

Synthetic Power Grid

Norwegian University of Science and Technology

Department of Computer Science

Supervisor: Ali Alsam

May 2022

Anders Hermanrud
Kenneth Solvoll
Hogne Winther

Anders Hermanrud
Kenneth Solvoll
Hogne Winther

Synthetic Power Grid

Bachelor Thesis
Supervisor: Ali Alsam
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Preface

The purpose of this report is to investigate the feasibility of generating synthetic power grid data, with machine learning as a tool. This implies exploring different solutions from other similar projects, figuring out how to process power grid data and deciding what machine learning model fits this assignment.

The authors of this report are computer engineering undergraduates, studying at NTNU in Trondheim. They have finished courses in machine learning and data structures throughout their study. These courses have provided the authors with a fundamental basis of knowledge for taking on this project. It is recommended to the reader of this report to have equivalent knowledge taught in these courses:

- IDATT2502 Applied machine learning with a project, a course at NTNU covering machine learning concepts and their usage.
- IDATT2101 Algorithms and data structures, a course at NTNU covering complexity theory and different data structures.

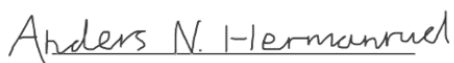
A significant part of this report is the project's process from the beginning to the end, mainly the data processing. A big part of producing synthetic data with the help of machine learning is understanding the original data needed for the input. Different machine learning methods have specific requirements for their data input, so analysing and processing the power grid data is key to succeeding with the given assignment.


Sincere gratitude to those who helped the project along the way:

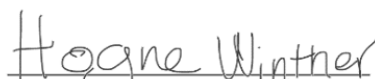
- Ali Alsam, Associate Professor at NTNU, as the team's academic advisor, for going above and beyond what is expected of him and helping the team whenever, wherever.
- Volue, especially Sunniva F. Hoel and Tonje Kroglund Rian, for providing a project workspace, technical resources and support throughout the project.

The authors of this report:

Trondheim, 20th May 2022


Anders Hermanrud


Kenneth Solvoll


Hogné Winther

Abstract

Designing, analysing and testing power grid structures today require topologies and geographical information of the real power grids. However, access to this data is restricted due to security concerns surrounding critical infrastructure. This project aims to generate synthetic power grid data with machine learning as the means in collaboration with Volue.

Volue, a Norwegian technology company, provided restricted power grid data to develop the synthetic data generator. This data represented the power grid infrastructure of a small town in northern Norway. Analysing the power grid data was done by inspecting the raw data files and consulting Volue. Representatives from Volue provided power grid expertise whenever it was needed.

The data provided by Volue was processed to fit the machine learning model requirements. Processing the data was needed as the raw data format of power grids cannot be interpreted by machine learning models. The data processing consisted of removing insignificant objects that had no impact on the power grid data structure. The power grid data was then transformed into lists of nodes and edges in order to be represented as graphs for graph-based machine learning. Power grid data have high variations in size and content, creating inconsistent input for training. This was challenging as machine learning requires a fixed input size to work well. The transformed graphs were used as input for the chosen geometric deep learning models, a graph autoencoder and a variational graph autoencoder. The team managed to generate synthetic power grid topologies, with a similar structure as the real topologies, with machine learning. However, Volue had no evaluator for the generated power grid topologies. With no metric for success, there is no way to tell if the generated topologies are of any use.

Volue stated that development will continue after the end of the project. With this in mind, the authors of this report will present recommended paths for future work.

Assignment

Original assignment text

"Power grid infrastructure is considered critical infrastructure, and data from some parts e.g. infrastructure related to military facilities, are highly restricted. We need to have a reliable test grid with real-life credibility without any identifying or connection to actual grids, but still covering all the complexity. We want to investigate how to use Machine learning to "learn" power grid structures from real-life structures and train a model to use as a basis to create a "test data generator" to avoid using real data for test purposes and ensure safety and security of customers' real data."

In discussion with Volue, the team and the supervisor, it was agreed to make the following modifications to the assignment.

Modifications to the assignment text

Volue wishes to generate synthetic data for testing, cost evaluation and preparation of new grid infrastructures. As a first step, we agreed to create a prototype that learns and mimics the grid topologies of individual real-world power grids. As a second stage, Volue wishes to evolve the data generation to accommodate additional parameters, like geographical positions.

The assignment was to research how synthetic power grid data that statistically resembles real-world data may be generated using machine learning. Knowing that grid data can be presented as graphs, we adjusted the assignment to investigate the suitability of different machine learning models for generating synthetic graph data.

Table of Contents

List of Figures

List of Tables

1	Introduction	1
1.1	Background	1
1.2	Research Questions	1
1.3	Structure	2
1.4	Acronyms	3
2	Theory	4
2.1	Nodes, edges and graphs	4
2.1.1	Graph Embedding	4
2.1.2	Node Embedding	4
2.1.3	Node2Vec	5
2.2	Graph Edit Distance	5
2.3	Machine learning	6
2.4	Geometric Deep Learning	6
2.4.1	Graph neural networks	7
2.5	Generative Models	7
2.5.1	Generative adversarial networks	7
2.5.2	Autoencoder	8
2.5.2.1	Variational Autoencoder	8
2.6	Synthetic Data	9
2.6.1	Common information model	9
2.7	Related Work	9
2.7.1	Synthetic Power Grids	10
2.7.1.1	Statistical synthetic grid generation	10
2.7.1.2	Synthetic Power Grids from Real World Models	10
2.7.1.3	A Learning-Based Method for Generating Synthetic Power Grids	10
2.7.1.4	Deep Generative Graph Distribution Learning for Synthetic Power Grids	10
2.7.2	GraphRNN	11
2.7.3	MolGAN: An implicit generative model for small molecular graphs	11
2.7.4	GraphGAN	11

2.7.5	Variational Graph Autoencoder	12
2.7.6	GraphVAE	12
3	Method	13
3.1	Process	13
3.1.1	Early stages	13
3.1.1.1	Assignment Scoping	13
3.1.1.2	Team Roles and Work	13
3.1.1.3	Project Environment	14
3.1.1.4	Work Process	14
3.1.2	Process Execution	14
3.1.3	Research	15
3.1.4	Data	16
3.1.4.1	Power Grid Data	16
3.1.4.2	NETBAS	16
3.1.4.3	Data Analysis	17
3.1.4.4	Data Processing	20
3.1.5	Machine Learning	22
3.1.5.1	Choice of Machine learning method	22
3.1.5.2	Choice of graph learning model	22
3.1.5.3	Experiment Setup	23
3.1.5.4	Hyperparameters	24
3.1.5.5	Evaluation of models	24
3.2	Choice of technologies	24
3.2.1	PyTorch	25
3.2.1.1	PyTorch Geometric	25
3.2.2	TensorBoard	25
3.2.3	Pandas	25
3.2.4	Numpy	25
3.2.5	RDFlib	25
3.2.6	Table Evaluator	26
3.2.7	SDV	26
3.2.8	Matplotlib	26
3.2.9	Jupyter Notebook	26
3.2.10	NetworkX	26

3.2.11	Intellij IDEA	26
3.2.12	Git	26
3.2.13	CUDA supported GPU	27
4	Results	28
4.1	Data Processing	28
4.2	Node Generation	30
4.2.1	CTGAN	31
4.2.2	TVAE	34
4.2.3	Comparing CTGAN with TVAE	37
4.3	Link Prediction	37
4.3.1	Graph Auto Encoder	38
4.3.2	Variational Graph Auto Encoder	39
4.3.3	Variational Graph Auto Encoder - Planetoid dataset	41
4.3.4	Comparing GAE with VGAE	42
5	Discussion	44
5.1	Similar Solutions	44
5.2	Data	44
5.2.1	Requirements for synthetic data	44
5.2.2	Data analysis and Processing	44
5.3	Graph Generation	45
5.3.1	Best performing model	45
5.3.2	Machine learning as the tool	45
5.4	Broader Impact	46
5.4.1	Sustainability	46
5.5	Reflection Students	46
6	Conclusion	47
6.1	Future work	48
	Bibliography	49
	Attachments	52
	List of Figures	
1	Basic Graph	4

2	Node Embeddings	5
3	Node2Vec	5
4	Graph Edit Distance	6
5	Graph neural network	7
6	GAN architecture	8
7	AutoEncoder model	8
8	Variational Autoencoder model	9
9	Triple illustration	9
10	MolGAN FrameWork	11
11	GraphGAN Framework	12
12	Scientific Method	14
13	PowerGrid Visualisation	16
14	Object proportions in raw data	18
15	Visualisation of raw data	19
16	Node degrees in one data file	20
17	CIM format	20
18	Ttl format	21
19	Visualisation of an edge list	21
20	Removed Data	28
21	Graph of processed data	28
22	Object proportions processed data	29
23	Processed node degrees	29
24	Node Clusters in a graph	30
25	CTGAN Loss	31
26	EnergyConsumer GAN Distribution	32
27	Breaker GAN Distribution	33
28	Correlation GAN	33
29	Correlation Difference GAN	34
30	TVAE Loss	35
31	EnergyConsumer Distribution TVAE	35
32	Breaker Distribution TVAE	36
33	Correlation TVAE	36
34	Correlation Difference TVAE	37
35	GAE Loss	38
36	GAE Confusion Matrix	39

37	Original - GAE generated	39
38	VGAE Loss	40
39	VGAE Confusion matrix	40
40	Original - VGAE generated	41
41	VGAE without node type	41
42	VGAE - Planetoid Loss	42
43	VGAE Planetoid Confusion matrix	42

List of Tables

1	File Summary	17
2	Object Example	18
3	Node2Vec configuration	22
4	Explanation for hyperparameters	24
5	CTGAN Hyperparameters	31
6	Hyperparameters for TVAE	34
7	Link prediction Hyperparameters	38

1 Introduction

This section describes in detail how the project started and why synthetic power grids were chosen as the assignment. Furthermore, the research questions will be explained, followed by the report's structure and acronyms.

1.1 Background

Volue is a Norwegian technology company founded in 2020. It is the result of a merger between four Nordic companies: Markedskraft, Powel, Scanmatic and Wattsight. Volue works in the fields of power grids, infrastructure, energy and water. Volue's Power Grid department focuses on delivering software solutions that enable grid companies to design, maintain, analyse and monitor their power grids in real-time.

The maintenance and improvement of specialised grid design software is a continuous process that requires intensive testing with real-world datasets. Power grid companies have access to large, real datasets that lend themselves to the task of improving software and developing new functionalities. Software development companies, including Volue, must request approval and follow strict rules to use real power grid data. Using these real data for software development is, however, problematic due to their sensitive nature. Power grid data are, in fact, classified and considered by most nations a matter of national security. Real-grid data contain information pertaining to peoples' locations, vital infrastructures, level of consumption as well as not publicly known military facilities. Nevertheless, masking sensitive information, including geographical information, is not a satisfactory solution as this information is needed in the testing, designing and analysing processes.

In the development of their grid design software, Volue wishes to collaborate with external developers and researchers to invent new solutions and optimise existing routines. Sharing real data with these actors is, however, prohibited. Further, Volue desires to estimate and optimise the cost and required infrastructure of power grids for new roads, towns and cities. Ideally, Volue's representatives desire the ability to select a region on the map, specify how many people are expected to live in it, and how many shops, schools and hospitals they might need. As a second step, they wish to use their software to generate an artificial grid that closely resembles what power engineering companies might design.

The ideal aim of this project is to use artificial intelligence to learn the underlying properties of real-world grid data and create resembling synthetic grids. As we will discuss in this report, this aim is marred with difficulties. Namely, the real grid data are not in a format that lends itself to machine learning without processing. The scope and size of the synthetic data are not clearly defined, and there is no metric that can be used to assess the difference between real and synthetically generated data. The latter point proved to be the most challenging. While we managed to produce a sample of artificial data, we were unable to improve the results or quantify their usefulness.

1.2 Research Questions

The purpose of this report is to investigate the feasibility of generating synthetic power grids through machine learning models and choosing the most suited models for the task. The team wanted to formulate research questions to help determine the feasibility of our assignment, taking the available capacity and resources into account. Our first research question was formulated as:

Has synthetic grid data been generated before? If yes, how?

Real-world power grid data have an underlying node structure representing the various components in the grid network. Manually creating these networks is time-consuming as they have different requirements. For instance, an industrial area is designed differently from a local neighbourhood,

as the networks contain various components. Therefore, the synthetic data needs to mimic the appropriate grid structures, which leads us to the following question:

What are the requirements of the synthetic data, and how does it resemble the real-world data?

There is a multitude of different grid networks, ranging from city grids to small neighbourhoods. The networks have varying sizes, locations and are constructed by different companies. This is not ideal for machine learning, as it relies on a fixed-sized input for consistent results; A regional grid network cannot be compared to a small mountain village. This leads to our third question:

Given real-world grid data, which parts of the data should be used for training?

The aforementioned question was our starting point in the project. Having analysed the real data, however, we found that most of the data is superfluous. Our analysis of the real-world data led us to a core question, which we believe to be a first step required in a successful project where synthetic data are generated:

What is a real-world bare-bone power grid?

Specifically, we wish to answer the question of how to filter extra information in real-world data, and what amounts to essential information.

Value has stated that this is just the beginning of the project and intends to continue the development after the bachelor team is finished. To create a good foundation for future development, the team wanted to investigate the efficiency of different machine learning models when working with power grid data, giving us our next research question:

Which machine learning model is best suited for learning power grid data?

This project aims to solve the assignment by using machine learning. In the past decade, machine learning has become a buzzword and "blossomed" into a solution for many data-related challenges. However, machine learning is a tool, and it is important to pick the right tool for the job, leading to the final research question:

Is machine learning the best approach to generate synthetic power grid data?

1.3 Structure

This report is divided into the following chapters:

Introduction - This chapter introduces our report. It includes the project background, research questions, structure, and acronyms used throughout the report.

Theory - This chapter contains all relevant theoretical information for understanding the methods and discussions in the report.

Method - This chapter describes the methodology and scientific process. Furthermore, the choice of technologies and team decisions is described.

Results - This chapter presents all obtained results. In addition, models are compared to give better insight into performance, accuracy, loss and score.

Discussion - This chapter reflects the obtained results. Broader impact and sustainability are also discussed.

Conclusion - This chapter attempts to answer the research questions presented in the introduction, based on the discussion of the results. Based on this conclusion, further work will be presented .

Bibliography - This chapter contains all references used throughout the report.

Attachments - This chapter contains all relevant attachments for the report.

1.4 Acronyms

AC - *Alternating Current*

AE - *Autoencoder*

CIM - *Common Information Model*

CSV - *Comma Separated Values*

CTGAN - *Conditional Tabular Generative Adversarial Network*

GAE - *Graph Autoencoder*

GAN - *Generative Adversarial Networks*

GGAN - *Graph Generative Adversarial Networks*

IDE - *Integrated Development Environment*

MB - *Mega Byte*

NIMBLE - *Network imitating method based learning*

NTNU - *Norges teknisk-naturvitenskapelige universitet*

OPF - *Optimal Power Flow*

RDF - *Resource Description Framework*

T-SNE - *T-Distributed Stochastic Neighbor Embedding*

TTL - *Terse RDF Triple Language (Turtle)*

TVAE - *Tabular Variational Auto Encoder*

UML - *Unified Model Language*

VAE - *Variational Autoencoder*

VGAE - *Variable Graph autoencoder*

2 Theory

This chapter describes all necessary information needed to understand the methodology and results.

2.1 Nodes, edges and graphs

A graph is one or more nodes with zero or more edges. Each node represents a component in the system, while the edges display the connections between the nodes [17].

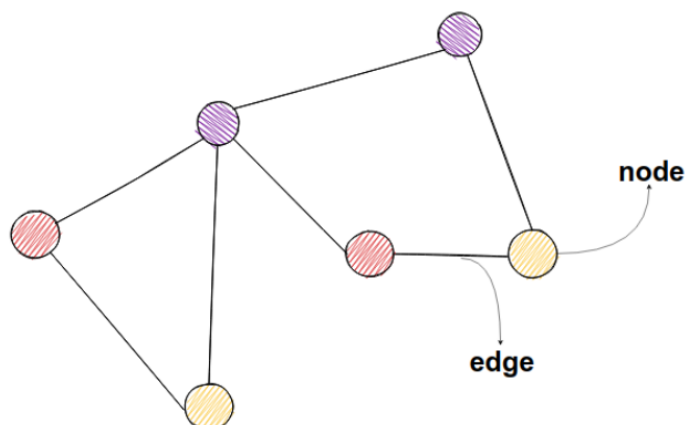


Figure 1: A basic graph. Taken from [20].

2.1.1 Graph Embedding

A graph embedding is a low dimensional vector representation of a graph [26]. These vectors are often called node embeddings. This embedding typically aims to preserve some key information about the original graph. What information is preserved is up to the algorithm creating this embedding. Some algorithms that create node embedding are: Random Walk, Hierarchical softmax and cosine similarity. Some machine learning on graphs can generate graph embeddings as a result of graph evaluation.

2.1.2 Node Embedding

Node embedding represents nodes as vectors in the Euclidian space [10]. With these embeddings, we can look at the topology of a network. Nodes residing in different parts of the graph can have similar structural roles within their local network topology. Identifying these roles can provide insight into how the network is structured, which can be used by machine learning algorithms.

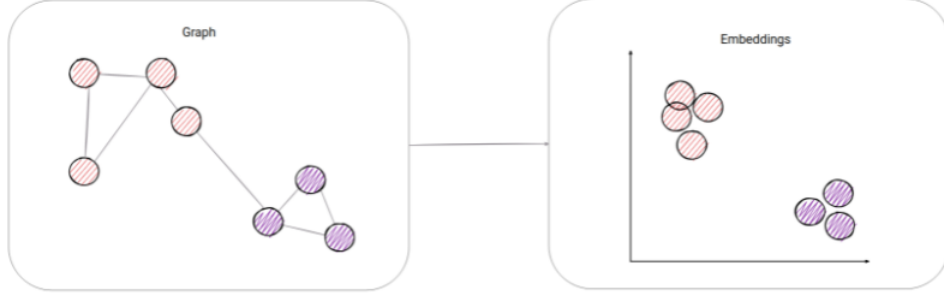


Figure 2: Before and after creating node embedding with cosine similarity. Taken from [20].

2.1.3 Node2Vec

Node2Vec is an algorithmic framework for learning continuous feature representations for nodes in networks [16]. It generates vector representations of nodes on a graph. The algorithm uses biased random walks from a targeted starting node and "walks" through a graph. These walks can be configured through multiple arguments: the graph g , dimensions d , amount of walks n , walk length l , different probability integers for visiting new nodes q , or previously visited ones p . More arguments are available for configuration, but they are not important. The random walk function is $F(g, d, n, l, q, p)$.

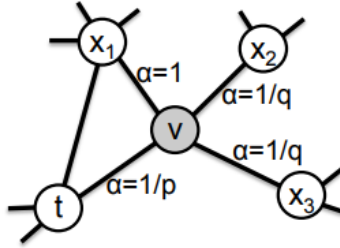


Figure 3: Illustration of the random walk procedure. The walk transitioned from t to v and is now evaluating its next step from v . Taken from [16].

2.2 Graph Edit Distance

Graph edit distance is a graph evaluation method that measures the similarity of two graphs [32]. The graph edit distance denotes how much one graph needs to be edited to be identical to the other. This can be defined as

$$GED(g_1, g_2) = \min_{(e_1, \dots, e_k) \in \mathcal{P}(g_1, g_2)} \sum_{i=1}^k c(e_i) \quad (1)$$

, where $\mathcal{P}(g_1, g_2)$ is the set of operations done to transform g_1 into g_2 and $c(e)$ is the cost of the operation.

The operations are usually:

- **Node insertion** to add a node to a graph
- **Node deletion** to delete a node in a graph

- **Node substitution** to change a node in a graph
- **Edge insertion** to add an edge between two nodes in a graph
- **Edge deletion** to delete an edge between two nodes in a graph

Based on the graph, additional operations can be added, for instance, changing edge weights on weighted graphs.

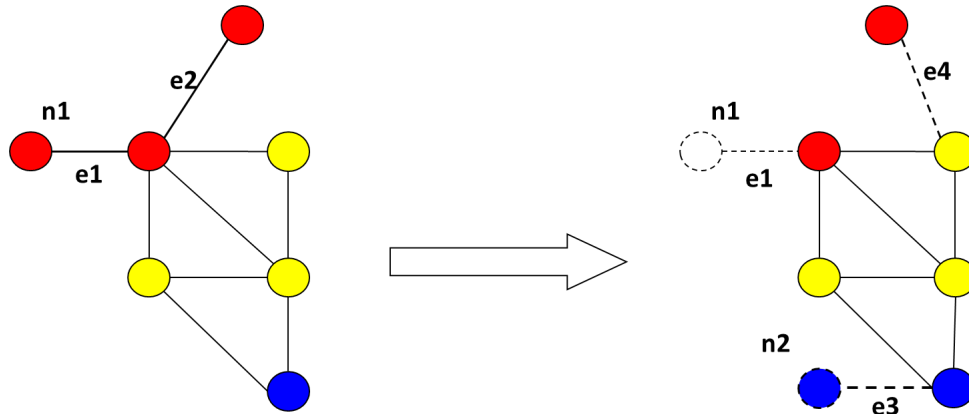


Figure 4: Illustration of how graph edit distance operations transforms $\mathbf{g1}$ on the left to $\mathbf{g2}$ on the right, taken from [14]. Transforming $\mathbf{G1}$ into $\mathbf{G2}$ takes six operations. Delete edge $\mathbf{e1}$, delete node $\mathbf{n1}$, delete edge $\mathbf{e2}$, insert edge $\mathbf{e4}$, insert node $\mathbf{n2}$, insert edge $\mathbf{e3}$. With all operations having a cost of 1, $GED(g_1, g_2) = 6$.

2.3 Machine learning

Machine learning is a field of AI that tries to solve the issue of making computers automatically learn through iterative improvement [21]. Using statistical algorithms, machine learning tries to learn the underlying structure of data. Machine learning can be split into three categories: Supervised, Unsupervised and Reinforcement Learning [25].

Supervised Learning is a technique used on datasets where each input has a predetermined output [25]. Supervised learning aims to create an inferred function that can predict correct outcomes for new input data. Typical uses for this are classification and regression problems.

Unsupervised Learning is a more exploratory approach than supervised learning [25]. This approach utilises data sets without any predetermined outcome and tries to learn underlying structures. This structure can be used to group data which at first glance seems to have no similarities. Typical uses include clustering data.

Reinforcement Learning is an approach where an algorithm is rewarded or punished based on its actions [6]. Through the rewards and punishments, the model learns desired actions. The long-term goal of the model is to receive the maximum reward by achieving the optimal solution. This approach can be used to make a computer how to play games [7], an example is Google's AlphaGo [4].

2.4 Geometric Deep Learning

Geometric Deep Learning is a small field within machine learning using neural networks that learns from non-Euclidean data [36]. An excellent example of non-Euclidean data is Graphs and Manifolds.

2.4.1 Graph neural networks

Graph neural networks are a subgroup of Geometric deep learning designed to be used directly on graphs-structured data [1]. GNNs are divided into two tasks: Node and Graph-focused. For node-focused tasks, GNN’s objective is to learn each node’s features to facilitate these tasks. For graph-focused tasks, their objective is to learn representative graph features [26]. The process can be explained as:

$$\mathbf{F}^{(of)} = h(\mathbf{A}, \mathbf{F}^{(if)}) \quad (2)$$

Where $\mathbf{A} \in \mathbb{R}^{N \times N}$ represents the graphs adjacency matrix with N nodes, $\mathbf{F}^{(if)} \in \mathbb{R}^{N \times d_{if}}$ and $\mathbf{F}^{(of)} \in \mathbb{R}^{N \times d_{of}}$ represents the input and output feature matrices where their dimensions are d_{if} and d_{of} . if and of represent the input and output filtering and $h(\cdot)$ is a graph filter.

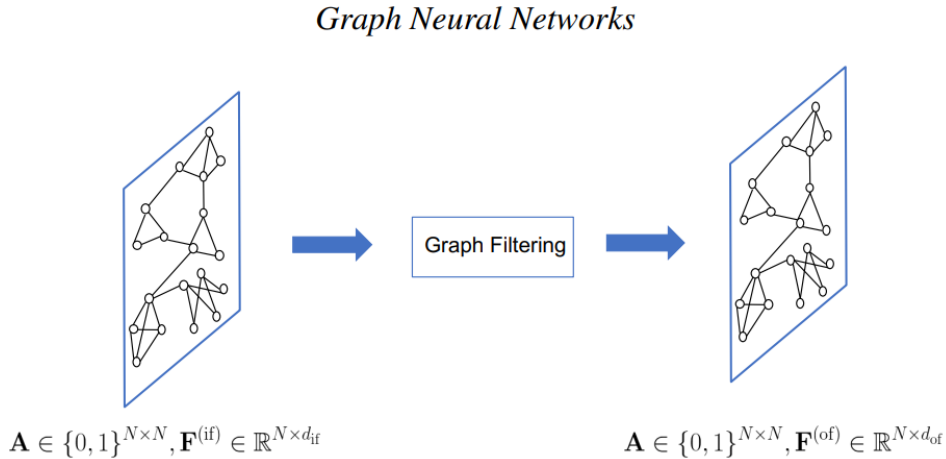


Figure 5: Illustration of Equation 2. Taken from [26].

By training on graphs directly, the goal is to learn representative features from the structure, such as clusters and corners. Due to the many types of graphs, there are also multiple variants of graph-based machine learning algorithms. An example is a graph convolutional network, which works as a CNN for graphs utilising weight sharing.

2.5 Generative Models

Generative models used in machine learning generate a data distribution that is similar to the original input distribution [5].

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | X_{\pi_i}) \quad (3)$$

Equation 3 shows the factorisation product of the conditional distribution over variables conditioned on parents π_i for variable X_i . Generative models are used when it is difficult to estimate the probability of output given an input.

2.5.1 Generative adversarial networks

A generative adversarial network is a machine learning framework used to generate and reconstruct data [15]. It implements two machine learning models, a discriminator, D , and a generator, G . The

discriminator uses a supervised model to distinguish real data from generated data. The generator attempts to fool the discriminator by falsely classifying the generated data. The two models are then put against each other in a zero-sum min-max game:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))], \quad (4)$$

where $D(x)$ is the discriminator's estimate of the probability that the data is real, E_x is the expected value of the real data, $G(z)$ is the generated output given noise z , $D(G(z))$ is the discriminator's estimate of the probability that the generated data is real, E_z is the expected value from random inputs to the generator. The models learn from each other and adjust accordingly.

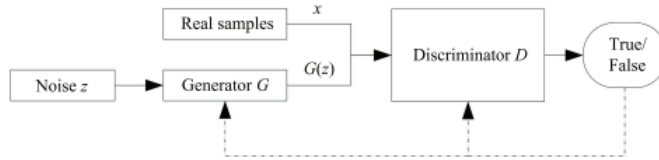


Figure 6: Visualization of the Generative adversarial architecture. Taken from [39].

2.5.2 Autoencoder

An autoencoder is a framework for learning latent-variable models [23]. AE is constructed from two different models, an encoder and a decoder. The encoder creates a latent distribution from the input data. The decoder reconstructs the data utilising the latent distribution. The AE model learns from similarities between real and generated data. Some data are lost because the latent space is in a lower dimension than the input data. The benefit of this approach is the least contributing data gets removed, thus increasing the probability of similarities between the input and the generated data.

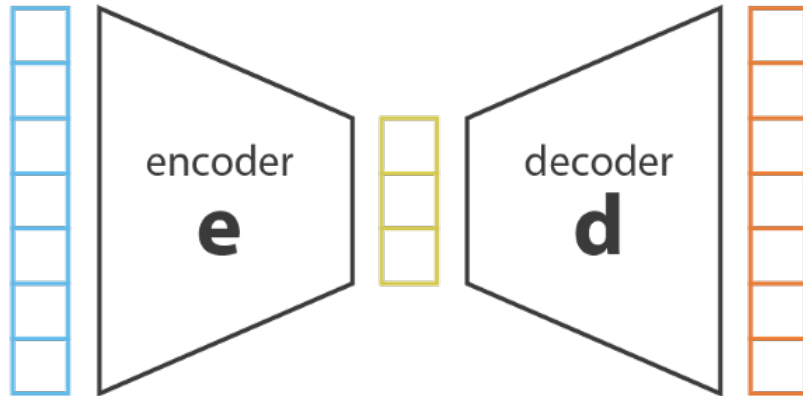


Figure 7: Visualisation of a conventional autoencoder. Taken from [31]

2.5.2.1 Variational Autoencoder

Variational autoencoder is a different implementation of the autoencoder [23]. The variational autoencoder addresses the issue of where the latent space distribution is not continuous. To solve this, it encodes the input into two vectors, one for the means and one for the standard deviation. These two vectors parameterise a continuous probability distribution.

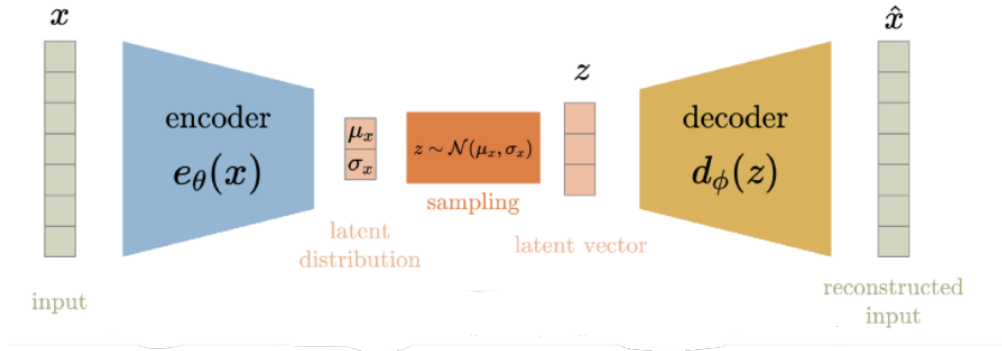


Figure 8: Visualisation of a conventional variational autoencoder. Taken from [31].

2.6 Synthetic Data

Synthetic data are data that are not obtained by measurement [40]. Synthetic data can be anything from randomly generated information to information generated by an algorithm. Synthetically generated data often substitutes real data. Synthetic data needs to resemble real data, which is usually achieved through statistical or machine learning algorithms.

Ideally, generated synthetic data are indistinguishable from real data and therefore come with many advantages. As the synthetic data contains no identifying information, companies are free to use these data without any restrictions [3]. Another advantage of synthetic data is generating specific data for a single purpose and simulating outliers.

2.6.1 Common information model

CIM (common information model) is an open standard that defines managed elements through objects and their relationships [33]. CIM is the industry standard for the International Electrotechnical Commission and is used when defining power grids. It enables multiple parties to exchange information regarding the managed elements. CIM can be implemented in multiple formats, for example, XML or JSON [8].

CIM is represented by triple statements [30]. A triple statement is divided into subject, predicate and object. The subject, or referring node, refers to the object, while the predicate describes how the subject refers. An example would be "Hogne drives a car", where the subject is "Hogne", the predicate is "drives", and the object is "a car".

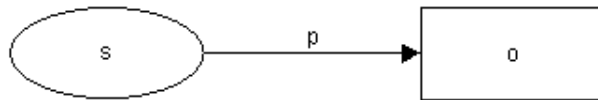


Figure 9: Illustration of a triple. Taken from [30].

CIM enables independent parties to develop software that works on the many implementations without changing or editing complicated operations. This allows multiple parties to exchange information without changing the format of the CIM implementation.

2.7 Related Work

This section contains papers and other research publications that have done similar or related work. The work included ranges from other projects on generating synthetic power grids to approaches

and models in geometric deep learning that is relevant.

2.7.1 Synthetic Power Grids

This section contains other projects on the generation of synthetic power grids.

2.7.1.1 Statistical synthetic grid generation

A case study in Singapore used a statistical approach for synthetic grid generation [37]. The model used a cost minimisation objective function, as power distribution system planning needs to be cost-efficient considering total investments- and operational costs.

The study used publicly available data from Singapore for the synthetic grid generation, as the data contained information about locations, substations, transformer dimensions, electricity pricing etc. This data represents 22kV power grid distributions.

2.7.1.2 Synthetic Power Grids from Real World Models

Synthetic Power Grids from Real World Models [43] is a Department of Energy sponsored article for the generation of realistic publicly available power system models. The article explains the issues relating to accessing real-world grid data and existing synthetic data problems.

The article presents a novel methodology for generating synthetic AC OPF grid data to tackle these issues. The method uses real-world topologies, parameters, and desired complexity of the data. It uses a statistical approach where topologies of real-world data are used to generate synthetic data.

Their proposed method takes in existing anonymised data and fragments it into smaller pieces. By using data from other power grids, these fragments are then reassembled into realistic synthetic power grids.

2.7.1.3 A Learning-Based Method for Generating Synthetic Power Grids

A Learning-Based Method for Generating Synthetic Power Grids [35] describes how one could use imitation learning for generating synthetic power grid data. In the article, the researchers focus on the Western Interconnection power grid, one of the two major power grid interconnections in North America. They evaluate the power grids based on five metrics: path length, clusters, degree distribution, line intersections and length distributions.

By applying these parameters to a NIMBLE model, the researchers presented that the generated power grids have similar topologies and the same robustness as the real data. However, as the parameters used are based on averages, it was concluded in the article that the synthetic data may not reflect the properties of single power grids. They also mention that several properties of power grids such as voltages and frequency were not used and therefore not generated.

2.7.1.4 Deep Generative Graph Distribution Learning for Synthetic Power Grids

Deep Generative Graph Distribution Learning for Synthetic Power Grids [22] describes the use of Deep graph distribution learning for generating synthetic power grids.

The article uses the Columbia University Synthetic Power Grid Dataset, which is a large dataset located in the Western US. It introduces a novel deep learning network to capture node and edge distributions of large, complex power grid networks. By using a deep architecture, DeepGDL uses complex nonlinear manifolds from real-life power grids. The network then generates synthetic power grids imitating the real power grids. DeepGDL shows impressive accuracy in multiple metrics, including node degrees and power flow measurements.

2.7.2 GraphRNN

GraphRNN is an RNN based model for generating graphs [42]. GraphRNN consists of a graph level RNN to generate nodes and an edge level RNN to generate edges. By sequentially generating nodes, then edges, GraphRNN can generate variable-sized graphs. GraphRNN implements a breadth-first search, which reduces the number of edges it needs to look at when updating the model.

2.7.3 MolGAN: An implicit generative model for small molecular graphs

MolGAN is a generative model for small, simple molecular graphs [9]. MolGAN uses a modified GAN model to operate on graph data while also utilising reinforcement learning to increase the model's accuracy further. The model consists of a generator and a discriminator combined with reinforcement learning. The reinforcement learning implementation consists of a domain-specific evaluator receiving feedback from an external source. By using a reward network, MolGAN can generate molecules with desired properties.

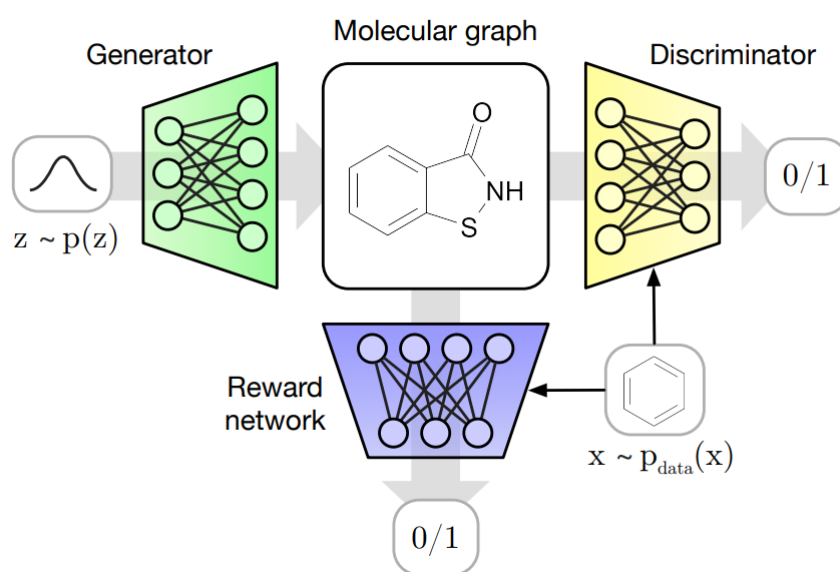


Figure 10: Figure displaying how the MolGan model works [9].

2.7.4 GraphGAN

GraphGAN is an innovative generative model for generating graph embeddings using the GAN architecture [38]. The generator G tries to estimate the connectivity distribution $p_{\text{true}}(v|v_c)$ and selects likely nodes connected with v_c . The discriminator tries to discriminate these connections (v, v_c) outputting a single scalar, describing the probability of the link being realistic.

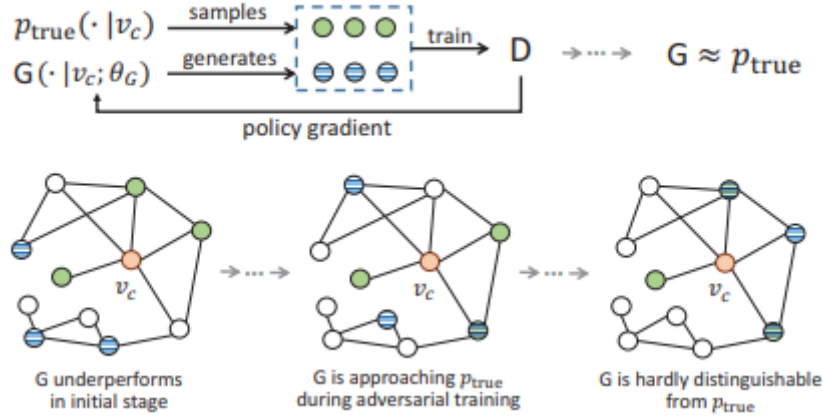


Figure 11: Figure displaying how GraphGAN works [38].

2.7.5 Variational Graph Autoencoder

Variational graph autoencoder is a framework used for unsupervised learning on graph represented data based on the VAE architecture [24]. It was introduced using a GCN encoder and a simple inner product as the decoder. The input is an undirected and unweighted graph. VGAE achieves competitive link prediction performance, and it can be configured with node features to increase the performance.

2.7.6 GraphVAE

GraphVAE is an implementation of the VGAE from Section 2.7.5. It changes the inner product decoder to a simple neural network. This makes the model produce a probabilistic, fully connected graph when decoding [34]. This graph comes in the form of a Bernoulli distribution that can be further decoded into a graph. It is then tested on the QM9 Dataset, where it achieves competitive graph generation results, however, this implementation is GPU computationally heavy and only works efficiently up to 38 nodes.

that

3 Method

This chapter describes our work process and methodology. By using the theory described in Section 2, we have conducted experiments to address the research questions introduced in Section 1.

3.1 Process

We will describe our process, which is divided up into early stages, our scoping, team roles, data analysis, and processing, and then discuss machine learning.

3.1.1 Early stages

The project started with meetings between the bachelor team, the supervisor and Volue. The agenda in those meetings was assignment scoping, scheduling and practical details, including assigning a workspace and ID cards.

After the initial meetings, we began structuring the project workdays and assigning responsibilities between the team based on individual strengths.

3.1.1.1 Assignment Scoping

Scoping the assignment was a large part of the initial meetings, as the assignment Volue presented was relatively high-level and not very concrete. Volue's representatives voiced different expectations about the project's goal. However, when we inquired about assignment constraints, no clear answers were given. With no real constraints, the team attempted to scope the assignment according to Volue's goals but with realistically adjusted steps. These steps included downscaling from generating functioning synthetic power grids to generating detailed synthetic topologies of power grids compatible with Volue's software. However, later on, this scope proved too demanding for the time available to the project. Finally, the project was scoped to: "Generate validated synthetic topologies."

Later in the project, we realised the assignment was scoped too ambitiously and had to adjust accordingly. As mentioned in Section 1, the initial assignment had many challenges to overcome. We were put in contact with machine learning experts at Volue, however, they had no experience surrounding generative graph models and synthetic data generation. The team wanted to define the scope better. To clarify the scope, the team asked Volue to specify what the synthetic power grid data was to be used for. Volue stated that the synthetic data were to be used for testing and improvement of their software, but this changed little in regard to assignment scoping, as it had no impact on how to define the requirements of synthetic power grid data. Finally, the assignment was re-scoped to: "Generate synthetic topologies", as we had no way of evaluating the generated topologies.

3.1.1.2 Team Roles and Work

Information about the team roles and responsibilities between team members.

Kenneth Solvoll was the team leader, as he has experience with team leadership from previous projects. He was responsible for the communication flow between all parties, ensuring that deadlines and goals were met, decision making and oversight of documentation.

Hogne Winther was responsible for all documentation and had the role of a minute taker during meetings, as he has been well structured in previous projects.

Anders Hermanrud was responsible for the architecture in the development process, as he has

shown competence with architecture in previous projects. This included structure, version control, technologies and environments.

3.1.1.3 Project Environment

Most of the bachelor process was done at Volue's offices in Trondheim. Cubicles were provided for each team member. The team were also given a small meeting room for the entire bachelor's duration, which was in use most of the time. The frequent use of the meeting room facilitated the information flow between the members and the representatives from Volue. The meeting room was equipped with two whiteboards and a projector. The whiteboards were used when explaining thought processes and daily stand-up meetings. The projector proved helpful in meetings and for discussing different documents.

3.1.1.4 Work Process

The team adopted methods from the agile software development process to increase our workflow. As seen in the project handbook (see attachments), we started the day at 8 am. These methods involved a morning stand-up meeting every day lasting 15 minutes. These meetings consisted of structuring and reviewing new tasks for the backlog. These stand-up meetings also helped address challenges that arose during the bachelor's process and helped synchronize the team.

3.1.2 Process Execution

To ensure a scientific workflow, we decided to tightly follow the scientific method as shown in Figure 12. As an examination of the scientific method, we are going to describe how we followed this method.

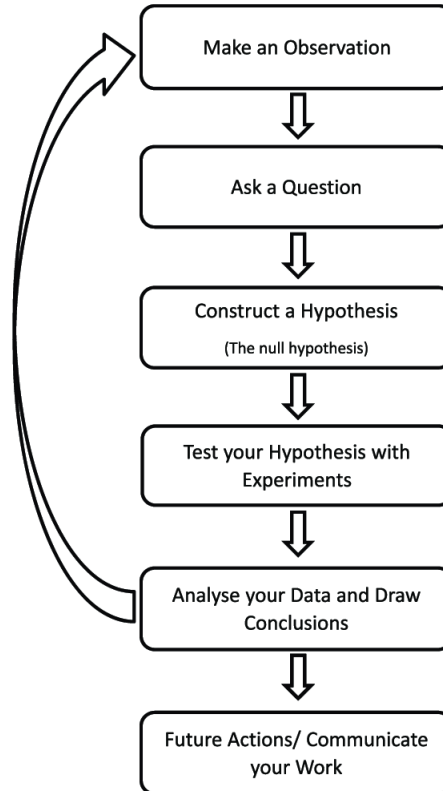


Figure 12: Visualisation of the scientific method [12]

At the initial meeting, we were presented with a visualisation of a power grid. We made the **observation** that the visualisation had graph similarities. From this we asked the **question** "*Can Power Grid Data be represented as Graphs?*". We **hypothesised** this to be the case and began conducting **experiments** to test this hypothesis. In Section 3.1.3 we **analysed** the results of our experiments and came to the **conclusion** that the hypothesis was correct.

3.1.3 Research

This section describes the entire research process from the beginning to the end. We will take a closer look at related work by others and important questions like "What is synthetic data?", to better understand the assignment and its challenges.

Due to the theoretical nature of the project and no prior knowledge among team members regarding synthetic data, the team had to do thorough research. This was done by gathering relevant articles about similar projects and data representations. The most notable examples here are "Statistical synthetic grid generation" Section 2.7.1.1 and "Synthetic Power Grids from Real World Models" in Section 2.7.1.2, as they resemble our project the most.

Each of the articles described in Section 2.7.1, explains how they managed to generate synthetic power grids. The proposed solutions in the articles are split between two approaches, statistical and machine learning. The statistical approach predicts the probability of a link between two components. For instance, a cost-evaluation algorithm predicts the probability of a connection between components, based on real-life costs for the installation of the aforementioned connection (see Section 2.7.1.1). The machine learning approach uses developed models to generate synthetic power grid data by inserting power grid metrics as model parameters. For instance: path length, clusters, degree distribution, line intersections, and length distributions (see Section 2.7.1.3).

The team had to define synthetic data in terms of the project. Synthetic data was defined as generated power grid topologies that mimic the real power grids. In addition, the team needed to figure out a good way of representing the data in a machine learning-friendly format.

After visualising Volue's data in NETBAS (seen in Figure 13), a grid management software product offered by Volue (described in Section 3.1.4.2), the team discovered that the power grid data and its topology displayed graph-like structures. Each component could be represented as a node with the connected cables as edges. By representing Volue's power grid data as graphs, the team could utilise Graph neural networks (see Section 2.4) models for training.

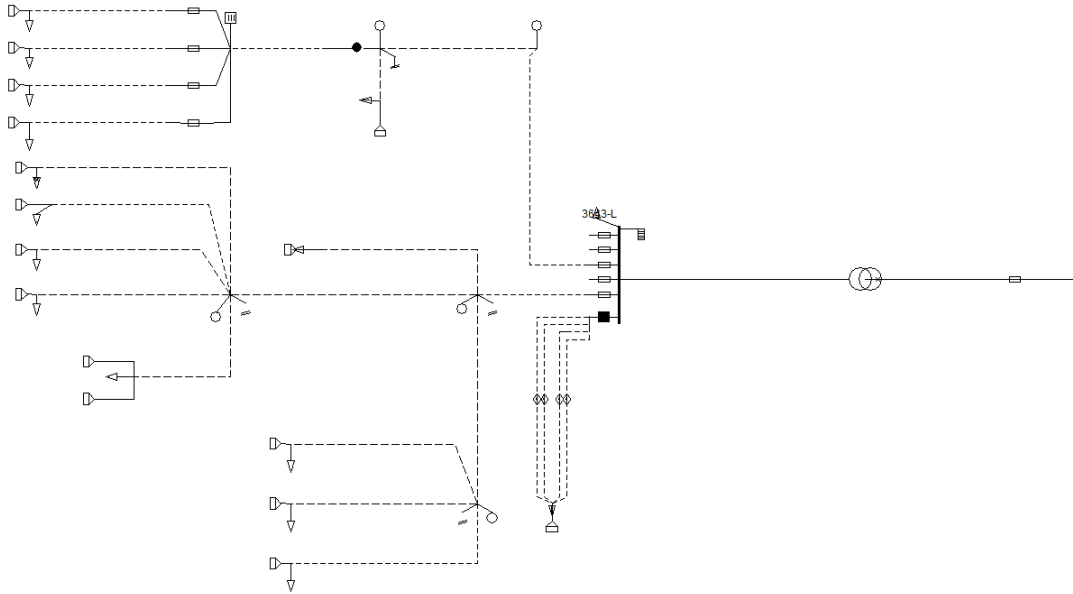


Figure 13: NETBAS representation of grid topology.

3.1.4 Data

This section describes the given dataset, its contents, and how it was formatted, analysed and processed.

3.1.4.1 Power Grid Data

In this project, we define a Power grid as a structured network supplying electricity from a source to consumers. Power grid data represents power grid networks in a data format. Objects represent these data. These objects are divided into different object types. The object's type defines which values an object can contain. For instance, the Fuse object type is defined by three different values: name, description and a boolean value describing whether it is open. Whereas another object type, the Terminal, has a single value describing its sequence number. These values then define the object's properties, such as the Fuse's boolean value. Values can be added retrospectively to an already created object, which enables power grid data to be updated as needed. Values can also be added to objects defining special properties. For instance, the solar panel property is represented as a single boolean value in another object. As a consequence, objects of the same type do not always have the same number of values.

Each object can be connected to other objects. Which connections an object has affects the object's properties. Some objects can only be connected to predetermined object types. For instance, the Terminal object is always connected to a ConnectivityNode object. However, a ConnectivityNode object can be connected to various other object types. The combination of different objects, object types and their connections define the power grid's properties, and thus the type of power grid these data represent. There exist multiple power grid types, and they can be Industrial areas, local neighbourhoods and single hospitals, each with its own defining structure.

3.1.4.2 NETBAS

Volue develops and delivers their own system software, NETBAS, for grid management. There exist over 1650 different object types in NETBAS [28]. As it is developed and maintained by Volue, new objects used in power grid structures must be implemented. These single objects are then

represented by predetermined objects and connections between them.

NETBAS exports the power grid data into a CIM formatted XML file. When exporting data, extra objects are added to represent certain values. This is done as different objects might have the same value. Both the Fuse- and Busbar objects have voltage values. If these values are identical, both objects would have a connection to the same Basevoltage object, where Basevoltage only contains a single voltage value.

3.1.4.3 Data Analysis

The dataset was provided by Volue. The data represents the structure of low voltage power grids in northern Norway. The team did data analysis to understand how to approach the data. This section describes how the data was defined and analysed.

Initial file summary				
File number	Number of Objects	Number of Edges	Number of Object types	Unique objects
1	258	324	34	SwitchInfo, Disconnecter
2	294	405	32	None
3	2091	3166	34	LoadBreakSwitch, SwitchInfo
4	2990	4403	34	LoadBreakSwitch, SwitchInfo
5	1710	2652	38	Breaker, LoadBreakSwitch, CurrentTransformerPhase, CurrentTransformer, CurrentTransformerInfo, SwitchInfo
6	1960	2972	35	LoadBreakSwitch, SwitchInfo, Disconnecter
7	912	1361	34	LoadBreakSwitch, SwitchInfo

Table 1: Information of the first seven data files given. The table shows how various the data was. To get unique objects, each file was compared with file number two, as file number two has the least number of object types. File number four was the largest file, with 2990 objects and 4403 connections.

The dataset was made up of 29 files, where each file represented a single power grid. The files were given to the team in multiple batches throughout the project. The team was initially given seven files at the start of the project, shown in Table 1. The remaining files were given in two more batches, with the last batch given towards the end of the project. The data did not contain information about the type of grid each file represented. However, Volue informed us all grids from the files all came from the same small town. This meant some files represented local neighbourhoods and others industrial areas. This inconsistency is reflected in the dataset, as all data files contain a different number of objects, edges and object types (See Table 1).

There were 38 different object types in the dataset, shown in Figure 14. However, multiple object types were not present in all files, such as the Jumper object. We learned from Volue that there are no prerequisites for using the Jumper object in any power grid. Therefore, using the Jumper object is up to the companies designing the power grids, which creates inherent randomness in power grid data.

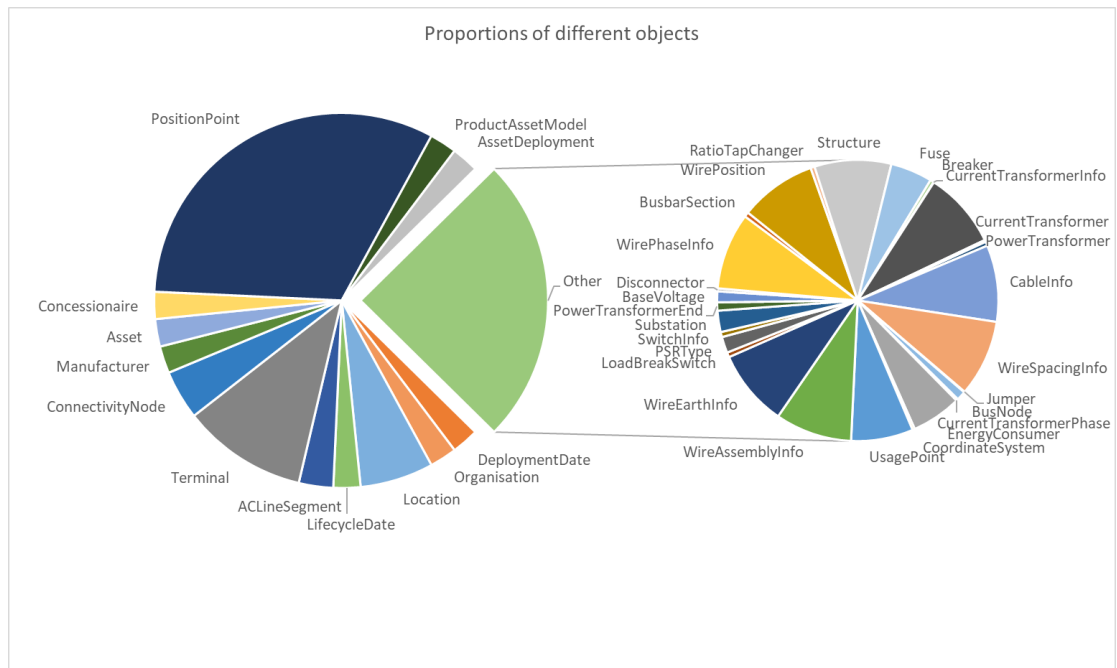


Figure 14: Illustration of a single file showing the different object types and their proportional numbers. From the figure, we see the object type with most objects is PositionPoint. The illustration also shows many object types with few instances in the dataset.

To start analysing the data, a single file was visualised in Python, as shown in Figure 15. To understand which data were important for the power grid structure, all files were distributed among the team members and analysed. We wanted, with the help of Volue, to define the least amount of object types and information needed to reconstruct the real-world power grid. We realised a large amount of data were unnecessary for this goal. This data did not affect the structure or properties of the power grid data. We had discussions with Volue to determine which data were unnecessary. We wanted, with the help of Volue, to define the least amount of object types and information needed to reconstruct

Relevant objects in the dataset			
Object	Number of edges	Number of Values	Removed
Terminal	3	1	No
PositionPoint	2	4	Yes
Fuse	6	3	No
Manufacturer	1	1	Yes
LifecycleDate	1	2	Yes
UsagePoint	4	2	No

Table 2: Table showing example object types in the dataset. The Object type Manufacturer describes the connected objects' manufacturer. The Lifecycle object type describes the connected objects production date and installment date. The PositionPoint object type describes a connected objects coordinates and has a connection to a location describing the address. The Fuse object is a physical object connected to six other physical objects power grid objects, with three values: Description, name and switch boolean value. The Terminal object is also a physical object connection to three other physical objects, with one value describing a sequence number. The UsagePoint is a special object that represent both a virtual or physical meter, its values are a name and an alias. As the table shows we removed the non physical objects as they have no influence on the power grid structure.

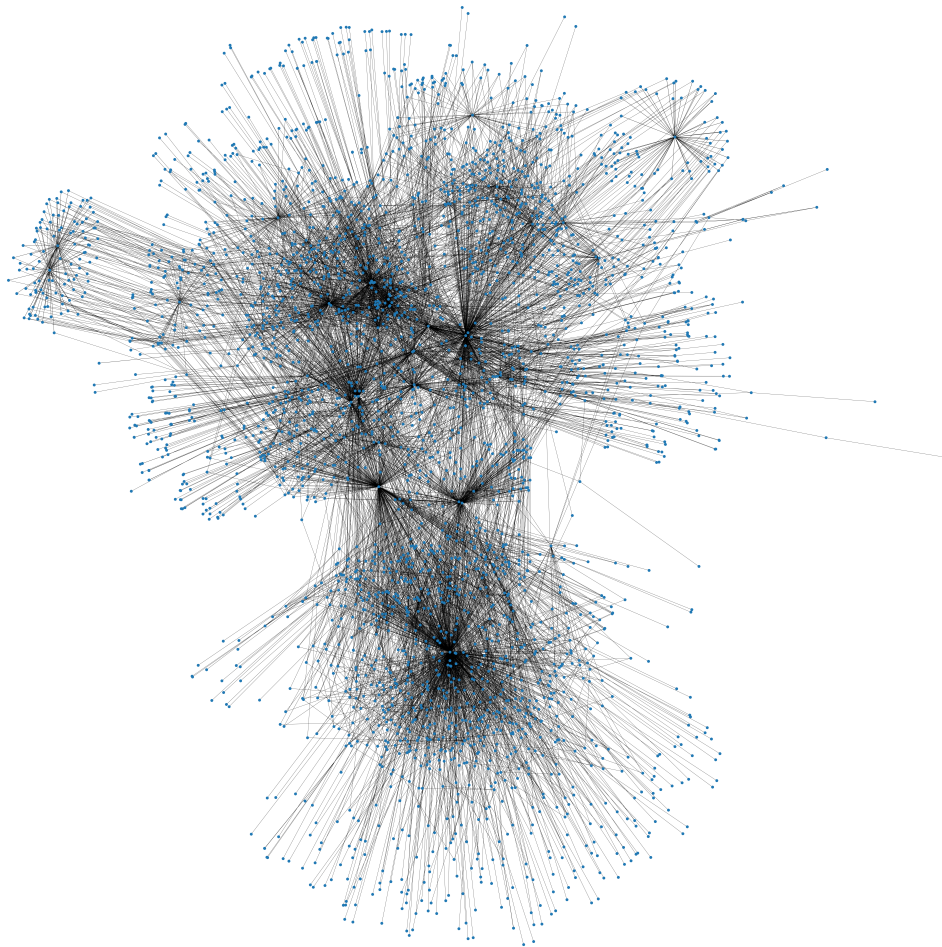


Figure 15: A graph visualisation of a single raw data file. The figure shows a large number of leaf nodes. Most of these nodes are objects, such as Manufacturer or PositionPoint.

These data included objects and object values NETBAS adds when importing or exporting the grid. For instance, the information of who converted the grid to NETBAS and when the object were converted. Twenty object types were unnecessary, such as the LifeCycleDate object seen in Table 2.

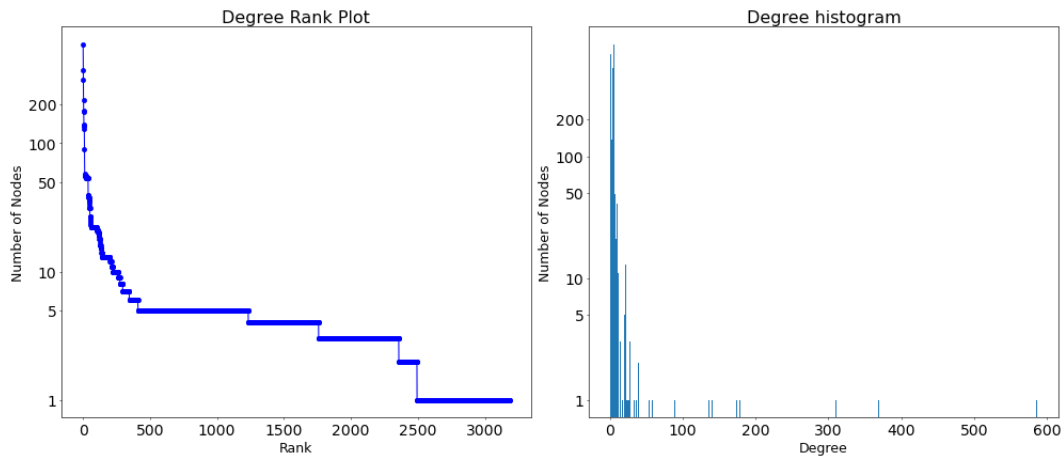


Figure 16: Illustration showing the different number of edges each node has in a single data file. From the figure, we can see there are a few nodes with a large number of edges, where the maximum number is 600. Most nodes have five or fewer edges.

3.1.4.4 Data Processing

The files were formatted as triples in the CIM format, as shown in Figure 17. The figure shows the objects are not separated. The files were converted to the TTL format as shown in Figure 18, which clearly separates each object while removing unnecessary syntax.

```
<cim:PowerTransformerEnd rdf:about="urn:uuid:1">
  <cim:ConductingEquipment.BaseVoltage rdf:resource="urn:uuid:2" />
  <cim:PowerTransformerEnd.PowerTransformer rdf:resource="urn:uuid:3" />
  <cim:TransformerEnd.Terminal rdf:resource="urn:uuid:4" />
</cim:PowerTransformerEnd>
<cim:Terminal rdf:about="urn:uuid:4">
  <cim:Terminal.ConductingEquipment rdf:resource="urn:uuid:5" />
  <cim:Terminal.ConnectivityNode rdf:resource="urn:uuid:6" />
  <cim:Terminal.sequenceNumber>5</cim:Terminal.sequenceNumber>
</cim:Terminal>
<cim:PowerTransformer rdf:about="urn:uuid:3">
  <cim:PowerSystemResource.PSRType rdf:resource="urn:uuid:7" />
  <cim:ConductingEquipment.BaseVoltage rdf:resource="urn:uuid:8" />
  <cim:Equipment.EquipmentContainer rdf:resource="urn:uuid:9" />
  <cim:PowerSystemResource.AssetDatasheet rdf:resource="urn:uuid:10" />
  <cim:PowerSystemResource.Concessionaire rdf:resource="urn:uuid:11" />
  <cim:IdentifiedObject.name>EXAMPLENAME</cim:IdentifiedObject.name>
  <cim:PowerTransformer.vector>EXAMPLEGROUP</cim:PowerTransformer.vector>
</cim:PowerTransformer>
```

Figure 17: Above is an example of how the raw CIM files are formatted. The data have been anonymised at Volue's request.

```

<urn:uuid:1> a cim:PowerTransformerEnd ;
  cim:ConductingEquipment.BaseVoltage <urn:uuid:2> ;
  cim:PowerTransformerEnd.PowerTransformer <urn:uuid:3> ;
  cim:TransformerEnd.Terminal <urn:uuid:4> .

<urn:uuid:4> a cim:ProductAssetModel ;
  cim:Terminal.ConductingEquipment <urn:uuid:5> ;
  cim:Terminal.ConnectivityNode <urn:uuid:6> ;
  cim:Terminal.sequenceNumber "5" .

<urn:uuid:3> a cim:PowerTransformer ;
  cim:PowerSystemResource.PSRType <urn:uuid:7> ;
  cim:ConductingEquipment.BaseVoltage <urn:uuid:8> ;
  cim:Equipment.EquipmentContainer <urn:uuid:9> ;
  cim:PowerSystemResource.AssetDatasheet <urn:uuid:10> ;
  cim:PowerSystemResource.Concessionaire <urn:uuid:11> ;
  cim:IdentifiedObject.name "EXAMPLENAME" ;
  cim:PowerTransformer.vector "EXAMPLEGROUP" .

```

Figure 18: Above is an example of how a file was converted to the TTL format. The figure clearly shows a more defined grouping and separation of each object. The TTL format also removes unnecessary syntax such as "rdf:resource" and "rdf:about". The data is the same as shown in Figure 17.

The files were parsed, and all triples were mapped (see Section 2.6.1). This mapping simplified removing the unnecessary power grid data. If a power grid object was unnecessary, all triples containing this object were removed from the files. This approach ensured no dangling pointers were created during processing. Dangling pointers are references to non-existing nodes, thus invalidating the graph [13]. The files were reconstructed with only confirmed triples, which ensures any dangling pointers present in the original files were removed.

To transform the data into a suitable format for machine learning, we had to find a format to keep the network structure intact while retaining object information. As machine learning frameworks prefer to use sequential data [11], we decided to use an edge list format. This format retains the network structure similarly to how triples work in CIM. An edge list represents the edges of the network by referencing the connecting nodes, as shown in Figure 19.

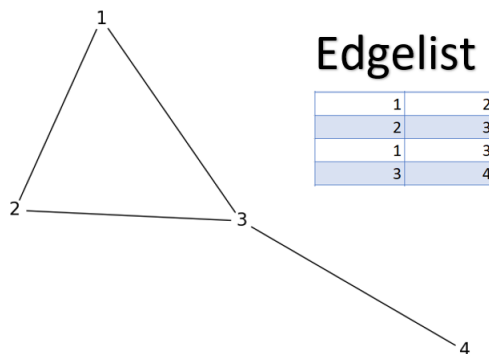


Figure 19: Visualisation of how a graph can be represented as an edge list.

To represent the relations of different nodes, we used node embeddings. These embeddings were generated with the Node2Vec algorithm. The embeddings were added as node features to each node in the edge list. This way, equal object types were represented differently based on the node's edges and its position in the graph. The team looked at including all the different node values as node features. However, this would change the graph type from a homogeneous to a heterogeneous graph. There were few frameworks and established ways of generating structured heterogeneous graphs. We did not explore this approach, but it can be promising [2].

Finally, node types were added to the node features. The combined node embedding was a good choice to represent each of the nodes' connections and properties.

Parameters for Node2Vec algorithm	
Dimensions	21
Walk length	16
Number of walks	10

Table 3: Table showing parameters used for Node2Vec algorithm. Dimensions are the embeddings dimensions used. Walk length define the number of nodes in each walk. Number of walks is how any walks per node.

3.1.5 Machine Learning

This section describes the process of finding machine learning models for synthetic graph generation. Furthermore, we will explain why we chose those models and our experiment setup.

3.1.5.1 Choice of Machine learning method

As explained in Section 3.1.3, a good representation of our data is graphs. However, conventional machine learning models train on the Euclidean space. This section describes our process of finding the best type of method to train on our dataset.

One approach was to convert the graphs into adjacency matrices and use the matrices as input for a tailored machine learning model. Conventional models can train on adjacency matrices, however, with reduced accuracy [19]. As our dataset has 60 to 852 nodes, the adjacency matrices would differ much in size. Since the matrices need to be the same size when training, they must be padded with zero vectors. The large difference in node count would lead to a small adjacency matrix containing 99.5% padded values. Furthermore, this would lead to models guessing correctly on up to 99.5% of values in these matrices, but only correctly guessing the padded zero vectors. Additionally, we wanted to generate data of different sizes, but we could not find any relevant literature on how to achieve this. Therefore, we chose not to use adjacency matrices in favour of trying other graph-based methods.

Since our data came in the form of graphs, one option was to train on the data directly without converting it to another format. Geometric deep learning (see Section 2.4), has had much development in the last few years [44], with multiple publications and a dedicated framework [29]. Geometric deep learning models could handle our dataset's variation in node count, edge count and node features. With this flexibility in mind, we decided to use geometric deep learning for training on our dataset.

3.1.5.2 Choice of graph learning model

We needed to pick the most tailored model to implement with respect to our assignment. This section will describe the most relevant models we reviewed and explain the ones we chose and what they can generate. For graph-based machine learning, we looked at three main methods for generating links, nodes or both: Graph Recurrent neural networks (GraphRNN), Graph Generative Adversarial Networks (GGAN), and Graph Autoencoders (GAE). Also, Not all types of models are capable of generating an entire graph but generate only edges, and therefore we also had to look at ways to generate nodes.

GraphRNN offers precise link prediction and node generation results on structured graphs [42], as explained in Section 2.7.2. Since GraphRNN generates graphs node-by-node, it can easily generate graphs of different node counts. Many other graph-based models are limited to learning from a single graph [23, 34], while GraphRNN can learn from the entire dataset, including distinct sized graphs. GraphRNN performs well on data with noise and has been used on datasets with up to 2025

nodes. GraphRNN cannot use node features, and in our dataset each node's edges affect the nodes it's connected to, therefore making features important, as shown in Section 3.1.4.3. It is possible to adapt GraphRNN to work with node features, but it is not well tested. Because we wanted to implement other models, this one was not, but is a good candidate for future development.

GGAN can perform link prediction and learns from a single graph [38]. It can also handle the noise in real-world data. GGAN only needs a generator and discriminator, but when performing graph generation, it needs an evaluator fitting the target domain [27]. This way, it can also learn from multiple graphs instead of one and generate with different node counts. To make GGAN work, we would have to make a domain-specific evaluator. After discussing with Volue, they stated it would take a minimum of three months to implement such an evaluator, placing it outside the scope of this project. An example of a GGAN implementation with similar properties as we want is MolGAN, explained in Section 2.7.3. The problem was the difference in our datasets. First, MolGAN cannot use node features. Secondly, MolGAN has a node count ranging from 1 to 9 and a maximum of 5 node types [9], while we have from 60 to 852 nodes and 18 different node types, suggesting this model may be designed for smaller graphs. We would have implemented this method had it not been for the evaluator.

GAE can do link prediction and learns from a single graph [24], the method is described in Section 2.7.5. GAE incorporating node features is an advantage. GAE is easy to implement and is fast to train, it is good for testing on the dataset. The idea was to use this model to identify unnecessary nodes and features. However, the best possible results are only variations of the graph it is trained on [18] and not new instances of data, posing a weakness in the model. We chose to implement this model.

VGAE is a variation of GAE with most of the same properties [18]. VGAE learns from a single graph, can use node features, and performs link prediction. A VGAE model learns the underlying structure and creates a distribution where new instances of data can be drawn [24]. By learning the underlying structure, the model becomes less affected by noise. Compared to the GAE, it performs better on structured data [24]. For these reasons, we wanted to implement VGAE. A relevant example of the use of VGAE to do graph generation [34], as described in Section 2.7.6. By using a neural net as the decoder, it has been used to generate graphs of up to 40 nodes. It outputs nodes, node features and edges for a graph. It serves as a proof of concept for full graph generation with VGAE, even if the generated graphs are too small for our purpose.

GAE and VGAE cannot generate new nodes. Therefore we used two other machine learning models to generate node distributions for our data. Here we chose both an AE and a GAN model to test different generative approaches. This was not graph-based but done in tabular form. We implemented CTGAN and TVAE [41].

3.1.5.3 Experiment Setup

Our experiment setup consists of two parts, each using a model. The first part is generating node distributions. The second part is training a link prediction model and using this model on the generated node distribution. The papers [24, 34] used BCEloss for their link prediction models. The BCEloss function gave accurate results, so we decided to use this loss function for our link prediction models.

- **Node Generation**

Setup: To train the Node Generation model, node distribution of the graphs was used. To find the node distribution of each graph, the processed data files were converted into tabular data where the columns were the different object types and the rows were each graph. The values of each column were the number of nodes of that object type in each graph. The team chose to implement two different types of Deep generative models for this task a GAN and VAE model. This was done to see the strength of each model and find which fit our project best.

CTGAN: The first node generation model implemented was a CTGAN model. It is implemented as described in [41].

TVAE: The second node generation model implemented was a TVAE model. It is implemented as described in [41].

- **Link Prediction**

GAE: The first link prediction model we implemented was a Graphical Auto Encoder using Torch Geometric. This model is implemented as described in the paper "Variational Graph Auto-Encoders" [24], see Section 2.7.5. The node features used were node type.

VGAE: The second link prediction model was a Variable Graphical Autoencoder implemented using Torch Geometric as described in "Variational Graph Auto-Encoders" [24], see Section 2.7.5. The node features used were node type.

3.1.5.4 Hyperparameters

The hyperparameters used were chosen after testing different values. To tune the hyperparameters, we tested on the dataset and recorded performance on different values. At last, the best performing values were selected.

Parameter Configuration	
Epochs	Number of times a model was trained on the training data
Optimizer	The algorithm that modifies the model inner parameters, such as weights
Loss function	Is evaluating how well the model predicts
Learnin Rate	Determines the step size when adjusting the weights
Weight Decay	Adds a cost function to the loss function, therefore shrinking weights to prevent overfitting
Embedding Size	Controls the dimensions of the encoded graph, higher means that the models can store more information
Decoder	The decoder controls how the graph embedding is reconstructed back into a graph.

Table 4: Explanation for the different hyperparameters we use. The decoder is unique for the GAE and VGAE model

3.1.5.5 Evaluation of models

We use a statistical approach to evaluate the node distribution model. There are three metrics that we use, a simple distribution probability, a Cumulative sum probability and object correlations. We compare the generated data with the real data.

To evaluate the link prediction models, we implemented a confusion matrix. The confusion matrix measures ground truth, called true positives, which is all links generated that are the same as in the original, divided by the number of links in the original. False positives are the number of links the model guessed that are not in the original, divided by the number of links in the generated graph. False negatives are the inverse of true positives. True negatives have been omitted, since the number of possible links in a graph is so high, making this too computational costly with the size of our graph.

By visualizing both the original and the generated graphs, we can visually see major differences. The relevant thing here is to see the difference in structure in the generated graphs. Instances of structures to look at are the number of nodes, number of connections per node and number of leaf nodes. We talked with Volue, and until they have implemented an evaluator, visually evaluating is sufficient.

3.2 Choice of technologies

This section provides an overview of the technologies used to carry out the experiment as described in Section 3.1.5.3. The experiment was written in python 3.9.

3.2.1 PyTorch

Pytorch¹ is an open-source framework for machine learning based on Torch. PyTorch is designed to be as close to Numpy as possible creating smooth interaction between the two frameworks. PyTorch comes with a vast amount of features such as libraries and tools. PyTorch was utilized over other machine learning frameworks as we have experience using the framework. PyTorch offers CUDA support making it able to allocate memory to both GPU and CPU dynamically. This proved useful as a GPU was used by us during the project.

3.2.1.1 PyTorch Geometric

PyTorch offers a library called Pytorch Geometric². PyTorch Geometric is a deep learning framework developed for Geometric Deep Learning. It is well documented and includes examples for most tasks. It implements the most common Geometric deep learning models and supports data containers compatible with these models. It has support for directed, undirected, homogeneous and heterogeneous graphs. We used PyTorch Geometric to build our models.

3.2.2 TensorBoard

TensorBoard³ is a visualisation library originally developed for Tensorflow, however it can also be used by PyTorch. Tensorboard provides the framework for visualizing different machine learning metrics such as accuracy and loss. It automatically saves this data, thus simplifying comparing models. Visualizations from TensorBoard gave the team overview of which models performed best and their hyperparameters.

3.2.3 Pandas

Pandas⁴ is a powerful open-source data analysis tool. Pandas simplified the processing of the dataset for our purpose. Pandas help represent tables and provide tools to analyze and process these tables. Pandas was used to represent our edge list and features.

3.2.4 Numpy

Numpy⁵ is a powerful python module for linear algebra operations. It can represent lists, matrices, and tensors, and perform operations on these. Numpy proved helpful in matrix multiplication used widely throughout the project and simple mathematical equations, our link prediction decoder was implemented using Numpy's matrix multiplication.

3.2.5 RDFlib

RDFLib⁶ is a python module developed for working with the RDF format. RDFLib provides a simple interface to parse our raw CIM files and convert them into graphs more suitable for machine learning. It also supports removing triples, which was extensively used to remove unwanted data.

¹PyTorch - <https://pytorch.org>

²PyTorch Geometric - https://github.com/pyg-team/pytorch_geometric

³TensorBoard - <https://www.tensorflow.org/tensorboard>

⁴Pandas - <https://pandas.pydata.org/>

⁵Numpy - <https://numpy.org/>

⁶RDFLib - <https://rdflib.readthedocs.io/en/stable/>

3.2.6 Table Evaluator

Table Evaluator⁷ is a python module used to evaluate synthetic tabular data. It contains a large number of statistical methods and visualises these in a clear and simple manner. It was used when comparing the generated node distribution with the real data.

3.2.7 SDV

Synthetic data vault⁸ is a machine learning module that implements different data generation models. The module was used for its intuitive and vast documentation. It was used for the generative node distribution models.

3.2.8 Matplotlib

Matplotlib⁹ is an open-source python module used for visualising data. Matplotlib was used extensively by both the team and imported modules to create graphs in this report.

3.2.9 Jupyter Notebook

Jupyter Notebook¹⁰ is open-source software that combines source code with markdown into a single document. Jupyter notebook is divided into individual cells where source code can be written. It saves each cell state enabling us to run single lines of code instead of whole programs. This reduced the time running code as unedited expensive code segments only needed to be run once.

3.2.10 NetworkX

NetworkX¹¹ is an open-source python module that creates and manipulates complex networks. This module was used to convert our data to a graph representation and visualise completed graphs. Networkx was also used to analyze the data by plotting node types and info onto the visualized graphs.

3.2.11 IntelliJ IDEA

IntelliJ IDEA¹² is an IDE developed by JetBrains used for software development. We were given IntelliJ Ultimate through NTNU and it simplified the structuring of our code and files. The IDE supports a large number of file extensions enabling us to use a single application for most files. It also supports extensions such as Git and Jupyter Notebook which increased our productivity.

3.2.12 Git

Git¹³ is a free and open-source version control software. Git enabled each of us to track changes done, revert changes and restore files. Through GitHub, we saved our work in the cloud, which simplified distributing and cooperating code written by each team member.

⁷Table Evaluator - <https://github.com/Baukebrenninkmeijer/Table-Evaluator>

⁸SDV - <https://sdv.dev/>

⁹Matplotlib - <https://matplotlib.org/>

¹⁰Jupyter Notebook - <https://jupyter.org/>

¹¹NetworkX - <https://networkx.org/>

¹²IntelliJ - <https://www.jetbrains.com/idea/>

¹³Git - <https://git-scm.com/>

3.2.13 CUDA supported GPU

Due to the sensitive nature of the data Volue provided each member with a laptop. These laptops were outfitted with an Nvidia Quadro T2000. This is a CUDA supported GPU and was used in together with PyTorch. By parallelising instructions to the GPU, we were able to reduce the time training the different models.

4 Results

This chapter presents a select set of results in line with the research questions introduced in Section 1. The results are based on the theory from Section 2 and our methodology from Section 3.

4.1 Data Processing

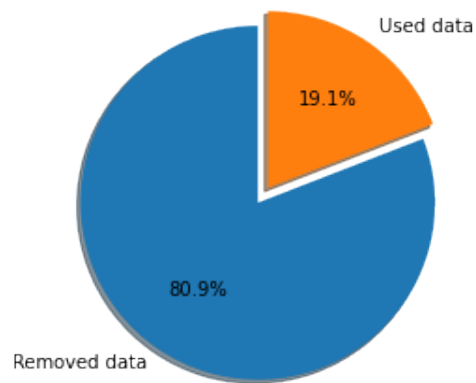


Figure 20: Pie graph showing portion of data removed from the original files.

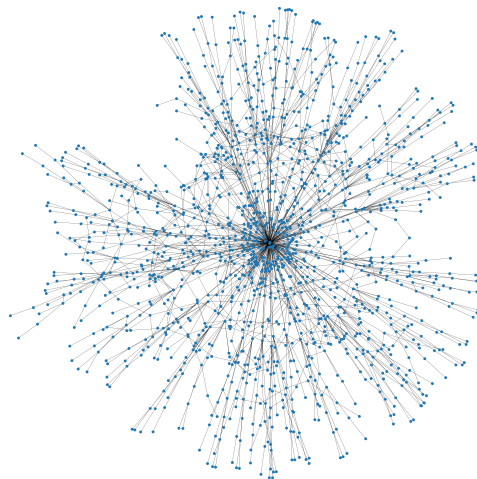


Figure 21: This is a new visualisation of Figure 15 after it has been processed. The figure shows a large reduction in the number of nodes and edges, however, it also shows that some nodes still have a large number of edges.

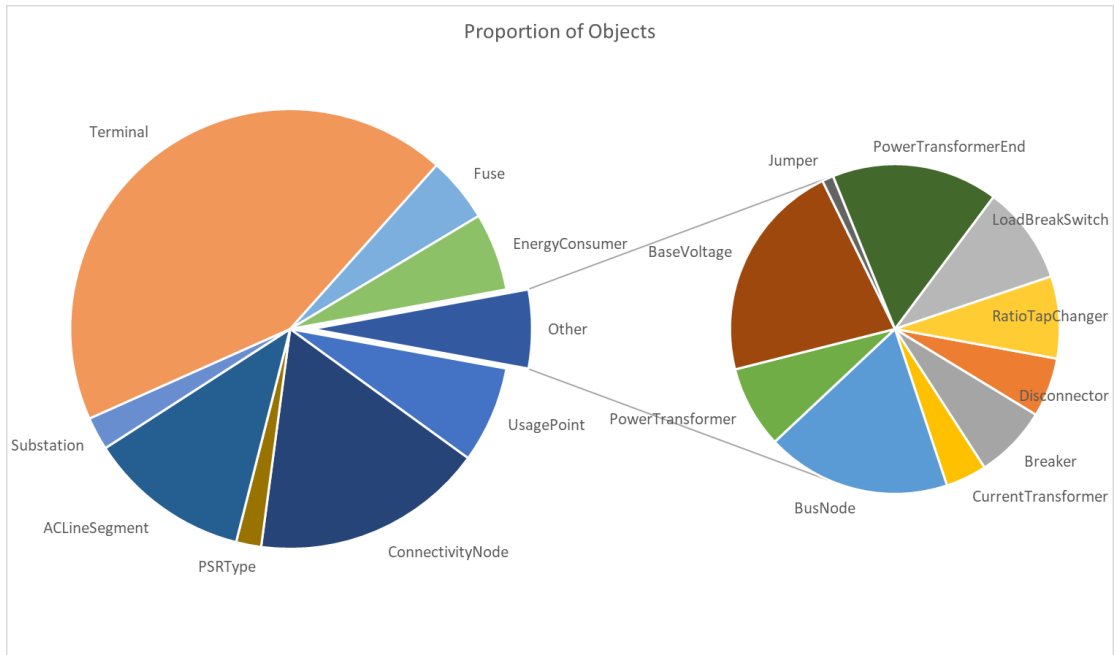


Figure 22: Illustrating the proportion of object types in the same file shown in Figure 14 after processing. By comparing the two graphs, we see the object types removed consisted of most of the objects. The graph also shows the most numerous object type that went from PositionPoint type to Terminal.

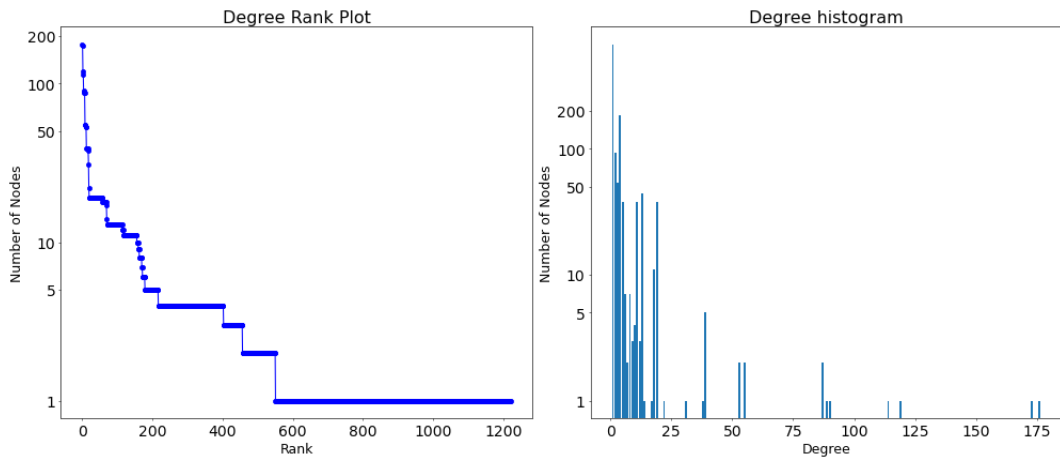


Figure 23: Illustration of the different degrees each node has in the same file as Figure 16. The illustration shows a large number of edges have been removed, however, there are still some nodes that have a significant number of edges. The maximum number of edges a single node has is 175.

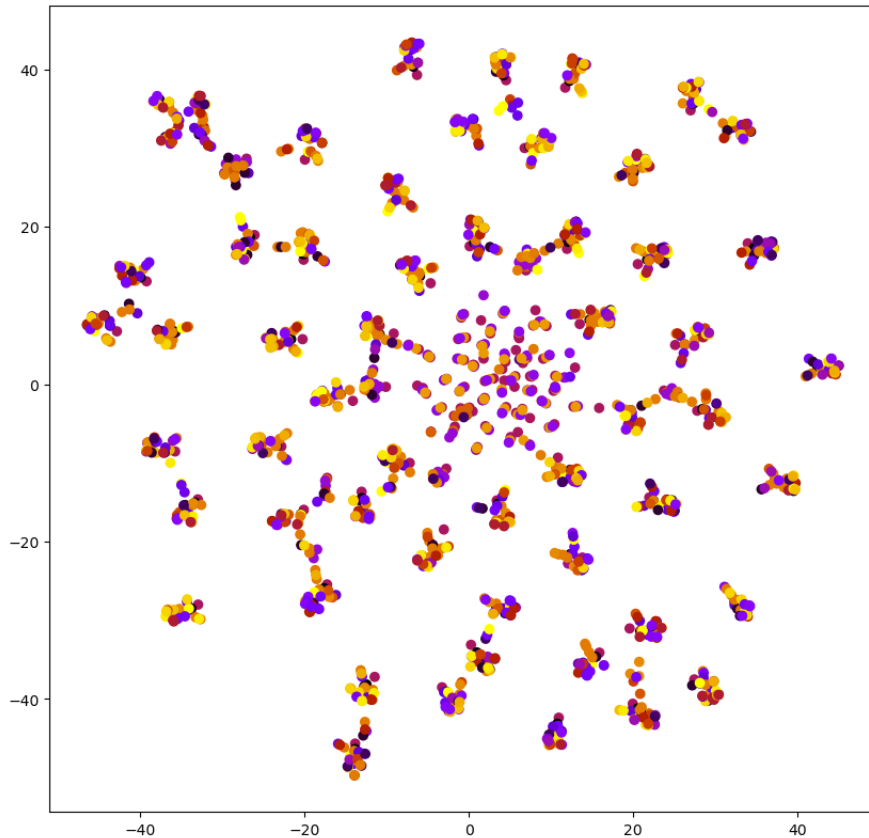


Figure 24: Graph illustrating node distribution in a two-dimensional space. This was created using the Node2Vec algorithm described in Section 3.1.4.4 with identical parameters. The dimensions were then reduced by using T-SNE to a two-dimensional space. The figure shows clustering tendencies in the data, which was important for the team as it shows a structure in the data. The clusters suggested the possibility for machine learning networks to learn the structures of power grid data. The team analysed the clusters more closely, however, it is unclear what makes the clusters. First, we thought the clustering was distributed based on the connecting node. However, after analysing the connected nodes, there was no indication of this. Thus the team believes the clusters are small subgraphs of a larger graph, with a coherent structure between closely connected nodes as described in Section 3.1.4.3.

4.2 Node Generation

This section describes the results obtained when running the node-link generation experiments. Our experiments consisted of the following models:

- CTGAN
- TVAE

All of the following models were trained with the same hyperparameters.

Hyperparameters for CTGAN model	
Embedding Size	128
Loss	Cross entropy, Tanh
Generator model	
Learnin Rate	2e-4
Weight Decay	1e-6
Dimension Size	256, 256
Optimizer	Adam
Discriminator model	
Learning Rate	2e-4
Weight Decay	1e-6
Dimension Size	256, 256
Optimizer	Adam

Table 5: Configuration used when training the CTGAN model. The parameters are explained in Table 7

4.2.1 CTGAN

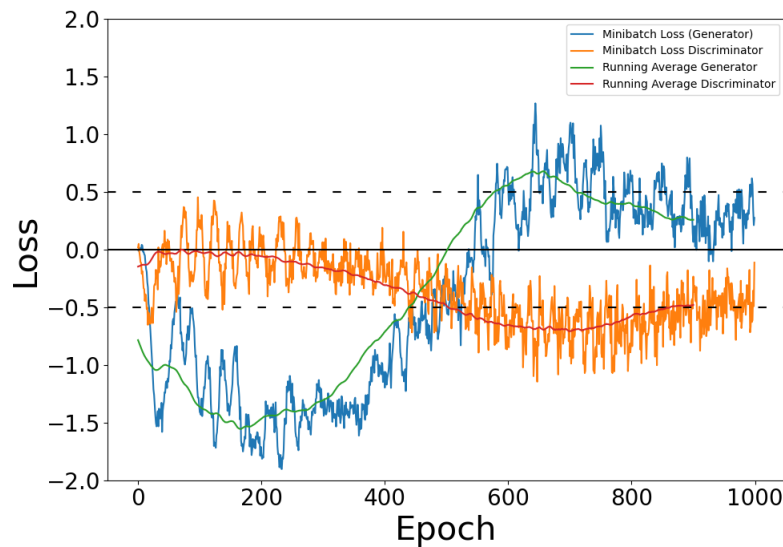


Figure 25: Graph illustrating the loss when training the Conditional tabular GAN. The figure shows the converging of the loss for both models. The Discriminator converges to -0.5 while the Generator converges to 0.5. This converging in both the discriminator and generator indicates the discriminator struggles to distinguish between the generated and real node distribution. This means the generator generates synthetic data resembling the real data.

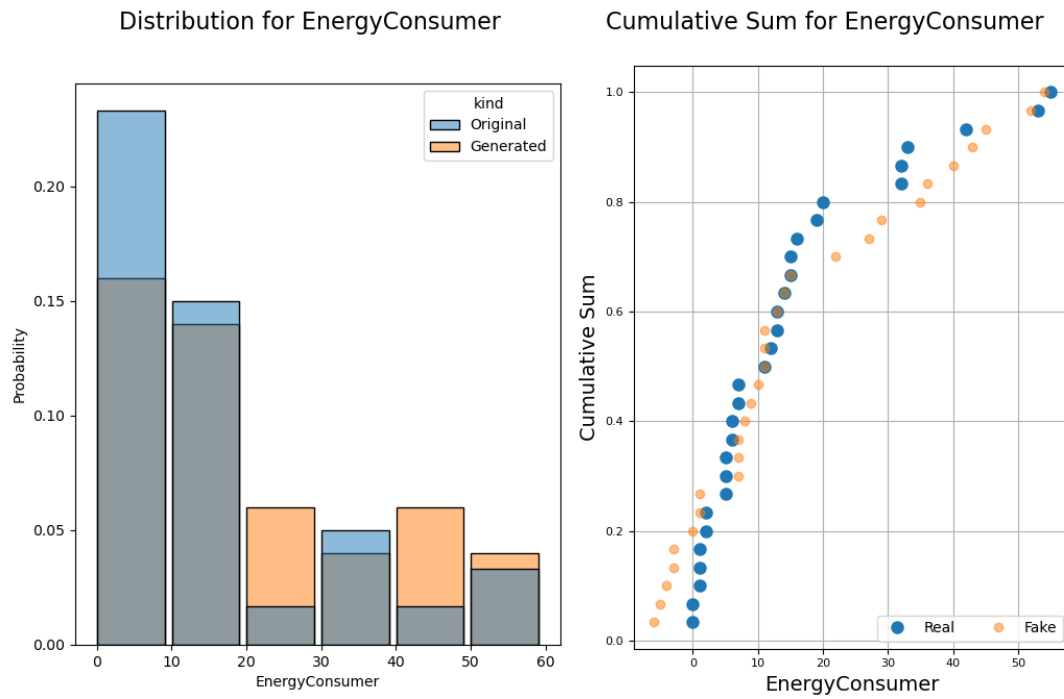


Figure 26: Illustration of how accurate the CTGAN model generated the EnergyConsumer object compared with the original file. This illustration includes two figures. One for the Distribution of an object and one for the cumulative sums for the same object. The object in this illustration is the Energyconsumer Object type. The EnergyConsumer was chosen to show the performance on an object with a high variance. The distribution of both the generated data and the original data are shown in a bar graph, where the x-axis represents the number of objects while the y-axis represents the probability of a file having a set number of objects. The Cumulative sums of both generated and original data are shown as a scatter plot, where the x-axis is the number of the object while the y-axis is the cumulative sum for the objects. The distribution figure shows the generated data performs well with the largest discrepancy being a 0.1 probability between 10 and 20 EnergyConsumers. The other figure shows high accuracy in the rest of the generated data when compared with the original data, however the generated data has some issues where it's generated negative values not present in the original data.

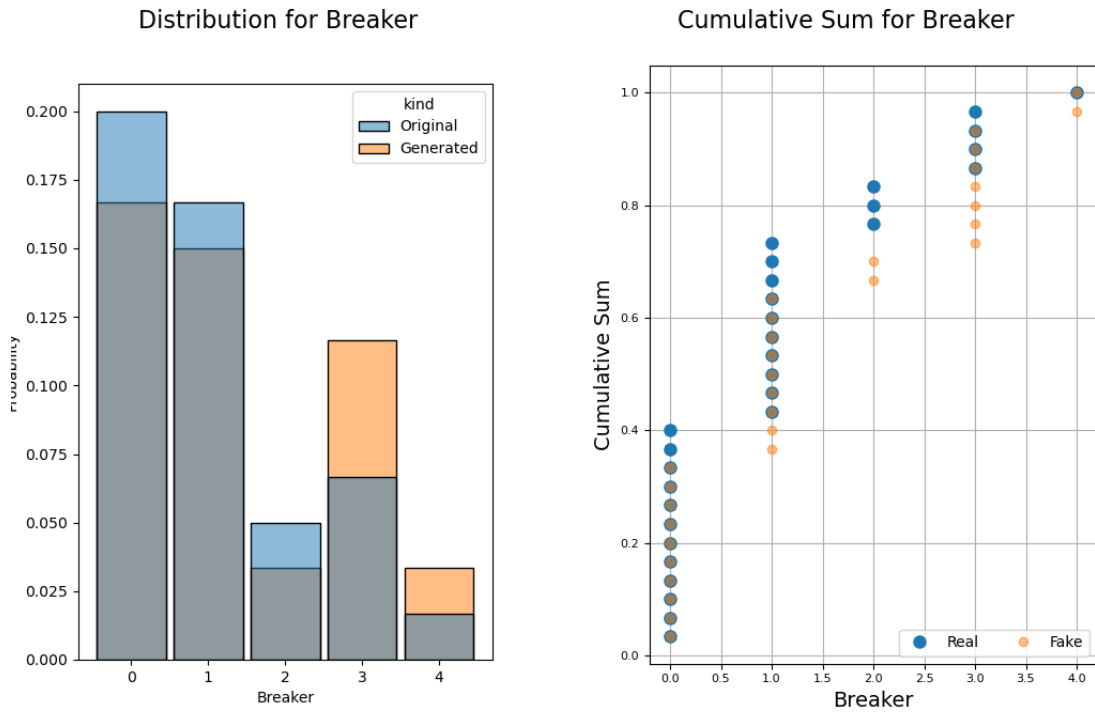


Figure 27: Illustration of how accurate the CTGAN model generated the Breaker object compared with the original file. The object in this illustration is the Breaker Object type. The Breaker object type was chosen to show the performance on an object type with a low variance. The distribution figure shows the generated data performs well with the largest discrepancy being a 0.05 probability at 3 Breakers. The other figure shows high accuracy in the rest of the generated data when compared with the original data.

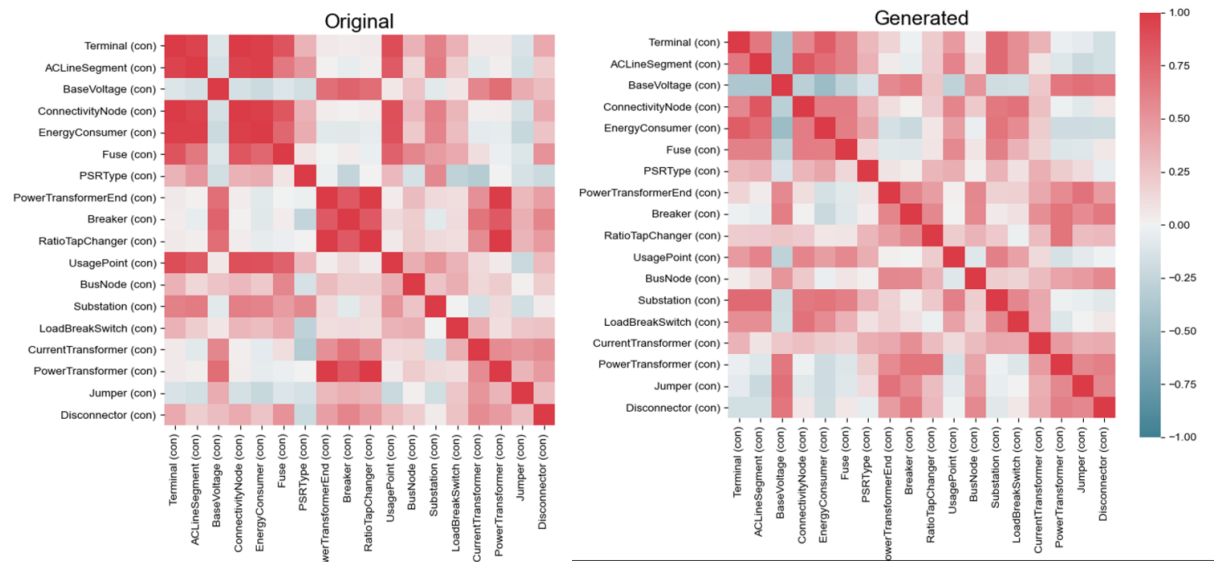


Figure 28: Visualisation showing the correlation between different object types in the data. Higher value shows higher correlation. From the figures we see the generated data performs well on objects with high correlations, however seems to have an correlation that is overall lower than the original data.

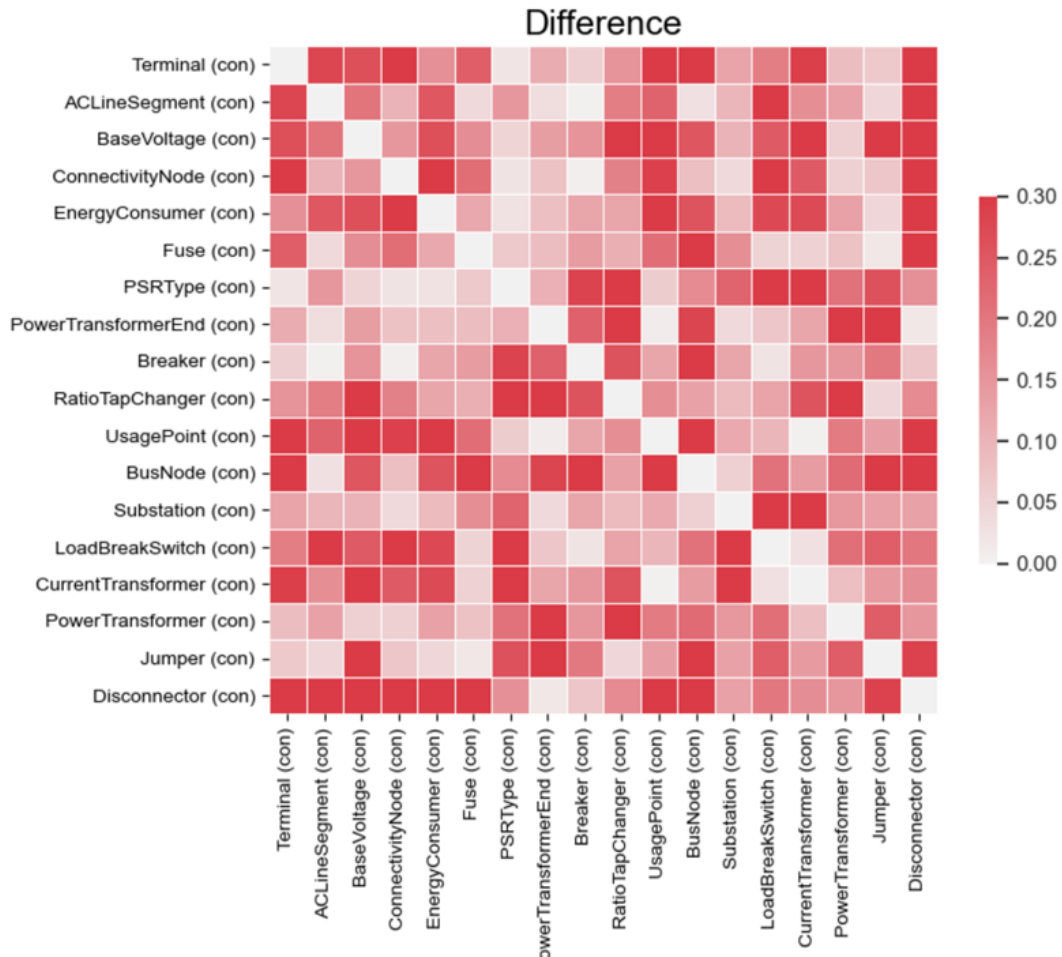


Figure 29: Visualisation of the difference in correlation between the original and generated data shown in Figure 28. Here lower value is better. From the illustration we can see a large difference in where the generated data is lacking correlation where the original data has high correlation.

4.2.2 TVAE

Hyperparameters for CTGAN Node Generation model	
Epochs	1000
Optimizer	Adam
Loss	Evidence lower bound, Softmax
Learnin Rate	1e-3
Weight Decay	1e-5
Embedding Size	128

Table 6: Configuration used when training the TVAE model. The parameters are explained in Table 7.

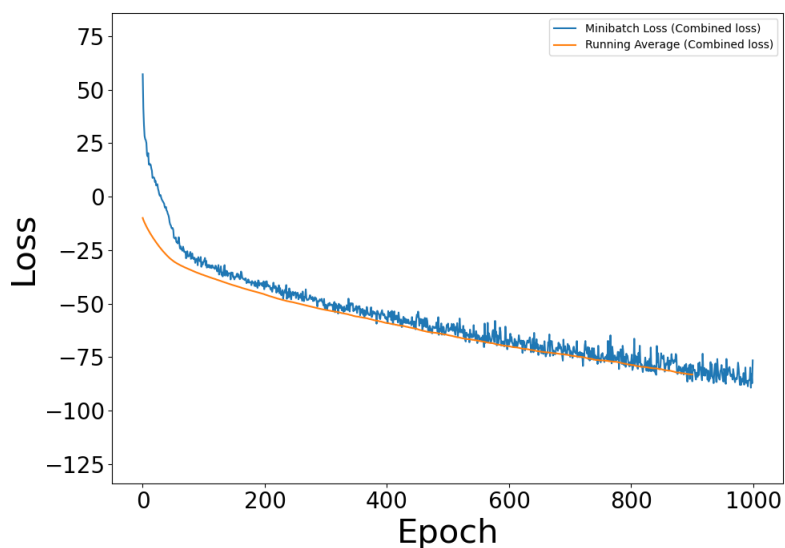


Figure 30: Graph illustrating the loss when training the TVAE. The figure shows the loss converges. This indicates the model improves for each epoch, however the improvement rate decreases.

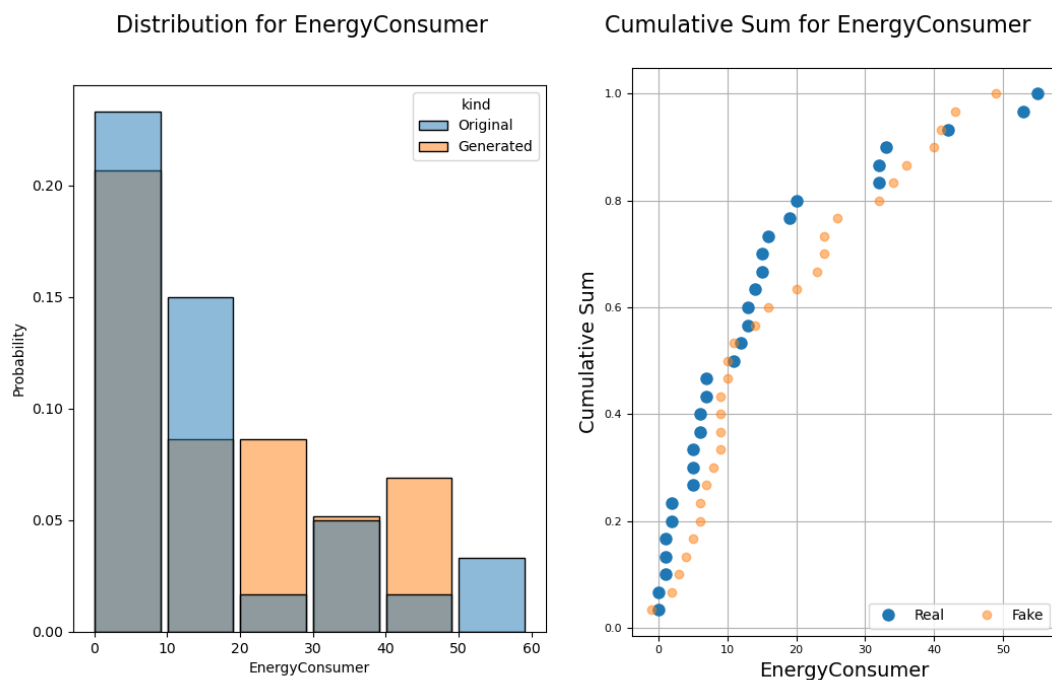


Figure 31: Illustration of how accurate the TVAE model generated the EnergyConsumer object compared with the original file. The distribution figure shows the generated data performs well with the largest discrepancy being a 0.07 probability between 10 and 20 EnergyConsumers. The other figure shows high accuracy in the rest of the generated data when compared with the original data, and unlike the CTGAN shown in Figure 26 there are no negative generated data.

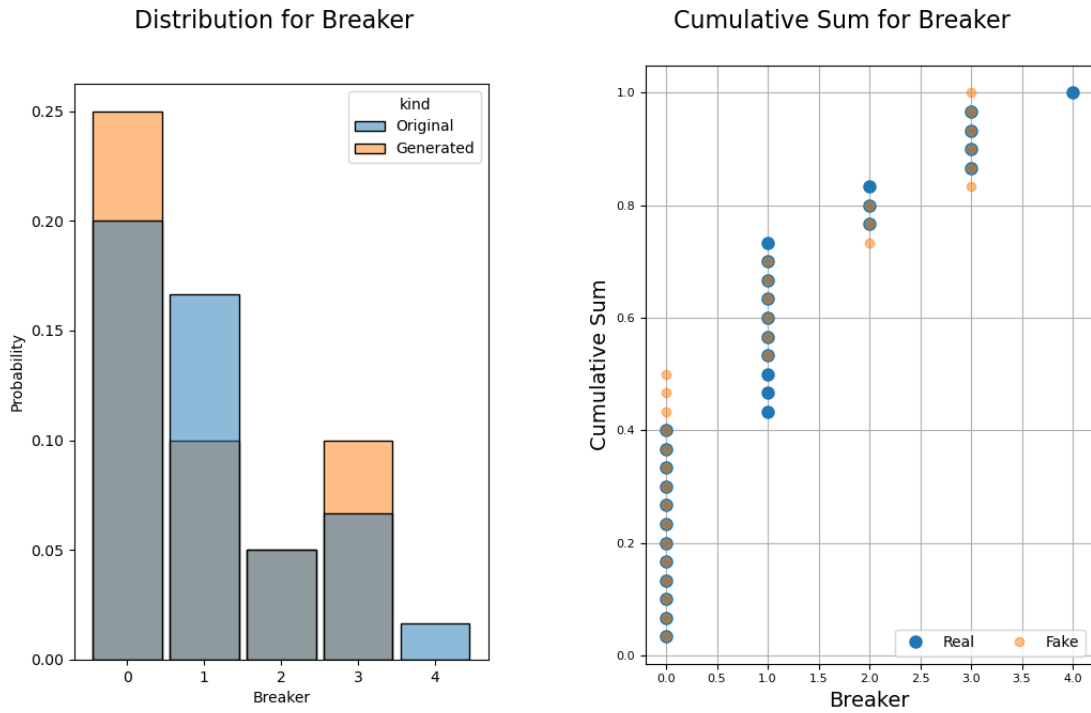


Figure 32: Illustration of how accurate the TVAE model generated the Breaker object compared with the original file. The distribution figure shows the generated data performs well with the largest discrepancy being a 0.06 probability at 1 Breaker. The other figure shows high accuracy in the rest of the generated data when compared with the original data.

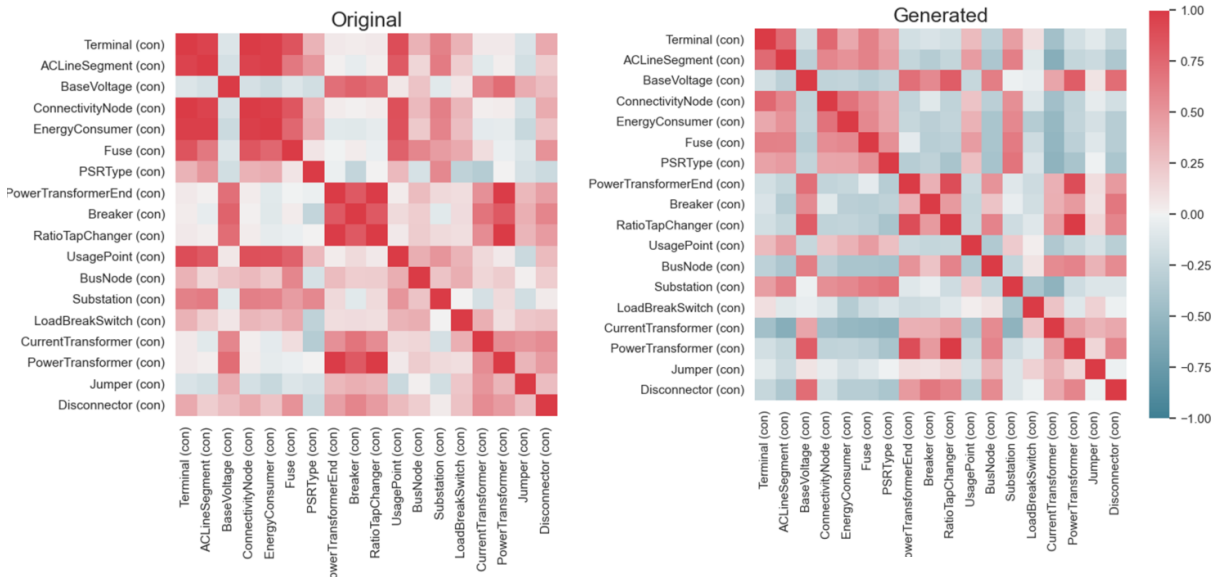


Figure 33: Visualisation showing the correlation between different object types in the data. A higher value shows a higher correlation. From the figures, we see the generated data performs poorly on objects without high correlations. The original data has few objects with high inverted correlations, however, the generated data has a significant number of objects with high inverted correlations.

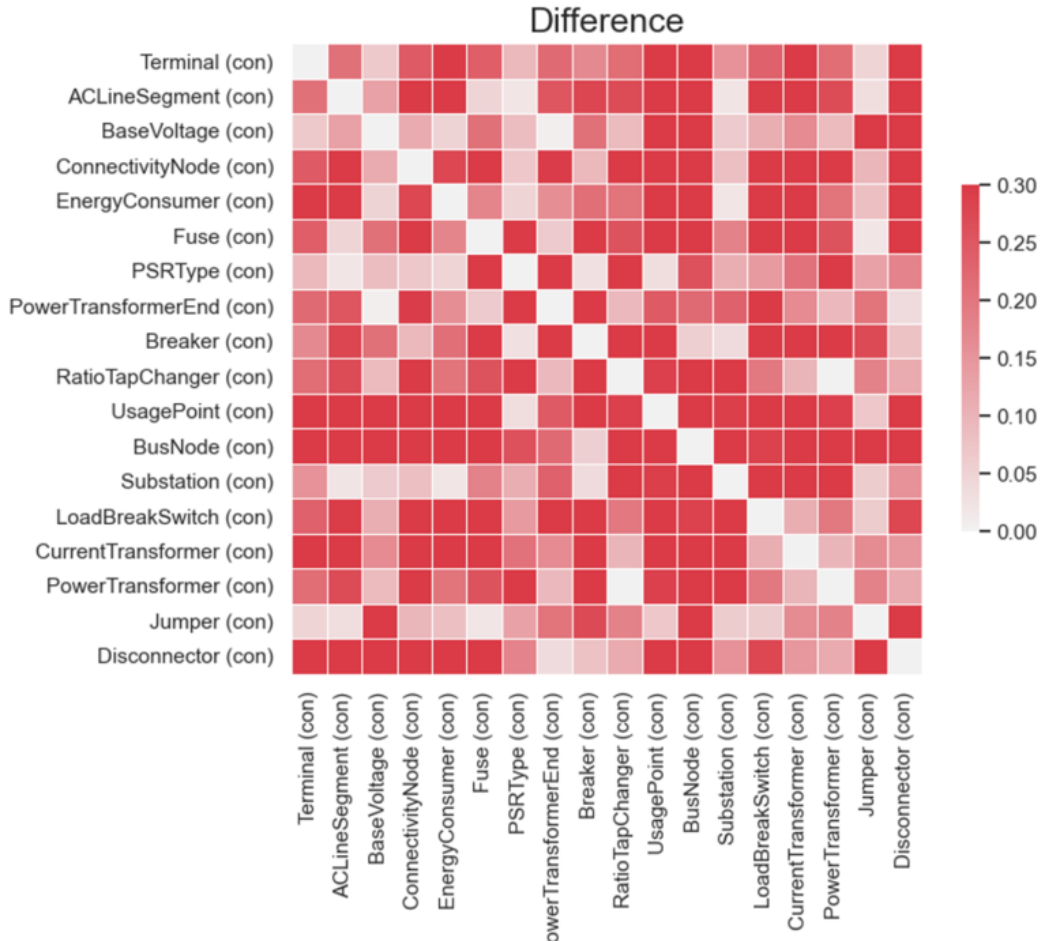


Figure 34: Visualisation of the difference in correlation between the original and generated data shown in Figure 33. Here lower value is better. From the illustration we can see the model has performed poorly, being a large difference over the entire graph. The difference is less on objects that have a high correlation.

4.2.3 Comparing CTGAN with TVAE

By comparing the CTGAN and TVAE distribution and cumulative sum plots, we can see that both node generation models produce competitive results. However, due to the CTGAN model generating some negative values, that cannot be in the original data, the TVAE model performs better on these metrics. The correlation differences tell another story. From the Figure 29 and Figure 34 we can clearly see that the CTGAN outperforms TVAE here. As the discrepancy in the correlation metric is significantly larger than the difference in the distributions, we can say the CTGAN model is the overall best performing model.

4.3 Link Prediction

This section describes the results obtained when running the link prediction experiments. The experiments consist of the following models:

- **GAE**
- **VGA:** There are two VGAE models, one trained on the power grid data provided by Volue, while the other is trained on the Planetoid dataset.

All of the models were trained with the same hyperparameters, and all models trained on our dataset use node type as the only feature.

Hyperparameters for Link prediction models	
Epochs	400
Optimizer	Adam
Loss function	Binary Cross-Entropy with Logits
Learnin Rate	0.1
Weight Decay	5e-4
Embedding Size	16
Decoder	Inner product decoder

Table 7: The configuration used when training all of the link prediction models. For explanation about each hyperparameters meaning, see Section 3.1.5.4

4.3.1 Graph Auto Encoder

The first experiment consisted of a simple graph autoencoder. This was to get a baseline for more complex models and to see how a simpler model performs on the data. This section describes the results obtained from the autoencoder model.

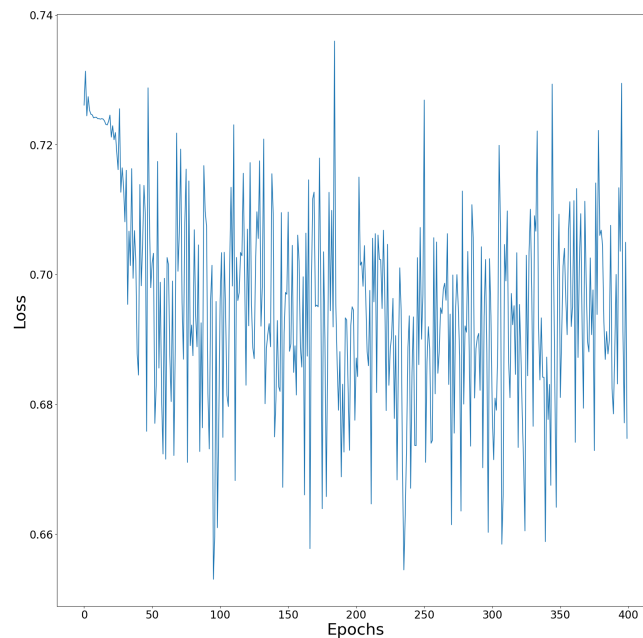


Figure 35: Graph illustrating the loss when training the Graph Auto Encoder. The loss varies between 0,64 and 0,74. The average remains almost the same through the epochs. The unchanging loss gives little information about model performance, how many epochs are optimal, and if the model is overfitted.

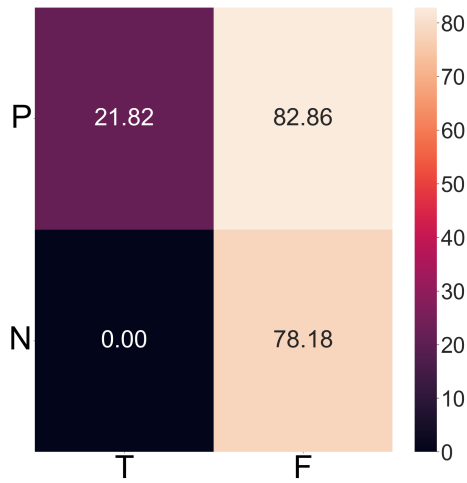


Figure 36: This figure are an illustration of the confusion matrix for the GAE model prediction. GAE guesses 22% of links are correct, but 83% of generated links do not match the original. This means the generated graph is 22% equal to the original. Summing true positives and false positives to over a hundred per cent means the model has generated more links than in the original

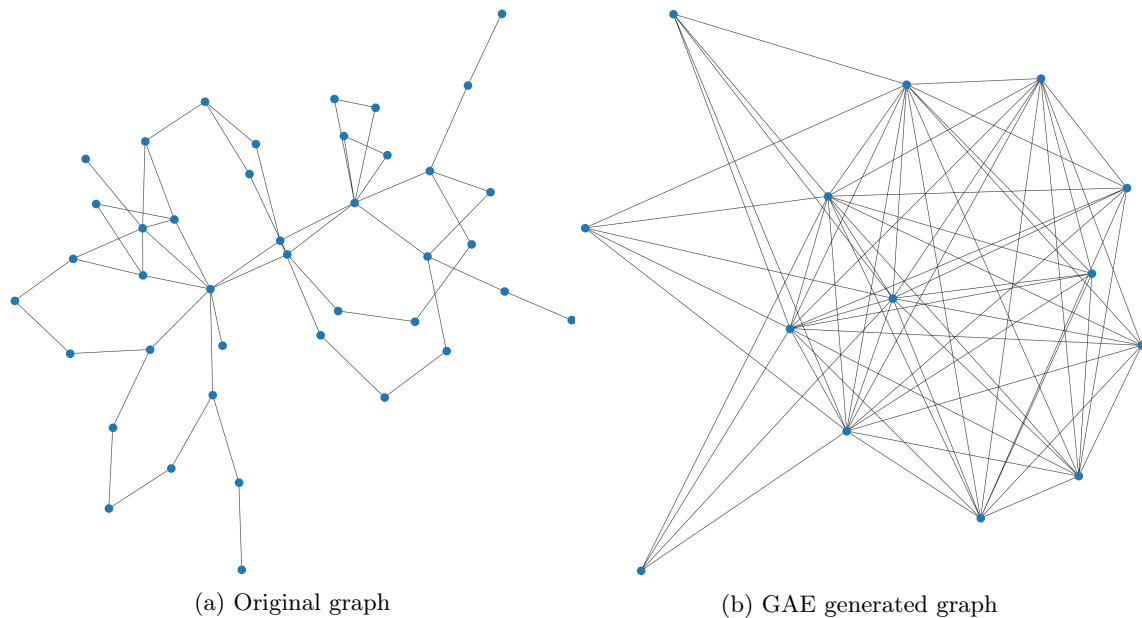


Figure 37: Plot displaying the difference between the original- and generated graph. Isolated nodes in the generated graph are not plotted. The generated shows that the model has chosen a small number of nodes and placed a lot of links between them, forgetting lower degree nodes. There is no visual signs of the structure from the original graph, in number of connections per node and what object types are connected to what object types.

4.3.2 Variational Graph Auto Encoder

The second link prediction model is a variational graph encoder. This section contains the results gathered from this model.

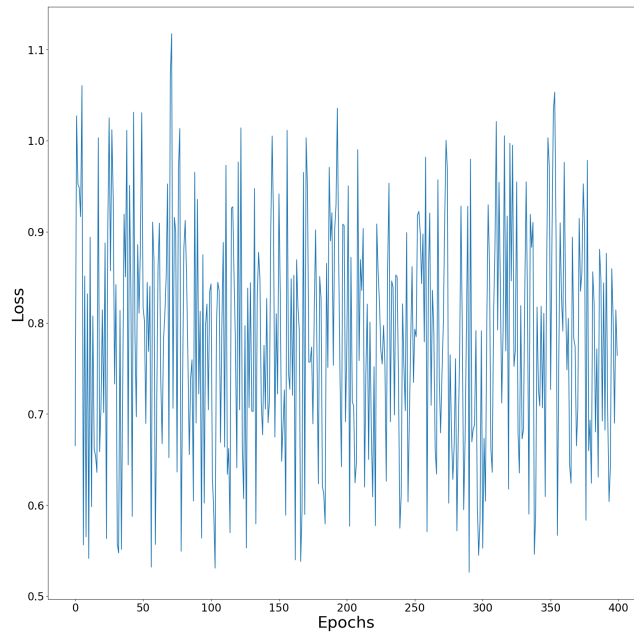


Figure 38: Graph representing the loss when training the Variational Graph Auto Encoder. The loss varies between 0,5 and 1,1. The average loss does not change much throughout training. The unchanging loss gives little information about model performance, how many epochs are optimal, and if the model is overfitted.

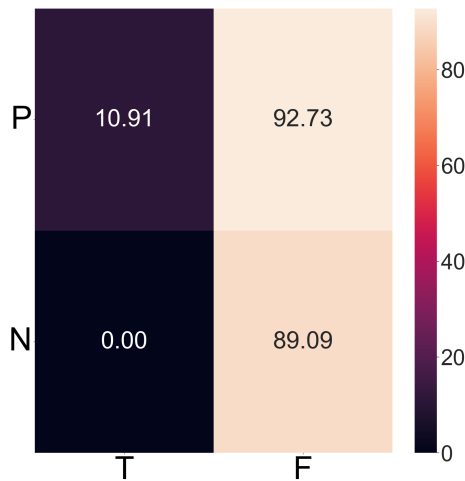


Figure 39: VGAE Confusion matrix. The number of correct guessed links is only 11%, and 93% of links are placed incorrectly. This makes sense as VGAE generates a variation with the same structure and not a copy of the graph itself. The true positives and false positives summing to over a hundred means the model has guessed more links than in the original

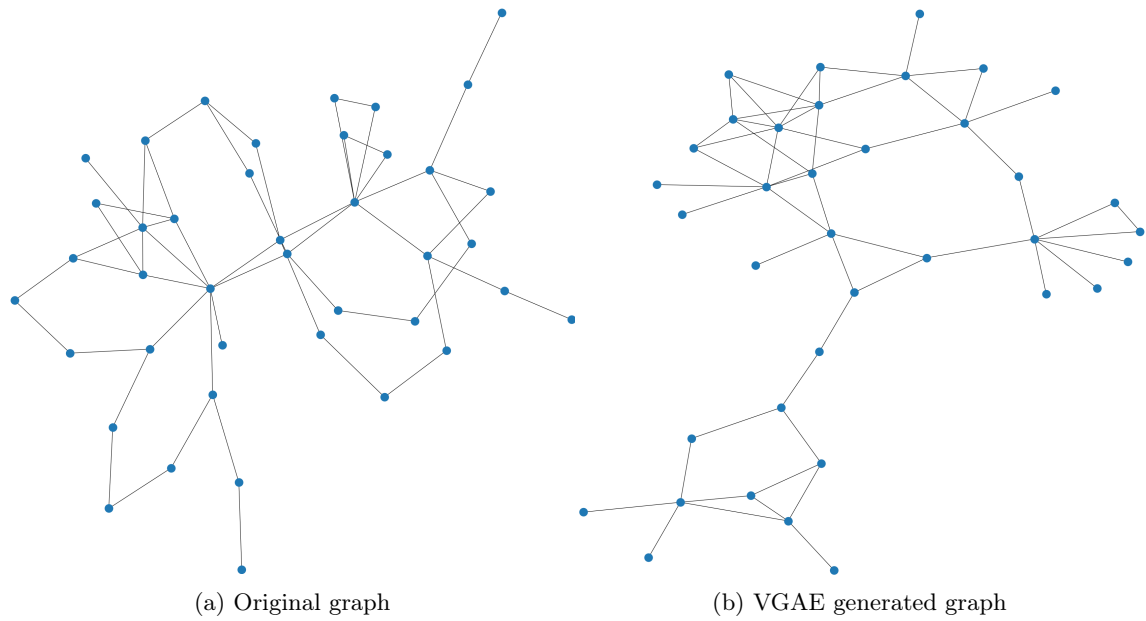


Figure 40: Plot displaying the difference between the original and generated graph. The VGAE generated graph has retained the structure from the original, but still has generated a new graph that looks different. The number of non-isolated nodes is similar, as well as the edge count. The VGAE has generated more leaf nodes.

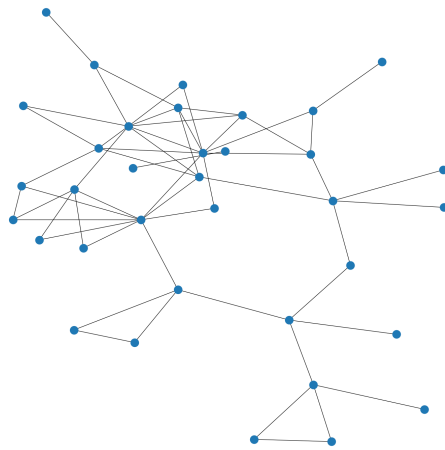


Figure 41: Plot of VGAE trained without node type. There are some connections that are very random, especially the cluster of nodes with many connections. There are multiple instances where three nodes are connected. With this, we can see that the node type is important in the structure of the grid, and therefore the model

4.3.3 Variational Graph Auto Encoder - Planetoid dataset

This section contains a simple implementation of a variational graph autoencoder on the Planetoid dataset, to serve as a comparison to the results from the other models. The Planetoid graph is larger than the power grid graphs, therefore the model's embedding size has been changed from 16 to 64, see Table 7. The model uses all features that come with the dataset.

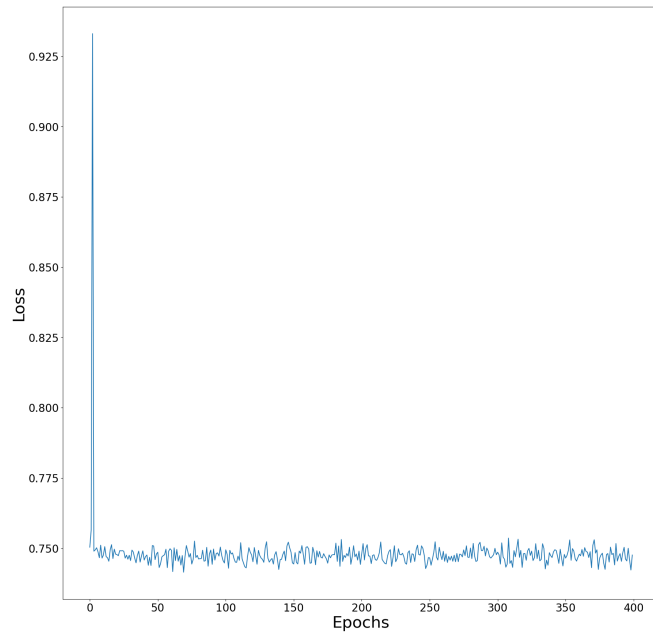


Figure 42: Illustrates the loss when training the Variational Graph Auto Encoder on the Planetoid dataset. The loss inside 0.755 and 0.745 except for the one outlier. The average loss remains the same throughout the training. The unchanging loss gives little information about model performance, how many epochs are optimal, and if the model is overfitted.

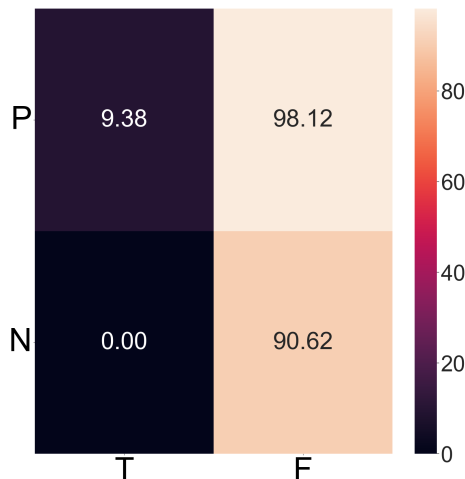


Figure 43: This figure illustrates the confusion matrix of the VGAE model trained on the planetoid dataset. It shows that only 10% of guessed edges are correct, while 98% of edges are incorrect. This means the generated graph is not very similar to the original, and that the generated graph has many more edges.

4.3.4 Comparing GAE with VGAE

The losses of our models give little information about model performance, how many epochs are optimal, and if the model is overfitted. The VGAE-Planetoid loss (Figure 42) also shows the same trend as in the GAE and VGAE, indicating that the constant average loss is not caused by our dataset.

By comparing Figure 36 with Figure 39, we can see that the GAE has the best results, however, the model guesses links between nodes based on a probable cluster. By looking at Figure 37b and Figure 40b, we observe that the VGAE has a more similar structure to the original graph than the

GAE. As the VGAE planetoid obtains similar results as the VGAE on our dataset, indicating that the VGAE model manages to train on our dataset equally well as on an established, well-processed dataset. There is no metric to evaluate if this structure is valid, however, by a visual comparison of the figures, the VGAE seems most promising.

5 Discussion

In this section we will reflect on the research questions presented in Section 1.2 chronologically. The reflection is rooted in our results and methodology.

5.1 Similar Solutions

In Section 2.7.1, articles about synthetic power grid generation are presented. After reading these articles, we discovered that generating synthetic power grid data have been done previously. However, these projects worked with larger power grid distributions, like big cities or regions. As opposed to this project, where we worked with distributions representing grids from a small town in northern Norway. With this in mind, we decided to draw inspiration from the projects instead of implementing their solutions. Rather than discovering if synthetic power grid data had ever been generated, the team learned there are multiple ways of doing this.

From Section 3.1.3, two approaches for generating synthetic power grids are described. Both approaches achieve promising results, however, they are not comparable as the chosen approach is dictated by the dataset. This is the case because each dataset is unique as they have different origins.

No international standard is present for developing power grid infrastructure. This leads to tailored synthetic data generators, as the power grid data used in each article (see Section 2.7.1) have different origins. The result of this is clear, none of the articles' proposed solutions can be copied as a tool for our assignment. Even heavy modifications to the closest, most relevant solution for our assignment, would be just as time-consuming as creating our own custom data generator.

5.2 Data

In this section we will discuss based on the results shown in Section 4.1 and the process described in Section 3.1.4.

5.2.1 Requirements for synthetic data

Volue's desire for synthetic data was to test their software. We had no access to this software, therefore we could not test our generated synthetic data, or quantify the resemblance to real data. Volue's requirements for the synthetic data were stated to be synthetic power grid data, resembling real power grid data. A possible way of evaluating power grid data is to compare a generated grid to a grid that is a statistical mean of all grids in the dataset. Using our dataset of 29 files of varying, unspecified types of grids, to learn about the mean structure of power grids is challenging. Since we had different types of grids, a generated mean structure based on our dataset is not guaranteed to have the properties of a real-world power grid. Therefore we had nothing to measure our results with. The structure of an industrial power grid differs so much from the grid of a mountain village, that a synthetically generated mean grid would make no sense. Therefore we could not clearly define what good synthetic power grid data is for Volue's purpose. The requirements for the synthetic data will differ from grid type to grid type, therefore we recommend having a larger dataset grouped by similar structures, and learning characteristics from these types. By looking at the characteristics of these groups, Volue can define requirements for each grid type and create a metric from these requirements.

5.2.2 Data analysis and Processing

As stated in Section 3.1.4.3, the data Volue gave us included a significant amount of superfluous information. Some examples are single objects describing the manufacturer, time and date the

components were installed, and by whom. Unnecessary data in each object would be information such as the description or name of a single object. This redundant information does not affect the power grid structure. The given examples are objects easily identifiable as excessive information, however, filtering all of this extra data proved challenging. Volue’s planned use of the synthetic data was unclear, and therefore it was unclear what data should be generated and which data should be removed.

As we lack domain-specific knowledge of both power grids and NETBAS, manually filtering the data would require extensive guidance from domain experts. From our data analysis, we tried only to retain the physical data, however, Volue informed us that some object types could represent both physical and virtual points of a network. For instance, the UsagePoint object type can represent either a virtual or physical meter. This complicated which objects to remove from the data and which objects the machine learning algorithms are required to generate.

The ideal power grid data to use for training the machine learning models would be the minimal structure that can represent the power grid. However, acquiring this bare-bone structure proved to be the most challenging part of the project. In retrospect, a viable approach to achieving a bare-bone grid could be to consider each object in the grid individually. To filter the unneeded data, we could represent the information of each object as a set. By removing the intersection of the data in the set, we could directly remove the information not pertaining to the bare-bone structure. Given enough data, this approach should converge the power grid data into a bare-bone structure. Although, convergence cannot be guaranteed for objects with repeated unique information only appearing once in a file. While this approach should filter most data automatically, further experiments would be needed to create a real bare-bone power grid.

5.3 Graph Generation

In this section we will reflect on the machine learning aspect of the project. We look at the best performing model, and why machine learning is a tool used for the assignment.

5.3.1 Best performing model

As shown by our experiments, the VGAE combined with CTGAN is our most promising model. To evaluate the results from our link prediction experiment, a measure of success is needed. It is essential for a good machine learning model to receive feedback from a metric to improve. Our research indicates the machine learning model with the highest potential is a modified GAN network, combined with a domain-specific evaluator.

5.3.2 Machine learning as the tool

In the past decade, the popularity of machine learning has exploded. Machine learning has been used to solve a myriad of problems where extensive data sets are available. During this project, we realized the availability of large datasets is not, in itself, sufficient to arrive at suitable machine learning solutions. This made us think about whether machine learning is the best solution for this project. A different approach for generating power grid data would be a statistical approach.

A statistical approach could be to look at the statistics pertaining to each object in real-world data. By using an object in the data, which in our case, is represented as a node, and then evaluate the likelihood that this node is connected to another node. By looking at the probabilities of the node connections, we can take a node and connect it to possible nodes based on the probability distribution. For instance, if a node has a 100% probability to be connected to another node, we connect these nodes. On the other side, a node with a 1% probability of being connected to another node would be represented in the dataset. This statistical approach would sequentially generate the grid with each node. This way of sequentially generating nodes is also an approach that can be combined with machine learning and should be investigated further.

5.4 Broader Impact

Power grids are considered critical infrastructure within a nation. One primary reason for generating synthetic power grid data is to counteract the security risks of using actual data. However, could the synthetic data be exploited maliciously by a hostile nation? One could speculate a scenario where a hostile nation could exploit the synthetic data. It does not contain the sensitive geographical locations of the power grids, but it has the structural patterns of the real power grids. This could be scrutinised to discover weak points in the power grid. However, this is unlikely as a drone- or satellite footage would probably achieve better results with less effort.

5.4.1 Sustainability

Power grids today are built with oversized components "just in case" there is an oversight with a total number of customers, as expanding a constructed power grid is expensive. However, building every power grid with oversized components because it is cheaper than expanding upon it later, is an ineffective use of resources and not sustainable. Increasing grid effectiveness would reduce construction costs and decrease passive energy loss.

Easier access to data while developing new power grids results in a more streamlined process. A more streamlined process means reduced costs and time, making power grids more readily available for construction. Further refining this approach can be used for other graph-structured networks, such as water and gas pipe networks. A refined approach creates an opportunity to build more efficient, streamlined grids. This would represent a decrease in raw materials early in the life cycle of the grid. A decrease in raw materials would reduce the environmental footprint of the construction of these grids.

5.5 Reflection Students

The team members have worked together on multiple projects in the past. As the members already had experience working together, the work progress has been fluid from beginning to end. This was enhanced by a thorough planning of the project, and effective sub-goals along the way.

Value has been great from the very beginning. Friendly, professional, and always providing help when needed. These circumstances have been very comforting for us and our goals. Working at Value's office increased the availability of expert advice resulting in a more efficient workflow.

We would also like to revisit the process and consider what we would have done differently with the given knowledge acquired throughout the previous months. We spent most of the project time analysing, understanding and processing the data. "How do we go from real power grid data to a bare-bone structure?" was a recurrent question throughout the project. Filtering the data was not seen as a core element at the start of the project. We decided to filter the data to the best of our abilities, but in the early stages, we did not fully understand the importance of filtering. We felt some results with Machine learning were important as this was the main part and the core element of the project. We believe we would have achieved better results and knowledge by focusing more on the data analysis part of the project. By doing this, we could come up with suitable methods to filter the data, try out set selections and other ways to remove excess data. This would have given us a better foundation for designing our machine learning models. Another significant issue was how to evaluate these methods. To address this challenge we would look at objects with certain attributes, link them together and then evaluate the smaller subgraphs. This approach would bypass the need for a metric to evaluate large synthetic power grids against large real-world power grids.

6 Conclusion

The main focus of the report was to investigate the feasibility of using machine learning to generate synthetic power grid data. By continuously researching the topic, analysing and processing the data provided by Volue, and experimenting with different machine learning methods, we managed to answer each formulated research question presented in Section 1.2.

Has synthetic grid data been generated before? If yes, how?

Synthetic power grid data have been generated by multiple parties in the past. Each party with their own unique solution, but with similar approaches. Section 5.1 discusses why the approaches do not really matter, as the data have the most influence on the solution. After analysing our data, we learned that it were too unique for any pre-existing solutions, considering the data's difference in size, structure and properties. Rather than copying any of the solutions found, we drew inspiration from them and tailored our own model.

What are the requirements of the synthetic data and how does it resemble the real data?

In Section 5.2.1 we discuss how our dataset is too small to create a usable mean grid. A bigger dataset grouped by grid type has the possibility of defining a usable mean grid that could serve as a point of evaluation for this grid type. The requirements for synthetic power grid data are based on which type of grid we want to generate. Hence, for synthetic data to resemble real data, a dataset that is grouped by grid types is needed.

Given real-world grid data, which parts of the data should be used for training?

In Section 5.2.2, the given dataset and the dataset's contents are discussed. A myriad of possible objects and information in the dataset implied that domain knowledge is needed to confidently remove all superfluous data from real-world power grid data. The discussion concludes the data that should be used for machine learning is the minimal network structure that can represent the power grid data.

What is a real-world bare-bone power grid?

In Section 5.2.2, it is discussed how acquiring a bare-bone power grid is a challenging process. By removing intersected values and objects throughout a dataset, we guarantee that the remaining data are unique and closely pertain to a bare-bone structure. However, in a small dataset there exist unique data appearing once and thus are not removed. A bare-bone power grid is therefore the minimal structure needed to represent the physical components in a power grid and their connections.

Which machine learning model is best suited for learning power grid data?

Evaluating generated synthetic power grid data is challenging. We had to define a way to evaluate power grid data ourselves. We found that a confusion matrix and visual evaluation were the best approximation for power grid data. From these evaluations, it is clear that a combination of the CTGAN and VGAE gives the best results. Without an evaluator, however, it is unclear which model is best suited to generate power grid data. If such a power grid evaluator existed, a GGAN has the best potential.

Is machine learning the best approach to generate synthetic power grid data?

There are two approaches to generate synthetic power grid data, that we know of. These are a machine learning approach and a statistical approach. Other projects have successfully used the statistical approach to generate power grid data. Section 5.3.2 present a proof of concept for a statistical approach that should be investigated before a generative approach is chosen. We could not investigate this approach as Volue specifically told us to use machine learning. However, a large dataset, in itself, does not guarantee good machine learning results. Power grid data in its raw form have too much irrelevant data. We have proved that machine learning models can generate a structure that is visually similar to real power grid data. But without seeing the results from a statistical approach, it is unclear which approach is best.

Generating synthetic power grid data is motivated by the need for data that is not characterised by confidentiality. To generate the synthetic data, two approaches have been used so far, the statistical approach and the machine learning approach. However, we discovered that the approaches did not matter as much as the individual solutions. Each solution ends up tailored for their respective dataset, becoming so unique that datasets with other origins are incompatible. Additionally, generating synthetic data requires a metric to evaluate the quality of the generated data. With no metric for comparing the generated data to the real data, there is no way to tell if the generated synthetic data can be used.

6.1 Future work

Volue has stated that this project is the first step toward creating a complete product capable of Volue's initial vision.

As Volue wants to use synthetic data in their software testing, they need a solution for transforming the data back to CIM. Currently, Volue has no way of transforming any generated data back to CIM, meaning the synthetic data cannot be used for their intended purpose. Solving this obstacle should be one of the first steps forward.

Consulting domain experts to manually define a bare-bone power grid, would help much with generating synthetic data and its evaluation. The generated power grids could be compared to the bare-bone power grid, as it is defined as a valid grid. This would make the bare-bone power grid a metric for success, through grid-comparing methods.

Defining a method for evaluating power grid data would provide multiple benefits. With an evaluator you could:

- Compare different synthetic generators based on their accuracy.
- Develop a model loss function that improves the model accuracy while training.
- Tell if a generated synthetic power grid is good, or not.

The evaluator would create a metric for the generated data. With this metric, Volue could start the process of gradually producing better synthetic data, with the help of the aforementioned benefits.

Bibliography

- [1] Oct. 2021. URL: <https://arxiv.org/ftp/arxiv/papers/1812/1812.08434.pdf> (cit. on p. 7).
- [2] Sept. 2021. URL: https://cs.emory.edu/~lzhao41/materials/papers/Heterogeneous_Graph_Generation.pdf (visited on 15th May 2022) (cit. on p. 21).
- [3] John M. Abowd and Lars Vilhuber. ‘How Protective Are Synthetic Data?’ In: *Privacy in Statistical Databases*. 2008, pp. 239–246. ISBN: 978-3-540-87471-3 (cit. on p. 9).
- [4] *AlphaGo*. Apr. 2022. URL: <https://www.deepmind.com/research/highlighted-research/alphago> (visited on 8th Apr. 2022) (cit. on p. 6).
- [5] *Background: What is a Generative Model? | Generative Adversarial Networks | Google Developers*. Feb. 2021. URL: <https://developers.google.com/machine-learning/gan/generative> (visited on 20th May 2022) (cit. on p. 7).
- [6] Joseph M. Carew. ‘reinforcement learning’. In: *SearchEnterpriseAI* (Mar. 2021). URL: <https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning> (visited on 10th May 2022) (cit. on p. 6).
- [7] Joseph M. Carew. ‘reinforcement learning’. In: (Mar. 2021). URL: <https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning> (visited on 8th Apr. 2022) (cit. on p. 6).
- [8] *CIM Schema | DMTF*. Apr. 2022. URL: https://www.dmtf.org/standards/cim/cim_schema_v2541 (cit. on p. 9).
- [9] Nicola De Cao and Thomas Kipf. *MolGAN: An implicit generative model for small molecular graphs*. 2018. DOI: [10.48550/ARXIV.1805.11973](https://doi.org/10.48550/ARXIV.1805.11973) (cit. on pp. 11, 23).
- [10] Claire Donnat et al. ‘Learning Structural Node Embeddings via Diffusion Wavelets’. In: (2018), pp. 1320–1329. DOI: [10.1145/3219819.3220025](https://doi.org/10.1145/3219819.3220025) (cit. on p. 4).
- [11] Jim Dowling. ‘Guide to File Formats for Machine Learning: Columnar, Training, Inferencing, and the Feature Store’. In: *Medium* (Dec. 2021). URL: <https://towardsdatascience.com/guide-to-file-formats-for-machine-learning-columnar-training-inferencing-and-the-feature-store-2e0c3d18d4f9> (visited on 5th May 2022) (cit. on p. 21).
- [12] *Figure 1. A schematic of the scientific method*. May 2021. URL: https://www.researchgate.net/figure/A-schematic-of-the-scientific-method_fig2_287358730 (visited on 5th Apr. 2022) (cit. on p. 14).
- [13] Luka Fürst, Marjan Mernik and Viljan Mahnič. ‘Graph grammar induction’. In: *Advances in Computers*. Vol. 116. 1. Jan. 2020, pp. 133–181. DOI: [10.1016/bs.adcom.2019.07.003](https://doi.org/10.1016/bs.adcom.2019.07.003) (cit. on p. 21).
- [14] *GEDEVO (Graph Edit Distance and EVolutionary algorithm for network alignment)*. May 2018. URL: <https://gedevo.mpi-inf.mpg.de> (visited on 15th May 2022) (cit. on p. 6).
- [15] Ian J. Goodfellow et al. ‘Generative Adversarial Networks’. In: (June 2014). DOI: [10.48550/arXiv.1406.2661](https://doi.org/10.48550/arXiv.1406.2661) (cit. on p. 7).
- [16] Aditya Grover and Jure Leskovec. ‘node2vec: Scalable Feature Learning for Networks’. In: (July 2016). DOI: [10.48550/arXiv.1607.00653](https://doi.org/10.48550/arXiv.1607.00653) (cit. on p. 5).
- [17] H. Hafting. *Graph Theory*. 2020. URL: http://www.iie.ntnu.no/fag/_alg/uv-graf/graf-uv.pdf (cit. on p. 4).
- [18] Fanghao Han. *Tutorial on Variational Graph Auto-Encoders - Towards Data Science*. Dec. 2021. URL: <https://towardsdatascience.com/tutorial-on-variational-graph-auto-encoders-da9333281129> (cit. on p. 23).
- [19] Kshiteesh Hegde et al. ‘Network Signatures from Image Representation of Adjacency Matrices: Deep/Transfer Learning for Subgraph Classification’. In: (Apr. 2018). DOI: [10.48550/arXiv.1804.06275](https://doi.org/10.48550/arXiv.1804.06275) (cit. on p. 22).
- [20] *Introduction to Node Embedding*. Oct. 2021. URL: <https://memgraph.com/blog/introduction-to-node-embedding> (cit. on pp. 4, 5).

- [21] M. I. Jordan and T. M. Mitchell. ‘Machine learning: Trends, perspectives, and prospects’. In: *Science* 349.6245 (2015), pp. 255–260. DOI: [10.1126/science.aaa8415](https://doi.org/10.1126/science.aaa8415) (cit. on p. 6).
- [22] Mahdi Khodayar, Jianhui Wang and Zhaoyu Wang. ‘Deep Generative Graph Distribution Learning for Synthetic Power Grids’. In: *arXiv* (Jan. 2019). DOI: [10.48550/arXiv.1901.09674](https://doi.org/10.48550/arXiv.1901.09674) (cit. on p. 10).
- [23] Diederik P. Kingma and Max Welling. ‘An Introduction to Variational Autoencoders’. In: *arXiv* (June 2019). DOI: [10.1561/22000000056](https://doi.org/10.1561/22000000056) (cit. on pp. 8, 22).
- [24] Thomas N. Kipf and Max Welling. ‘Variational Graph Auto-Encoders’. In: (2016). DOI: [10.48550/ARXIV.1611.07308](https://doi.org/10.48550/ARXIV.1611.07308) (cit. on pp. 12, 23, 24).
- [25] Serafeim Loukas. ‘What is Machine Learning: Supervised, Unsupervised, Semi-Supervised and Reinforcement learning method’. In: *Medium* (Dec. 2021). URL: <https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb> (visited on 8th Apr. 2022) (cit. on p. 6).
- [26] Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021 (cit. on pp. 4, 7).
- [27] Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021, pp. 201–203 (cit. on p. 23).
- [28] *Ontology Documentation*. Feb. 2019. URL: https://ontology.tno.nl/IEC_CIM (visited on 16th May 2022) (cit. on p. 16).
- [29] *PyG Documentation — pytorch_geometric 2.0.5 documentation*. May 2022. URL: <https://pytorch-geometric.readthedocs.io/en/latest> (cit. on p. 22).
- [30] *Resource Description Framework (RDF) Model and Syntax Specification*. Oct. 2017. URL: <https://www.w3.org/TR/PR-rdf-syntax> (visited on 5th May 2022) (cit. on p. 9).
- [31] Joseph Rocca. *Understanding Variational Autoencoders (VAEs) - Towards Data Science*. Towards Data Science, Dec. 2021. ISBN: 978-705109197. URL: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73> (cit. on pp. 8, 9).
- [32] Alberto Sanfeliu and King-Sun Fu. ‘A distance measure between attributed relational graphs for pattern recognition’. In: *SMC-13.3* (1983), pp. 353–362. DOI: [10.1109/TSMC.1983.6313167](https://doi.org/10.1109/TSMC.1983.6313167) (cit. on p. 5).
- [33] John J. Simmins. ‘The impact of PAP 8 on the Common Information Model (CIM)’. In: *2011 IEEE/PES Power Systems Conference and Exposition*. 2011, pp. 1–2. DOI: [10.1109/PSCE.2011.5772503](https://doi.org/10.1109/PSCE.2011.5772503) (cit. on p. 9).
- [34] Martin Simonovsky and Nikos Komodakis. ‘GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders’. In: *arXiv* (Feb. 2018). DOI: [10.48550/arXiv.1802.03480](https://doi.org/10.48550/arXiv.1802.03480) (cit. on pp. 12, 22, 23).
- [35] Saleh Soltan, Alexander Loh and Gil Zussman. ‘A Learning-Based Method for Generating Synthetic Power Grids’. In: *IEEE Systems Journal* 13.1 (2019), pp. 625–634. DOI: [10.1109/JSYST.2018.2825785](https://doi.org/10.1109/JSYST.2018.2825785) (cit. on p. 10).
- [36] Flawnsong Tong. ‘What is Geometric Deep Learning? - Flawnsong Tong - Medium’. In: (Dec. 2021). URL: <https://flawnsontong.medium.com/what-is-geometric-deep-learning-b2adb662d91d> (visited on 12th May 2022) (cit. on p. 6).
- [37] Andrej Trpovski, Dante Recalde Melo and Thomas Hamacher. ‘Synthetic Distribution Grid Generation Using Power System Planning: Case Study of Singapore’. In: Aug. 2018. DOI: [10.1109/UPEC.2018.8542054](https://doi.org/10.1109/UPEC.2018.8542054) (cit. on p. 10).
- [38] Hongwei Wang et al. ‘GraphGAN: graph representation learning with generative adversarial nets’. In: *AAAI’18/IAAI’18/EAAI’18: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. Feb. 2018, pp. 2508–2515. DOI: [10.5555/3504035.3504341](https://doi.org/10.5555/3504035.3504341) (cit. on pp. 11, 12, 23).
- [39] Kunfeng Wang et al. ‘Generative adversarial networks: introduction and outlook’. In: *IEEE/CAA Journal of Automatica Sinica* 4.4 (2017), pp. 588–598. DOI: [10.1109/JAS.2017.7510583](https://doi.org/10.1109/JAS.2017.7510583) (cit. on p. 8).

- [40] *Why synthetic data is important for your business*. Mar. 2022. URL: <https://mostly.ai/synthetic-data/how-synthetic-data-changes-the-business-world> (visited on 10th May 2022) (cit. on p. 9).
- [41] Lei Xu et al. ‘Modeling Tabular data using Conditional GAN’. In: (July 2019). DOI: [10.48550/arXiv.1907.00503](https://doi.org/10.48550/arXiv.1907.00503) (cit. on pp. 23, 24).
- [42] Jiaxuan You et al. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models*. 2018. DOI: [10.48550/ARXIV.1802.08773](https://doi.org/10.48550/ARXIV.1802.08773) (cit. on pp. 11, 22).
- [43] Stephen J. Young et al. ‘Synthetic Power Grids from Real World Models’. In: *2018 IEEE Power Energy Society General Meeting (PESGM)*. 2018, pp. 1–5. DOI: [10.1109/PESGM.2018.8585792](https://doi.org/10.1109/PESGM.2018.8585792) (cit. on p. 10).
- [44] Jie Zhou et al. ‘Graph neural networks: A review of methods and applications’. In: 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: <https://doi.org/10.1016/j.aiopen.2021.01.001> (cit. on p. 22).

Attachments

Attachment 1 - Project Handbook

