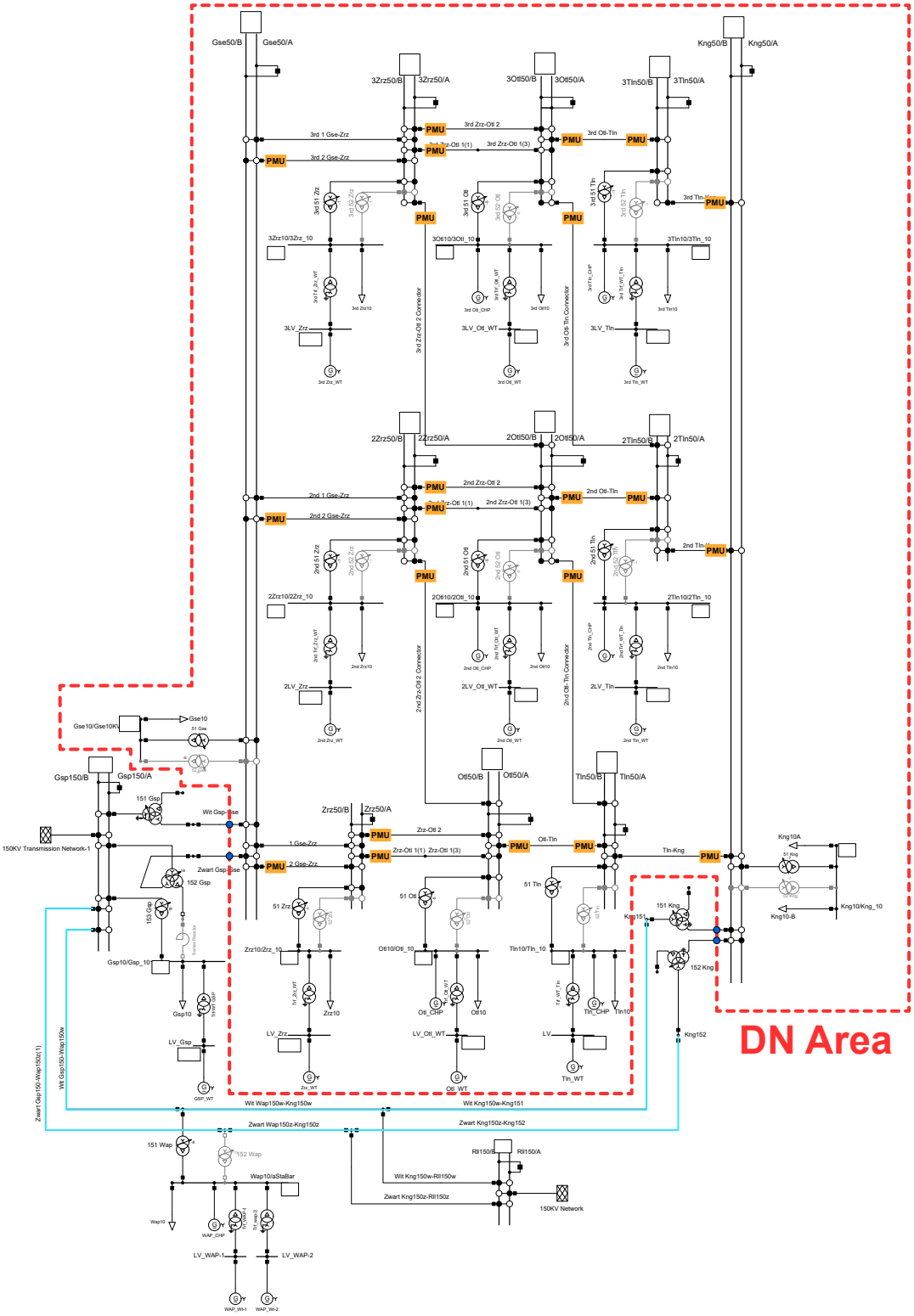


Figure A.3: The Zeeland 50 kV ring whose 50 kV busses have been copied two times yields a bigger and meshed ADN. Only the 50 kV area of the network is considered to be the DN (▬ ▬). The DN has 22 PMUs (PMU), at least one near each 50 kV bus. The DN is connected to the 150 kV TN at two points by four PCC terminals (●). The fault events are generated on the 150 kV transmission lines between Gsp and Kng (▬).

A.2. Full results

This section contains the results of the four models on three datasets: one contains fault events (Tab. A.1), the second has operating conditions (Tab. A.2) and the third focuses on topological changes (Tab. A.3). The best results are marked in green. The values represent the average ± first standard deviation of the MSE of all scenarios in a particular split. Please see Fig. 4.7 for the effect different MSE values have on the equivalent's accuracy.

Table A.2: Performance of the four models of the triple Zeeland 50 kV network for different operating conditions.

Target	Model	Training	Validation	Test	Training time
P	GTCNN	0.012 (± 0.005)	0.011 (± 0.005)	0.012 (± 0.005)	01:19:04
	GTCNN-LSTM	0.010 (± 0.007)	0.017 (± 0.024)	0.018 (± 0.023)	05:51:20*
	LSTM100	0.007 (± 0.005)	0.013 (± 0.016)	0.017 (± 0.037)	05:55:09*
	LSTM5	0.004 (± 0.003)	0.004 (± 0.004)	0.005 (± 0.004)	05:57:18*
Q	GTCNN	0.024 (± 0.013)	0.029 (± 0.027)	0.028 (± 0.024)	03:07:23**
	GTCNN-LSTM	0.066 (± 0.044)	0.071 (± 0.057)	0.072 (± 0.059)	05:50:08*
	LSTM100	0.087 (± 0.051)	0.103 (± 0.065)	0.097 (± 0.057)	05:55:20*
	LSTM5	0.039 (± 0.025)	0.050 (± 0.043)	0.046 (± 0.031)	05:56:33*

*The training was stopped early not to exceed the maximum training time of 6 hours.
**The training was stopped early as there was no improvement over 100 epochs.

Table A.3: Performance of the four models of the triple Zeeland 50 kV network for different topological changes.

Target	Model	Training	Validation	Test	Training time
P	GTCNN	0.006 (± 0.003)	0.011 (± 0.003)	0.053 (± 0.008)	00:21:31
	GTCNN-LSTM	0.013 (± 0.012)	0.013 (± 0.011)	0.011 (± 0.006)	03:09:18
	LSTM100	0.013 (± 0.005)	0.013 (± 0.005)	0.013 (± 0.005)	01:29:11*
	LSTM5	0.008 (± 0.004)	0.008 (± 0.005)	0.010 (± 0.009)	02:25:16
Q	GTCNN	0.051 (± 0.043)	0.040 (± 0.026)	0.056 (± 0.047)	00:58:14
	GTCNN-LSTM	0.043 (± 0.030)	0.042 (± 0.020)	0.051 (± 0.032)	01:44:03*
	LSTM100	0.038 (± 0.029)	0.034 (± 0.019)	0.046 (± 0.035)	01:11:41*
	LSTM5	0.053 (± 0.028)	0.054 (± 0.025)	0.056 (± 0.029)	02:23:19

*The training was stopped early as there was no improvement over 20 epochs.

Table A.1: Performance of the four models of the triple Zeeland 50 kV network for different fault locations.

Target	Model	Training	Validation	Test	Test (no outliers)*	Training time
P	GTCNN	0.014 (± 0.006)	0.011 (± 0.006)	0.039 (± 0.084)	0.009 (± 0.003)	00:18:48
	GTCNN-LSTM	0.024 (± 0.019)	0.024 (± 0.018)	0.054 (± 0.081)	0.029 (± 0.026)	02:45:50**
	LSTM100	0.038 (± 0.029)	0.033 (± 0.026)	0.080 (± 0.132)	0.027 (± 0.017)	02:12:48
	LSTM5	0.022 (± 0.014)	0.021 (± 0.015)	0.288 (± 0.798)	0.022 (± 0.015)	02:06:40
Q	GTCNN	0.130 (± 0.092)	0.085 (± 0.040)	0.426 (± 1.069)	0.070 (± 0.031)	00:50:02
	GTCNN-LSTM	0.141 (± 0.085)	0.129 (± 0.091)	0.189 (± 0.233)	0.114 (± 0.066)	02:56:15
	LSTM100	0.113 (± 0.076)	0.118 (± 0.084)	0.192 (± 0.318)	0.088 (± 0.059)	02:13:09
	LSTM5	0.149 (± 0.112)	0.112 (± 0.080)	0.317 (± 0.646)	0.102 (± 0.058)	01:21:00**

*The outlier is caused by a fault in *Wit Gsp150-Wap150w* at a 99% fault percentage.
**The training was stopped early as there was no improvement over 20 epochs.

Bibliography

[1] G. Kalt, P. Thunshirn, and H. Haberl, "A global inventory of electricity infrastructures from 1980 to 2017: Country-level data on power plants, grids and transformers," *Data Brief*, vol. 38, p. 107 351, 2021, issn: 2352-3409. doi: 10.1016/j.dib.2021.107351.

[2] I. F. E. Research, *History of electricity*, Web Page, [accessed 2 March 2023]. [Online]. Available: https://www.instituteforenergyresearch.org/history-electricity/#Dawn.

[3] B. Nussey, *How edison, tesla, and other visionaries invented the modern grid (part 1 of 2)*, Web Page, [accessed 2 March 2023], 2018. [Online]. Available: https://www.freeingenergy.com/how-edison-tesla-and-other-visionaries-invented-the-modern-grid-part-1-of-3/.

[4] IEA, *Sankey diagram world: Balance*, Web Page, [accessed 2 March 2023], 2018. [Online]. Available: https://www.iea.org/sankey/#?c=World&s=Balance.

[5] J. Muelaner, *Grid frequency stability and renewable power*, Web Page, [accessed 2 March 2023], 2021. [Online]. Available: https://www.engineering.com/story/grid-frequency-stability-and-renewable-power.

[6] J. Rueda-Torres, *ET4113: Lecture 1. Introduction to power system stability and synchronous generator*, TU Delft, Apr. 2021.

[7] T. Weckesser, H. Jóhannsson, and J. Østergaard, *Impact of model detail of synchronous machines on real-time transient stability assessment*. 2013, pp. 1–9. doi: 10.1109/IREP.2013.6629364.

[8] ENTSO-E, *Entso-e releases pan-european network development plan for 2030 & 2040*, Web Page, [accessed 6 March 2023], 2022. [Online]. Available: https://tyndp.entsoe.eu/news/2022/07/entso-e-releases-pan-european-network-development-plan-for-2030-and-2040-tyndp-2022-for-consultation-until-16-september/.

[9] TenneT, *Market facilitation*, Web Page, [accessed 6 March 2023], 2023. [Online]. Available: https://www.tennet.eu/about-tennet/our-tasks/market-facilitation.

[10] N. Grid, *What's the difference between electricity transmission and distribution?* Web Page, [accessed 6 March 2023], 2023. [Online]. Available: https://www.nationalgrid.com/stories/energy-explained/electricity-transmission-vs-electricity-distribution.

[11] G. Chaspierre, G. Denis, P. Panciatici, and T. V. Cutsem, "An active distribution network equivalent derived from large-disturbance simulations with uncertainty," *IEEE Transactions on Smart Grid*, vol. 11, no. 6, pp. 4749–4759, 2020, issn: 1949-3061. doi: 10.1109/TSG.2020.2999114.

[12] L. D. P. Ospina, V. U. Salazar, and D. P. Ospina, "Dynamic equivalents of nonlinear active distribution networks based on hammerstein-wiener models: An application for long-term power system phenomena," *IEEE Transactions on Power Systems*, vol. 37, no. 6, pp. 4286–4296, 2022, issn: 1558-0679. doi: 10.1109/TPWRS.2022.3153117.

[13] P. Djapic, C. Ramsay, D. Pudjianto, *et al.*, "Taking an active approach," *Power and Energy Magazine, IEEE*, vol. 5, pp. 68–77, 2007. doi: 10.1109/MPAE.2007.376582.

[14] L. Martin and K. Brehm, *Clean energy 101: Virtual power plants*, Web Page, [accessed 6 March 2023], 2023. [Online]. Available: https://rmi.org/clean-energy-101-virtual-power-plants/.

[15] E. Wood, *What is a microgrid?* Web Page, [accessed 6 March 2023], 2020. [Online]. Available: https://www.microgridknowledge.com/about-microgrids/article/11429017/what-is-a-microgrid.

[16] TenneT, *Crowd balancing platform - blockchain technology*, Web Page, [accessed 6 March 2023], 2023. [Online]. Available: https://www.tennet.eu/about-tennet/innovations/crowd-balancing-platform-blockchain-technology.

[17] F. O. Resende, J. Matevosyan, and J. V. Milanovic, "Application of dynamic equivalence techniques to derive aggregated models of active distribution network cells and microgrids," in *2013 IEEE Grenoble Conference*, pp. 1–6. doi: 10.1109/PTC.2013.6652356.

[18] M. Khodayar and J. Wang, "Probabilistic time-varying parameter identification for load modeling: A deep generative approach," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1625–1636, 2021, issn: 1941-0050. doi: 10.1109/TII.2020.2971014.

[19] G. Mitrentsis and H. Lens, "Probabilistic dynamic model of active distribution networks using gaussian processes," in *2021 IEEE Madrid PowerTech*, pp. 1–6. doi: 10.1109/PowerTech46648.2021.9494816.

[20] Q. Lai, C. Liu, and K. Sun, "A network decoupling method for voltage stability analysis based on holomorphic embedding," Unpublished Work, arXiv, 2020. doi: 10.48550/ARXIV.2003.12287. [Online]. Available: https://arxiv.org/abs/2003.12287.

[21] N. Yadaiah and N. Venkata Ramana, "Linearisation of multi-machine power system: Modeling and control – a survey," *International Journal of Electrical Power & Energy Systems*, vol. 29, no. 4, pp. 297–311, 2007, issn: 0142-0615. doi: https://doi.org/10.1016/j.ijepes.2006.06.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061506001554.

[22] J. M. Undrill and A. E. Turner, "Construction of power system electromechanical equivalents by modal analysis," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-90, no. 5, pp. 2049–2059, 1971, issn: 0018-9510. doi: 10.1109/TPAS.1971.293000.

[23] B. Marinescu, B. Mallem, and L. Rouco, "Large-scale power system dynamic equivalents based on standard and border synchrony," *IEEE Transactions on Power Systems*, vol. 25, no. 4, pp. 1873–1882, 2010, issn: 1558-0679. doi: 10.1109/TPWRS.2010.2043548.

[24] J. Machowski, "Dynamic equivalents for transient stability studies of electrical power systems," *International Journal of Electrical Power & Energy Systems*, vol. 7, no. 4, pp. 215–224, 1985, issn: 0142-0615. doi: https://doi.org/10.1016/0142-0615(85)90023-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0142061585900237.

[25] E. J. S. Pires de Souza, "Identification of coherent generators considering the electrical proximity for drastic dynamic equivalents," *Electric Power Systems Research*, vol. 78, no. 7, pp. 1169–1174, 2008, issn: 0378-7796. doi: https://doi.org/10.1016/j.epsr.2007.09.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779607002118.

[26] X. Shang, Z. Li, T. Ji, P. Z. Wu, and Q. Wu, *Online area load modeling in power systems using enhanced reinforcement learning*, Electronic Article, 2017. doi: 10.3390/en10111852.

[27] J. B. Ward, "Equivalent circuits for power-flow studies," *Transactions of the American Institute of Electrical Engineers*, vol. 68, no. 1, pp. 373–382, 1949, issn: 2330-9431. doi: 10.1109/T-AIEE.1949.5059947.

[28] F. O. Resende and J. A. P. Lopes, "Development of dynamic equivalents for microgrids using system identification theory," in *2007 IEEE Lausanne Power Tech*, pp. 1033–1038. doi: 10.1109/PCT.2007.4538457.

[29] K. S. Metallinos, T. A. Papadopoulos, and C. A. Charalambous, "Derivation and evaluation of generic measurement-based dynamic load models," *Electric Power Systems Research*, vol. 140, pp. 193–200, 2016, issn: 0378-7796. doi: https://doi.org/10.1016/j.epsr.2016.06.022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779616302309.

[30] G. A. Barzegkar-Ntovom, E. O. Kontis, T. A. Papadopoulos, and P. N. Papadopoulos, "Methodology for evaluating equivalent models for the dynamic analysis of power systems," *IEEE Transactions on Power Delivery*, vol. 37, no. 6, pp. 5059–5070, 2022, issn: 1937-4208. doi: 10.1109/TPWRD.2022.3167136.

[31] J. V. Milanović and S. M. Zali, "Validation of equivalent dynamic model of active distribution network cell," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2101–2110, 2013, issn: 1558-0679. doi: 10.1109/TPWRS.2012.2227844.

[32] S. M. Zali, "Equivalent dynamic model of distribution network with distributed generation," Thesis, 2012.

[33] R. A. Ramos, A. P. Grilo-Pavani, A. B. Piardi, and T. C. C. Fernandes, *Method to build equivalent models of microgrids for rms dynamic simulation of power systems*, Conference Paper, 2021. doi: 10.48550/ARXIV.2110.06160. [Online]. Available: https://arxiv.org/abs/2110.06160.

[34] F. Conte, F. D. Agostino, S. Massucco, *et al.*, "Dynamic equivalent modelling of active distribution networks for tso-dso interactions," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pp. 1–6. doi: 10.1109/ISGTEurope.2017.8260235.

[35] G. Chaspierre, P. Panciatici, and T. V. Cutsem, "Modelling active distribution networks under uncertainty: Extracting parameter sets from randomized dynamic responses," in *2018 Power Systems Computation Conference (PSCC)*, pp. 1–7. doi: 10.23919/PSCC.2018.8442573.

[36] G. Chaspierre, G. Denis, P. Panciatici, and T. V. Cutsem, "A dynamic equivalent of active distribution network: Derivation, update, validation and use cases," *IEEE Open Access Journal of Power and Energy*, vol. 8, pp. 497–509, 2021, issn: 2687-7910. doi: 10.1109/OAJPE.2021.3102499.

[37] X. Shang, Z. Li, J. Zheng, and Q. H. Wu, "Equivalent modeling of active distribution network considering the spatial uncertainty of renewable energy resources," *International Journal of Electrical Power & Energy Systems*, vol. 112, pp. 83–91, 2019, issn: 0142-0615. doi: https://doi.org/10.1016/j.ijepes.2019.04.029. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014206151832756X.

[38] G. Chaspierre, G. Denis, P. Panciatici, and T. V. Cutsem, "An active distribution network equivalent derived from large-disturbance simulations with uncertainty," *IEEE Transactions on Smart Grid*, vol. 11, no. 6, pp. 4749–4759, 2020, issn: 1949-3061. doi: 10.1109/TSG.2020.2999114.

[39] G. Chaspierre, P. Panciatici, and T. V. Cutsem, "Aggregated dynamic equivalent of a distribution system hosting inverter-based generators," in *2018 Power Systems Computation Conference (PSCC)*, pp. 1–7. doi: 10.23919/PSCC.2018.8442968.

[40] G. Mitrentsis and H. Lens, "Dynamic modeling of active distribution networks using cluster analysis of field measurement data," in *NEIS 2020; Conference on Sustainable Energy Supply and Energy Storage Systems*, pp. 1–7.

[41] G. Mitrentsis and H. Lens, "Data-driven dynamic models of active distribution networks using unsupervised learning techniques on field measurements," *IEEE Transactions on Smart Grid*, vol. 12, no. 4, pp. 2952–2965, 2021, issn: 1949-3061. doi: 10.1109/TSG.2021.3057763.

[42] E. O. Kontis, T. A. Papadopoulos, M. H. Syed, E. Guillo-Sansano, G. M. Burt, and G. K. Papagiannis, "Artificial-intelligence method for the derivation of generic aggregated dynamic equivalent models," *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 2947–2956, 2019, issn: 1558-0679. doi: 10.1109/TPWRS.2019.2894185.

[43] P. Wang, Z. Zhang, Q. Huang, X. Tang, and W. J. Lee, "A robust-improved method for dynamic equivalent modeling of active distribution network," in *2020 IEEE/IAS 56th Industrial and Commercial Power Systems Technical Conference (I&CPS)*, pp. 1–8. doi: 10.1109/ICPS48389.2020.9176767.

[44] C. Zheng, S. Wang, Y. Liu, *et al.*, "A novel equivalent model of active distribution networks based on lstm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2611–2624, 2019, issn: 2162-2388. doi: 10.1109/TNNLS.2018.2885219.

[45] Y. Li, Z. Wang, J. Yang, X. Wang, and J. Feng, "Dynamic equivalence modeling for microgrid cluster by using physical-data-driven method," *IEEE Transactions on Applied Superconductivity*, vol. 31, no. 8, pp. 1–4, 2021, issn: 1558-2515. doi: 10.1109/TASC.2021.3091065.

[46] P. N. Papadopoulos, T. A. Papadopoulos, P. Crolla, A. J. Roscoe, G. K. Papagiannis, and G. M. Burt, “Black-box dynamic equivalent model for microgrids using measurement data,” *IET Generation, Transmission & Distribution*, vol. 8, no. 5, pp. 851–861, 2014, https://doi.org/10.1049/iet-gtd.2013.0524, issn: 1751-8687. doi: https://doi.org/10.1049/iet-gtd.2013.0524. [Online]. Available: https://doi.org/10.1049/iet-gtd.2013.0524.

[47] MathWorks, *What are hammerstein-wiener models?* Web Page, [accessed 17 March 2023], 2023. [Online]. Available: https://uk.mathworks.com/help/ident/ug/what-are-hammerstein-wiener-models.html.

[48] B. Fortuner, *Can neural networks solve any problem? - visualizing the universal approximation theorem*, medium.com, Web Page, [accessed 20 November 2023], 2017. [Online]. Available: https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6.

[49] F. Milano, *Power System Modelling and Scripting*. 2010, vol. 54, isbn: 978-3-642-13668-9. doi: 10.1007/978-3-642-13669-6.

[50] P. Kundur, *Power System Stability And Control*. McGraw-Hill, 1994, isbn: 9780070635159. [Online]. Available: https://books.google.nl/books?id=v3RxH_GkwmsC.

[51] A. R. Sobbouhi and A. Vahedi, “Transient stability prediction of power system; a review on methods, classification and considerations,” *Electric Power Systems Research*, vol. 190, p. 106 853, 2021, issn: 0378-7796. doi: https://doi.org/10.1016/j.epsr.2020.106853. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779620306520.

[52] C. Olah, *Understanding lstm networks*, Web Page, [accessed 20 November 2023], 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/?utm_source=pocket_saves.

[53] IBM, *What is machine learning?* Web Page, [accessed 21 November 2023], 2023. [Online]. Available: https://www.ibm.com/topics/machine-learning.

[54] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[55] J. Brownlee, *How to choose an activation function for deep learning*, Machine Learning Mastery, [accessed 22 November 2023], 2023. [Online]. Available: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

[56] R. B. Arif, M. A. Siddique, M. M. R. Khan, and M. Oishe, “Study and observation of the variations of accuracies for handwritten digits recognition with various hidden layers and epochs using convolutional neural network,” Sep. 2018. doi: 10.1109/CEEICT.2018.8628078.

[57] M. Mishra, *Convolutional neural networks, explained*, medium.com, [accessed 24 November 2023], 2020. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[58] A. Rosebrock, *Are cnns invariant to translation, rotation, and scaling?* pyimagesearch.com, [accessed 24 November 2023], 2021. [Online]. Available: https://pyimagesearch.com/2021/05/14/are-cnns-invariant-to-translation-rotation-and-scaling/.

[59] A. Kaligambe and G. Fujita, “Short-term load forecasting for commercial buildings using 1d convolutional neural networks,” in *2020 IEEE PES/IAS PowerAfrica*, 2020, pp. 1–5. doi: 10.1109/PowerAfrica49420.2020.9219934.

[60] A. P. Wibawa, A. B. P. Utama, H. Elmunsyah, U. Pujianto, F. A. Dwiyanto, and L. Hernandez, “Time-series analysis with smoothed convolutional neural network,” *Journal of Big Data*, vol. 9, no. 1, p. 44, 2022, issn: 2196-1115. doi: 10.1186/s40537-022-00599-y. [Online]. Available: https://doi.org/10.1186/s40537-022-00599-y.

[61] J. Leskovec, *CS224W: Lecture 1. Introduction*, Stanford University, Sep. 2023. [Online]. Available: http://web.stanford.edu/class/cs224w/.

[62] S. Sankar, *Did you mean ”galene”?* LinkedIn Engineering, Jun. 2014. [Online]. Available: https://engineering.linkedin.com/search/did-you-mean-galene.

[63] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” *Computer Networks*, vol. 30, pp. 107–117, 1998. [Online]. Available: http://www-db.stanford.edu/~backrub/google.html.

[64] J. Zhou, G. Cui, S. Hu, *et al.*, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020, issn: 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[65] W. Liao, B. Bak-Jensen, J. R. Pillai, Y. Wang, and Y. Wang, “A review of graph neural networks and their applications in power systems,” *Journal of Modern Power Systems and Clean Energy*, vol. 10, no. 2, pp. 345–360, 2022, issn: 2196-5420. doi: 10.35833/MPCE.2021.000058.

[66] J. Fan, S. Rao, G. Muniraju, C. Tepedelenlioglu, and A. Spanias, “Fault classification in photovoltaic arrays using graph signal processing,” in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, pp. 315–319. doi: 10.1109/ICPS48405.2020.9274763.

[67] K. Chen, J. Hu, Y. Zhang, Z. Yu, and J. He, “Fault location in power distribution systems via deep graph convolutional networks,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 1, pp. 119–131, 2020, issn: 1558-0008. doi: 10.1109/JSAC.2019.2951964.

[68] Z. Wang, Y. Chen, S. Huang, X. Zhang, and X. Liu, “Temporal graph super resolution on power distribution network measurements,” *IEEE Access*, vol. 9, pp. 70 628–70 638, 2021, issn: 2169-3536. doi: 10.1109/ACCESS.2021.3054034.

[69] C. Wang, J. An, and G. Mu, “Power system network topology identification based on knowledge graph and graph neural network,” *Frontiers in Energy Research*, vol. 8, 2021, issn: 2296-598X. doi: 10.3389/fenrg.2020.613331. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fenrg.2020.613331.

[70] C. Dinesh, S. Makonin, and I. V. Bajić, “Residential power forecasting using load identification and graph spectral clustering,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 11, pp. 1900–1904, 2019, issn: 1558-3791. doi: 10.1109/TCSII.2019.2891704.

[71] E. Isufi, *CS4350: Lecture 8. Graph-Time Learning*, TU Delft, Apr. 2023.

[72] J. Huang, L. Guan, Y. Su, H. Yao, M. Guo, and Z. Zhong, “Recurrent graph convolutional network-based multi-task transient stability assessment framework in power system,” *IEEE Access*, vol. 8, pp. 93 283–93 296, 2020, issn: 2169-3536. doi: 10.1109/ACCESS.2020.2991263.

[73] D. Huang, H. Liu, T. Bi, and Q. Yang, “Gcn-lstm spatiotemporal-network-based method for post-disturbance frequency prediction of power systems,” *Global Energy Interconnection*, vol. 5, no. 1, pp. 96–107, 2022, issn: 2096-5117. doi: https://doi.org/10.1016/j.gloei.2022.04.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096511722000287.

[74] X. Jiao, X. Li, D. Lin, and W. Xiao, “A graph neural network based deep learning predictor for spatio-temporal group solar irradiance forecasting,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6142–6149, 2022, issn: 1941-0050. doi: 10.1109/TII.2021.3133289.

[75] J. T. de Freitas and F. G. F. Coelho, “Fault localization method for power distribution systems based on gated graph neural networks,” *Electrical Engineering*, vol. 103, no. 5, pp. 2259–2266, 2021, issn: 1432-0487. doi: 10.1007/s00202-021-01223-7. [Online]. Available: https://doi.org/10.1007/s00202-021-01223-7.

[76] J. J. Q. Yu, D. J. Hill, V. O. K. Li, and Y. Hou, “Synchrophasor recovery and prediction: A graph-based deep learning approach,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7348–7359, 2019, issn: 2327-4662. doi: 10.1109/JIOT.2019.2899395.

[77] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured sequence modeling with graph convolutional recurrent networks,” in *Neural Information Processing*, L. Cheng, A. C. S. Leung, and S. Ozawa, Eds., Springer International Publishing, pp. 362–373, isbn: 978-3-030-04167-0.

[78] B. Y. Zhu, H. Yin, and Zhanxing, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, pp. 3634–3640. doi: 10.24963/ijcai.2018/505. [Online]. Available: https://doi.org/10.24963/ijcai.2018/505.

[79] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, *Language modeling with gated convolutional networks*, 2017. arXiv: 1612.08083 [cs.CL].

[80] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, "Spatiotemporal graph neural network for performance prediction of photovoltaic power systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 17, pp. 15 323–15 330, 2021. doi: 10.1609/aaai.v35i17.17799. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17799.

[81] Y. Zhang, Y. Li, X. Wei, and L. Jia, "Adaptive spatio-temporal graph convolutional neural network for remaining useful life estimation," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, isbn: 2161-4407. doi: 10.1109/IJCNN48605.2020.9206739.

[82] E. Isufi and G. Mazzola, "Graph-time convolutional neural networks," in *2021 IEEE Data Science and Learning Workshop (DSLW)*, pp. 1–6. doi: 10.1109/DSLW51110.2021.9523412.

[83] T. Wu, Y. J. A. Zhang, Y. Liu, W. C. Lau, and H. Xu, "Missing data recovery in large power systems using network embedding," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 680–691, 2021, issn: 1949-3061. doi: 10.1109/TSG.2020.3014813.

[84] *Grid diagram: The dutch hv network*, TenneT, Nov. 2019. [Online]. Available: https://netztransparenz.tennet.eu/electricity-market/transparency-netherlands/grid-diagram/.

[85] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1034–1049, 2019, issn: 1941-0476. doi: 10.1109/TSP.2018.2887403.

[86] E. Isufi, *CS4350: Lecture 5. GraphConvolutional Neural Networks*, TU Delft, May 2023.

[87] *Transparency platform*, ENTSO-E, [accessed 11 January 2023], 2024. [Online]. Available: https://transparency.entsoe.eu/.

[88] G. Rietveld, A. Jongepier, J. Seters, *et al.*, *APPLICATION OF PMUS FOR MONITORING A 50 KV DISTRIBUTION GRID*. 2015.

[89] Wikipedia contributors, *Delta — Wikipedia, the free encyclopedia*, [accessed 31-July-2023], 2023. [Online]. Available: https://nl.wikipedia.org/wiki/DELTA.

[90] KiesZeker, *Enduris netbeheer*, Web Page, [accessed 31 July 2023], 2023. [Online]. Available: https://www.kieszeker.nl/energie/netbeheerder/enduris/.

[91] N. Save, *Phasor measurement unit (pmu) based power system analysis of mv distribution grid*, Thesis, 2016. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A9b192f84-f3f7-483e-9b67-99b1480e29c7.

[92] L. D. P. Ospina, V. U. Salazar, and D. P. Ospina, "Dynamic equivalents of nonlinear active distribution networks based on hammerstein-wiener models: An application for long-term power system phenomena," *IEEE Transactions on Power Systems*, vol. 37, no. 6, pp. 4286–4296, 2022, issn: 1558-0679. doi: 10.1109/TPWRS.2022.3153117.

[93] *Apache parque*, [accessed 15 January 2023], 2024. [Online]. Available: https://parquet.apache.org/#td-block-1.

[94] D. Chopra, *Unveiling the battle: Apache parquet vs csv — exploring the pros and cons of data formats*, medium.com, Web Page, [accessed 15 January 2024], 2023. [Online]. Available: https://medium.com/@dinesh1.chopra/unveiling-the-battle-apache-parquet-vs-csv-exploring-the-pros-and-cons-of-data-formats-b6bfd8e43107.

[95] E. O. Schweitzer, B. Kasztenny, A. Guzmán, V. Skendzic, and M. V. Mynam, "Speed of line protection - can we break free of phasor limitations?" In *2015 68th Annual Conference for Protective Relay Engineers*, 2015, pp. 448–461. doi: 10.1109/CPRE.2015.7102184.

[96] “Ieee/iec international standard - measuring relays and protection equipment - part 118-1: Synchrophasor for power systems - measurements,” *IEC/IEEE 60255-118-1:2018*, pp. 1–78, 2018. doi: 10.1109/IEEESTD.2018.8577045.

[97] *DelftBlue Supercomputer (Phase 1)*, Delft High Performance Computing Centre (DHPC), 2022. [Online]. Available: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1.

[98] J. Leskovec, *CS224W: Lecture 7. Heterogenous Graphs*, Stanford University, Oct. 2023. [Online]. Available: http://web.stanford.edu/class/cs224w/.