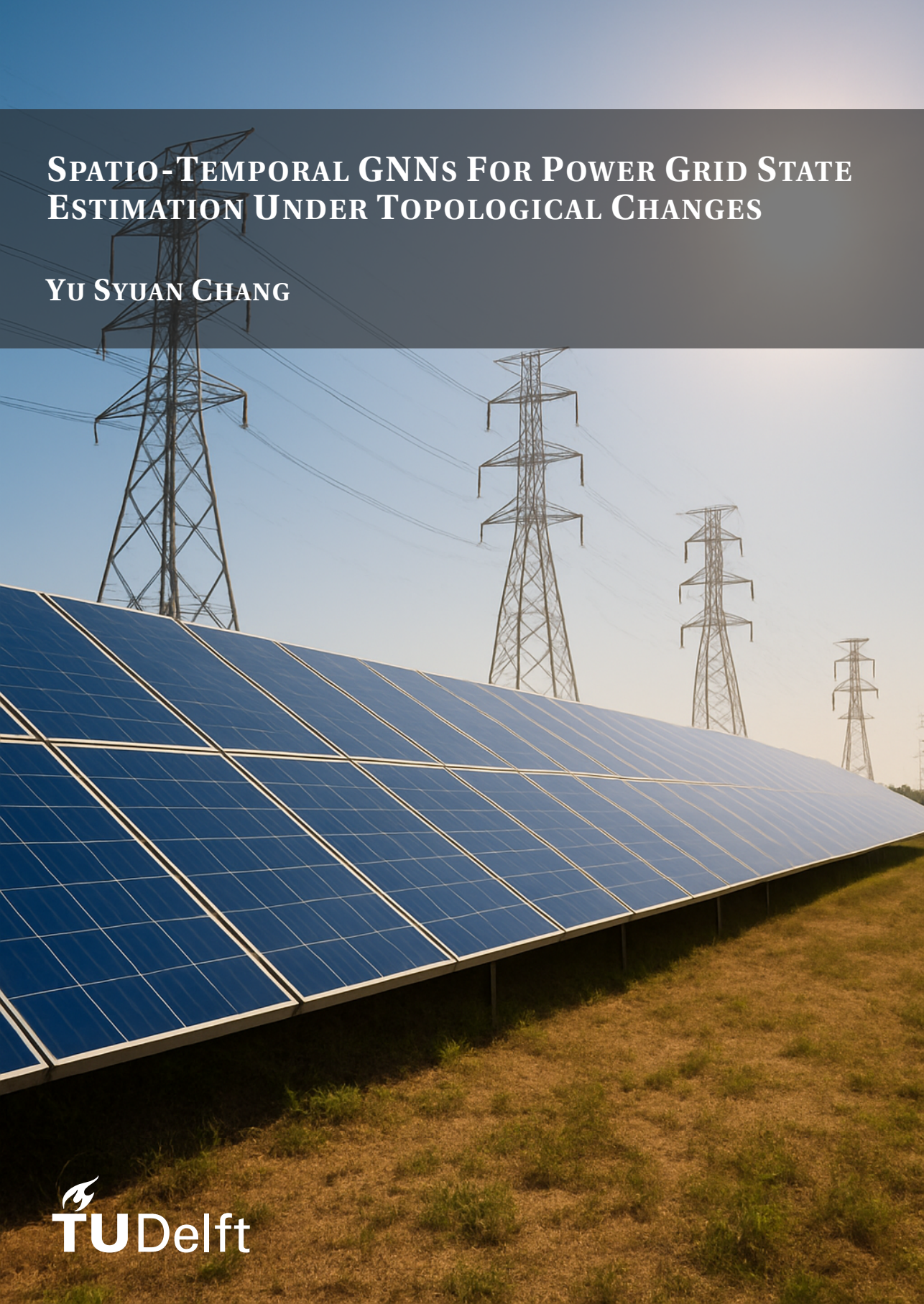
Spatio-Temporal GNNs For Power Grid State Estimation Under Topological Changes

Yu Syuan Chang



TU Delft

SPATIO-TEMPORAL GNNS FOR POWER GRID STATE ESTIMATION UNDER TOPOLOGICAL CHANGES

by

Yu Syuan CHANG

to obtain the degree of Master of Science

in
Sustainable Energy Technology

at the Delft University of Technology,
to be defended publicly on Wednesday, 18 June, 2025 at 10:00.

Student Number:	5998476	
Project Duration:	1 November, 2024 – 18 June, 2025	
Thesis Committee:	Dr. J. L. Cremer,	TU Delft, Thesis supervisor
	Dr. J. Dong,	TU Delft, Committee member
	J.B. Stiasny,	TU Delft, Daily co-supervisor

An electronic version of this thesis is available at
http://repository.tudelft.nl/.

TU Delft Delft University of Technology

SUMMARY

State estimation (SE) plays a critical role as a prerequisite for grid control and operation. However, the increasing penetration of distributed energy resources (DERs) and integrated energy systems (IES) introduces new challenges—such as unreliable pseudo-measurements and time-varying slack bus conditions—which make traditional methods like weighted least squares (WLS) increasingly difficult to apply. Moreover, DER integration leads to more frequent topological changes due to safety requirements and economic considerations. Yet, most existing machine learning methods for SE do not explicitly consider the topological changes. Therefore, this study evaluates three GNN-based models—GCN, GAT, and EvolveGCN—for state estimation in power grids under topological changes.

First, in the scenario without topological changes, where only the phase angle of the slack bus is fixed and noisy measurements are used, WLS performs worse than the three GNNs, indicating its susceptibility to interference under non-ideal conditions. Second, among the static GNNs, GAT performs best when topological changes are visible during training, but it cannot fully predict the voltage drops of unseen topological changes. GCN, on the other hand, demonstrates better generalization to unseen topologies and effectively suppress overfitting caused by noise. Third, although EvolveGCN is less accurate overall, it shows greater stability on nodes near PV buses, highlighting its potential to use historical information to enhance the time dimension when dealing with problems such as weak spatial correlation or local information loss.

These results suggest that GCN and GAT are potentially more suitable than WLS for SE in distribution grids with high DER penetration or IES. However, PV buses pose unique modeling challenges: they are physically but not numerically correlated with neighboring nodes, which make static GNNs hard to model their value change. Accurate estimation at PV buses is crucial, as they inject power into the system; in this context, EvolveGCN shows promise due to its stable predictions at these nodes.

ACKNOWLEDGMENTS

I would first like to thank my professor Jochen Cremer for taking the initiative to ask me again in 2024 after I expressed my interest in this topic at the end of 2023, and allowing me to do my research under the joint supervision of my daily supervisor, Jochen Stiasny.

I sincerely thank them for allowing me, who had no relevant background, to join the research group to work on this topic. They took time out to meet with me regularly before the research officially started. Under their earnest and serious guidance, they gradually led me to learn how to do research from the shallower to the deeper level, and finally enabled me to truly learn the techniques and methods that interested me. I also really appreciate their willingness to take time off, even from their lunch hours, to allow me to have frequent meetings with them.

Finally, I would also like to thank my landlord for being a great spiritual mentor to me when I was very anxious, and for going back to China for vacation when my research was at its busiest, allowing me to have the entire apartment to myself. I am also very grateful to my friends and family for being my good confidants during a random stressful weekend.

Yu Syuan Chang
Delft, May, 2025

List of Abbreviations

Abbreviation	Full Term
ANN	Artificial Neural Network
Conv	Convolutional
DERs	Distributed Energy Resources
DL	Deep Learning
DNN	Deep Neural Network
EvolveGCN	Evolutionary Graph Convolution Network
GAT	Graph Attention Network
GATv2	Graph Attention Network v2
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
IES	Integrated Energy Systems
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NR	Newton-Raphson (Method)
ReLU	Rectified Linear Unit Activation Function
RNN	Recurrent Neural Network
SE	State Estimation
tanh	Hyperbolic Tangent Activation Function
WLS	Weighted Least Squares

CONTENTS

Summary ii

Acknowledgments iii

List of Abbreviations iv

List of Figures viii

List of Tables x

1 Introduction 1

1.1 Background . . . 1

1.2 Problem Statement . . . 2

1.3 Research Questions . . . 3

1.4 Research Objectives, Contribution and Scope . . . 4

1.5 Key Terminology . . . 4

2 Literature Review 6

2.1 State Estimation: Challenge . . . 6

2.2 From ANN to GNN in Power Systems . . . 7

2.3 GNN in State Estimation . . . 8

2.4 Static Graphs with Temporal Dynamics . . . 9

2.5 Dynamic Graphs with Temporal Dynamics . . . 9

2.6 Application of EvolveGCN in other fields . . . 11

2.7 Research Gaps and Justification . . . 11

3 Theory and Methodologies **13**

3.1 Weight Least Square Method in State Estimation . . . 13

3.2 Artificial Neural Networks . . . 14

3.2.1 Workflow of Neural Networks . . . 16

3.2.2 Activation Functions . . . 18

3.2.3 Graph Neural Network . . . 18

3.3 Different kinds of layers in GNNs . . . 19

3.3.1 Convolutional Layer . . . 19

3.3.2 Graph Attention Layer . . . 20

3.3.3 Multilayer Perceptron Network (Fully Connected Layers) . . . 21

3.3.4 Recurrent Layer . . . 22

3.4 Evolve GCN . . . 23

3.5 Model Setup and Configuration . . . 25

3.5.1 WLS Model Architecture . . . 25

3.5.2 GCN and GAT Model Architecture . . . 26

3.5.3 EvolveGCN Model Architecture . . . 27

4 Research Setup **31**

4.1 Research Setup . . . 31

4.1.1 Test Network Selection . . . 31

4.1.2 Data Generation . . . 33

4.1.3 Data storage and pre-processing . . . 35

4.1.4 Workflow . . . 36

4.1.5 Standardization Process . . . 37

4.1.6 Residual Function . . . 38

4.1.7 Loss Function . . . 39

5 Case Study and Results **41**

5.1 Hyperparameter Tuning 41

5.2 Performance of the Modified EvolveGCN 43

5.3 Case Definition and Topology 44

5.4 Performance Comparison of GNN Models in Default Cases 47

5.5 GAT's Superior Performance in Seen Topologies 48

5.6 GCN's Superiority in Unseen Cases 49

5.7 Impact of Topological Change Frequency on Performance 52

5.8 Stable Performance of EvolveGCN Near PV Buses 53

5.9 Summary of the Result 57

6 Discussion **58**

6.1 Result Discussion 58

6.1.1 Training Insights and Hyperparameter Tuning 58

6.1.2 Overall Performance 60

6.1.3 Seen vs. Unseen Case Performance 62

6.2 Answers to Research Questions 63

6.3 Use Cases in Real Life 65

6.4 Future Research Direction 66

7 Conclusion **68**

List of Figures

3.1 Relationship between Machine Learning terminologies 15

3.2 The architecture of Artificial Neural Network(Bre et al., 2018) 16

3.3 The workflow of Artificial Neural Network 17

3.4 Model configuration of original EvolveGCN (Pareja et al., 2020) 23

3.5 Message passing diagram of EvolveGCN-O (Pareja et al., 2020) 25

3.6 Model configuration of GCN and GAT 27

3.7 Comparison between Original and Modified EvolveGCN: Backward frequency between sequence learning and batch-wise learning 28

3.8 Model configuration of EvolveGCN 30

4.1 IEEE 14-bus network topology (Illinois Center for a Smarter Electric Grid [ICSEG], n.d.-a) 32

4.2 IEEE 57-bus network topology (ICSEG, n.d.-b) 32

4.3 Load profile: 10 days 34

4.4 Topology change case design concept diagram 35

4.5 The workflow of data generation 36

4.6 The workflow of training 37

4.7 The workflow of testing 37

4.8 The extra averaging process 38

5.1 RMSE of predicted voltage for different batch sizes for the original and modified EvolveGCN 43

5.2 IEEE 14-bus network topology 46

5.3 IEEE 57-bus network topology 46

5.4 RMSE of Voltage prediction on 14-Bus Default case, Noise Std = 0.005 (def) 47

5.5 RMSE of Voltage prediction on 57-Bus Default case, noise std = 0.005 (def) 47

5.6 Comparison of Bus 8 voltage prediction in Case 3.1 and 3.3: Zoom in to 40 - 100 time steps 51

5.7 Comparison of model performance with seen and unseen cases 51

5.8 RMSE of Voltage prediction near the topological change on unseen cases: GCN vs. GAT 52

5.9 Comparison of Bus 17 voltage prediction in Case 7.2 (rl_111316_3_95) . . . 54

5.10 Comparison of Bus 17 voltage prediction in Case 7.3 (rl_111316_30) 54

5.11 Comparison of Bus 15 voltage prediction in Case 10.1 (rl_51422_30) 56

5.12 Comparison of Bus 9 voltage prediction in Case 10.2 (rl_16824_30) 56

5.13 Comparison of Bus 7 voltage prediction in Case 10.3 (rl_2918_30) 56

List of Tables

3.1 Activation functions and formulas 18

3.2 Comparison of Neural Networks and Graph Neural Networks 19

3.3 The trainable elements of three models 30

4.1 The elements of the networks 31

4.2 Topological change scenarios and their practical mapping 35

5.1 Number of nodes with the given measurement value in both test networks 41

5.2 Summary table of hyperparameters 42

5.3 Topological change cases: 14-bus network 44

5.4 Topological change cases: 57-bus network 45

5.5 Voltage RMSE comparison: default case (def) 48

5.6 Average RMSE of Voltage prediction on seen cases 49

5.7 Average RMSE of Voltage prediction on unseen cases 50

5.8 Average RMSE under topological change time step (buses directly related to the link removal) 52

5.9 The RMSE of the unseen cases: Case 7 to Case 9 54

5.10 The RMSE of all 57-bus network cases: PV buses' neighbors 55

5.11 The RMSE of Case 10.1 to Case 10.3: PV buses' neighbors 55

1

INTRODUCTION

1.1. BACKGROUND

Importance of State Estimation

State Estimation (SE) in the power system plays a foundational role in both conventional and smart grids. SE is used to estimate the voltage phasors in all system buses, which are necessary for designing control schemes to improve the stability and reliability of the system (Sharma & Jain, 2018). The method currently widely applied for SE is Weighted Least Squares (WLS), which is implemented under the static topology assumption and based on physical models (Abur & Exposito, 2004).

Challenges from Distributed Energy Resources (DERs)

However, in recent years, as more distributed energy resources (DERs) have been integrated into modern grids, they face significant variability in power generation due to weather dependency (H. Fan et al., 2022). This variability creates two problems. First, there is a lack of historical data, as many DER installations are relatively new or exhibit large variability over time, making historical information of limited use. Second, prior knowledge is of limited usefulness because weather-driven generation does not follow predictable patterns. Additionally, the sparse measurement problem becomes more pronounced in high DER penetration grids, such as rooftop solar panels and regional wind turbines (Ramachandran et al., 2019), causing pseudo-measurements—used to fill in missing sensor data and traditionally relying on historical data and prior knowledge—to become more necessary but less reliable. Therefore, when using WLS for state estimation, its result becomes inaccurate due to its reliance on the quality of these pseudo-measurements.

Challenges from Integrated Energy Systems (IES)

Moreover, the Integrated Energy Systems (IES) are gaining popularity: they improve energy efficiency by coupling electricity, gas and heat networks. However, this integration of different subsystems also has a side effect—affecting the consistency of voltage conditions in the power system. For instance, the power-gas system, whose compressor requires electrical load as temporal demand, may cause irregular compensation of the slack bus, leading to varying voltage conditions on the slack bus (Y. Huang et al., 2021). These variations of the slack bus, which are assumed to be fixed in WLS-based estimation. Such varying voltage conditions at the slack bus violate the assumptions of WLS and reduce its effectiveness.

Challenges of topological changes in ML-based state estimation

The application of machine learning in state estimation, which can tolerate sparser measurements and varying slack bus conditions, has received increasing attention. There have been related studies on the integration of state estimation and machine learning methods, which allow the model to learn this changing ecology and better adjust its calculations by considering specific physical constraints.

Nonetheless, most existing methods discuss spatial or temporal effects in fixed topology networks, rather than considering the impact of topological changes, which is unrealistic in modern grids. In traditional systems, security issues may lead to topological changes; however, these changes occur less frequently compared to systems with higher DER penetration. The intermittency of renewable energy may cause generators to be temporarily online or offline, and links to be removed more frequently due to operational constraints, fault isolation (IEEE Standard Association, 2018, 2022), or economic dispatch strategies—all of which can lead to frequent and dynamic topological changes. Moreover, such disconnections may cause temporary overvoltage (TOV) or voltage violations. Accurate SE is specifically vital for designing effective control schemes to restore normal operation.

As a result, the fixed-topology machine learning models may not be suitable for systems where the topology changes frequently due to renewable energy integration, component failures, or operational decisions. This leads to a critical gap between the capabilities of current models and the requirements of real-world power system with renewable sources integrated.

1.2. PROBLEM STATEMENT

Currently, most existing methods assume a fixed network topology during training and inference. As a result, static GNNs—which operate on a single graph snapshot—are typically trained on one topology and may fail to generalize when the graph structure changes. This limitation implies that whenever a topology change occurs, the model might need to be retrained, which is inefficient in real-time operations.

While many types of static GNNs exist, including Graph Convolutional Networks (GCNs)

and Graph Attention Networks (GATs), it is unclear whether they completely fail under topological changes, or whether they might still retain some robustness. In contrast, temporal GNNs are designed to model evolving graphs by incorporating historical information to learn temporal patterns. One example is EvolveGCN. However, no prior work has applied EvolveGCN to the power system state estimation problem, despite the fact that topological changes in power systems naturally introduce temporal effects.

To answer these questions, this study aims to evaluate and compare the performance of three neural network architectures: Graph Attention Mechanism (GAT), Graph Convolutional Network (GCN) and EvolveGCN with traditional methods: WLS, in the context of state estimation under topological changes.

1.3. RESEARCH QUESTIONS

Based on the problem statement, both static GNNs (GCN and GAT) and temporal GNN (EvolveGCN) are experimented in this research, applying WLS as the base model, to verify their performance. The focus is on evaluating the model's ability of state estimation to handle topological changes at different time scales and its generalization performance in processing unseen data.

Based on the motivation, the two overarching questions are:

- **Do static GNNs maintain prediction performance under topological changes, particularly when such changes are (not) included during training?**
- **Can temporal GNNs (e.g., EvolveGCN), which model evolving topologies, improve state estimation performance under dynamic structural conditions of the power grid?**

To provide a more fine-grained analysis of model performance, the following research questions are defined to detail the performance in different conditions.

- How do WLS, GAT, GCN, and EvolveGCN compare in terms of the prediction accuracy when no topological changes are present, particularly when noise is introduced in the measurement data (with vs. without noise)?
- How do GAT, GCN, and EvolveGCN perform in terms of prediction accuracy when faced with different topological changes, different time scales, and different frequency of changes? How do these factors affect the model's ability to adapt and maintain accurate predictions?
- How do GAT, GCN and EvolveGCN perform with and without topological change data during training?

1

1.4. RESEARCH OBJECTIVES, CONTRIBUTION AND SCOPE

This study aims to explore the feasibility of the selected methods to consider topological changes in state estimation tasks. Specifically, this study evaluates whether models designed for static graphs (e.g., GCN and GAT) have intrinsic limitations under topological changes, and explores the potential of under-experimented temporal GNNs (e.g., EvolveGCN) to overcome these limitations.

To achieve this goal, this study compares three types of neural network architectures - GCN, GAT and EvolveGCN - alongside the WLS method. EvolveGCN, namely Evolutionary Graph Convolution Network, combines static GCN with RNN to enable it to adapt to topological structure changes. The models are evaluated under different time scales and frequencies of topological changes.

The contributions of this study are as follows:

- Examine the performance of static GNNs (GCN, GAT) in state estimation when faced with topological changes in power grid.
- Test and verify the effectiveness of EvolveGCN in state estimation under topological changes.
- Compare the performance of the three GNN models and WLS under static topology, and further compare the three GNNs under dynamic topology. Focus on prediction accuracy, noise tolerance, and adaptability to topological changes.

This thesis report is structured as follows. The literature review is shown in Chapter 2, including the development of state estimation, the artificial neural network in power systems, and applied methodologies. Chapter 3 introduces the fundamental theory related to the project and the methodologies used, including the rationale of state estimation, neural networks, characteristics of different layers, and the applied EvolveGCN. The workflow applied and the research setup are presented in Chapter 4. The case studies and results are shown in Chapter 5. A discussion of the results and answers to the research questions are provided in Chapter 6.

1.5. KEY TERMINOLOGY

This section briefly introduces three types of buses in the power system—slack buses, PV buses, and PQ buses—which play different roles in power system.

Slack Bus

The slack bus is the reference node with a specified voltage and phase angle, and it is applied to balance the power flow equations. In conventional systems, this node is assumed to maintain a fixed voltage. However, with increasing integration of other energy

systems (such as gas or heat networks), slack buses may experience voltage variations that make them less stable than assumed by traditional models.

PV Bus

A PV bus is a generator bus with specified voltage amplitude and active power. In conventional systems, these buses represent controllable generators, such as turbines. However, in modern distribution grids with renewable energy sources such as solar panels or wind turbines, the active power output is affected by the weather and therefore varies. Although the voltage is theoretically specified and can be regulated by a control unit, in practice variations can still occur, especially when there is a lack of control infrastructure.

PQ Bus

A PQ bus is a load bus without generating units, where the active power and reactive power are specified. As part of the state estimation process, the voltage magnitude and phase angle are estimated. While PQ buses traditionally represent fixed loads, modern systems may involve demand-side flexibility or time-varying consumption patterns, introducing additional uncertainties into the state estimation.

2

Literature Review

This chapter explains the research progress related to this study from state estimation, artificial neural networks in power systems and other fields, and graph convolutional networks under static and dynamic graphs. Later, following Chapter 1, more detailed research gaps will be explained based on these historical approaches, which can provide a solid basis for discussing the validity of the results.

2.1. State Estimation: Challenge

State Estimation (SE) was first proposed by Fred Schweppe in 1970, aiming to estimate the operating state of the system by utilizing the redundancy of the real-time measurement system. The states of the system refer to internal physical variables that characterize the temporal evolution, which is crucial for system detection or analysis (Bai et al., 2023). When state information is not directly measured or there is lack of accuracy, SE is applied to reconstruct unknown state variables through prior knowledge and limited measurement of the system (Barfoot, 2024).

The Weighted Least Squares method (WLS) is a classic analytical tool for state estimation in engineering, and is widely used in power grid state estimation. According to (Basetti et al., 2018), the ratio of independent measurement variables to state variables is typically 1.5–3.0. However, compared to traditional independently planned and designed energy supply systems, the data redundancy of integrated energy systems or multi-energy systems is less than 1.5 (Zhou et al., 2020). This makes it necessary to refer to pseudo-measurement. Once a node is estimated incorrectly or the noise is too large, error propogation will occur. Furthermore, in renewable energy systems such as the power-gas integrated system, the slack node introduced as a gas load makes the entire network unstable, which makes it difficult to have a stable and unrestricted electricity slack node as

a reference point (Y. Huang et al., 2021).

Jin et al. (Jin et al., 2021) proposed a review paper covering the development of state estimation, starting from model-based methods. This paper starts with the least squares method for static state estimation and later introduces the Kalman filter (KF) and its modifications, which enhance stability and address nonlinear issues for dynamic state estimation while taking noise into account. However, all of the above methods assume Gaussian noise, and performance degrades when noise distributions are complex or non-Gaussian, as mixtures of Gaussian distributions fail to accurately represent such noise. Robust filters have been proposed to address this issue, though they often require significant computational resources. Recently, data-driven models have gained attraction with the availability of Big Data, and studies have shown that purely data-driven approaches can enhance estimation performance, though they depend heavily on data quality and quantity. To maximize accuracy, hybrid models that combine model-based and data-driven methods have become popular, and they can apply in various ways.

2.2. FROM ANN TO GNN IN POWER SYSTEMS

Artificial Neural Networks (ANNs) have been used in different fields such as finance, social networks, and transportation. The choice of ANN depends on the characteristics of the dataset in different case studies. Datasets with explicit time series characteristics require more time series-dependent models to learn ecology. EvoNet designed a model for learning evolutionary state diagrams and applied it to case studies such as network data and daily transaction data (Hu et al., 2021). Hinton et al. (Hinton et al., 2012) used Deep Neural Network (DNN) to analyze speech recognition to increase transcript accuracy. Oliveira et al. (Oliveira et al., 2016) used different ANN models such as Multi-Layer Perceptron (MLP) and Recurrent Neural Network (RNN) to predict computer network traffic. These datasets are closely related to time series. However, these approaches are appropriate when only the time series is important.

Choosing Neural Networks based on data structure

Data levels, where not only time series but also spatial relationships are important, require different types of ANN. Fan et al. (W. Fan et al., 2019) use GNN to learn user-item interactions and social relationships to improve the accuracy of social recommendation. Rico et al. (Rico et al., 2021) use GNN to capture spatial and temporal dependencies to predict traffic flow. You et al. (Y. Yu et al., 2022) also use GNN, high-order neighbor information to improve product recommendations on social networks. For data sets such as social networks and traffic flows, the image relationship between the data has a great impact on prediction, so Graph-based ANN is more suitable.

GNNs for power systems with spatial-topological characteristics

For power grid data, the measurement of the power system is limited by the number of smart meters. The relationships between data have non-Euclidean structures, and there are many types of data-driven models designed to process Euclidean data. In addition,

traditional NNs such as feedforward networks cannot handle changes in the number of nodes due to changes in topological structures. GNNs with dynamic batching functions or learning methods that use GNNs to process single graphs (non-batch learning) can solve this problem. Therefore, when applying neural networks in power systems, the graph structure in the power system is indispensable. In 2020, Liao et al. shows an overview of the application of GNN in power system in (Liao et al., 2021), illustrating that Graph Convolutional Network (GCN) is one of the classical ways to deal with graph data, and is applied to solve power grid problems such as power quality disturbance classification, optimal power flow, and transient stability assessment of power systems.

2.3. GNN IN STATE ESTIMATION

As mentioned in Section 2.1, combining model-based methods and data-driven methods is an experimental direction to try to improve the accuracy of GNN in state estimation tasks. Model-based means using mathematical models or physical formulas for optimization, and its combination with data-driven models is to consider them as soft constraints that are part of the loss function. There are different ways to integrate physical constraints with data-driven models.

Physics-informed GNN models for SE

Ngo et al.(Ngo et al., 2024) proposed a physical information neural network (PINN), which includes both a model-based model (including the phasor measurement unit (PMU) part) and a data-driven model (the rest), and incorporates the results of KF estimation into model training. This is a semi-supervised model similar to the fusion of data-driven models and physical constraints. It uses approximately half of the measured voltage as input and uses it as labels at the beginning of the loss function calculation. This approach attempts to reduce the risk that the simulation results will not be accurate enough for the actual power system. Habib et al.(Habib et al., 2023) proposed a Deep Statistical Solver (DSS^2) to construct a weakly supervised method. This method substituted the state variables predicted by the Graph Neural Network (GNN) into the power flow equations, which served as a soft constraint to include physical relations into the model.

Temporal modeling with GNN in SE

Wang et al. (Wang et al., 2020) applied the power flow equation at the decoder to transfer the estimated state from data-driven model to estimated measured data based on physical constraints. It applied the Long-Short-Term Memory (LSTM) as its Dynamic Neural Network (DNN) model to consider the temporal effect. This model has studied the time correlation in the power system to some extent, but has not yet considered topological changes.

Topology-aware GNN for SE

Moshtagh et al. (Moshtagh et al., 2023) use a combination of Graph Convolutional Network (GCN) and Graph Attention Network (GAT) layers for state estimation: they use the

measured voltages from PMUs (All nodes either have a PMU or are adjacent to at least one node with a PMU) for prediction and as labels in the loss function calculation. Furthermore, Moshtagh et al. The impact of topological changes of link removal type on the overall state estimation is tested.

2.4. STATIC GRAPHS WITH TEMPORAL DYNAMICS

However, the sole application of GCN model cannot simulate all possibilities in the modern power system, especially the temporal effect and the topological change. To make GCN able to adapt to the power system's dynamics, there are many problems to be solved, for instance: First, the currently better-developed GCN method is spectral, which requires high computational demand and has lower potential to analyze larger-scale distribution. The spatial GCN grows rapidly in recent years and is not yet a mature topic to put into realistic settings. Second, the size of adjacency matrix depends on the number of samples, thus the GCN model may need to be retrained once the topology changes leading of number of samples changes (Liao et al., 2021). As a result, researches are trying to integrate GNN with other tools to make it adaptable to dynamic environment.

The development of the dynamic GNN first lies into the simulation of temporal effect simulation in static graph. To allow the time dynamism to be predictable in the model, there are extended models combining the GNN and recurrent architectures, whereby the former deals with the graph information, and the latter deals with the dynamism and temporal effect. Do et al. (Do et al., 2019) integrated the RNN methods such as LSTM with GCN to deal with time dependent problem in the chemical reaction field. In 2018, Yu et al. (B. Yu et al., 2017) proposed the ST-Conv blocks, which applied the similar combination techniques to solve the spatiotemporal traffic data. In 2020, Huang et al (J. Huang et al., 2020) proposed a Recurrent Graph Convolutional Network (RGCN), integrating the LSTM into GCN method to address the early-warning of stability classification of the power system, it applies the LSTM in the learning process of but not yet the graph's dynamic nature. The previous researches model the dynamics only within a static graph structure rather than considering the topological change, and most researches are in traffic flows field, which contains less physical constraints compared to power systems.

2.5. DYNAMIC GRAPHS WITH TEMPORAL DYNAMICS

RNN-GCN hybrid approaches for dynamic graphs

Combining RNN with GNN is found to be an experimental approach for dynamic graph prediction, because GNN is designed for structure extraction, while RNN is designed for sequence learning (Lan et al., 2021). This category model uses GCN to learn node embedding and RNN for sequence learning. GCRN published by Seo et al. (Seo et al., 2018) designed the Graph Convolutional Recurrent Network (GCRN) by using LSTM to learn sequence dynamics along the time axis. Manessia et al. (Manessi et al., 2020) proposed

WD-GCN/CD-GCN to learn dynamic graph embeddings, which combines an improved LSTM and an extension of graph convolution operations to learn long- and short-term dependencies and graph structures. However, this kind of method requires knowledge of the nodes over the entire time span, so it is difficult to guarantee generalization to unseen topological changes (Pareja et al., 2020).

Evolving GCN weights to handle graph dynamics

In later 2020, Pareja et al.(Pareja et al., 2020) published the EvolveGCN method, which applied RNN in the graph level rather than the node embedding, that is, applied the recurrent architecture (Gated Recurrent Units (GRU) & LSTM) to dynamically update the GCN weight matrix, rather than only update the weight after the backward process. In this way, the model does not need to know all nodes and learn their characteristics in advance, making new nodes independent and the dynamics of the model more flexible. Moreover, the GCN model does not need to be retrained entirely when the topology is changed. However, the EvolveGCN did not save explicit historical information such as the past transaction information, which may rise the inaccuracy when the number of time step snapshots is large (You et al., 2022). Based on the method from EvolveGCN, Gao. et al. (Gao et al., 2022) proposed the Dynamic Graph Convolutional Network (DGCN) in 2023 to further consider the global feature integration and feature importance. DGCN adjusted GCN weight parameters by LSTM, at the same time compared the local data with global feature with Mutual Information (MI), and guided the aggregation by the importance of nodes obtained by the Dice similarity.

Further improvements: DynGCN and ROLAND

Li et al. (Lan et al., 2021) proposed the Dynamic Graph Convolutional Network (DynGCN), trying to convolve both the graph structural dynamism and weight parameters. Besides, DynGCN applies Convolutional Neural Network (CNN) method (TCN): every time step is aggregated by convolution operations which guarantees rich information and at the same time. You et al. (You et al., 2022) proposed the ROLAND method, which is a design principle starting from mature static GNN and adapting it for dynamic graphs. ROLAND views the node embeddings at different GNN layers as hierarchical node states, and updates the states based on the newly observed nodes and edges. Besides, it applies the truncated backward process to calculate the weight matrix, saving vast amount of computational demand. ROLAND keeps the effective design from static GNN, and also make the model scalable. Dileo et al. (Dileo et al., 2024) published a research related to the link prediction in the dynamic social network based on ROLAND method in 2024.

In addition, there are also studies on modeling topological changes without reducing computational requirements. Shi et al. (Shi et al., 2021) published the Graph Attention Evolutionary Network (GAEN), which uses GAT as a base method, adds an additional three-dimensional tensor to record the changes in the adjacency matrix at each time step, learns features and considers it when doing node embedding. It enforces relationships at different time steps by actually learning graph structures. However, its applicability depends on the specific circumstances. For example, the power system has a fixed physical structure, so the node relationships are not as irregular as the communication

network to which this paper is applied.

To sum up, there are many methods been tested under dynamic graph structure, either combining GCN as feature extractor, RNN for sequence learning, or EvolveGCN, which adjusts the weight parameters in the graph level in each time step. Those methods apply the recurrent architectures to save the historical data, and thus enable the graph structure to change and prevent the model from being retrained entirely. However, many of those methods are not yet been applied in the power system.

2.6. APPLICATION OF EVOLVEGCN IN OTHER FIELDS

In the EvolveGCN method paper published by Pareja (Pareja et al., 2020), they apply Stochastic Block Models, social networks, and traffic flows as case studies. The DEDGCN published by Zhong et al. (Zhong et al., 2022) reveals a model based on EvolveGCN, and models the adjacency matrix and node embeddings both by recurrent networks to increase node stability and better capture both feature and structural dynamics. The case studies of this model include networks such as autonomous systems, Bitcoin trading platforms, and social networks. This reveals that most successful model applications are in the social network field, and more cases can be seen from methods in Section 2.5. Models designed for social networks tend to include mechanisms for learning graph structures and node relationships. However, this is less relevant for power systems, where node relationships are fixed by physical laws, leading to less uncertainty and fewer degrees of freedom. On the other hand, the case of state estimation applies extra physical laws in the model training, thus the model can be regarded as semi-supervising, and not only the node embedding can be used directly, but no extra clustering work is required.

2.7. RESEARCH GAPS AND JUSTIFICATION

GNN models are promising but limited under topological changes

By reviewing relevant literature, it is found that: First, the GNN model is considered to have advantages over other data-driven models in tasks related to power systems, and the GNN in the power system has also been proven to be able to simulate the power grid structure. Second, hybrid models have been experimented with by many studies, especially on the task of state estimation, testing different approaches combining physical constraints with GNNs. However, adapting GNNs to dynamic topologies remains a challenge. When topological changes occur over time, it is often assumed that retraining is required for each new topology, which limits the practicality of GNN-based methods. Although some recent works explore ways to improve model generalization, the problem of how to effectively handle frequent or unpredictable topological changes without full retraining remains open.

Dynamic graph structure models remain underexplored in power systems

Meanwhile, the emergence of methods such as dynamic graph structure models makes graph structures changeable, and has been applied in areas such as social networks and transportation, but has not yet been applied to power system. Therefore, this study hopes to start from this and explore the performance of these dynamic graph structure models in power grid state estimation. Considering the unpredictable topological changes that may occur in the power system, it should also have a certain generalization ability. Compared with GCN+RNN, EvolveGCN explicitly learns temporal effects instead of learning from node embeddings and has the potential to be used as a testing model for power system topological changes. Therefore, this study aims to investigate whether such models can be used for the state estimation under topological changes. The performance of EvolveGCN will be compared with GCN and GAT models under different conditions, including static and dynamic topologies. This comparison also aims to verify whether static GNN needs to be retrained due to changes in graph structure.

EvolveGCN shows potential but faces performance limitations

However, as mentioned in the previous section, EvolveGCN may encounter performance issues when processing long sequences of graph snapshots, especially since it lacks explicit memory of past transactions (as discussed in the ROLAND study (You et al., 2022)), while power system state estimation typically requires processing large amounts of graph data. Therefore, this study will process the number of graph data and sequences separately, trying to make the modified EvolveGCN able to process a large amount of data containing multiple sequences to better adapt to the research topic of this study. The details of this modification are discussed in Section 3.5.3. In addition, (Moshtagh et al., 2023) has attempted to use GCN+GAT to perform state estimation and prediction of topological changes, but it has not yet involved topological changes of different time lengths and frequencies. Therefore, this study will design case studies for the diversity of topological changes.

3

THEORY AND METHODOLOGIES

This chapter introduces the background theory used to solve the research problem, the theory of the models used, and the model architecture design based on the research question. Regarding background theory, Section 3.1 first introduces the concept of state estimation, and Section 3.2 defines important terms related to Artificial Neural Network (ANN) and explains its workflow. Regarding the theory related to the models, Section 3.3 details the characteristics of different layers used in the model, while Section 3.4 outlines the core methods applied in this study.

Section 3.5 introduces the models used in the study, including their initial settings, adjustments made for this experiment, and hyperparameter tuning options, and details the model architecture design.

3.1. WEIGHT LEAST SQUARE METHOD IN STATE ESTIMATION

Power systems monitor state variables to ensure the stability of the system. State variables are values used to describe the internal state, such as voltage amplitude, phase angle, power flow, etc. However, some of the state variables cannot be directly measured; thus, they require to be inferred through state estimation. State estimation is the process of estimating the remaining unknown state variables through partial measurements. The goal is to develop an optimal representation of the state variables.

Most systems in power networks have limited measurement data due to limitations in sensor deployment. Therefore, estimating the data without measurement is crucial. Weighted Least Squares (WLS) is a common way to implement the state estimation. It combines available measurements and system models to infer unmeasured states, providing a best-fit solution by minimizing the weighted error between measurements and

estimates.

Measurements often have different variances, meaning that some measurements are more reliable than others. Besides, all measurements are subject to random error. Therefore, WLS is designed to focus on the difference between the actual measured value and the estimated measured value, and the reliability on different measurement data. In addition, it is important to note that because the measurement data in the power grid is mostly sparse, which results in an unobservable system, many measurement data are replaced by pseudo-measurements, which are usually estimated based on historical data or average values (Clements, 2011). (3.1) to (3.3) are the core concept of WLS (Rakpenthai et al., 2011). z is the measurement vector, $h(x)$ is the function of the system state vector, its specific form depends on the nonlinear function used. x is the unknown state variable, e is the measurement error, and the core of the WLS method is to minimize the error in estimating x. In order to minimize errors and consider the reliability of different data at the same time, W is added as the weight matrix, which is higher when the measurement is more reliable. Together with W, the sum of the error becomes J.

$$z = h(x) + e \tag{3.1}$$

$$L = \sum_{i=1}^{n} W_i \left(z_i - h_i(x)\right)^2 = (z - h(x))^T W (z - h(x)) \tag{3.2}$$

$$\Delta x = \left(J_{WLS}\right)^{-1} (r_{WLS}) = \left(J^T W J\right)^{-1} (J^T W (z - h(x))) \tag{3.3}$$

(3.3) highlights how WLS updates the state variable x. To minimize the function L, use the Jacobian matrix J, which is the partial derivative matrix of $h(x)$ with respect to x. J is used to express the degree of impact of changes in x on $h(x)$. r refers to the residual function, that is, $z - h(x)$. x will use the iterative method to continuously update in multiple calculations of L. Before calculating the update step size of x, both J and r are multiplied by W. $J^T W J$ serves as the total weighted influence matrix, taking into account the sensitivity of the system to variables, the reliability and contribution of multiple measurements. r is first multiplied by the weight matrix W, and then the direction is adjusted by mapping it to the corresponding state variable through the Jacobian matrix. The new state variable x_{new} is then updated via Δx in the calculation, as shown in the (3.4). After the final J or its norm is less than the allowable value as the tolerance, the iteration stops, and WLS completes the calculation.

$$x_{\text{new}} = x_{\text{old}} + \Delta x \tag{3.4}$$

3.2. ARTIFICIAL NEURAL NETWORKS

This section provides an overview of the concept of the Machine Learning (ML), the ANN, and the Deep learning (DL), and details the workflow and related functions of ANN. It helps to build the necessary foundational knowledge for the method applied

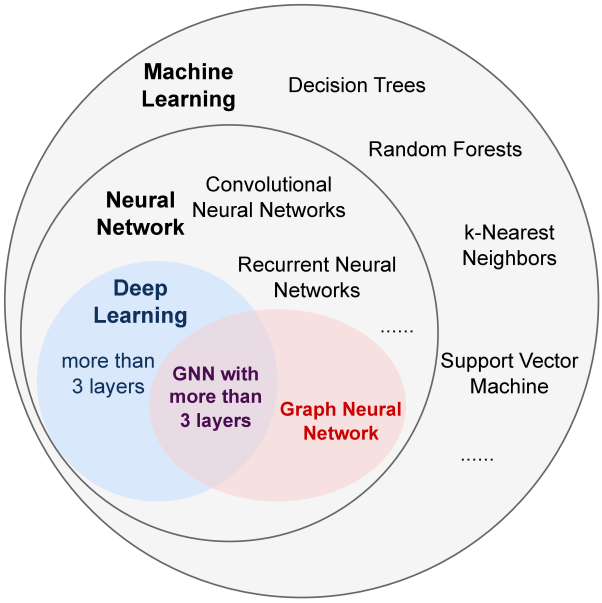


Figure 3.1: Relationship between Machine Learning terminologies

in this research. The relationship within ML terminology is shown in Fig. 3.1. When ranked inductively from most specific to most inclusive, the three terms should be: $DL \subset ANN \subset ML$ (Janiesch et al., 2021). ML is a subset of Artificial Intelligence (AI), and it involves designing different types of algorithms that are capable of learning the inherent decision boundaries for classification or prediction tasks based on the provided data. The core of ML is to learn patterns from data and make predictions. ML uses the available domain knowledge to derive features from the existing data, and applies the purposeful selection of the features to tackle a certain part of the problem (Badillo et al., 2020). Common ML algorithms include decision trees, random forests, k-nearest neighbors, Gaussian mixture models, support vector machines, etc.

ANN, also known as Neural Network (NN), is a subset of ML, and is a computational model that mimics the structure of biological neural networks. Fig. 3.2 shows a feed-forward multilayer NN architecture. ANN is composed of different layers, including an input layer, one or multiple hidden layers and an output layer. Neurons learn through estimation functions and learn the weights required to predict output through hidden layers: Each layer receives input from the previous layer, updates it with its own weight and bias matrices and passes it to the next layer. These matrices will be calibrated during training. The neurons in each layer can be set with activation functions, commonly used ones are tangent sigmoid, rectified linear units (ReLU), and softmax (Bre et al., 2018). When there are more than three layers in the model, the network is referred to as a DL of Deep Neural Network (DNN).

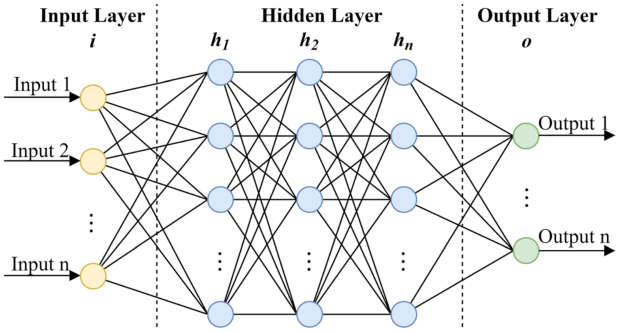


Figure 3.2: The architecture of Artificial Neural Network(Bre et al., 2018)

DL is a subset of NNs that represents NNs with more than three layers, also known as DNNs. NN models with 1-2 hidden layers are called shallow networks. The size of the hidden layer becomes a crucial hyperparameter because it directly influences the network capacity. Due to their simple structure, shallow networks can learn important features independently of other input features, making them suitable for learning tasks involving low-dimensional materials (Mhaskar & Poggio, 2016). However, at the same time, manual feature selection has a greater impact on the results. DNNs aim to learn the useful characteristics from the raw data automatically through multiple layers of processing. Filtering the data through multiple layers enables the model to understand the complex relationships within input data, making DNN more suitable for processing complex high-dimensional tasks (Mhaskar & Poggio, 2016). Besides, because DL is a data-driven approach, it requires a large amount of data to perform effectively.

3.2.1. WORKFLOW OF NEURAL NETWORKS

Since the methods to be discussed and applied in this study are subsets or extensions of NNs, this section briefly introduces its workflow. The workflow is shown in Fig. 3.3, which is modified from the flowchart in (Verma et al., 2013). First, data is collected, preprocessed and splitted before training. Preprocessing involves normalizing or standardizing the raw data, handling missing or non-convergent values, and adding or filtering out noise to improve the model's generalization ability. Afterwards, the cleaned data was divided into training, validation, and test data sets. The training and validation data sets are used for the model training and validation process of hyperparameter tuning, and the test set is treated as unseen data to verify performance.

Second, a model needs to be established, including architecture design, the definition of the activation function, and weight initialization. The activation function is added after the layers to let the layer introduce nonlinearity, and the weight initialization refers to setting the initial values of the weights and biases, which are crucial for proper gradient flow during training. Third, input the preprocessed data into the model, and the entire training process includes forward propagation, loss function calculation, backward

propagation, and optimization. Fourth, the model evaluation is implemented through a validation data set. Finally, after the specified training period is completed, model testing applies the test data set to evaluate performance.

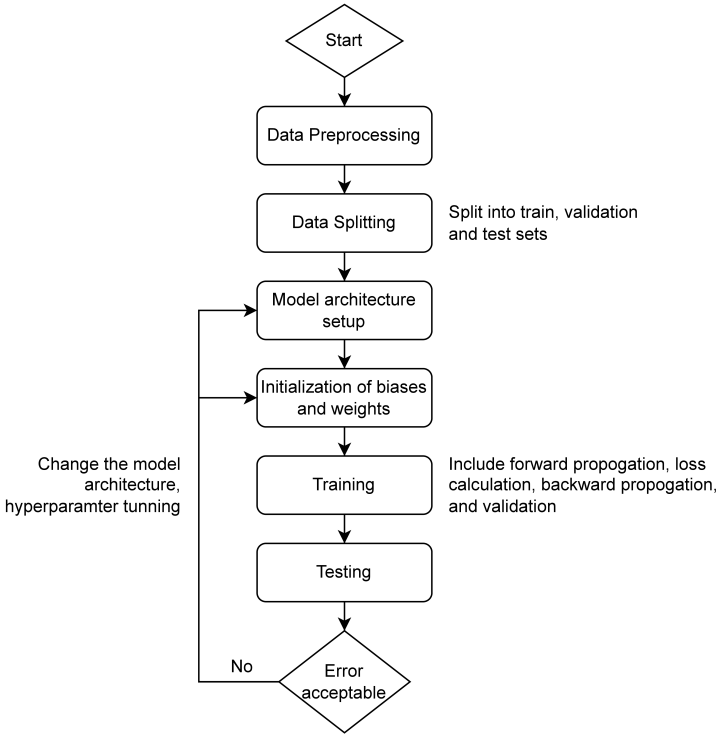


Figure 3.3: The workflow of Artificial Neural Network

Regardless of the type of layers in the NN, they share a similar method of gradient propagation. In the forward propogation, the input is fed into each layer, and the intermediate computations are performed based on the current weight and bias matrices. The model then uses the predicted values and the target values to calculate the loss function. During the backward propogation, the gradient obtained from the loss function is used to update the weight and bias matrices. In different types of layers, which may have different forward passes within the layers, the gradient transmission process for the entire model remains the same, which is also why the layers can be stacked in one model to better utilize the characteristics of different layers.

3.2.2. ACTIVATION FUNCTIONS

The activation function is defined as "a function $g : R \rightarrow R$ that is differentiable almost everywhere. " by Calgar et.al (Gulcehre et al., 2016). The activation function brings the nonlinear capabilities to neurons in layers, meaning that the model can differentiate data which cannot be classified linearly. The choice of activation function for different layers depends on the role and characteristic of each layer. Commonly used activation functions are shown in Table 3.1.

Sigmoid and hyperbolic tangent (tanh) are continuous functions, so they are differentiable everywhere. Due to the simple calculation of derivatives, they are mainly used in shallow networks, and due to their value distribution, they can also be used in the output layer of DNNs. However, they are less advantageous in the hidden layers of DNNs because they achieve soft saturation in the extreme range, leading to the vanishing gradient problem when training DNNs (Ding et al., 2018).

The definition of the activation function provided by Caglar (Gulcehre et al., 2016), that it should be differentiable everywhere, is not always true for DNNs training, since the increasing number of activated neurons may make training difficult. Sigmoid and tanh functions activate almost half of neurons at the same time, which is inconsistent with the finding in neuroscience that only 1-4% of the neurons in the brain are activated simultaneously (Lennie, 2003). Therefore, to obtain sparse output matrices, ReLU, first proposed by Nair and Hinton (Nair & Hinton, 2010), are now widely used in DNNs training. ReLU improves training efficiency due to sparse representation (Ding et al., 2018). In addition, when the input is greater than 0, the derivative is always 1, which solves the vanishing gradient problem. However, the derivative is always 0 when the input is smaller than 0, causing saturation on the negative side, and thus Leaky ReLU is introduced. Leaky ReLU allows small and non-zero gradients to pass when the units are saturated. In this way, no neuron is always inactive during the training (Ding et al., 2018).

Table 3.1: Activation functions and formulas

Activation Function	**Formula**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
ReLU	$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{if } x \leq 0. \end{cases}$
Leaky ReLU	$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{if } x \leq 0,\ \alpha > 0. \end{cases}$

3.2.3. GRAPH NEURAL NETWORK

Graph Neural Networks (GNNs) are the main focus of this research and are a subset of NNs. The main differences between NNs and GNNs are as follows. First, most of the data

in the NNs are text or tabular structures, which are input into the model and processed independently. That means that the learning of different features is completely independent. Each neuron in the layer can see the entire input because it is fully connected. The information in NNs is transferred directly to the next layer after the neurons in one layer are updated. For GNNs, data are processed into nodes and edges, and messages are transmitted within a graph, called message passing. A graph is regarded as an input data, and each neuron corresponds to a node of the graph and is connected according to the connectivity of the graph. These neurons update their status and exchange messages based on connectivity until a stable equilibrium is reached (Scarselli et al., 2008). Therefore, the output of GNN is calculated locally based on neuron connections, rather than fully connected. The main comparison is shown in Table 3.2.

Table 3.2: Comparison of Neural Networks and Graph Neural Networks

Models	**Neural Network (NN)**	**Graph Neural Network (GNN)**
Input data type	Structured (text, tabular)	Graph-structured (nodes, edges)
Training in one layer	Each data point is processed independently	Nodes exchange information with neighbors
Training within layers	Layer-wise transformation	Neighbor aggregation + Layer-wise transformation
Neurons	Each neuron sees the whole input	Each node(neuron) only sees its neighbors
Feature Dependency	Each function is independent	Interrelated through graph structure

3.3. DIFFERENT KINDS OF LAYERS IN GNNS

Both NNs and GNNs consist of different kinds of layers: each layer applies a different activation function, batch normalization, or dropout to the output, allowing the network to learn complex relationships in the dataset. This section focuses on how these layers are applied in GNNs. To design the architecture of a GNN, different layers are added, and the type of layers often determines how the model is named: depending on which layers are incorporated to handle the training process. In this project, four main types of layers are used, each chosen to fit specific characteristics of the data set or different phases of training.

3.3.1. CONVOLUTIONAL LAYER

The main concept of convolutional (Conv) layers is to aggregate information from neighbors to learn changes in the node data itself. Conv layers in traditional NNs compute outputs based on nearby pixels, whereas in GNNs, node aggregation is implemented

based on graph-connected neighbors rather than spatial neighbors. The GNN model constructed mainly using Conv layers during the training process is called Graph Convolutional Network (GCN).

The model can be set up to consider the value of the node itself or only the information from its neighbors. The basic formula of the Conv layer in GCNs is shown in (3.5) (Xu et al., 2023): $H^{(l)}$ is the matrix of node features in layer l, $W^{(l)}$ is the weight matrix in layer l, $\hat{A}$ is the normalized adjecency matrix, and σ is the activation function. D is the degree matrix, which represents the number of neighbors for node i.

$$H^{(l+1)} = \sigma\left(\tilde{D}_i^{-1/2}\tilde{A}\tilde{D}_i^{-1/2}H^{(l)}W^{(l)}\right) \tag{3.5}$$

$\hat{A}$ preserves the graph structure and encodes it to record which nodes are neighbors. Structure information is converted to edge indices before being inputted into the model. The input feature matrix includes the node or edge features and is also the optimization object for model training. $W^{(l)}$ is a trainable element that updates its value after each batch. In the Conv layer, all nodes share the same weight and bias matrices in the same features, making the model computationally efficient and translation invariant.

3.3.2. GRAPH ATTENTION LAYER

Compared to the typical GCN model, which is built by multiple Conv layers, and those layers by default have the same weight at all edges. The edge weights can also be set in the Conv layers, but the manual setting of edge weight sometimes cannot really reflect what exactly is going on in the model. Graph Attention (GAT) layer applies the attention mechanism to capture the attention scores of different edges in the graph, making the model able to catch the subtle differences in importance within edges (Vrahatis et al., 2024). Since this mechanism is especially designed for node-connected structures like GNNs, traditional NNs have no correlated type of layer. Instead of treating all neighbors equally, GAT layer dynamically adjusts the attention vector which is used to compute the attention score and the final attention coefficient to different edges, allowing the model to focus more on important connections while reducing the impact of less relevant ones. The core formulas of the static attention GAT layer are shown in (3.6) to (3.9).

The input node features x_i is first linearly transformed using a shared weight matrix W, often denoted as W_l or W_r, resulting in h_i. Then, the transformed features h_i and h_j (for node i and j) are concatenated and passed through an attention vector a, producing the attention score e_{ij}, which measures how much attention node i pays to node j. After that, a softmax function is applied to normalize the attention scores across all neighbors of node i, producing the final attention coefficients α_{ij}. This coefficient is then applied to re-update the node feature from h_i to h'_i by performing a weighted sum of the neighbor features.

$$h_i = W x_i \tag{3.6}$$

$$e_{ij} = \text{LeakyReLU}\left(a^\top [h_i \parallel h_j]\right) \tag{3.7}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{3.8}$$

$$h'_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} h_j\right) \tag{3.9}$$

However, in the GAT layer with static attention, the order of operations can reduce the individual differences between nodes. It first applies a linear transformation to both the query node (i) and the key node (j) before concatenation (like (3.6)). Since both nodes are projected into the same space before computing the attention score with a shared vector a, the resulting attention ranking tends to become over-smoothed across different query nodes.

To address this limitation, Graph Attention Network v2 (GATv2) introduces the dynamic attention by changing the order of linear transformation and concatenation (Vrahatis et al., 2024). This change makes the individual difference among nodes more obvious, and the attention mechanism to better preserve node-specific differences. The modified process is shown in (3.10). Furthermore, edge properties can be included in the score calculation as additional information, which can be added to the attention score as a bias term, as shown in (3.11). The bias term introduces an additional trainable parameter, usually expressed as w_{edge}.

$$e_{ij,dynamic} = \text{LeakyReLU}\left(a^\top W [x_i \parallel x_j]\right) \tag{3.10}$$

$$e_{ij} = \text{LeakyReLU}\left(a^\top [h_i \parallel h_j] + w_{edge}^T e_{ij}\right) \tag{3.11}$$

3.3.3. MULTILAYER PERCEPTRON NETWORK (FULLY CONNECTED LAYERS)

The fully connected feedforward layer, often called the multilayer perceptron (MLP) layer, applies multiple linear transformations, possibly with activation functions, to process inputs to outputs in one direction (Popescu et al., 2009). While the MLP layer in NNs is structurally the same as in GNNs, its role in GNNs is to transform the node-level outputs that have already incorporated structure-aware features from previous graph-based operations. The key difference between MLP layers and GNN layers is that, In MLP layers, although all nodes still share the trainable matrix, it updates the representation of each node independently without considering its neighbors. (3.12) shows its simplified version of forward propogation (Popescu et al., 2009):

$$MLP(x) = \sigma(W \cdot x + b) \tag{3.12}$$

x is the input, W is the weight matrix, b is the bias, and σ is the activation function. MLP layers are applied to ensure that each node is updated individually after Conv layers. Besides, multiple MLP layers can adjust the hidden size of it, and larger hidden size can catch more individual characteristic of the predicted features.

3.3.4. RECURRENT LAYER

Recurrent Neural (RNN) layers are used for data with temporal relations. The difference between the structure of RNN and MLP is that information not only runs in the forward pass, but also in the backward pass: the output of an intermediate node at the previous time step is recorded by its built-in memory unit, and is fed back to the network as the input of the next time step (Venkateswarlu & Karri, 2022). This mechanism allows RNN to save past information, and also makes RNN layers more suitable for dynamic data sets. RNN contains inner and outer recurrent networks: An inner one establishes a recurrent connection between the input and the hidden state, while an outer one establishes it between the input and the output. (Venkateswarlu & Karri, 2022) That is, the weights inside the inner RNN can be adjusted through hidden states, but not in the outer RNN. Both the Long-Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) are regarded as inner recurrent network. GRU is a class of RNN to increase the speed performance of LSTM model when dealing with the massive number of data (Cho et al., 2014), and the concept of GRU as an example is shown from (3.13) to (3.16).

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{3.13}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{3.14}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1})) \tag{3.15}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.16}$$

W_* are the weight matrices from the input to the gates, also referred to as W_{ih}, and U_* are the weight matrices from the hidden state to the gates, referred to as W_{hh}. The input node features are combined with the hidden state from the previous time step h_{t-1} to get the update gate z_t and reset gate r_t. These two gates determine how much past information should be carried forward or forgotten at each time step.

In GNN, RNN is used when nodes or graph topology changes over time, and RNN is used to capture the temporal dependencies between graph snapshots. The combination of RNN and GCN varies depending on the time series data that different models or cases aim to target. Graph Convolutional Recurrent Network (GCRN) from Seo et al. (Seo et

al., 2018) is a combined RNN and GCN layer model. It first applies GCN layer to achieve node embedding, and then applies RNN for sequence learning. While, in the model EvolveGCN studied in the project, the RNN layer is first applied to obtain the weight matrix, and then the weights are passed to the GCN layer for node embedding. In this way, the the hidden state is regarded as output of the RNN layer in EvolveGCN design, making the RNN an outer recurrent network. The details of EvolveGCN are explained in the next section.

3.4. EVOLVE GCN

EvolveGCN is a method modified from the typical GCN to extend it to multiple time steps and allow changes in topology (Pareja et al., 2020). As mentioned in the previous section, GCN involves the adjacency matrix A, the node embedding matrix H, and the weight matrix W. When it comes to batch-wise learning, the graphs in the same batch share the same weight matrix during training. When there is a change in topology, although the A matrix is updated to reflect the new graph structure, the W matrix cannot be changed within a batch during training, resulting in the matrices used to update the trainable parameters possibly failing to capture the changes caused by the topological changes.

Applied RNN's hidden state to learn the weight matrix

To address the weight update problem, EvolveGCN is considered as a potential model in this research to help reduce the mismatch between the weight matrix and the changing graph structure. The original EvolveGCN introduces RNN layers into the model to update the weight matrix used by the GCN layers. Instead of training the weight matrix by Conv layers independently, EvolveGCN directly uses the hidden state computed by the RNN layers as the weight matrices. Each Conv layer uses a separately RNN and is updated only based on the weight history of that layer, as shown in Fig. 3.4. Therefore, the Conv layer in EvolveGCN obtains the weight matrix from the RNN instead of learning it by itself. This approach allows the weight matrix to include time effects from the data.

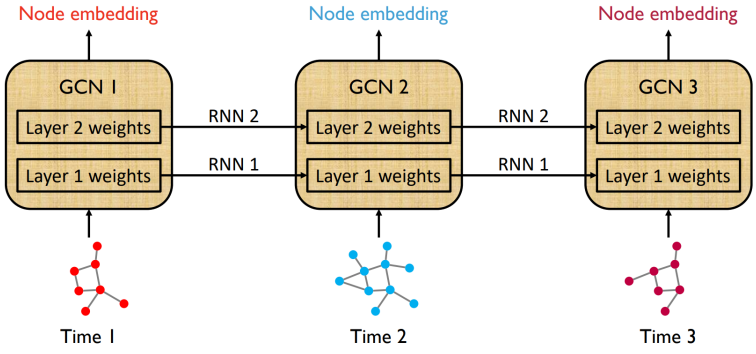


Figure 3.4: Model configuration of original EvolveGCN (Pareja et al., 2020)

Shifting RNN from learning node embeddings to updating the weight matrix

Moreover, when there is a topological change, the RNN layer in EvolveGCN can update the hidden state within the batch using information from previous time steps, and also enabling the model to learn the outputs while considering the topological changes. In contrast to approaches that learn the sequential relationship after node embedding from GCN layers (as seen in combinations of GNN and RNN like (Seo et al., 2018)), the role of the RNN in EvolveGCN is shifted to learning the weight matrix itself beforehand. In methods that apply RNNs as sequence learners for node embedding, a limitation arises in that the node information and size need to be known in advance, making prediction or regression of new or reappearing nodes difficult. By shifting the task of the RNN from learning node embeddings to updating the weight matrix, EvolveGCN resolves the size difference problem of nodes.

EvolveGCN provides two implementation approaches: EvolveGCN-O and EvolveGCN-H. The -O version extends the standard LSTM from vector to matrix form, and considers only the hidden state as input. The -H version considers also the node embedding as input. The RNN of both versions can be changed to other RNN architectures. Since this study mainly compares the impact of changes in graph structure, the -O version, which is more focused on graph structure, was selected (Pareja et al., 2020). The detailed explanation is as follows.

EvolveGCN-O model architecture

In -O version, the model treats $W_t^l(x)$ as the output of the dynamical system. In this case, the LSTM network is used to model the input-output relationship. The formula is written as:

$$W_t^{(l)} = \text{LSTM}(W_{t-1}^{(l)}) := f(W_{t-1}^{(l)}) \tag{3.17}$$

Where $f()$ is the RNN architecture, as shown in Section 3.3.4. Combining these steps with the remaining calculations in GCN, the overall function is written as:

Algorithm 1 EvolveGCN Implementation: EGCU-O

1: **function** EGCU-O($A_t, H_t^{(l)}, W_{t-1}^{(l)}$)
2: $\quad W_t^{(l)} \leftarrow \text{LSTM}(W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} \leftarrow \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

The overall message passing diagram of original EvolveGCN-O is shown in Fig. 3.5. The original EvolveGCN used sequence learning instead of batch-wise learning, that is, a backward process was implemented at the end of the sequence, which is the entire dataset. However, this study used batch-wise learning, which means that the frequency of gradient updates is inconsistent with the original EvolveGCN design, so relevant adjustments must be made. In addition, based on the considerations of computational efficiency and required memory units, the RNN network was changed from LSTM to GRU. See Section 3.5.3 for more details on the actual implementation.

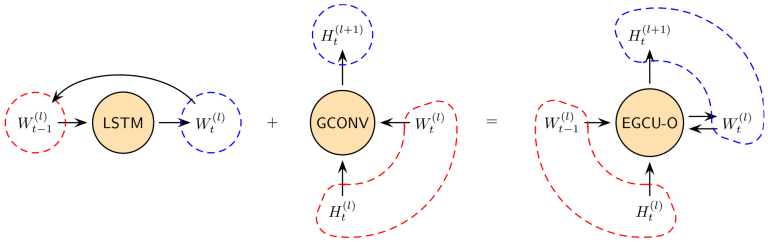


Figure 3.5: Message passing diagram of EvolveGCN-O (Pareja et al., 2020)

3.5. MODEL SETUP AND CONFIGURATION

After detailing the required background and model theory, this section explains the three GNN model settings used in the study. Firstly, under the default condition of not changing the topology, three GNN models and the WLS method were used for verification. Secondly, three GNN models are adopted under different topological changes and their characteristics are analyzed. The three GNN models are: GCN, GAT and EvolveGCN. The comparison between GAT and EvolveGCN aims to explore the impact of temporal and spatial effects in a single graph structure on state estimation of topologically varying data and further discuss their performance.

3.5.1. WLS MODEL ARCHITECTURE

This study applies the Weighted Least Squares (WLS) method as the basic method to evaluate the state estimation performance of the three test models under noise. This method is only compared with the other three models in the default case. Using the traditional WLS method, the process is as follows.

The WLS method views the voltage and angle as the state vectors, and it begins with a flat start, where the voltage magnitude V is initialized to 1 and the phase angle ϕ is set to 0. The phase angle of the slack bus is set to be fixed during the iteration, and the voltage of the slack bus may be fixed in some cases. By extending the original formula in Section 3.1 and applying the actual variables in this study, the applied formulas are shown in (3.18) and (3.19). r_{WLS} is the residual value as in (3.3). Given the admittance matrix Y and the power measurements P and Q, the residual function R is calculated as the difference between the calculated and measured active power, more details are illustrated in Section 4.1.6. After that, the real and imaginary parts are saved separately.

$$\begin{aligned} r_{\text{WLS}} &= z - h(x) \\ &= R(Y, P_{\text{meas}}, Q_{\text{meas}}, V, \phi) \\ &= \left[(P_{\text{meas}} - P_{\text{pred}}), (Q_{\text{meas}} - Q_{\text{pred}})\right] \end{aligned} \tag{3.18}$$

$$J = \begin{bmatrix} \frac{\partial p}{\partial v} & \frac{\partial p}{\partial \phi} \\ \frac{\partial q}{\partial v} & \frac{\partial q}{\partial \phi} \end{bmatrix} \tag{3.19}$$

Using the residual values, the Jacobian matrix J in (3.19) is then computed based on the inputs, capturing the sensitivity of the power flow equations to voltage magnitudes and angles. The weight matrix W is initialized as the identity matrix and can be adjusted as an adjustable parameter, and its size is adjusted according to the number of nodes. The r_{WLS} and the J are first transferred by the W, and then are used to calculate the update vector. The vector is calculated by solving the linear system, and the process is iterated repeatedly after updating the voltage and angle until the norm of the residual vector is small enough.

3.5.2. GCN AND GAT MODEL ARCHITECTURE

The static GNN models contain multiple message-passing (Conv or GAT) layers, with self-loop added, Leaky ReLU, ReLU and tanh as the options of the activation function, and the dropout is applied after every Conv (GAT) layer. The edge weight is fixed and equal on the whole graph during training, and two MLP layers are added after. The trainable elements are the weight and bias matrices in every layer, and hyperparameter tuning includes the number of layers, dropout rate, activation function, hidden size, learning rate (lr), and the L2 regularizer (weight decay).

GCN model architecture

The model configuration is shown in Fig. 3.6. The GCN model applies Conv layers as the message-passing layers, and utilizes the Glorot initialization scheme for weight initialization. The GCN model starts with the construction of the input features. The node feature matrix X is defined with a shape of $4 \times N$, where 4 is the number of features, namely active power, reactive power, voltage and phase angle, and N represents the number of buses in the network. The node feature matrix is fed into the model along with the edge index converted from the adjacency matrix A. Following the Conv layer, MLP layers are applied to transform node embeddings into the desired output representation. The MLP consists of hidden layers with a predefined size, and the final output layer maps the hidden representation to the output channel. The model output retains the same shape as the input. This structure ensures that the final predictions maintain the spatial and feature relationships embedded within the input data.

GAT model architecture

Similar to GCN, they are models that focus on the optimization of a single graph feature. The GAT model changes the Conv layer in GCN to the GAT layer. Instead of fixing the edge weight among the entire graph, GAT applies additional edge features as edge attributes, learns attention weights through the node embedding results after linear transformation of node features and shared weight matrix, and calculates the attention score of each graph. This mechanism makes the edge weights different in each graph. Its shared weight, bias matrices and attention vector are all trainable parameters. The edge

properties applied in this model include active and reactive power("p_from_mw" and "q_from_mvar"). The rest of the settings are the same as GCN. The model configuration is also shown in Fig. 3.6. The trainable parameters of both methods are listed in Table 3.3.

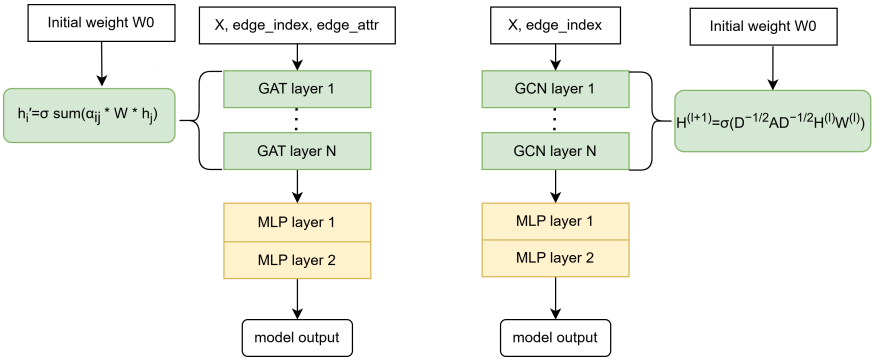


Figure 3.6: Model configuration of GCN and GAT

3.5.3. EVOLVEGCN MODEL ARCHITECTURE

The implemented EvolveGCN model is modified from the EvolveGCN source code in the Pytorch Geometric temporal ("PyTorch Geometric Temporal", n.d.), which is also originally from (Pareja et al., 2020). The source code contains one single RNN layer, one single Conv layer of GCN with no specific edge weight assigned, no explicit activation function, and only one linear out layer.

Modifying EvolveGCN for batch-wise training

This study adjusted the several points of the model. First, the original EvolveGCN model is applied to a small dataset, it uses fewer time steps and sequential learning, which means it updates the weight matrix after each graph, transmits the weights to the entire dataset, and the backward process is only performed at the end of the dataset. The weight matrix is updated directly by the hidden states in the RNN layer and adjusted at each time step. However, the problem setting of this project is semi-supervised learning of state estimation in the network, which requires a large amount of data to help capture the feature values, so batch-wise training is more appropriate than pure sequence learning. However, general batch-wise training does not update the weights for each graph, which does not follow the concept of EvolveGCN. Additionally, the complexity of the model architecture and RNN layers may lead to training instability, exploding gradients, or a higher likelihood of overfitting. Therefore, this study designed a modified EvolveGCN to make its weight update concept suitable for batch-wise training.

Modified EvolveGCN

Considering computational efficiency and memory units, this study replaced the LSTM in the EvolveGCN-O method with GRU (as stated in the method paper, the RNN layer can be replaced with other types), and the RNN layer in the Pytorch geometric source

code is also GRU. If the original method is applied, i.e. self-evolution of weights, and LSTM is replaced with GRU, one needs to substitute both the GRU input and hidden state with the weight matrix of the previous time step. Besides, the weight as GRU input is transmitted throughout the entire dataset. The example code is shown in Algorithm 2, and only the forward pass of a single RNN + GCN is shown.

In the modified EvolveGCN, the RNN still updates the weights after each graph and regards its weights as the hidden state of the next time step (the next graph), but the input is still the initial weight. The purpose of this change is to allow the trainable weights within the RNN (W_{ih}, and U_*) to have more possibilities for weight updating directions and avoid relying entirely on historical states. The comparison of weight updates between the original and modified EvolveGCN can be seen in (3.20) and (3.21). Moreover, the modified EvolveGCN treats the batch size as a sliding window in sequential learning, that is, the backward process is performed after each batch, as shown in Fig. 3.7. In this way, the training logic of the same batch is similar to the original entire dataset in EvolveGCN. In order to avoid gradient explosion, the modified EvolveGCN reduces the frequency of weight transmission (only records in the hidden state instead of the input value of GRU) and increases the number of backward times. This method aims to imitate the idea of the EvolveGCN weight update method, so that each graph in the sliding window has a weight and the transmission of weights is continuous. The example code is shown in Algorithm 3.

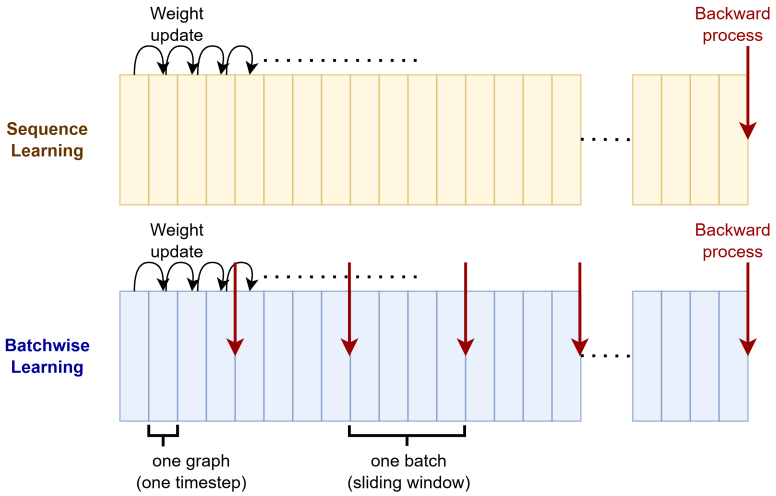


Figure 3.7: Comparison between Original and Modified EvolveGCN: Backward frequency between sequence learning and batch-wise learning

$$\text{Original EvolveGCN: } W_t^{(l)} = \text{GRU}(W_{t-1}^{(l)}, W_{t-1}^{(l)}) \tag{3.20}$$

$$\text{Modified EvolveGCN: } W_t^{(l)} = \text{GRU}(W_{initial}^{(l)}, W_{t-1}^{(l)}) \tag{3.21}$$

The modified weight update method is independently applied to each RNN layer and its connected GCN layer, which is similar to the original design of EvolveGCN, but the frequency of the backward process and the way the weights are transmitted are different. Multiple RNN+GCN groups are implemented by adding one more layer of loop outside the loop within a single batch.

Second, in addition to the method of updating the weight in the recurrent layer, this project adds the functions that multiple Conv layers and linear layers are allowed compared to the source code. Third, add the activation function, dropout, and batch normalization. The hyperparameter tuning includes the amount of Conv layers, the dropout rate, the activation function, the hidden size, the learning rate (lr) and the L2 regularizer (weight decay). The whole model configuration is shown in Fig. 3.8, and the trainable parameters are listed in Table 3.3.

Algorithm 2 Original EvolveGCN: Weight Evolution

Initialize $W \leftarrow W_0$
Initialize hidden state h
for $t = 1$ to T **do**
    $h \leftarrow \text{GRU}(W, h)$
    $W \leftarrow \text{extract\_weight}(h)$
    $X_t \leftarrow X_t \cdot W$
end for

Algorithm 3 Modified EvolveGCN: Weight Evolution

Initialize $W \leftarrow W_0$
Initialize hidden state h
for $t = 1$ to T **do**
    $h \leftarrow \text{GRU}(W, h)$
    $W_t \leftarrow \text{extract\_weight}(h)$
    $X_t \leftarrow X_t \cdot W_t$
end for

Note: In Algorithm 2, T refers to the entire time range of the dataset, allowing the weight W both in the input and hidden state of GRU to evolve across all time steps. In contrast, Algorithm 3 only evolves W in the hidden state throughout the entire dataset, does not propagate the evolved weight in the input to the next graph, and the backward process is implemented after each batch (T in Algorithm 3).

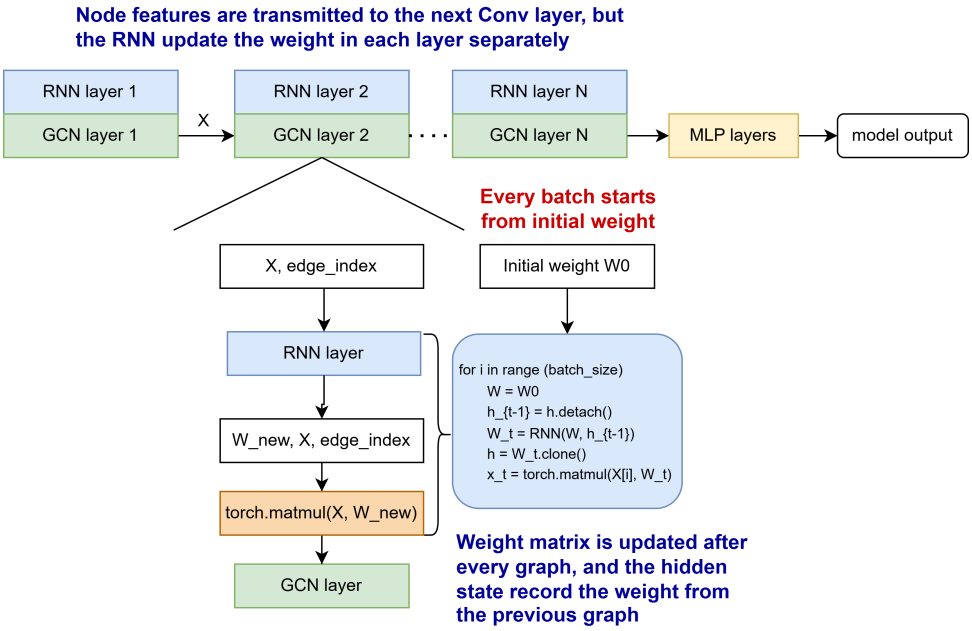


Figure 3.8: Model configuration of EvolveGCN

Table 3.3: The trainable elements of three models

GCN		GAT		EvolveGCN	
Element	Parameters	Element	Parameters	Element	Parameters
GCN	weight / bias	Attention	att / bias	RNN	ih: weight / bias hh: weight bias
MLP	weight / bias	Shared weight	lin_l: weight / bias lin_r: weight / bias	Batch norm	rnn: weight / bias gcn: weight / bias
		Edge weight	weight	MLP	weight / bias
		MLP	weight / bias		

4

RESEARCH SETUP

This chapter elaborates the research setup, including the power networks used, the data generation and storage, the workflow, and the loss functions applied.

4.1. RESEARCH SETUP

4.1.1. TEST NETWORK SELECTION

In this research, two testing power networks are applied to verify the performance in different cases: IEEE 14-bus, and IEEE 57-bus standard test networks. The summary of the elements of the network is shown in Table 4.1. The topologies of the networks are shown in Figs. 4.1 and 4.2. Although topological changes involving DERs are more frequently observed in distribution grids, two factors are taken into consideration when choosing transmission networks for testing:

Table 4.1: The elements of the networks

Network	Bus	Load	Link	Slack bus	Transfo	Gen
14-Bus	14	11	17	1	3	5
57-Bus	57	42	65	1	16	7

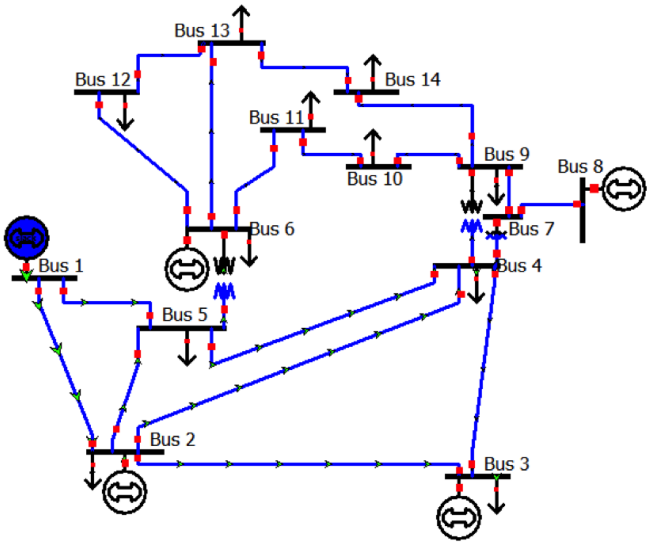


Figure 4.1: IEEE 14-bus network topology (ICSEG, n.d.-a)

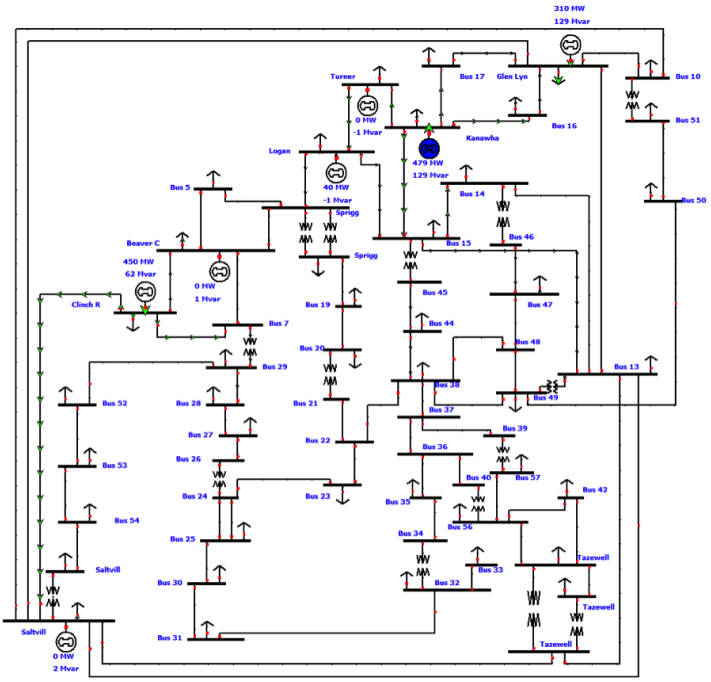


Figure 4.2: IEEE 57-bus network topology (ICSEG, n.d.-b)

Selection of network structure for topological change simulation

First, the standard testing distribution networks, such as the IEEE 33-bus or IEEE 123-bus systems, are radial in structure rather than meshed. The radial structure limits the possibilities for simulating different kinds of topological changes, as experiments are typically restricted to switching operations. In contrast, meshed structure offers greater possibility to perform different types of topological changes. Therefore, by adjusting the system voltage of the transmission networks to medium-voltage levels (13 kV), this research aims to leverage the topology of meshed systems while maintaining relevance to DER-related dynamics.

Relevance of IEEE transmission grids to DER contexts

Second, under the growing trend of smart grids, the boundary between transmission and distribution networks has become increasingly blurred. In such environments, DERs may be connected at medium-voltage levels, and their event-driven actions, such as trips or reactive power injections, can resemble the topological change scenarios simulated in this study. Hence, although the networks used here are based on standard transmission test cases, they can reasonably reflect medium-voltage grids under high DER penetration conditions.

Data generation and training framework with physical constraints

The measurement data which are viewed as the target values for the training process are taken from the simulation of these two networks. The entire training process is considered semi-supervising for the target values to not participate in the training process directly. The output of the model is further calculated through the added physical relation of the power system as a soft constraint: The residual function is implemented on the predicted output and sends out the predicted features to compare with the target values. The details of data generation and storage from simulation, and how the residual function and loss function interact during the training process, are discussed below.

4.1.2. DATA GENERATION

The measurement data are generated from the simulation. Firstly, the load profile of the European Network of Transmission System Operators for Electricity (ENTSO-E) (“Power Statistics”, n.d.) is considered. The load profile for multiple consecutive days is saved, normalized, and applied as a scaling factor to the default load values in the testing networks, so that the actual demand change can be simulated. To simulate the occurrence of topological changes, the scaling factor in the 14-bus network is shifted by 1.1, and by 0.4 in the 57-bus network to highlight the topological changes. The stored load profile is shown in Fig. 4.3, with an interval of 15 minutes and a total of 960 data points.

Secondly, the topological changes are performed before the Pandapower power flow calculation to ensure convergence and that the network is reloaded to its original state after every time step. In this way, during the simulation, each time step is treated as an independent graph. By running simulations separately, topological changes can be saved

along with the graph, and relationships between graphs are maintained through time-load profile data. In order to balance the amount of data with and without topological changes and make the error in the test set more balanced, the performance of data points with and without topological changes will be evaluated separately.

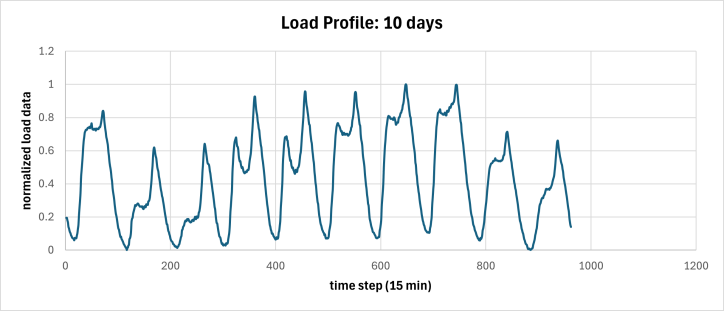


Figure 4.3: Load profile: 10 days

Topology change case design

The topological changes are set as different test cases as shown in Table 5.3 in Chapter 5. The test cases were designed to investigate the effectiveness of the model for the removal or interruption of different electrical components over different time frames: Certain generators or links are taken offline or removed in specific time steps. These scenarios are designed to simulate a variety of realistic events. Shorter-duration changes emulate fast events such as DER tripping or localized faults, whereas longer-duration changes are intended to represent slower dynamics like scheduled maintenance or extended component outages. In addition, the number of times a topological change occurs within a fixed length of data, i.e., the frequency of occurrence, is also tested. The electrical elements, duration, and frequency covered in this study are shown in Table 4.2. These events may occur due to IEEE 1547 safety standards for DERs systems (IEEE Standard Association, 2022), economic stretegies or operational considerations, or due to accidents or manual scheduling.

It is important to note that the resolution of the ENTSO-E data is 15 minutes, which is not precisely aligned with the fast time scales at which some of these physical events may occur. However, from a modeling perspective, each time step is treated as a snapshot and simulated independently, which allows the model to numerically evaluate its robustness and adaptability under structural disruptions of different frequencies and durations. In particular, these setups function as numerical stress tests to investigate model sensitivity to both abrupt and gradual topological changes. If higher-resolution data were available, the same methodology could be applied to better capture the temporal dynamics and cascading effects over time.

Create "Unseen cases"

In addition, the cases are categorized based on whether the model is exposed to topolog-

ical changes during the training phase. For cases that are not given topological change data in the training phase, the model will use the model trained with the default case, and only add the topological change to the test data to measure the performance of the model in predicting "unseen topological changes". The concept of topological change case design, including the unseen cases, is shown in Fig. 4.4.

Table 4.2: Topological change scenarios and their practical mapping

Type	Timescale	Frequency	Practical Scenario
Line Removal	Short (3 steps)	Once	Fault in local line, protection-induced line isolation
		Frequent	Recurrent protection-triggered fault clearing under instability
	Long (10-40 steps)	Once	Long-term feeder outage due to maintenance or damage
		Frequent	DERs output oscillation causing disconnection of lines
Generator Offline	Short (3 steps)	Once	Momentary trip due to overvoltage or anti-islanding trigger
		Frequent	Short-circuit-induced repeated inverter tripping
	Long (10-40 steps)	Once	Scheduled shutdown due to grid congestion or curtailment
		Frequent	Load displacement strategy due to dynamic pricing

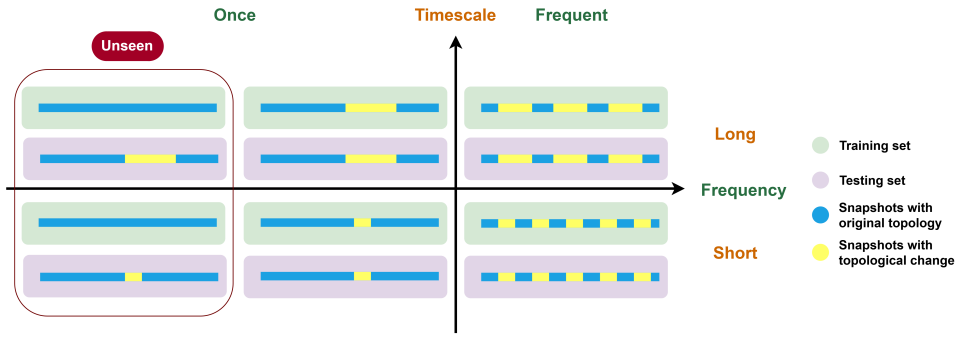


Figure 4.4: Topology change case design concept diagram

4.1.3. DATA STORAGE AND PRE-PROCESSING

After the Pandapwer network simulation, the adjacency matrix A, and the feature matrix X need to be saved for models' training, the admittance matrices G and B are needed for residual function calculation, and extra link features L are saved as edge attribute for

GAT. The saved features contain active power (P), reactive power (Q), voltage (V) and phase angle (ϕ) of the buses. The active and reactive power of buses are calculated by (4.1) and (4.2), and later on divided by the base power sn_mva to convert their units into pu system.

The A, X, L, G and B are saved in the pickle. In the simulation, all data are simulated based on the constructed network and are therefore considered fully observable. However, state estimation is implemented when only part of the data is observable. Furthermore, the measurement data usually contains noise. Therefore, noises and masks are added to the data before the training phase.

$$P_{bus} = P_{gen} - P_{load} + P_{slack} \tag{4.1}$$

$$Q_{bus} = Q_{gen} - Q_{load} + Q_{slack} \tag{4.2}$$

The measurements entered into the model are only part of the saved features, which will be done by setting the mask in preprocessing. In preprocessing, P, Q and part of V and ϕ containing noise will be used as input data, while the remaining data will be masked as non-observable. The masked data remains non-observable throughout the training process, and no additional labels are provided for the non-measured data in the loss function calculation. The models are expected to execute the semi-supervise learning: predicting those non-observable values through the given measurement data. The workflow of data generation is shown in Fig. 4.5.

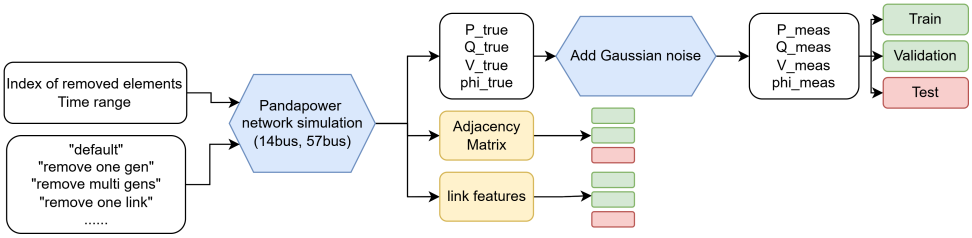


Figure 4.5: The workflow of data generation

4.1.4. WORKFLOW

After the data are generated, saved, and preprocessed, they are split into training, validation, and test set to run the training process. The training workflow is shown in Fig. 4.6. The chosen features are first standardized to the abstract domain, blocked the masked data, and fed into the model as inputs. After the forward propogation process, the model outputs are destandardized to the physical domain. Second, together with the admittance matrices saved from the simulation, the destandardized prediction is fed into the residual function to get the predicted active and reactive power. Third, the loss function is calculated for a single batch size and finally fed into the optimizer for optimization. Run the next batch and later the next epoch. After the training phase, the model is

tested in a similar workflow without gradient update, as shown in Fig. 4.7. The detailed explanation of the standardization process, the loss function and the residual function calculation is provided in Sections 4.1.5 to 4.1.7.

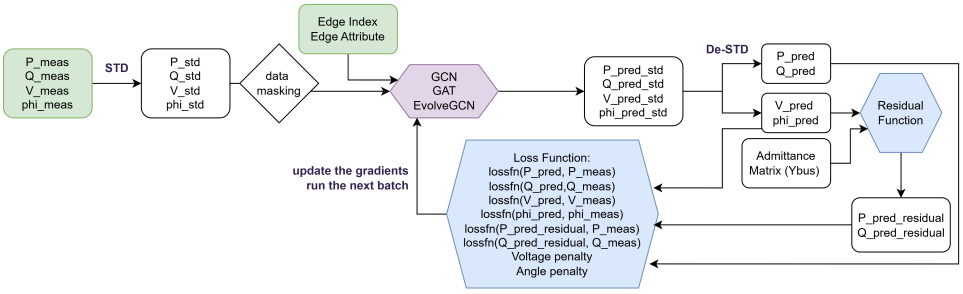


Figure 4.6: The workflow of training

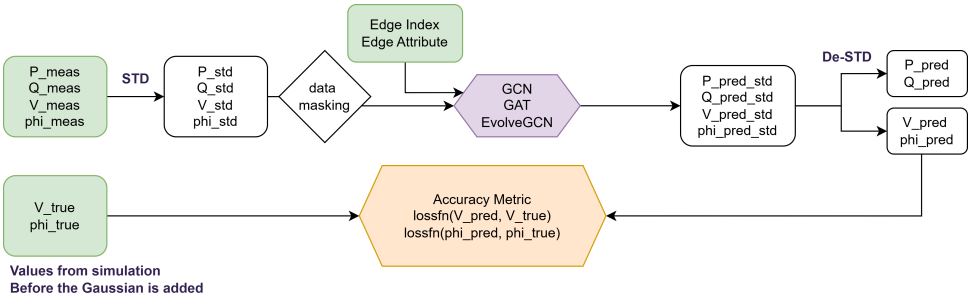


Figure 4.7: The workflow of testing

4.1.5. STANDARDIZATION PROCESS

When the input data has a similar scale and are well distributed, it helps prevent certain features from dominating others, speeds up convergence, and alleviates vanishing gradient problems, especially in models that applied gradient-based optimization, such as NNs. To convert the input data from the physical domain to a well-distributed abstract domain, several considerations are taken into account.

First, only the training data set is used to calculate the mean and standard deviation to prevent data leakage. Second, this project computes the mean and standard deviation not only for each feature but also for each node. In other words, each node has its own mean and standard deviation in every feature, and there are also global mean and standard deviation for every feature. Third, in a semi-supervised setting, some buses may not have available measurement data, so their averages and standard deviations are inaccurate, requiring an additional process: for the standardization process in the input data, the masked part is forced to be zero, and for the destandardization process in the

output data, if the originally masked data is a voltage, it applies the global standard deviation value and an offset to implement the destandardization process: one is added as the voltage offset, and zero is set as the phase angle offset, which are a bit like the initial condition for the model to start iterating and optimizing.

Taking into account the above considerations, the (de)standardization formula is as follows. X is the node feature, including P, Q, V, ϕ. i represents the node index, σ is the standard deviation, $X_{i,pred\_std}$ is the output of the model, and mask is an array that records the index of masked data.

$$X_{i,std} = 0, \quad i \in \text{mask} \tag{4.3}$$

$$X_{i,std} = \frac{(X_{i,meas} - X_{i,mean})}{X_{i,\sigma} + 10^{-5}}, \quad i \notin \text{mask} \tag{4.4}$$

$$X_{i,pred} = X_{i,pred\_std} \cdot (X_{i,\sigma} - 10^{-5}) + X_{i,mean}, \quad i \notin \text{mask} \tag{4.5}$$

$$V_{i,pred} = V_{i,pred\_std} \cdot (V_{global\_\sigma} - 10^{-5}) + 1, \quad i \in \text{Vmask} \tag{4.6}$$

$$\Phi_{i,pred} = \Phi_{i,pred\_std} \cdot (\Phi_{global\_\sigma} - 10^{-5}) + 0, \quad i \in \phi\text{mask} \tag{4.7}$$

Besides the initial offset setting for the non-observable data points, another additional process is done after destandardization: taking the average of its neighbors (similar to the extra step after the neighbor process in a Conv layer), as shown in Fig. 4.8. This step helps limit the values of the bus without measurements, preventing them from straying too far from their neighbors.

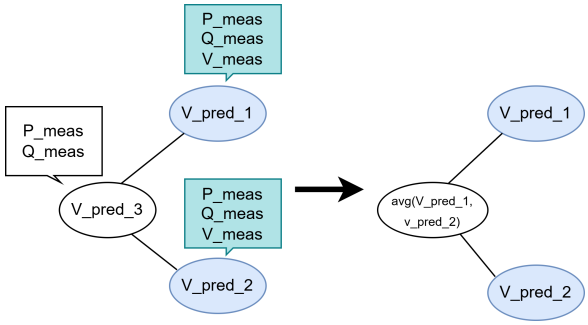


Figure 4.8: The extra averaging process

4.1.6. RESIDUAL FUNCTION

This study applies the residual function as a physical soft constraint to guide the model. This function applies model predictions of V and ϕ and the admittance matrix stored

from the simulation to calculate P and Q values. Because the functions use the power flow equations to calculate predicted P and Q, they essentially follow the physical laws of the power system. Furthermore, using a residual function will increase the results and comparability with traditional methods such as the Newton-Raphson (NR) method, since the iterations of both methods are based on the residual function. The original residual function is shown in the following formula. The traditional method will directly subtract the measured values of P and Q after the calculation is completed (such as iteration), while the GNN model will splice the difference between the predicted value and the measured value with other different loss functions.

$$Y = \mathrm{G} + j \cdot \mathrm{B} \tag{4.8}$$

$$v_{\text{complex}} = V \cdot \exp(j \cdot \phi) \tag{4.9}$$

$$S_{\text{pred}} = \text{diag}(v_{\text{complex}}) \cdot \text{conj}(Y \cdot v_{\text{complex}}) - (P_{\text{pred}} + j \cdot Q_{\text{pred}}) \tag{4.10}$$

$$\text{residual} = [(P_{\text{meas}} - P_{\text{pred}}), (Q_{\text{meas}} - Q_{\text{pred}})] \tag{4.11}$$

4.1.7. LOSS FUNCTION

The loss function is used to calculate the losses of the targeted parameters, which are then converted into gradients for the weight and bias in each layer during the backward process. In this research, depends on the characteristic of the model and target values, different loss function are applied, including Mean Square Error (MSE) and Mean Absolute Error (MAE). For example, MSE may be suitable for numerical data that tends to be stable because it is more sensitive to small changes. If the data is overall stable and has no significant outliers or large deviations, MSE is better at capturing the overall trend. MAE may perform better for drastic topological changes on short time scales, which is because: MAE treats all errors equally, so it does not let short-lived drastic changes dominate the loss function. The formula of loss functions are shown below.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{4.12}$$

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{4.13}$$

The components used in the loss function in this research are as follows. First, for nodes with measured values, the loss between predicted and measured data values is included, while those that are masked do not have this value. Second, voltage and angle penalties are added. These penalties serve as an additional regularization term after the training process, with stability criteria established based on physical constraints. The main formulas are shown in (4.14). W is the weight of different features, and w is the weight of different nodes. The weight adjustment within the nodes is tuned during training to help the model prioritize nodes that are more challenging to predict. Additionally, the

weight adjustment within features is based on the characteristics of different topological change scenarios. For instance, in some cases, the PV bus might be primarily responsible for active power adjustment, and its voltage may remain relatively stable. Therefore, the weights assigned to both the nodes and features are used to correct the imbalance effects between features and buses.

$$L = \sum_{j=1}^{4} \left(W_j \cdot \sum_{i=1}^{num\_buses} (w_i \cdot \text{lossfn}(X_{\text{i,pred}}, X_{\text{i,meas}})) \right) + \text{ReLU}(V - V_{max}) + \text{ReLU}(V_{min} - V) + \text{ReLU}(\phi - \phi_{max}) + \text{ReLU}(\phi_{min} - \phi) \tag{4.14}$$

5

Case Study and Results

In this chapter, the case study of both networks are discussed. First, the scope and decisions for hyperparameter tuning are explained in Section 5.1. Section 5.2 illustrates the performance comparison between the original EvolveGCN design and the modified version. Section 5.3 presents the topology of two networks and the case studies defined for the experiment. Afterwards, based on the experimental results of the two networks, the main findings will be described separately.

5.1. Hyperparameter Tuning

In this research, approximately 30% of the voltage and phase angle data is provided as measurement data, as shown in Table 5.1. The selection of given data is based on practical considerations, including data from the slack bus and PV bus, as well as concerns about the neighboring effects that the models need to capture. Some bus data are provided to account for message-passing distance.

Table 5.1: Number of nodes with the given measurement value in both test networks

	Number of nodes given measurement	
Network	Voltage	Phase angle
14-bus	7	2
57-bus	22	8

Number of layers

For EvolveGCN, the number of Conv layers ranges from 1 to 3, and each Conv layer is paired with an RNN layer for weight updates. The reason for not having too many layers

is that RNN layers are prone to overfitting due to their complexity based on gradient accumulation. Both the original design and the modified method are experimented with different batch sizes, and Section 5.2 details the performance of the modified method compared to the original EvolveGCN. For GCN and GAT, the number of layers is set between 3 and 8 because the distance the message is transmitted has a great impact on the results. Additionally, integrating information from distant nodes may also affect the results.

Hidden sizes, batch sizes, epochs and early stopping

The hidden size of the MLP is adjusted in the range of 16 to 64 to evaluate how well the representation learned by the GCN captures relevant feature features. The final size was chosen on the basis of the verification performance. The batch size of EvolveGCN varies between 2 and 48 because it is directly related to the length of the sliding window that captures sequential information over time. This parameter is adjusted based on the length of the topological change event. For GAT and GCN, a fixed batch size of 48 is used since they do not rely on temporal modeling across batches. The learning rate is adjusted on the basis of the complexity of the model, with weight decay and dropout rates modified accordingly. The maximum number of epochs is set to 120 for both networks. The minimum learning rate is set at 0.0005, and early stopping is triggered when there is no improvement greater than 0.0001 for five epochs. The γ set in the learning rate scheduler is 0.8.

Weights of different elements in loss functions

The weights of different elements in the loss function are adjusted based on their initial values at the beginning of training. This ensures that the model optimally balances its focus across all elements. Since numerous complex factors influence performance across different nodes—such as whether data is given or not, the level of oscillation caused by a topological change event, and the inherent characteristics of each bus (e.g., PV bus, PQ bus)—the node-specific weights are fine-tuned through trial and error. The model is first trained, and then its performance is analyzed to identify buses that require more focus, after which the training phase is repeated with adjusted weights. The hyperparameter values are summarized in Table 5.2.

Table 5.2: Summary table of hyperparameters

	epoch (max)	learning rate			MLP	layers		weight decay	drop-out rate	batch size
		initial lr	step size	γ	hidden size	RNN	Conv/ GAT			
Evolve GCN	120	0.003	5	0.8	16-64	-	1-3	0.03	0.45	1-48
GCN/ GAT	120	0.003	5	0.8	16-64	-	3-8	0.03	0.45	48

5.2. PERFORMANCE OF THE MODIFIED EVOLVEGCN

You et al. (You et al., 2022) have mentioned that the shortcomings of the original design of EvolveGCN on large sequence data: it is impossible to save the explicit past transaction data. This study designs a modified EvolveGCN architecture by adjusting the weight transmission method and the frequency of weight initialization, and this section presents the results of the modified version along with the original design.

To implement the original design in the current setting, the weights used as the GRU input and hidden states are transmitted across the dataset. However, since the original dataset in this method paper is quite small and the dataset used in this study is relatively large, to prevent the gradient explosion, the frequency of the backward process will not be the same as the original method, but will depend on the batch size. The modified version was also tested with different batch sizes. This method does not pass W in the input: it will use the initial weight after each graph as the input to the GRU for initialization. The hidden state record of the weight updates is still passed through the entire dataset, and the batch size indicates the frequency of backward updates. The RMSE results of the voltage prediction in Case 6 of Table 5.4 are shown in Fig. 5.1.

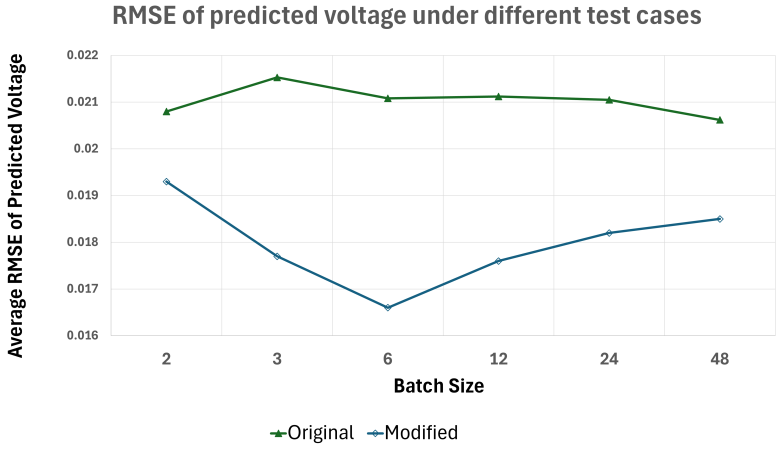


Figure 5.1: RMSE of predicted voltage for different batch sizes for the original and modified EvolveGCN

Compared to the original design, the modified EvolveGCN does not explicitly transfer weights as inputs in the same batch, nor does it continuously pass weights as inputs to the entire dataset. The results show that the RMSE prediction performance of the modified method is better than the original design regardless of the batch size. The batch size applicable to the case studies is determined by testing the default case. After testing with the default cases, it was decided to run additional case studies with a batch size of 3 for the 14-bus network and a batch size of 6 for the 57-bus network.

5.3. CASE DEFINITION AND TOPOLOGY

In the following case studies, each dataset contains only topological variations of a specific electrical component. Although the clear consequence of the topological change is voltage change, the difference in removing the generator and the connection is whether the graph structure changes or not. In the Pandapower simulation, the generators are first connected to the bus and then to the rest of the network, so removing a generator does not cause the graph structure to change, but removing a link does. For the case where the topological change is visible (referred to as "Seen Case" in the following results discussion), the model relies on incomplete voltage and phase angle labels, as well as predicted labels calculated by the residual function, to learn the topological change during the training phase, and applies the same topological change to the test data as validation. The goal is to evaluate the performance of each model for different types of topological changes. For topology-invisible cases (hereinafter referred to as "Unseen Case"), the model will be trained using the default data, and the unseen topological data will be added to the test set.

The topology and electrical components to be removed in the 14-bus network case study are shown in Fig. 5.2. The case study will discuss the removal of generators connected to buses 1 and 2, and links 5 and 7 for different lengths of time. Table 5.3 lists selected case studies of 14-bus network.

Table 5.3: Topological change cases: 14-bus network

Case	Name	Removed elements	Number of occurrences	Timestep	Present in Training
1	def	Default	-	-	-
2.1	gen_1_40	Gen 1	1	40	Yes
2.2	gen_1_10_31	Gen 1	31	10	Yes
2.3	gen_1_40_unseen	Gen 1	1	40	No
3.1	gen_12_3	Gen 1, Gen 2	1	3	Yes
3.2	gen_12_3_95	Gen 1, Gen 2	95	3	Yes
3.3	gen_12_3_unseen	Gen 1, Gen 2	1	3	No
4.1	rl_57_3	Link 5, Link 7	1	3	Yes
4.2	rl_57_3_95	Link 5, Link 7	95	3	Yes
4.3	rl_57_3_unseen	Link 5, Link 7	1	3	No
5.1	rl_57_30	Link 5, Link 7	1	30	Yes
5.2	rl_57_10_31	Link 5, Link 7	31	10	Yes
5.3	rl_57_30_unseen	Link 5, Link 7	1	30	No

The case naming follows the format: "`gen/link_<num>_<duration>_<frequency>_unseen`", where:

- `gen/link`: the type of the element (generator or transmission link).
- `<num>`: number of affected elements.

- `<duration>`: number of timesteps.
- `<frequency>` (optional): number of occurrences; omitted if 1.
- `unseen` (optional): is added for event data not included during training.

To further verify the graph structure-related model prediction performance phenomenon observed in the 14-bus network, a 57-bus network was selected as a larger network for testing. Since the generator offline has no direct relationship with the graph structure change, the generator test case tests the model's ability to cope with extreme feature mutations rather than graph structure changes. Therefore, the 57-bus network focuses on the changes in the graph structure and tests different link removal scenarios. The topology and the electrical components to be removed are shown in Fig. 5.3 and Table 5.4 lists selected case studies of the 57-bus network.

Table 5.4: Topological change cases: 57-bus network

Case	Name	Removed elements	Number of occurrences	Timestep	Present in Training
6	def_57	Default	-	-	-
7.1	rl_111316_3	Link 11, Link 13, Link 16	1	3	Yes
7.2	rl_111316_3_95	Link 11, Link 13, Link 16	95	3	Yes
7.3	rl_111316_30	Link 11, Link 13, Link 16	1	30	Yes
7.4	rl_111316_3_unseen	Link 11, Link 13, Link 16	1	3	No
7.5	rl_111316_30_unseen	Link 11, Link 13, Link 16	1	30	No
8.1	rl_1640_3_unseen	Link 16, Link 40	1	3	No
8.2	rl_1640_30_unseen	Link 16, Link 40	1	30	No
9.1	rl_40_3_unseen	Link 40	1	3	No
9.2	rl_40_30_unseen	Link 40	1	30	No
10.1	rl_51422_30_unseen	Link 5, Link 14, Link 22	1	30	No
10.2	rl_6824_30_unseen	Link 6, Link 8, Link 24	1	30	No
10.3	rl_2918_30_unseen	Link 2, Link 9, Link 18	1	30	No

In the result discussion, "whether it is the time when topological change occurs" and "whether it is the area where topological change occurs" will be discussed separately in some sections. "The time when topological change occurs" is referred to as "tp time" hereinafter. Consider the location of the topological event points in the network: “direct

neighbors" represent nodes directly related to the link removal, and "hop=1" represents the indirect neighbor that is one node away from the link removal.

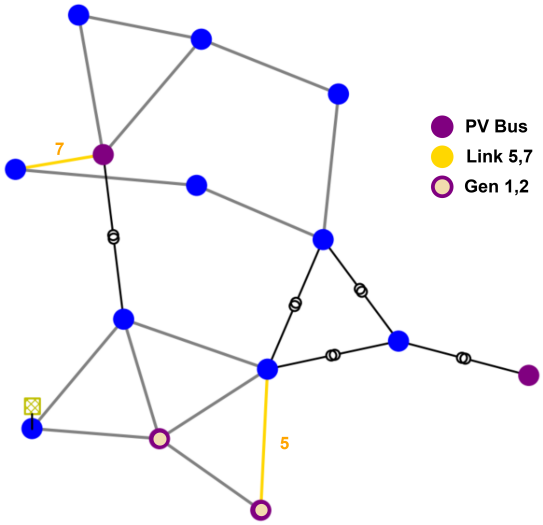


Figure 5.2: IEEE 14-bus network topology

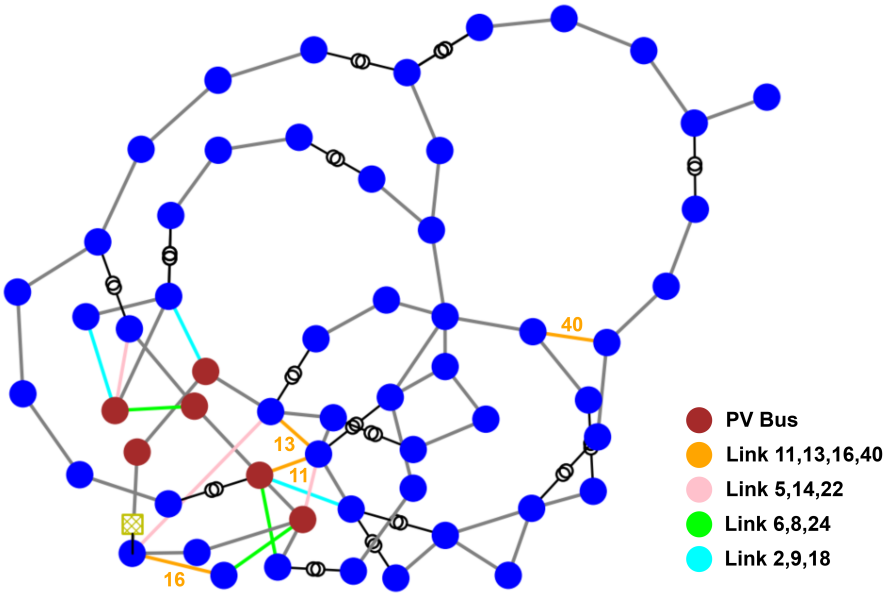


Figure 5.3: IEEE 57-bus network topology

5.4. PERFORMANCE COMPARISON OF GNN MODELS IN DEFAULT CASES

In the default cases, no electrical components are removed, aiming to establish the baseline performance of the three GNNs and compare their performance with the WLS method. Table 5.5 summarizes the performance of all methods in the two networks, $WLS*$ means that the voltage of the slack bus is also set to be fixed during the iteration process. The individual performance of each bus in both networks is shown in Figs. 5.4 and 5.5. The WLS in both figures is the case of "with noise, only fixed slack bus phase angle".

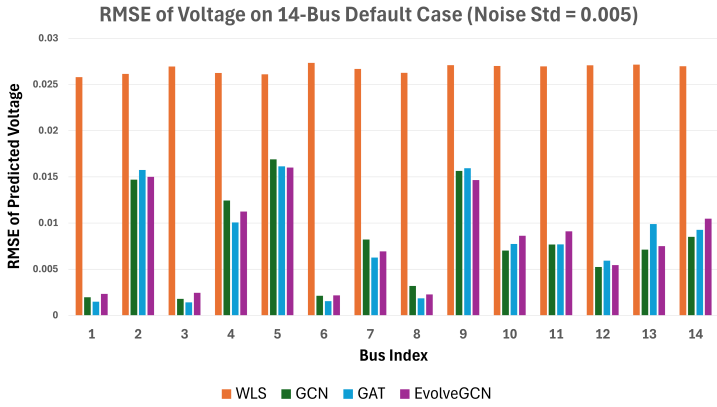


Figure 5.4: RMSE of Voltage prediction on 14-Bus Default case, Noise Std = 0.005 (def)

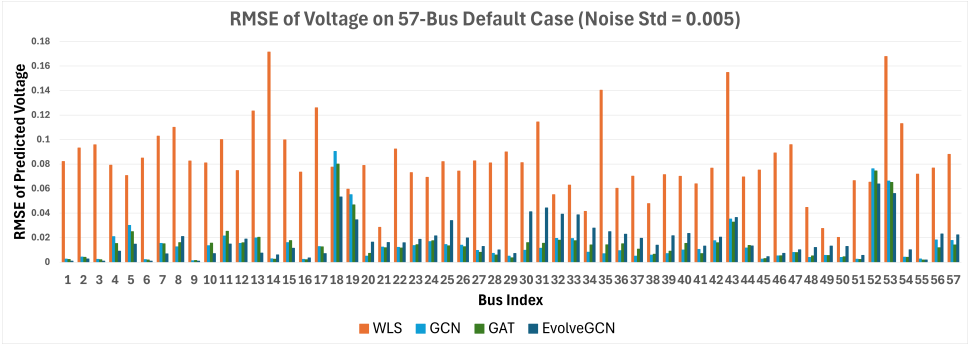


Figure 5.5: RMSE of Voltage prediction on 57-Bus Default case, noise std = 0.005 (def)

Performance differences between different buses

First, from the performance of different buses of a single model, it is found that: three GNNs have different prediction performances for different buses, while WLS has similar

prediction performances for each bus. For GNNs, in the 14-bus network model, the performance of different buses is different, which mainly depends on whether the measurement data is visible. For example, buses 1, 2, 6, and 8 are slack buses and PV buses, and are considered as measurement visible data (or pseudo-measurement available buses) during training, and their performance is more stable. However, this is not the same on all buses. For example, the measurement data for bus 14 is also given for spatial considerations, but its performance is not particularly stable. The predicted performance of the bus depends not only on whether measurement data is given, but also on the variation range of the target value itself, the number of neighbors, whether its value change is related to the neighbors, etc. This is even more obvious in the results of the 57-bus network: whether or not measurement data is given has no absolute relationship with the prediction performance.

Table 5.5: Voltage RMSE comparison: default case (def)

	Average RMSE	clear		with noise				
		WLS	WLS*	WLS	WLS*	GCN	GAT	EGCN
14-Bus	with_meas	0.0002	0.0001	0.0267	0.0088	0.0080	0.0079	0.0082
	no_meas	-	-	-	-	0.0114	0.0116	0.0113
57-Bus	with_meas	0.1129	0.003	0.0828	0.004	0.0149	0.0149	0.0178
	no_meas	-	-	-	-	0.0223	0.0221	0.0219

Performance Comparison Across Methods and Networks

Next, when looking at the overall numerical performance, it is found that: first, the performance of the WLS method in 14-bus network is strongly affected by noise: their overall performance drops from RMSE = 0.0002 to 0.0267 when noise is present in the measurements. However, in actual measurements, noise is inevitable. Secondly, it is found that the convergence rate of WLS of 57-bus network decreases: the convergence rate of $WLS*$ is 91.7% when there is noise and 95.8% when there is no noise, and additional fixing of the voltage of the slack bus is required to maintain a better performance. Such results indicate that in larger networks, the prediction performance of WLS is not very stable and is prone to non-convergence or estimation deviation, so the voltage of the slack bus is also preferably fixed.

Third, under the default conditions, GAT and GCN outperform EvolveGCN overall, but for buses without measurement, EvolveGCN performs slightly better. This shows that GAT and GCN, which process graphs independently and aim to capture the exact shape of value changes, can perform slightly better than EvolveGCN when there are no topological changes in the network operations (i.e., events with temporal effects), but the performance varies greatly between different buses.

5.5. GAT'S SUPERIOR PERFORMANCE IN SEEN TOPOLOGIES

The overall result of seen cases in both networks are shown in Table 5.6. The colors in the table represent the performance of each model in the same case and the same time

step. In the same case, the performance of "all timesteps" and "tp time" are compared respectively. Among the three models, blue performs the best, followed by light orange, and pink is the worst. For example, in Case 2.1, GAT performs best (0.0093), which is blue, followed by EvolveGCN (0.0098), which is light orange, and the worst performance is GCN (0.0103), which is pink.

The results show that GAT outperforms EvolveGCN and GCN overall in almost all cases, but the difference in RMSE is not significant. Regarding the comparison between Evolve GCN and GCN, in the 14-bus network (Case 2.1 to 5.2), EvolveGCN performs better than GCN in the link removal cases, but performs poorly for high-frequency events; in the generator offline cases, for short-term events, EvolveGCN's overall performance is better than GCN, but worse than GCN in events with longer duration.

In the 57-bus network (Case 7.1 to 7.3), GCN performs better than EvolveGCN. This is because the topological changes occur regionally, so most of the bus voltages are maintained at the default state, and GAT and GCN perform better under the default conditions. This shows that when dealing with state estimation of large networks, even with temporal effects such as topological changes, considering the importance of locality and graph structure, the overall performance of the model that handles individual graph structure is better. In addition, noise overfitting and scale imbalance are found near the PV bus, among which GCN has the most serious fluctuation, followed by GAT, and EvolveGCN has the smallest fluctuation, more details are shown in Section 5.6.

Table 5.6: Average RMSE of Voltage prediction on seen cases

Case	Name	All timesteps			tp time		
		GCN	GAT	EGCN	GCN	GAT	EGCN
2.1	gen_1_40	0.0103	0.0093	0.0098	0.0089	0.0084	0.0085
2.2	gen_1_10_31	0.0103	0.0095	0.0099	0.0110	0.010	0.0102
3.1	gen_12_3	0.0103	0.0090	0.0108	0.0150	0.0085	0.0158
3.2	gen_12_3_95	0.0193	0.0188	0.0270	0.0227	0.0298	0.0272
4.1	rl_57_3	0.0096	0.0089	0.0094	0.0093	0.0090	0.0107
4.2	rl_57_3_95	0.0111	0.0098	0.0113	0.0134	0.0111	0.0118
5.1	rl_57_30	0.0110	0.0088	0.0091	0.0108	0.0084	0.0096
5.2	rl_57_10_31	0.0123	0.0103	0.0108	0.0133	0.0109	0.0116
7.1	rl_111316_3	0.0144	0.0145	0.0175	0.0183	0.0182	0.0222
7.2	rl_111316_3_95	0.0150	0.0142	0.0187	0.0152	0.0143	0.0187
7.3	rl_111316_30	0.0153	0.0145	0.0170	0.0189	0.0185	0.0220

5.6. GCN'S SUPERIORITY IN UNSEEN CASES

Table 5.7 shows the performance of the overall unseen case. It can be found that the overall performance of EvolveGCN is worse than the other two models, but its performance in the tp time of 14-bus networks is better, and the overall numerical difference is not large. In the 57-bus network, the performance of GAT and GCN is significantly better

than EvolveGCN, and the overall performance of GCN is slightly better than GAT, which will be explained in detail below.

GCN's noise overfitting is reduced

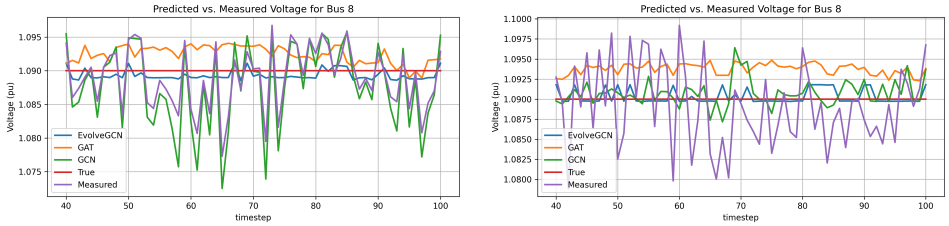
First, GCN was found to have obvious noise overfitting and scale imbalance in the seen cases: during the learning process, because the dynamics of the PV bus are different from those of its neighbors, it will start to learn noise near the PV bus when it cannot distinguish the edge weights, resulting in large fluctuations in voltage prediction, as shown in Fig. 5.6(a), where bus 8 is a PV bus in 14-bus network. However, the situation improves in the unseen case. From the comparison of Fig. 5.6, it is found that GCN is prone to learning noise from measurement during the training process of seen cases, but for unseen cases, the model learns a more basic voltage distribution law by the default data, making the noise overfitting of bus 8 smaller in the unseen cases. Besides, in this case, it can be found that EvolveGCN performs more stably than the other two for PV bus prediction that lacks spatial correlation.

Table 5.7: Average RMSE of Voltage prediction on unseen cases

Case	Name	All timesteps			tp time		
		GCN	GAT	EGCN	GCN	GAT	EGCN
2.3	gen_1_40_unseen	0.0089	0.0122	0.0100	0.0071	0.0134	0.0098
3.3	gen_12_3_unseen	0.0095	0.0098	0.0100	0.0145	0.0197	0.0181
4.3	rl_57_3_unseen	0.0091	0.0088	0.0092	0.0097	0.0092	0.0092
5.3	rl_57_30_unseen	0.0089	0.0087	0.0091	0.0084	0.0088	0.0084
7.4	rl_111316_3_unseen	0.0147	0.0147	0.0178	0.0200	0.0207	0.0259
7.5	rl_111316_30_unseen	0.0146	0.0148	0.0176	0.0180	0.0181	0.0231
8.1	rl_1640_3_unseen	0.0152	0.0149	0.0177	0.0217	0.0214	0.0256
8.2	rl_1640_30_unseen	0.0156	0.0148	0.0173	0.0197	0.0193	0.0223
9.1	rl_40_3_unseen	0.0142	0.0152	0.0178	0.0198	0.0222	0.0261
9.2	rl_40_30_unseen	0.0145	0.0152	0.0173	0.0184	0.0196	0.0225
10.1	rl_51422_30_unseen	0.0152	0.0151	0.0158	0.0211	0.0187	0.0186
10.2	rl_6824_30_unseen	0.0138	0.0150	0.0166	0.0168	0.0191	0.0213
10.3	rl_2918_30_unseen	0.0153	0.0155	0.0162	0.0207	0.0199	0.0202

GCN shows better generalization ability than GAT

In addition, GCN is found to have overall better generalization ability in unseen cases. Fig. 5.7 compares the performance of the seen and unseen cases in the 14-bus network. First, it can be seen that whether it is link removal or generator offline, the overall performance of GCN is better in the unseen case than in the seen case. In the link removal case (Figs. 5.7(c) and 5.7(d)), the overall performance of the three models in unseen cases is better than that in seen cases, indicating that the model trained using default cases has advantages in generalization ability and may be able to handle unseen topology change scenarios more stably. Since some topological changes occur only in a certain area, their impact on all buses is limited. Therefore, a better generalization ability helps

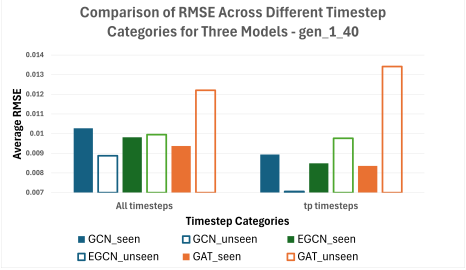


(a) Case 3.1, Seen

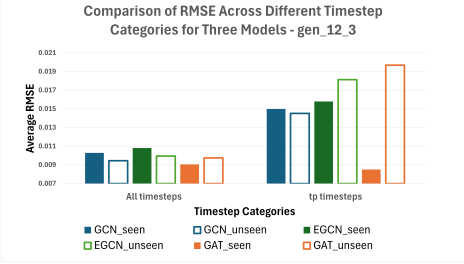
(b) Case 3.3, Unseen

Figure 5.6: Comparison of Bus 8 voltage prediction in Case 3.1 and 3.3: Zoom in to 40 - 100 time steps

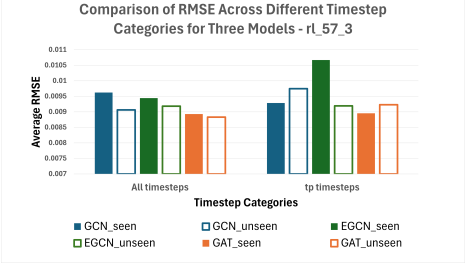
the model maintain stable performance on other buses. Second, when the generator is offline (Figs. 5.7(a) and 5.7(b)), the performance of GCN is significantly improved in the unseen case, while the opposite is true for GAT. Regardless of the length of time, the performance of GAT in the unseen case is significantly worse than that in the seen case in terms of both overall time and tp time.



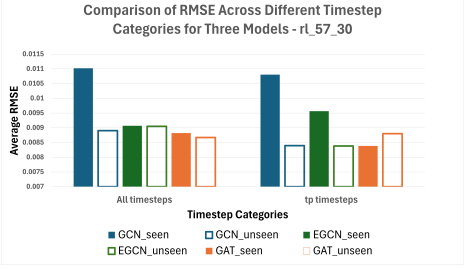
(a) Case 2.1 and Case 2.3 (gen_1_40)



(b) Case 3.1 and Case 3.3 (gen_12_3)



(c) Case 4.1 and Case 4.3 (rl_57_3)



(d) Case 5.1 and Case 5.3 (rl_57_30)

Figure 5.7: Comparison of model performance with seen and unseen cases

GCN performs better than GAT for buses close to the topological change event

Fig. 5.8 shows the voltage prediction performance of GCN and GAT for direct neighbors of topological changes in unseen cases in the 57-bus network. It is found that although the numerical performance of GCN and GAT shows no obvious difference in Table 5.7,

GCN performs better for buses close to the event, namely having more impact from the topological change. Considering the unseen case is to test model's generalization to adapt to unseen change, GCN shows better performance when not only the temporal but the spatial effect are taken into consideration.

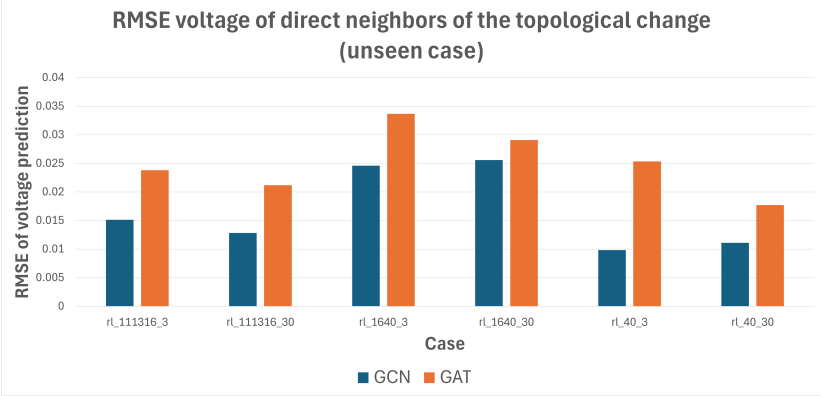


Figure 5.8: RMSE of Voltage prediction near the topological change on unseen cases: GCN vs. GAT

5.7. IMPACT OF TOPOLOGICAL CHANGE FREQUENCY ON PERFORMANCE

Table 5.8: Average RMSE under topological change time step (buses directly related to the link removal)

rl_57_3	**GCN**	**GAT**	**EGCN**	**rl_57_3_95**	**GCN**	**GAT**	**EGCN**
Bus 3	0.0007	0.0015	0.0013	Bus 3	0.0044	0.0023	0.0013
Bus 4	0.0175	0.0150	**0.0098**	Bus 4	0.0297	0.0136	**0.0077**
Bus 6	0.0021	0.0008	0.0010	Bus 6	0.0039	0.0029	0.0015
Bus 11	0.0074	0.0017	0.0041	Bus 11	0.0163	**0.0227**	**0.0265**
avg RMSE	0.0069	0.0047	0.0040	avg RMSE	0.0136	0.0104	0.0093
rl_57_30	**GCN**	**GAT**	**EGCN**	**rl_57_10_31**	**GCN**	**GAT**	**EGCN**
Bus 3	0.0042	0.0019	0.0009	Bus 3	0.0044	0.0019	0.0003
Bus 4	0.0228	0.0082	**0.0047**	Bus 4	0.0281	0.0168	**0.0089**
Bus 6	0.0030	0.0018	0.0014	Bus 6	0.0040	0.0022	0.0004
Bus 11	0.0106	0.0076	0.0092	Bus 11	0.0143	**0.0188**	**0.0263**
avg RMSE	0.0101	0.0049	0.0041	avg RMSE	0.0127	0.0099	0.0090

Regarding the impact of increasing event frequency, different findings were found in the two networks. First, in the 14-bus network, as shown in Table 5.6, increasing the frequency of the same event does not improve the prediction performance. On the contrary, there is a slight decrease in performance overall: the increase in RMSE is less than 0.005, and the increase in RMSE within tp time ranges from 0.0002 to 0.0037. Table 5.8

lists the prediction performance of buses directly related to the link removal in cases in 14-bus network. When looking at the RMSE values of bus 11, it is found that at low frequencies, the performance of the three models is similar, and GCN has the worst prediction performance. However, when the frequency increases, the performance of GAT and EvolveGCN is lower than that of GCN. At higher frequencies, both models predict the actual value of the voltage drop less accurately. Since the prediction performance of GCN on some other buses is still worse than that of GAT and EvolveGCN, its overall performance is still the lowest. However, its bus directly connected to the activity behaves more stable under the influence of frequency than GAT and EvolveGCN.

However, the same phenomenon did not occur in the 57-bus network. When high-frequency topological changes occur (Case 7.1 vs. Case 7.2), the prediction of a specific node is not found to become unstable. On the contrary, the performance in predicting tp time improves, which is not observed in the 14-node network. This suggests that larger networks are more stable to changes in information flows: Removing a link does not cause large-scale value changes and has a higher prediction stability of wording efficacy of GAT and EvolveGCN.

5.8. STABLE PERFORMANCE OF EVOLVEGCN NEAR PV BUSES

Initial Observation: Local Effects Near Topological Events

Although EvolveGCN performs worse than GAT and GCN in overall seen cases and unseen cases, it performs significantly better than GCN and GAT in some specific nodes. First, in the link removal case of the 14-bus network, it was found that regardless of the frequency of the event, EvolveGCN's prediction performance for bus 4 was significantly higher than that of GAT and GCN, as shown in Table 5.8, and bus 4 was closer to the PV bus and the topological change point. The same was observed when the generator was offline.

In the case studies of the 57-bus network, it was found that EvolveGCN seemed to consistently outperform GAT and GCN near event points. Figs. 5.9 and 5.10 show the RMSE results for bus 17 in Cases 7.2 and 7.3. The results show that even when the graph structure is explicitly provided in the input as the edge index, GCN and GAT cannot accurately capture the voltage drop for each topology change in the bus directly connected to the event point.

Testing Hypotheses about Local Effects

Since these topological change points are close to the PV Bus locations, and considering the stable performance of EvolveGCN in PV bus prediction, Cases 8.1 and 9.1 are set to explore whether topological changes or large voltage variations affect EvolveGCN's prediction performance for neighboring nodes by removing links at different locations from the cases. The main difference between these three cases (Cases 7.1 to 9.1) lies in the distance between the PV bus and the incident site. Links 11, 13, and 16 are close to one of the PV bus, while link 40 is farther from the PV bus, as shown in Fig. 5.3.

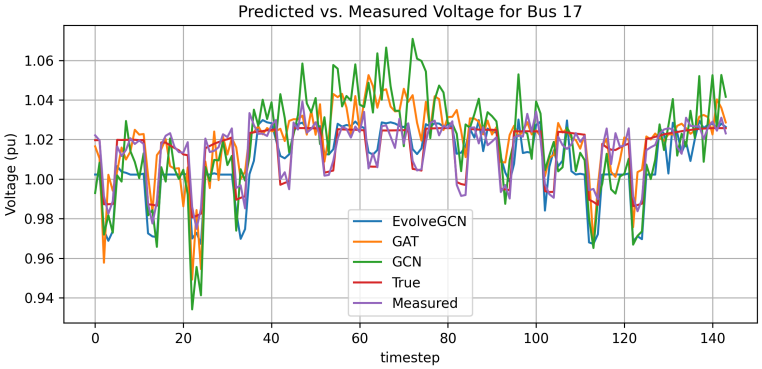


Figure 5.9: Comparison of Bus 17 voltage prediction in Case 7.2 (rl_111316_3_95)

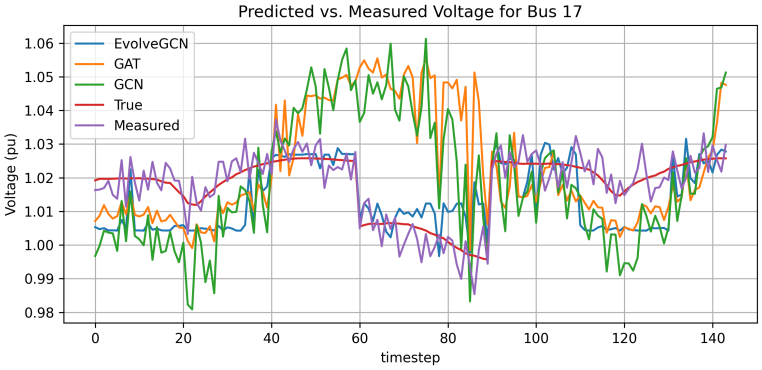


Figure 5.10: Comparison of Bus 17 voltage prediction in Case 7.3 (rl_111316_30)

Table 5.9: The RMSE of the unseen cases: Case 7 to Case 9

Case	Name	direct neighbors			hop = 1		
		GCN	GAT	EGCN	GCN	GAT	EGCN
7.4	rl_111316_3_unseen	0.0152	0.0238	0.0078	0.0101	0.0140	0.0091
7.5	rl_111316_30_unseen	0.0128	0.0212	0.0086	0.0088	0.0126	0.0093
8.1	rl_1640_3_unseen	0.0246	0.0337	0.0272	0.0164	0.0276	0.0300
8.2	rl_1640_30_unseen	0.0256	0.0291	0.0243	0.0170	0.0221	0.0257
9.1	rl_40_3_unseen	0.0099	0.0254	0.0379	0.0099	0.0249	0.0386
9.2	rl_40_30_unseen	0.0111	0.0177	0.0302	0.0108	0.0176	0.0309

Table 5.9 shows the event neighbor RMSE prediction results of Cases 8.1 and 9.1. It is found that as the event location becomes further away from the PV bus, the performance of EvolveGCN at the direct neighbors and hop=1 becomes worse, indicating that

EvolveGCN does not perform better near the event location, but performs better near the PV buses.

Verifying the Role of PV Bus Proximity

To further validate: "EvolveGCN performs better on the bus close to the PV bus, regardless of the location of the event." Buses as neighbors of PV buses are taken and observed its performance separately. The results are shown in Table 5.10. It can be seen that EvolveGCN has a clear advantage in estimating the bus voltage near the PV bus, while GCN, although it has the best generalization ability, performs the worst in this case. To sum up, Case 7.1 to Case 7.5 have shown that when a topological change event occurs near the PV bus, EvolveGCN can better predict the neighbors of the PV bus, while Case 6, Case 8.1 to Case 9.2 reveal that even if there is no topological change near the EvolveGCN, EvolveGCN still performs well in predicting the neighbors of the PV bus. Next, Case 10.1 to Case 10.3 will try to make more disturbances around the PV bus to confirm that Case 7.1 to Case 7.5 are not independent cases and that EvolveGCN does have a more stable performance near the PV buses.

Case 10.1 to Case 10.3 select three links directly connected to the PV bus for unseen case testing. The average RMSE results of voltage prediction are shown in Table 5.11. "Neighbors" means only the neighbors of PV buses are considered, "PV buses + Neighbors" means both the RMSE value of PV buses and Neighbors are considered. The buses selected in Figs. 5.11 to 5.13 are neighbors of PV buses. It can be seen that EvolveGCN indeed performs the most stably near PV buses in every case, and its numerical performance is better than GAT and GCN.

Table 5.10: The RMSE of all 57-bus network cases: PV buses' neighbors

Case	GCN	GAT	EGCN
6	0.0168	0.0164	0.0098
7.1	0.0245	0.0242	0.0113
7.2	0.0173	0.0148	0.012
7.3	0.0247	0.0238	0.0134
7.4	0.0248	0.0284	0.0105
7.5	0.0206	0.0243	0.0111
8.1	0.0341	0.0306	0.0105
8.2	0.0300	0.0303	0.0113
9.1	0.0341	0.0296	0.0097
9.2	0.0288	0.0285	0.0101

Table 5.11: The RMSE of Case 10.1 to Case 10.3: PV buses' neighbors

rl_51422_30	**GCN**	**GAT**	**EGCN**
Neighbors	0.0357	0.0331	0.0181
PV Buses + Neighbors	0.0227	0.0208	0.0125
rl_6824_30	**GCN**	**GAT**	**EGCN**
Neighbors	0.0270	0.0269	0.0136
PV Buses + Neighbors	0.0173	0.0175	0.0094
rl_2918_30	**GCN**	**GAT**	**EGCN**
Neighbors	0.0456	0.0388	0.0179
PV Buses + Neighbors	0.0282	0.0239	0.0121

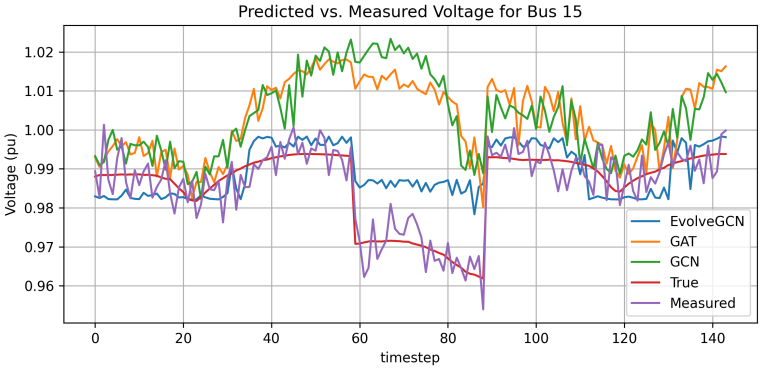


Figure 5.11: Comparison of Bus 15 voltage prediction in Case 10.1 (rl_51422_30)

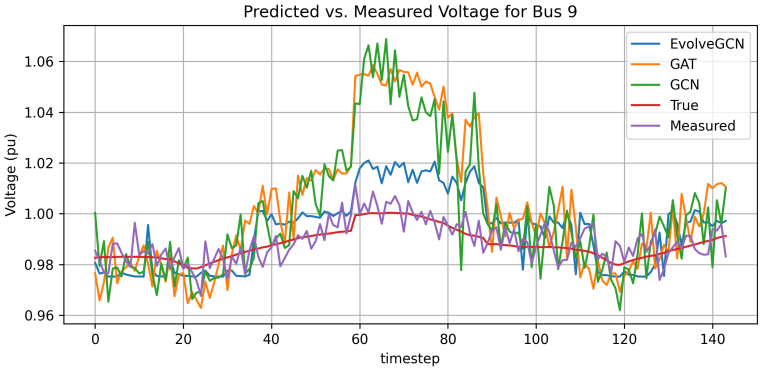


Figure 5.12: Comparison of Bus 9 voltage prediction in Case 10.2 (rl_16824_30)

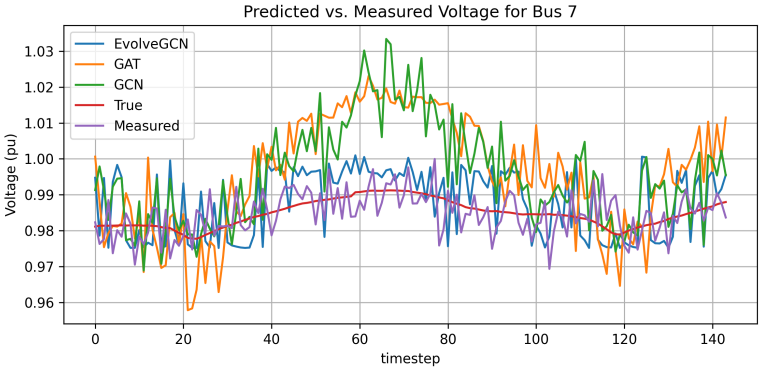


Figure 5.13: Comparison of Bus 7 voltage prediction in Case 10.3 (rl_2918_30)

5.9. SUMMARY OF THE RESULT

In the default case, it is found that the prediction performance of WLS is easily affected by the noise level, and more complex networks need to provide more slack bus information to ensure stable prediction performance. The performances of the other three models are similar in the default case.

In the seen cases of both networks, GAT has the best overall performance under both link removal and generator offline conditions. The overall performance of EvolveGCN is comparable to that of GCN in a 14-bus network, but is the worst in a 57-bus network. This may be because the topological change event only affects a part of the entire network, i.e., most of the graph still behaves like the default case, and EvolveGCN exhibits lower performance in the default case due to its inability to capture the values in the stable graph. However, the study also found that although GAT has good overall performance, it seems unable to capture the precise topological changes near the event location (see Figs. 5.9 and 5.10).

In the unseen case, GCN has the best overall performance and the most stable prediction results. The GAT in the generator offline case is obviously inferior to the seen case. On the other hand, the generalization effect brought by the default case-training model makes the prediction of the PV bus in unseen cases more stable and can effectively alleviate the noise overfitting phenomenon in seen cases. In the 57-bus network, GCN has better performance than GAT in direct neighbors of the topological change event, showing its better generalization ability when it comes to unseen change.

Experiments on the 14-bus network show that EvolveGCN is relatively less affected by noise or topological leakage when processing events adjacent to the PV bus (especially events related to graph structure changes), and its prediction performance for buses directly related to the graph structure or PV bus is generally better than the other two models. In the 57-bus network, link removal cases with different distances between event locations and the PV bus are tested as unseen cases, and it is found that EvolveGCN performs better only near the PV bus, but performs worse near the event location, indicating that its performance is independent of the event location. In addition, the 14-bus network cannot fully release this phenomenon due to the high density of PV nodes, which improves the overall performance of EvolveGCN to a certain extent. In summary, EvolveGCN is shown to have a clear advantage in predicting the bus voltage close to the photovoltaic bus, while GCN performs the worst in this case.

6

Discussion

This chapter aims to provide the discussion of the results, bring the experimental results back to the reality level for discussion, and provide answers to research questions, and future research directions. First, Section 6.1 summarizes the limitations and experiments of the two networks and interprets their training processes and phenomena. Afterwards, Section 6.2 answers research questions based on the result and observations of comparisons of the model's internal structure. Next, real-world examples where each model might be useful will be discussed in Section 6.3. Finally, the future research directions will be explained in Section 6.4.

6.1. Result Discussion

6.1.1. Training Insights and Hyperparameter Tuning

Model-based and data-driven models are not completely comparable

First, in the default case, this study compares the WLS with GCN, GAT, and EvolveGCN. However, since WLS is a model-based method, its operating mechanism and data input method are different from those of deep learning models. Therefore, it is difficult to make a completely fair comparison between the two. For example, the performance of WLS is better with flat start than with masked data, and the results of "fixing only the phase angle of the slack node" and "fixing both the phase angle and voltage of the slack node" are very different in a 57-bus network. The other three GNN models do not fix the state of any nodes. But what is certain is that under the influence of certain noise, if only the slack bus phase angle is fixed, the performance of the three models is better than that of WLS.

Limitation of data granularity and temporal simulation

Second, as stated in Section 4.1.2, the granularity of the data obtained from ENTSO-E is 15 minutes, which is not aligned with the actual timescale of topological change events in DER-penetration system. Events such as the short circuits and inverter protection triggers usually occur and are resolved within seconds or minutes. Moreover, the state estimation in real-world scenarios is used to detect the system restoration after the events occur, typically within few timesteps. To capture this, the data simulation requires continuous temporal effect storage and interaction between time steps.

However, in the data simulation method applied in this research, each snapshot is simulated in pandapower independently, and only the final power flow results are retained, without tracking dynamic transition. leading to not accurate enough temporal effect being recorded. As a result, neither fine-grained temporal effects nor the evolution of the system state over time can be accurately represented.

The usefulness of the case studies in this research lies mainly in providing a numerical stress test for these three GNNs, testing whether those models can learn the subtle change in the training set or generalize to adapt to unseen topological change. If higher-resolution data is available or the simulation allows for the inter-snapshot temporal effect to be recorded, the result can better align with the actual situation.

Limitations regarding measurement representativeness

Third, this study has limitations in dealing with measurement uncertainty and availability. The experiment simulated some nodes where pseudo-measurements are completely unavailable (for example, no historical data is available, they are easily affected by weather, they are too young to be installed, etc.). The measurements of these nodes are invisible during the training input, loss function calculation, and testing phases.

However, in the experiment, different nodes were not assigned different noise levels, that is, their trustworthiness was considered the same. This treatment ignores the different reliabilities of different pseudomeasurements, which in reality can vary greatly depending on local conditions and historical accuracy. Furthermore, this study did not specify the location of actual measurements, assuming that all measurements were pseudomeasurements. The reason why the impact of specific PMU placement or measurement coverage on state estimation performance is not explored is because: determining the optimal measurement nodes is a complex topic in itself. Therefore, in order to focus on studying the impact of topological changes on state estimation, these considerations were omitted in the experimental design.

Hyperparameter tuning cannot guarantee the optimal data driven models

Although this study attempts to compare the performance of EvolveGCN, GCN, and GAT models, there are still several challenges in actual comparison. First of all, the accuracy and comprehensiveness of hyperparameter adjustment affects the performance of the model. Although this study adjusts parameters and selects the best combination for training, it still cannot guarantee that the result is absolutely optimal. In particular, EvolveGCN is more sensitive to changes in batch size than GAT and GCN, which may

be related to the fact that this study modified its original architecture. Although a better batch size has been selected in the default case and used for experiments in other cases, the performance may still be improved if it is readjusted in other cases. In addition, EvolveGCN is a model that contains recurrent layers. If a deeper network is used on a large graph with a large number of nodes, overfitting problems may occur easily, and the number of layers, dropout rate, or regularization parameters need to be further adjusted.

EvolveGCN converges too quickly, GAT and GCN are prone to overfitting

Next, this study also observed differences in the convergence behaviors of the three models during training. The convergence speed of EvolveGCN is significantly faster than that of GCN and GAT. In most cases, the latter two need to be trained for more than 100 epochs to reach the early stopping condition, and in some cases the model shows the phenomenon of fitting to noise, which may be related to the static graph training. This study has added a regularization term to the loss function, but there has been no significant change. Therefore, if this type of model is employed in the future, it may be beneficial to refine the regularization terms, or set a stop loss point for training — storing and retrieving data after a fixed number of epochs to observe overfitting — to prevent overfitting of noise. As for EvolveGCN, since it focuses on the changes in graph structure in time series, if the graph structure does not change much or the numerical changes are small, the model will converge quickly, but it may stop early due to the inability to fully learn the structural details related to numerical difference of the graph during training, resulting in a decrease in accuracy. In this case, the design of EvolveGCN-H may provide more stable performance when facing more complex node information in the graph.

Weight adjustment of different nodes

Last, the common challenge faced by the three GNN models is to set appropriate item weights in the loss function and node weights in one graph. Since the graph distance in the power grid is not consistent with the actual electrical distance, and a single event has different impacts on each node, it is difficult for the model to handle all nodes with a unified standard. This study chose to adjust the weight parameters in the preset case and keep them unchanged in other cases. However, if the regional weights can be adjusted based on the location of the event or the affected area, the model may be able to focus more on “important areas” and improve the accuracy of estimates.

6.1.2. OVERALL PERFORMANCE

14-bus network

In the 14-bus network, the overall performance of the three models is actually not much different, but GAT’s performance is slightly better than the other two models. First, GCN has obvious noise overfitting in PV bus prediction, followed by GAT and EvolveGCN. This is related to the training method of the single graph structure mentioned in Section 6.1.1. GCN’s message passing cannot adjust the weights of different edges. Therefore, especially near the PV bus, in nodes with large numerical differences between neighbors,

scale imbalance will occur due to topology leakage. EvolveGCN performs best in this situation where the differences between neighbors are large because it can see the explicit weights of the previous time step - as the hidden state in the recurrent layer- and the internal parameters of the RNN that perform gradient updates based on historical information.

Secondly, the prediction stability of GAT and EvolveGCN decrease under high-frequency event cases. Although the overall value is not deteriorate significantly, there are prediction deviations at specific nodes. On the contrary, GCN has the best stability in high-frequency events, which may be related to the effect that GAT and EvolveGCN amplify structural information or temporal information: their strong response to high-frequency events may cause the model to pay too much attention to certain nodes during the learning process, or to pay too much attention to certain historical information when updating weights. GCN uses a simpler message passing mechanism and does not pay special attention to specific nodes, which may allow it to stably predict overall performance when high-frequency changes cause numerical instability. In addition, it is also found that EvolveGCN performs particularly well on nodes close to event points. A full comparison is given in Section 6.1.3.

57-bus network

In the 57-bus network, the performance differences between models are more obvious, and the overall performance of GAT and GCN is better than EvolveGCN. This suggests two possibilities: first, EvolveGCN has limited prediction capabilities when dealing with larger graphs; second, the impact of topological events is usually regional, so nodes far away from the event point behave similarly to the default case. In this case, the prediction performance of EvolveGCN will decline due to its poor grasp of the overall graph in the stable state, which shows that its ability to capture numerical changes in stable graph structures is worse than that of GAT and GCN.

Regarding high-frequency events, GAT and EvolveGCN do not become unstable as the 14-bus network cases when the frequency becomes higher. This can be attributed to the fact that there are more link connections in large graphs. The removal of a small number of links does not significantly change the message transmission process and cause excessive impact, thereby affecting the stability of GAT and EvolveGCN learning. Instead, it allows the model to adapt to high-frequency changes and obtain more topological structure information in the training set for learning. Therefore, when the numerical fluctuations caused by topological changes are not large enough to severely disrupt the information flow across the entire graph, and further affect the models' prediction stability, having more topological information can indeed effectively improve the prediction performance.

EvolveGCN is still found to perform particularly well on nodes close to event points, but since these events including the 14-bus event are all close to the PV bus, considering that EvolveGCN itself has been found to have better prediction stability near the PV bus, whether this advantage comes from topological events will be explained in Section 6.1.3.

6.1.3. SEEN VS. UNSEEN CASE PERFORMANCE

The generalization advantage of GCN and the limitation of GAT model training

Although the overall RMSE value of voltage prediction of GAT is similar to that of GCN, its prediction performance degrades in unseen scenarios, especially for neighboring nodes with topology changes and generators offline. Generator offline is not a topic of graph structure change, but a feature value that predicts extreme changes. The experiment also shows that in the unseen case, GAT's performance in the generator offline mode drops the most, indicating that it performs poorly for unseen events and fuzzy topological structure changes (which cannot appear explicitly in the edge index). On the other hand, GCN performs better in the unseen case than in the seen case, and the noise overfitting problem caused by the single image structure is also reduced.

This trend is consistent on both 14- and 57- bus networks. One possible explanation is that GAT is a model that relies on a more complex procedure for information transmission, so it cannot learn how to change attention scores without relevant historical data. When the voltage oscillation caused by topological changes is large, the model that requires "attention score" as a premise may not be able to grasp the exact direction of information transmission, resulting in unstable predictions. In contrast, GCN has good generalization ability and is equally effective in conveying global information. Similar to the leakage of the PV bus topology that is considered a disadvantage in prediction of buses near PV buses, the infectiousness of the topology in GCN training allows the characteristic changes to be transmitted to multiple nodes with high sensitivity.

EvolveGCN has high prediction stability near the PV bus

More examples of topological changes in different areas of the power grid are experimented in unseen to explore the regional effectiveness of EvolveGCN. In the neighboring node prediction performance of EvolveGCN, the unseen cases are better than the seen cases, but its overall performance tends to be regional: the nodes close to the PV bus perform significantly better, while the farther nodes perform worse than the other two models, showing their performance differences. A phenomenon is observed here: regardless of seen or unseen cases, EvolveGCN's prediction ability for the adjacent nodes of the PV bus not only performs better in different situations, but also has the smallest topological leakage and imbalance scale among the three models.

Further analysis revealed that the illusion that EvolveGCN performed better near the event point in 14-bus network was because the overall network was too small and almost all links were close to the PV bus. However, in 57-bus network, this phenomenon is clearly distinguished, proving that it is related to the distance from the PV bus but not to the distance from the event point. Therefore, when focusing on the topological events themselves, GAT demonstrates better overall performance in seen cases, whereas GCN exhibits stronger generalization capability in unseen cases.

6.2. ANSWERS TO RESEARCH QUESTIONS

Two overarching questions

Do static GNNs maintain prediction performance under topological changes, particularly when such changes are (not) included during training?

As stated in the literature review, static GNN models may need to be retrained when a topological change occurs. However, the results show that static GNNs still retain some capability to handle data with topological changes, although their performance drops slightly compared to the default case with no topological change. GAT demonstrates strong performance when changes are included during training, indicating that its attention mechanism can learn how to adjust edge weights in response to sudden feature value shifts. On the other hand, GCN shows good generalization ability when changes are not included during training. This suggests that, by universally transmitting messages between nodes based on the edge index (which represents the graph structure), GCN can capture unseen changes while also avoiding overfitting, as there are no drastic feature value shifts during training.

Can temporal GNNs (e.g., EvolveGCN), which model evolving topologies, improve state estimation performance under dynamic structural conditions of the power grid?

The experimental results show that EvolveGCN-O does not have a clear advantage over GCN or GAT in estimating state variables under topological changes. In both seen and unseen cases, EvolveGCN-O does not outperform the static GNNs. However, this does not imply that temporal GNNs are inherently less effective for this task, as there are various training mechanisms within the temporal GNN family. One possible reason for EvolveGCN-O's underperformance is that its node embeddings do not participate in the recurrent layer's learning process. Since state estimation requires more information from feature values compared to tasks like node classification or link prediction, the lack of direct feature integration may lead to poorer numerical prediction. Interestingly, the experiment found that EvolveGCN-O performs better in voltage prediction at PV buses—its results are significantly better than those of the static GNNs. This suggests that for buses with weak spatial correlation, incorporating historical information into training provides an additional useful reference.

Detailed research questions

How do WLS, GAT, GCN, and EvolveGCN compare in terms of the prediction accuracy when no topological changes are present, particularly when noise is introduced in the measurement data (with vs. without noise)?

When the prediction is performed without topology changes, without the introduction of noise, and given fixed slack bus phase angles and voltages, the WLS performs accurately in both networks. This indicates that WLS requires clean data as input for accurate model-based predictions, which is often difficult to obtain in practical situations. Furthermore, in this study setting, if the true values of active power (P) and reactive power (Q) are available, with the help of the complete admittance matrix, the voltage and phase

angle can be directly calculated. Therefore, the predictions from the clean data set are not applicable in this study setting.

If Gaussian noise is introduced into the measurement data, the prediction accuracy of GCN, GAT and EvolveGCN is much higher than that of WLS when only the slack bus phase angle is fixed. As a traditional entity model-based method, WLS is easily affected by starting conditions (such as flat start or masked data) and constraints (such as whether the voltage of the slack bus is fixed), resulting in large differences in accuracy in different situations. Among the three models, GAT has the best overall performance, and EvolveGCN performs the best in predicting nodes without measurements. This shows the robustness of the GNN model in the state estimation task in the presence of noise.

How do GAT, GCN, and EvolveGCN perform in terms of prediction accuracy when faced with different topological changes, different time scales, and different frequency of changes? How do these factors affect the model's ability to adapt and maintain accurate predictions?

This study experimented with link removal and generator offline cases, and found that link removal is directly related to changes in graph structure and can be directly learned during message passing, making the overall prediction of the model better. Generator offline involves extreme feature value changes that are not directly related to the graph structure. Therefore, especially in unseen cases, the prediction performance deteriorates because the model cannot adjust the trained weights.

From the experimental cases, it is found that: in the topological change-visible cases, the length of topological time change does not significantly affect the prediction results of the three models. This can be explained by the way the model is trained. First, both GAT and GCN are single-graph training models. Therefore, in the link removal case, they can directly learn about topological changes through the transmission of edge indices, thereby affecting message passing, without the need for historical information. The impact of long-term and short-term topological changes on the two is the proportion of snapshots with topological change graph structures in a single batch, which can significantly affect gradient updates. EvolveGCN is relatively more dependent on historical information for weight updates, but its gradient updates are also performed after the last batch, so having more topological change data should also help improve performance. In the topological change-invisible cases, it can be found that the long-term events of GAT and EvolveGCN perform slightly better than the short-term events, while GCN does not have this phenomenon.

For events of different frequencies, the study found that in small networks, the prediction performance of GAT and EvolveGCN may become unstable due to high-frequency events. The possible reason is that topological changes have a greater impact on the overall message transmission of small networks, causing the attention learning of GAT or the weight update of EvolveGCN to differ too much in a short period of time. In large networks, because the impact of message transmission caused by topological changes is relatively small, high-frequency events can improve the prediction performance of the

three models by providing more topological change data.

How do GAT, GCN and EvolveGCN perform with and without topological change data during training?

GAT was found to have the best overall performance when trained with topological variation data, but despite having the best overall performance, it seems to be unable to capture the exact topological variation in some cases. It is found that GCN has the best overall performance when trained without topology-changing data. Compared with the case where topological changes are visible, GCN has smaller topological leakage and scale imbalance, and the overall generalization ability is improved, but it still has the problem of scale imbalance near the PV bus compared with the other two models. EvolveGCN performs worse than GAT and GCN in overall performance, but it can stably predict the values of PV bus and its neighbors that lack spatial correlation, which is related to its model characteristics that rely on historical information for training.

6.3. USE CASES IN REAL LIFE

As stated in the introduction, modern power systems face challenges from both Distributed Energy Resources (DERs) and Integrated Energy Systems (IES) when using WLS for State Estimation (SE). In smart grids, data redundancy is reduced, slack bus conditions become less stable, and pseudo-measurements are increasingly unreliable. This section highlights how the experimental results of the three GNN-based models tested in this study can address these challenges in realistic power system scenarios.

GNNs outperforms WLS in case of slack bus voltage variation

In the default case, all three GNN models outperform WLS when the slack bus voltage is not fixed. This shows that in real-world IES environments—such as power-gas coupled systems—where gas compressors place irregular and time-dependent electrical loads on the grid. These fluctuations make the slack bus voltage unstable. The superior performance of GNNs under such conditions compared to WLS suggests that they are more suitable for SE in IES contexts, where the impact of coupled subsystems on grid stability cannot be ignored.

Static GNNs effectively handle topological changes in high-DER systems

In high DER-penetration systems, topological changes occur frequently due to fault isolation, protective disconnection (e.g., IEEE 1547 compliance), scheduled maintenance, or economic dispatch strategies such as curtailing wind power to stabilize prices. The case studies in this research demonstrate that the duration and frequency of these topological changes do not significantly affect the performance of the GNN models, indicating a strong degree of robustness to various types of network reconfigurations.

Although EvolveGCN-O does not outperform GCN or GAT during topological changes, the static GNNs still show practical value. GAT performs well in scenarios where potential topology changes are known and can be included in training, such as scheduled

maintenance or controlled disconnections of DERs. GCN, by contrast, adapts better to unseen topological changes, making it more suitable for real-time SE applications where unexpected line outages or generator dropouts occur. In systems with high rooftop solar or regional wind turbine deployment, measurement devices are often sparse, and pseudo-measurements become less reliable due to unpredictable weather and lack of historical data. In these contexts, GCN and GAT exhibit a clear advantage over traditional WLS. GNN-based SE offers an opportunity to fill in the gaps using learned patterns rather than relying on fragile statistical assumptions.

EvolveGCN improves voltage prediction accuracy at PV buses for state estimation

Furthermore, although EvolveGCN-O does not show an advantage around the topological change moment, it provides higher stability in estimating the state of PV bus and its neighbors. In distribution grids, small generators are often connected without a centralized control unit such as an inverter. In this case, the PV bus voltage is highly dependent on local weather conditions, and the system operator cannot actively regulate it. Even if an inverter exists, its control logic cannot be successfully understood through the information in a single snapshot. Therefore, regardless of whether there is a control unit or not, the voltage curve of the PV bus lacks a strong spatial correlation with its neighbors — either affected by weather conditions or by control logic, so it is difficult for static GNNs that only rely on single-snapshot information for message passing to capture this. However, EvolveGCN-O's ability to model temporal evolution and retain memory allows it to better handle these local dynamics. In practice, this could be very useful for microgrid operators or rural distribution system operators managing areas with fluctuating PV generation.

6.4. FUTURE RESEARCH DIRECTION

Based on the model selection and experimental results of this study, three directions are considered to be worth trying in the future to achieve better performance in state estimation under topological changes.

Try EvolveGCN-H or other GCN + RNN combinations

First, the reason why GCN and GAT perform better overall may be that they still use a single static graph structure, which makes them more accurate in predicting regression problems. Although there is no historical information available, most nodes do not have the lack of spatial correlation like the PV bus. Therefore, the impact caused by topological changes can be mostly known through edge index and message passing. The poor performance of EvolveGCN may be because it was originally designed for node classification or edge prediction and is not complex enough for regression problems such as state estimation. Although we chose EvolveGCN-O to focus on changes in graph structure, the results show that numerical prediction capabilities still play an important role in this type of prediction tasks.

Therefore, in the future, it can be considered to use structures that bring graph informa-

tion into the RNN layer for learning, such as EvolveGCN-H or other GCN+RNN combinations. Although many GCN+RNN combinations were found in the literature review to be ineffective in predicting newly emerged nodes, in larger graph structures, the importance of predicting new nodes and overall prediction performance may vary from task to task, so it is still worth trying. In addition, in the training of GNN based models, general batch learning cannot accept graphs with different numbers of nodes in the same batch, which makes it difficult to simulate the actual "node removal" or "new node appearance". This might be possible using masking techniques to simulate node removal, but be aware of the differences in graph structure between this and actual removal.

GCN + EvolveGCN combination

Second, the results of this study found that GCN has better generalization ability, while EvolveGCN has stable prediction ability on the PV bus itself and its neighboring nodes. Considering the increasing number of multienergy source systems, the combination of GCN and EvolveGCN may be a method that can stabilize the prediction near the PV bus while maintaining a good generalization ability to predict larger graph structures.

Actual implementation

Third, considering that the case designs are all based on simulations and 'where should the measurement points for the estimation of the state of the grid be' is another topic in itself, this study did not distinguish between 'actual measurements' and 'pseudo-measurements' in the simulated measurement data. In addition, the data granularity is not completely consistent with the actual case due to data accessibility issues. In the future, it is expected to conduct actual tests in small power grids with high penetration rates, and use WLS and three GNN models to perform state estimation through actual measurement values, so that the advantages and disadvantages of the four methods will be more clear and practical.

7

CONCLUSION

This study helps to understand the performance of GNNs in power grid state estimation under topological changes. It verifies that static GNNs (GCN and GAT) have certain adaptability under different circumstances due to their training mechanism. The performance of GCN and GAT varies depending on data visibility, and EvolveGCN provides more stable predictions for weakly spatial-correlated nodes such as PV buses.

By building this framework in a simulated environment, this study validated the numerical stress testing of GCN, GAT, and EvolveGCN under drastic changes in feature values under different topologies. This study also adapts temporal GNNs to the State Estimation (SE) task by modifying the recurrent model from sequence to batch-wise learning. Specifically, varying the frequency of weight updates during training enables snapshot-based learning to incorporate temporal effects. This helps make the batch-wise learning method adaptable to learning temporal effects.

A key insight emerging from this study is that combining static GNNs with temporal GNNs (e.g., EvolveGCN + GCN) may produce more stable and general predictions. This is motivated by the generalizability of GCN and the prediction stability of EvolveGCN near the PV buses. For further development, future steps include applying this method to real datasets with finer temporal resolution and diverse noise characteristics reflecting sensor reliability and differentiation between the actual and pseudo- measurements. Exploring other temporal GNNs (GNNs that incorporate node embeddings in recurrent layer training) is also a promising direction.

Overall, this work highlights that static GNNs show some ability to adapt to topological changes in the SE task, and temporal GNNs can enhance the stability of prediction near PV buses. These findings are relevant to future applications of machine learning in Distributed Energy Resource (DER)-dominated grid environments.

BIBLIOGRAPHY

Abur, A., & Exposito, A. G. (2004). *Power system state estimation: Theory and implementation*. CRC press.

Badillo, S., Banfai, B., Birzele, F., Davydov, I. I., Hutchinson, L., Kam-Thong, T., Siebourg-Polster, J., Steiert, B., & Zhang, J. D. (2020). An introduction to machine learning. *Clinical pharmacology & therapeutics*, *107*(4), 871–885.

Bai, Y., Yan, B., Zhou, C., Su, T., & Jin, X. (2023). State of art on state estimation: Kalman filter driven by machine learning. *Annual Reviews in Control*, *56*, 100909. https://doi.org/https://doi.org/10.1016/j.arcontrol.2023.100909

Barfoot, T. D. (2024). *State estimation for robotics*. Cambridge University Press.

Basetti, V., Chandel, A. K., & Subramanyam, K. (2018). Power system static state estimation using jade-adaptive differential evolution technique. *Soft Computing*, *22*, 7157–7176.

Bre, F., Gimenez, J. M., & Fachinotti, V. D. (2018). Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, *158*, 1429–1441.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Clements, K. A. (2011). The impact of pseudo-measurements on state estimator accuracy. *2011 IEEE Power and Energy Society General Meeting*, 1–4.

Dileo, M., Zignani, M., & Gaito, S. (2024). Temporal graph learning for dynamic link prediction with text in online social networks. *Machine Learning*, *113*(4), 2207–2226.

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. *2018 Chinese control and decision conference (CCDC)*, 1836–1841.

Do, K., Tran, T., & Venkatesh, S. (2019). Graph transformation policy network for chemical reaction prediction. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 750–760.

Fan, H., Wang, C., Liu, L., & Li, X. (2022). Review of uncertainty modeling for optimal operation of integrated energy system. *Frontiers in energy research*, *9*, 641337.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., & Yin, D. (2019). Graph neural networks for social recommendation. *The world wide web conference*, 417–426.

Gao, C., Zhu, J., Zhang, F., Wang, Z., & Li, X. (2022). A novel representation learning for dynamic graphs based on graph convolutional networks. *IEEE Transactions on Cybernetics*, *53*(6), 3599–3612.

Gulcehre, C., Moczulski, M., Denil, M., & Bengio, Y. (2016). Noisy activation functions. *International conference on machine learning*, 3059–3068.

Habib, B., Isufi, E., van Breda, W., Jongepier, A., & Cremer, J. L. (2023). Deep statistical solver for distribution system state estimation. *IEEE Transactions on Power Systems*.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, *29*(6), 82–97.

Hu, W., Yang, Y., Cheng, Z., Yang, C., & Ren, X. (2021). Time-series event prediction with evolutionary state graph. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 580–588.

Huang, J., Guan, L., Su, Y., Yao, H., Guo, M., & Zhong, Z. (2020). Recurrent graph convolutional network-based multi-task transient stability assessment framework in power system. *IEEE Access*, *8*, 93283–93296.

Huang, Y., Sun, Q., Zhang, N., & Wang, R. (2021). A multi-slack bus model for bi-directional energy flow analysis of integrated power-gas systems. *CSEE Journal of Power and Energy Systems*.

IEEE Standard Association. (2018). *IEEE Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces*. https://doi.org/10.1109/IEEESTD.2018.8332112

IEEE Standard Association. (2022). *IEEE Guide for Using IEEE Std 1547 for Interconnection of Energy Storage Distributed Energy Resources with Electric Power Systems* [IEEE Standard Association].

Illinois Center for a Smarter Electric Grid. (n.d.-a). Ieee-14 bus system [Accessed: 2025-03-17].

Illinois Center for a Smarter Electric Grid. (n.d.-b). Ieee-57 bus system [Accessed: 2025-03-17].

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic markets*, *31*(3), 685–695.

Jin, X.-B., Robert Jeremiah, R. J., Su, T.-L., Bai, Y.-T., & Kong, J.-L. (2021). The new trend of state estimation: From model-driven to hybrid-driven methods. *Sensors*, *21*(6), 2085.

Lan, L., Li, J., & Fu, Y.-G. (2021). Developing gcn: Graph convolutional network with evolving parameters for dynamic graphs. *International Conference on Neural Information Processing*, 369–376.

Lennie, P. (2003). The cost of cortical computation. *Current biology*, *13*(6), 493–497.

Liao, W., Bak-Jensen, B., Pillai, J. R., Wang, Y., & Wang, Y. (2021). A review of graph neural networks and their applications in power systems. *Journal of Modern Power Systems and Clean Energy*, *10*(2), 345–360.

Manessi, F., Rozza, A., & Manzo, M. (2020). Dynamic graph convolutional networks. *Pattern Recognition*, *97*, 107000.

Mhaskar, H. N., & Poggio, T. (2016). Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, *14*(06), 829–848.

Moshtagh, S., Sifat, A. I., Azimian, B., & Pal, A. (2023). Time-synchronized state estimation using graph neural networks in presence of topology changes. *2023 North American Power Symposium (NAPS)*, 1–6.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Ngo, Q.-H., Nguyen, B. L., Vu, T. V., Zhang, J., & Ngo, T. (2024). Physics-informed graphical neural network for power system state estimation. *Applied Energy*, *358*, 122602.

Oliveira, T. P., Barbar, J. S., & Soares, A. S. (2016). Computer network traffic prediction: A comparison between traditional and deep learning neural networks. *International Journal of Big Data Intelligence*, *3*(1), 28–37.

Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T., & Leiserson, C. (2020). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *Proceedings of the AAAI conference on artificial intelligence*, *34*(04), 5363–5370.

Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579–588.

Power statistics [Accessed: 2025-05-02]. (n.d.).

Pytorch geometric temporal [Accessed: 2025-05-02]. (n.d.).

Rakpenthai, C., Uatrongjit, S., & Premrudeepreechacharn, S. (2011). State estimation of power system considering network parameter uncertainty based on parametric interval linear systems. *IEEE Transactions on Power Systems*, *27*(1), 305–313.

Ramachandran, T., Reiman, A., Nandanoori, S. P., Rice, M., & Kundu, S. (2019). Distribution system state estimation in the presence of high solar penetration. *2019 American Control Conference (ACC)*, 3432–3437.

Rico, J., Barateiro, J., & Oliveira, A. (2021). Graph neural networks for traffic forecasting. *arXiv preprint arXiv:2104.13096*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, *20*(1), 61–80.

Seo, Y., Defferrard, M., Vandergheynst, P., & Bresson, X. (2018). Structured sequence modeling with graph convolutional recurrent networks. *Neural information processing: 25th international conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, proceedings, part I 25*, 362–373.

Sharma, A., & Jain, S. K. (2018). A review and performance comparison of power system state estimation techniques. *2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, 770–775. https://doi.org/10.1109/ISGT-Asia.2018.8467861

Shi, M., Huang, Y., Zhu, X., Tang, Y., Zhuang, Y., & Liu, J. (2021). Gaen: Graph attention evolving networks. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*.

Venkateswarlu, C., & Karri, R. R. (2022). Chapter 5 - data-driven modeling techniques for state estimation. In C. Venkateswarlu & R. R. Karri (Eds.), *Optimal state estimation for process monitoring, fault diagnosis and control* (pp. 91–111). Elsevier. https://doi.org/https://doi.org/10.1016/B978-0-323-85878-6.00010-5

Verma, A. K., Chaki, S., Routray, A., Mohanty, W. K., Jenamani, M., Chaudhuri, P., & Das, S. (2013). Quantifying sand fraction from seismic attributes using modular artificial neural network. *10th Biennial International Conference & Exposition*, 399–404.

Vrahatis, A. G., Lazaros, K., & Kotsiantis, S. (2024). Graph attention networks: A comprehensive review of methods and applications. *Future Internet*, *16*(9), 318.

Wang, L., Zhou, Q., & Jin, S. (2020). Physics-guided deep learning for power system state estimation. *Journal of Modern Power Systems and Clean Energy*, *8*(4), 607–615.

Xu, X., Zhao, X., Wei, M., & Li, Z. (2023). A comprehensive review of graph convolutional networks: Approaches and applications. *Electronic Research Archive*, *31*(7), 4185–4215.

You, J., Du, T., & Leskovec, J. (2022). Roland: Graph learning framework for dynamic graphs. *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2358–2366.

Yu, B., Yin, H., & Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

Yu, Y., Qian, W., Zhang, L., & Gao, R. (2022). A graph-neural-network-based social network recommendation algorithm using high-order neighbor information. *Sensors*, *22*(19), 7122.

Zhong, F., Liu, Y., Liu, L., Zhang, G., & Duan, S. (2022). Dedgcn: Dual evolving dynamic graph convolutional network. *Security and Communication Networks*, *2022*(1), 6945397.

Zhou, D., Ma, S., Huang, D., Zhang, H., & Weng, S. (2020). An operating state estimation model for integrated energy systems based on distributed solution. *Frontiers in Energy*, *14*, 801–816.