FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Graph Reinforcement Learning for Improving Smart Grid Services

António Bernardo Linhares Oliveira

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. António Costa & Prof. Rosaldo Rossetti

August 31, 2024

Graph Reinforcement Learning for Improving Smart Grid Services

António Bernardo Linhares Oliveira

Mestrado em Engenharia Informática e Computação

August 31, 2024

Resumo

Os grafos são representações que descrevem problemas e os seus objectos em domínios orientados para as redes, como as redes eléctricas, os transportes ou as redes sociais. Estas representações podem captar não só os conceitos e as suas respectivas propriedades, mas também as relações entre esses conceitos, resultando em estruturas de dados frequentemente complexas e esparsas que são especialmente úteis para representar problemas em que a topologia de uma rede desempenha um papel importante. No contexto da utilização destas estruturas em algoritmos de aprendizagem computacional, o seu desempenho torna-se dependente não só da sua conceção e seleção dos atributos relevantes e parâmetros, mas também das representações subjacentes utilizadas para captar a essência das estruturas em grafo.

A Aprendizagem por Reforço em Grafos ou *Graph Reinforcement Learning* (GRL) é um tópico que tem merecido grande atenção por parte dos académicos nos últimos anos. Ao permitir que as técnicas de Aprendizagem por Reforço aprendam e optimizem processos de decisão sequenciais em ambientes baseados em grafos, os sistemas podem ser melhorados de forma a tirar partido das características da topologia dos grafos em domínios de aplicação associados a redes. Com os avanços no final da década de 2010 em Redes Neuronais de Grafos, ou *Graph Neural Networks*, na aprendizagem e extração de representações eficientes de grafos, foram propostos métodos mais sofisticados de GRL e o tópico começou a atrair mais curiosidade dos académicos. Embora, nos últimos anos, muito trabalho tenha sido feito nesta área, a pesquisa à volta do GRL ainda é considerada estar em fase inicial.

Além disso, considerando os actuais desafios globais associados à sustentabilidade e aos sistemas energéticos, há uma necessidade crescente de avanços em sistemas inteligentes focados em modernizar as redes de distribuição e transmissão de energia. Atualmente, as fontes de energia renováveis desempenham um papel importante na redução da dependência dos combustíveis fósseis, o que altera a topologia dos sistemas de distribuição de energia à medida que os consumidores adquirem a capacidade de gerar energia renovável. Com as melhorias na Inteligência Artificial e Aprendizagem Computacional, os sistemas podem ser adaptados à descentralização da produção e gerir eficientemente a monitorização, distribuição e transmissão da energia. Neste trabalho, a ênfase principal reside na melhoria dos algoritmos de GRL que serão aplicados no contexto do problema da distribuição dinâmica e económica de energia como seu principal domínio de aplicação, considerando fontes de energia renováveis e sistemas de armazenamento de energia.

Desta forma, esta dissertação tem como objetivo fazer avançar a investigação existente sobre técnicas de Aprendizagem por Reforço em Grafos através de: (1) realizar uma revisão exaustiva da literatura recente relativa às várias abordagens de GRL propostas e de sistemas de distribuição dinâmica e económica de energia, de modo a obter uma perspetiva global das técnicas recentes mais avançadas e das suas limitações; (2) realizar um estudo empírico comparativo e sistemático das diferentes técnicas de GRL no problema de distribuição dinâmica e económica, considerando cenários de estudo de caso de diferentes dimensões modelados por uma simulação de uma rede de distribuição de energia eléctrica; (3) propor um modelo que melhor integre as capacidades das técnicas de *Deep Reinforcement Learning* e *Graph Neural Networks*, com base nos resultados do estudo empírico e com melhorias no desempenho e escalabilidade face aos modelos propostos pela literatura.

Abstract

Graphs are structures that depict problems and their objects in network-oriented domains such as power grids, transport or social networks. These representations can capture no only the concepts and their respective properties but the intricate relationships between those concepts, resulting in often complex and sparse data structures that are especially useful for representing problems where network topology plays a major role. In the context of using these structures in machine learning algorithms, their performance becomes not only dependent on its design and the selection of relevant features and parameters, but also on the underlying representations used to capture the essence of graph structures.

Graph Reinforcement Learning (GRL) is a topic that has earned significant attention from academics in the last few years. By enabling Reinforcement Learning techniques to learn and optimize sequential decision-making processes in graph-based environments, systems can be improved and gain the ability to leverage graph topology features in network-oriented application domains. With the advancements in the late 2010s on Graph Neural Networks on learning how to extract efficient graph representations from a given scenario, more sophisticated methods of GRL were proposed and the topic started to attract the curiosity of scholars. Although, in recent years, a lot of work has been done in this area, research on techniques is still considered to be in an early stage.

Furthermore, considering the current global challenges associated with sustainability and energy systems, there is an increasing need for advancements in energy-focused intelligent systems to modernize the current power grids. In the present, renewable energy sources play a major role in reducing the reliance on fossil fuels, which changes the topology of energy distribution systems as consumers gain the capability to generate renewable power. With the improvements in Artificial Intelligence and Machine Learning, systems can be adapted to the decentralization of energy production and efficiently manage the monitoring, distribution and transmission of energy systems. In this work, the primary emphasis lies on improving GRL algorithms which will be applied to solve the dynamic economic power dispatch problem as its main application domain, considering renewable energy sources and energy storage systems.

In this manner, this dissertation aims to advance the existing research on Graph Reinforcement Learning techniques by: (1) conducting a thorough review of the recent literature regarding various proposed GRL approaches and Dynamic Economic Dispatch Systems to gain an overall perspective of the recent state-of-the-art techniques and their limitations; (2) performing a comparative and systematic empirical study on the different GRL techniques on the Dynamic Economic Power Dispatch problem, considering case study scenarios of different sizes modelled by a power distribution grid simulation (3) propose a model that better integrates the capabilities of Deep Reinforcement Learning Agents with Graph Neural Networks, based on the results of the empirical study, with improvements in performance and scalability.

Keywords: Graph Reinforcement Learning, Graph Neural Networks, Deep Reinforcement Learning, Smart Grid, Dynamic Economic Dispatch

ACM Classification: Computing Methodologies $\rightarrow$ Machine Learning $\rightarrow$ Learning Paradigms $\rightarrow$ Reinforcement Learning

Acknowledgements

To my supervisors for all the guidance and teachings during the development of this work
To my parents and girlfriend who always supported me and expected me to become the best version of myself

António Oliveira

*“Man is not worried by real problems
so much as by his imagined anxieties about real problems”*

Epictetus

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

IT	Information Technology
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
RL	Reinforcement Learning
MDP	Markov Decision Process
DRL	Deep Reinforcement Learning
GNN	Graph Neural Network
GRL	Graph Reinforcement Learning
SAC	Soft Actor-Critic
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
GCN	Graph Convolutional Network
GAT	Graph Attention Network
GIN	Graph Isomorphism Network
ADN	Active Distribution Network
DED	Dynamic Economic Dispatch
ESS	Energy Storage System
RES	Renewable Energy Source
ML	Machine Learning

Chapter 1

Introduction

In this introductory chapter, the context and motivation regarding this dissertation, as well as its key objectives are presented in sections 1.1, 1.2 and 1.3, respectively. Additionally, the structure of this report is exposed and its logical divisions are described is section 1.4.

1.1 Context

Several real-world problems and their objects can be instinctively represented by graph structures. These representations not only capture the main properties of a given domain but also the intricate topology of relationships in a network-oriented problem. Graph representations are often sparse and complex and to appropriately leverage their various topology features machine learning algorithms require underlying methods to efficiently generalize and produce adequate representations from these structures, considering the trade-off data completeness and computational efficiency.

In the case of sequential decision-making problems, the same is verified. Learning how to map good sequences of decisions in network-oriented domains can depend, in some cases, on accounting for the environment topological features [**?**, **?**, **?**, **?**]. As the main paradigm of machine learning that addresses sequential decision-making problems, RL algorithms need to be adapted to reflect these considerations, which establishes the foundations of Graph Reinforcement Learning (GRL). This compromises the main focus of this work, which will be applied in the application domain of *Smart Grid* Services.

In other regards, this report is written in the context of the course of Dissertation Preparation (PDIS) inserted in the Master's Degree in Informatics and Computing Engineering (MEIC) of the Faculty of Engineering of the University of Porto (FEUP). In addition, this project is accommodated in the Artificial Intelligence and Computer Science Laboratory (LIACC).

1.2 Motivation

Reflecting on the current global issues associated with the energetic crisis and climate change, there is an increasing need for sustainable and economic energetic systems to modernize the current power grids. This modernization is translated into the transition to the *smart grid*, a power grid equipped with intelligent control and monitoring systems to efficiently manage power distribution [?, ?], voltage regulation, system restoration [?], grid reliability [?] or other associated processes. Currently, renewable energy sources play a major role in reducing the reliance on fossil fuels [?], which changes the topology of energy distribution systems as consumers gain the capability to generate renewable power. Furthermore, investment in energy storage is becoming a priority in the energy sector [?], resulting in improvements on storage capacity and approximating the current solutions to the average consumer [?].

The exposed issues serve as the prime motivation for performing this work on the application domain of *smart grid* services, with the expectation that by proposing concrete improvements in GRL techniques a well-performing solution to the dynamic economic dispatch problem can be presented. Beyond this, we hope the proposed solution and its architecture is also adaptable and applicable to other smart grid problems, resulting in a significant contribution to these services.

Furthermore, the main reasons for addressing GRL algorithms lies on their novelty and complexity, the lack of well-documented literature regarding these approaches, and the need for systematic and comparative studies confronting the different proposed techniques and architectures.

1.3 Objectives

Considering this work's context and motivations, we define its main objectives:

1. Perform a review of literature regarding GRL approaches and Dynamic Economic Dispatch (DED) systems
2. Conduct a comparative and systematic empirical study of different GRL solutions of the DED problem
3. Propose a GRL model and concrete improvements facing the literature proposed models

The first goal addresses the analysis of existent techniques, as well as its limitations, by reviewing the relevant research on GRL approaches and DED systems. Secondly, we will focus on implementing the different observed approaches to solve the DED problem and perform a comparative study to analyse and confront the gathered results. Lastly, we hope the accomplishment of the first to goals to enable the proposition of a state-of-the-art GRL model and specific improvements to these techniques.

1.4 Report Structure

This report is organized as follows: (1) an introductory chapter; (2) a chapter explaining the relevant background concepts to perform this study; (3) the review of the literature regarding GRL techniques and DED systems; the statement of the main problem and the presentation of the proposed solution and finally, (6) the main conclusions, reflections and expected contributions of this work.

Chapter 2

Background Knowledge

In this chapter, we will present the underlying concepts related to this dissertation. This knowledge consists in important context for the rest of this work by explaining the its background concepts.

This chapter is divided into section 2.1 addressing Artificial Neural Networks, section 2.2 regarding Reinforcement Learning problem and algorithms, 2.4 explaining Graph Representation Learning, 2.5 exposing the current GNN approaches and 2.6 addressing Smart Grid Services..

2.1 Artificial Neural Networks

For almost a century, academics ravelled around the concept of biological learning and how to replicate such behaviour with computational systems [**?**, **?**] . As inspiration for the fundamental concepts that the field of **Deep Learning** bases itself upon, scientists turned to the structure responsible for this process on all living beings, the brain [**?**, **?**]. As a consequence, some of the earlier learning algorithms were designed to model how learning happens in the brain.

Artificial Neural Networks (ANNs) are a foundational and ubiquitous class of machine learning algorithms that are also based on the neural process of biological learning [**?**, **?**]. These networks are composed of interconnected nodes known as *neurons* and can be generally described as *function approximators*. Given an unknown function f^* that models a complex relationship between input data x and ouput data y, the main goal of an ANN is to approximate to the considered function knowing x and y [**?**, **?**]. The new model of the relation between both samples can be then be further applied to new input data x' with the goal of predicting y' .

The simplest form of an ANN is a Multilayer Perceptron (MLP), also called a **Feedforward Neural Network** and is further described in the next subsection.

2.1.1 Feedforward Neural Networks

A Feedforward neural network defines a mapping $y = f(x;\theta)$ and learns the best composition of parameters θ to approximate it to the unknown model f^* [**?**, **?**]. The MLP serves as a fundamental part of developing the other more complex types and modern implementations of neural networks [**?**]. The main building block of a MLP is the **Perceptron**, pictured in figure 2.1, a simple

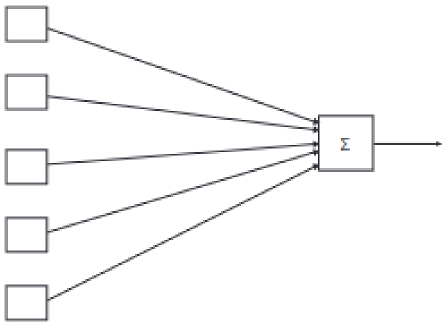

Figure 2.1: The Perceptron [**?**]

computational model initially designed as a binary classificator that mimics biological neurons' behaviour [**?**]. A neuron might have many inputs x and has a single output y. It contains a vector of *weights* $w = (w_1 ... w_m)$, each associated with a single input, and a special weight b called the *bias*. In this context, a perceptron defines a computational operation formulated by equation 2.1 [**?**].

$$f(x) = \begin{cases} 1 & \text{if } b + w \cdot x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Functions that compute $b + w \cdot x > 0$ are called *linear units* and are identified with Σ [**?**, **?**]. An activation function g was introduced to enable the output of non-linear data. The default recommendation is the *Rectified Linear Unit* (*ReLU*), with the Logistic curve (sigmoid) also being very common [**?**].

Feedforward networks are composed of layers of *neurons*, namely an input layer formed by the vector of input values, an arbitrary number of hidden layers and an output layer, which is the last layer of neurons that computes the final predicted output value y' [**?**]. The dimensionality of the hidden layers determine the overall width of the model and the greater the amount of layers the higher the *depth* of the network [**?**, **?**]. This last notion together with the process of how biological neurons interact also motivated the conceptualization of **Deep Neural Networks**, which consist in feedforward neural networks with a considerable amount of hidden layers. By increasing network depth, these models are able to learn highly complex patterns and abstractions in data which *shallow* networks cannot capture [**?**].

These structures are called networks because they consist of a chain of different functions, which correspond to the various layers. Feedforward networks can also be described by a directed acyclic graph that defines how these functions interact with each other to compose the complete network [**?**]. With real-world correspondence between input values and associated outputs, a feedforward network can be trained to approximate the unknown function of the environment. In more concrete terms, the training process encompasses an optimization problem, which in turn

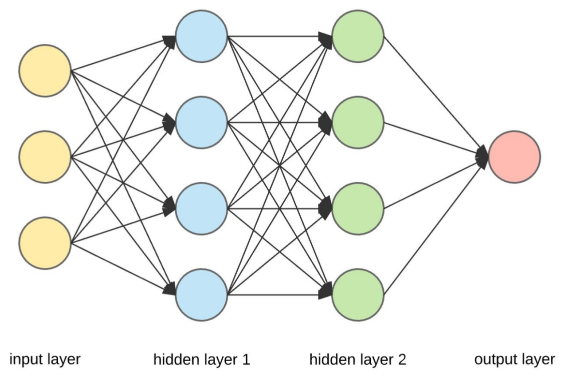

Figure 2.2: Architecture of a Feedforward Neural Network [?]

involves minimizing or maximizing another function that consists in a *objective function* $J(\theta)$. This function is also called the loss or cost function (when the objective is to minimize it) and, in the context of neural networks, indicates how distant the network model is to the real function [?]. Typically used loss functions include the mean squared error or mean absolute error. The objective functions can then be minimized with techniques such as *stochastic gradient descent* to optimize the neural network parameters [?, ?].

2.2 Reinforcement Learning

Reinforcement Learning (RL) consists of a field and a class of machine learning algorithms that study how to learn to take good sequences of actions to achieve a goal associated with a maximizing received numerical reward [?]. The main objective is to maximize the received cumulative reward by trying between the available actions and discovering which ones yield the most reward [?]. This sequential decision-making process becomes more complex when a delayed reward is considered, given that an action with immediate reward may not always reflect the delayed consequences of that decision [?]. It's also the learner's job to consider this during the learning process. These concepts of *delayed reward* and *trial-and-error search* make up the most important characteristics of Reinforcement Learning [?]. The classic formalisation of this problem is the MDP through defining the agent-environment interaction process, explained in the following subsection 2.2.1.

A major challenge in this machine learning paradigm is the trade-off between *exploring* new unknown actions and *exploiting* the already known "good" actions [?]. To choose the sequence of actions that return the highest reward, the agent must choose actions it found effective in similar past situations or *exploit* what it learned from experience. Furthermore, given that the agent may not know the action-reward mappings initially, it has to *explore* possible actions that were not selected previously or may initially seem to yield a low reward to compute accurate reward estimates. The main problem is that neither exploitation nor exploration can be favoured exclusively without failing at the task [?]. Additionally, an agent's environment is uncertain, and changes in the environment's dynamics may also involve re-estimating action rewards.

In conclusion, RL techniques enable the implementation of sequential decision-making agents that seek to maximize a reward signal analogous to an explicit (complex) goal. The agents need to balance between actions that yield a reward on posterior time steps and actions that produce immediate rewards. In addition, these agents are also faced with the task of balancing the exploitation of information from past experiences and the exploration of new decision paths that could potentially return a higher reward down the road [**?**].

2.2.1 Markov Decision Process

Markov Decision Processs (MDPs) are a classical formalization of a sequential decision-making process, constituting the mathematical definition of the RL problem [**?**, **?**]. Beyond estimating potential rewards for the available actions, the problem defined by MDPs involves learning which actions are optimal in specific situations, i.e. learning a mapping between states of the environment and actions [**?**].

The central component of MDPs is the agent, which acts as a decision-maker and learns from interactions with the environment it's inserted. In a continuous process, the agent takes actions that affect the environment's state, which in turn presents new situations [**?**]. The environment also responds with the reward signals which the agent aims to maximize over time through its decision process.

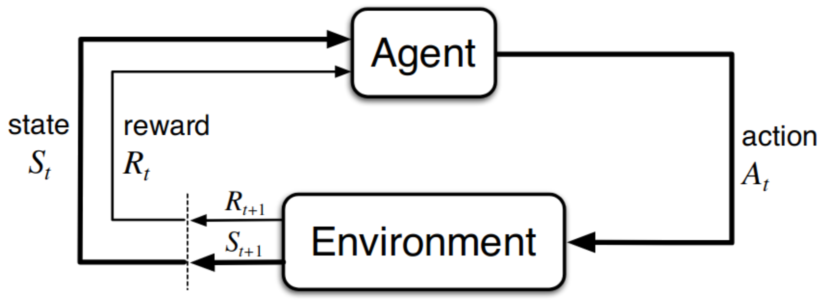

Figure 2.3: Agent-environment interaction in a MDP [**?**]

Formally, the agent-environment interactions, as figure 2.3 entails, occur in a sequence of discrete time steps t, where at each step, the agent receives a representation of the state of the environment $S_t \in \mathcal{S}$ which is used to select an appropriate action $A_t \in \mathcal{A}(s)$, where $\mathcal{S}$ is the set of possible states called the *state space* and $\mathcal{A}(s)$ is the set of available actions for state s [**?**, **?**]. In the next step, the agent receives, as a consequence of its decision, a numerical reward signal $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and is faced with a new state S_{t+1} [**?**]. Ultimately, the MDP agent follows a logical sequence that occurs as equation 2.2 states. The collection of a state S_t, action taken A_{t+1}, reward R_{t+1} received and next state S_{t+1} constitutes an *experience tuple* [**?**].

$$S_0, A_0, R_1, S_1, A_2, R_2, S_2, A_2, R_3, \ldots \tag{2.2}$$

In addition, when the set of possible actions, states and rewards ($\mathcal{A}$, $\mathcal{S}$ and $\mathcal{R}$) are finite, the MDP is said to be *finite* [?]. This results in S_t and R_t having well-defined discrete probability distributions in function of the preceding state and chosen action [?]. Therefore, the probability of receiving a particular reward and state given the previous state and selected action, which characterizes a finite MPD's dynamics, may be characterized by function p defined in equation 2.3

$$p(s',r|s,a) \doteq Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \tag{2.3}$$

For all $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$, where $\doteq$ denotes a mathematical formal defintion. This encompasses the assumption that the probability of each possible state, S_t, and reward, R_t, pair is only dependent on the preceding state, S_{t-1}, and action taken, A_{t-1} [?]. Instead of observing this as a restriction on the decision process, it's more convenient to view it as a constraint on the state variable, considering that it must contain all the necessary information from experience to make a valuable decision in the immediate step. If this condition is satisfied, the state is declared to have the *markov property* [?].

From function p in equation 2.3, the state-transtion probabilities, also called the *transition function*, can be computed as described by equation 2.4 [?, ?].

$$p(s'|s,a) \doteq Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s',r|s,a) \tag{2.4}$$

In addition, the expected rewards can be calculated for state-action pairs (equation 2.5) or state-action-next-action triples (equation 2.6) [?, ?].

$$r(s,a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s',r|s,a) \tag{2.5}$$

$$r(s,a,s') \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)} \tag{2.6}$$

2.2.2 Rewards and Returns

As stated in the previous subsections, the main goal of an RL agent is tied to the **numeric reward signals**, $R_t \in \mathbb{R}$, it receives from the environment [?]. In this context, the agent's objective is to maximize the total reward it receives, considering not only immediate but also the cumulative reward over time. In the ubiquitous work of [?], the *reward hypothesis* is stated as follows:

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). [?]

This also entails that the process of **reward maximization** from the agent has to be closely tied to it achieving its defined goals in a practical sense. Otherwise, the agent will fail at fulfilling the desired objectives [?].

Formally, the cumulative reward received over time is also called the **expected return**, G_t, and can be described by equation 2.7 [?].

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.7}$$

T describes the final time step. This definition can be applied in domains with a natural notion of a terminal state or final time step. In these cases, the agent-environment interaction process can be broken into logically independent subsequences called *episodes* [?]. Each episode ends in a special state, called the terminal state, restarting a new sequence of states and actions completely independent from the previous episode [?]. In this context, episodes can be considered to end in the same terminal state, with different accumulated rewards for the different outcomes [?].

In contrast, there are situations where the decision-making process doesn't divide itself into logically identifiable episodes but goes on indefinitely. In this case, $T = \infty$ and according to equation 2.7, the expected return the agent aims to maximize would be infinite [?]. In this manner, another concept is added in the expected return definition called the *discount rate*, γ where $0 \leq \gamma \leq 1$, representing how strongly the agent should account for future rewards in the expected return calculations, as equation 2.8 [?].

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.8}$$

From this equation, we can compute the expected discounted return on a given time step t in the function of the immediate reward signal received and the expected return for the next time step $t+1$, which eases the job of calculating expected returns for reward sequences [?]. This is entailed by equation 2.9.

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.9}$$

In this manner, a MDP can be defined by a tuple with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function p, a reward function r and a discount factor γ, as equation 2.10 portrays [?].

$$M = (\mathcal{S}, \mathcal{A}, p, r, \gamma) \tag{2.10}$$

2.2.3 Policies and Value Functions

RL techniques typically involve the estimation of what is understood as **value functions**. A value function estimates the expected return based on the current state value or state-action pair. They characterize how good it is for an agent to be in a specific state or to take an action in a specific state, respectively, using the expected return to characterize the overall *goodness* of these scenarios [?, ?]. These functions are tied to a specific way of determining the action in a given state. Formally, this is defined as a **policy** π, that defines the probability $\pi(a|s)$ of taking action a in state s [?]. In this context, the **state value function**, $v_\pi(s)$ and **action-value functions** $q_\pi(s,a)$ for policy π can be defined by equations 2.11 and 2.12, respectively [?].

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \forall s \in \mathcal{S} \tag{2.11}$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \tag{2.12}$$

The utility of such functions rely on the possibility of estimating them with regard to past experience of the agent [?]. A fundamental property of value functions, in a similar method as the expected return (equation 2.9), is that it can, satisfy recursive relationships with the next immediate value as equations 2.13 and 2.14 entail. [?]. These equations are called **Bellman equations**, and they characterize the relationship between the value of current state and state-action pairs and subsequent states.

$$v_\pi(s) \doteq \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)(r+\gamma\, v_\pi(s')) \tag{2.13}$$

$$q_\pi(s,a) \doteq \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)(r+\gamma\, v_\pi(s')) \tag{2.14}$$

The two equations can be connected by equation 2.15, which demonstrates that the value of state s under policy π is equivalent to the average of all possible action values in that state. The value of each state-value pair (s,a) is weighted with the probability of policy π taking said action in state s.

$$v_\pi(s) \doteq \sum_{a\in\mathcal{A}(s)} \pi(a|s) q_\pi(s,a) \tag{2.15}$$

2.2.4 Types of RL

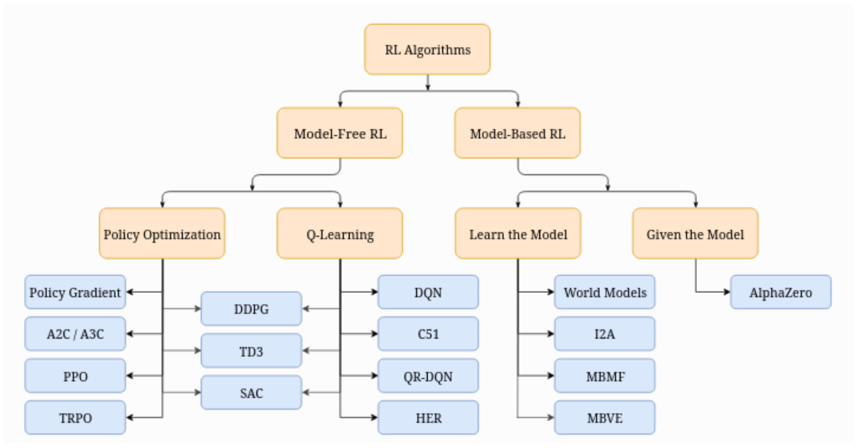

Figure 2.4: Taxonomy of algorithms in modern RL [?]

Regarding RL algorithms, they can be divided into **model-free** and **model-based** techniques [?]. These categories are distinguished by whether an agent uses a provided or learned *model* of the set of transition and reward functions, another optional element of RL techniques [**?**, **?**]. In the positive case, the method is said to be model-based, otherwise, it's model-free. Having an accurate model of the environment allows the RL agent to focus on planning ahead by calculating future scenarios and creating policies based on the results of the planning process. An example of a famous system of this kind is AlphaZero [**?**]. However, in most cases, agents can't access a ground-truth model of the environment, leaving only the scenario where an agent learns a model purely from experience. This creates several challenges, the most prominent of which relies on the fact that the model, in most times, doesn't fully capture the environment's transition dynamics, equipping it with bias in relation to the actual dynamics. With this, learning how to generalise the model to real-world environments so that the bias is not over-exploited becomes a very complex task [**?**]. Model-free algorithms can be also further divided into Q-learning and Policy Aproximmation techniques.

Furthermore, algorithms can also be subdivided into **on-policy** and **off-policy** methods. [**?**] On-policy algorithms evaluate and improve a single policy used to determine the agent's behaviour [**?**]. The methods under the policy optimization category such as A2C and A3C [**?**] or the Proximal Policy Optimization (PPO) [**?**] almost always fall into this label. In contrast, off-policy algorithms learn how to improve a different target policy based on the results that arise from the policy used to determine the system behaviour initially [**?**]. Such approaches include Q-Learning algorithms such as Deep Q-Networks (DQNs) [**?**, **?**]. Deep Deterministic Policy Gradient (DDPG) [**?**] combines policy optimization with q-learning, consisting of an off-policy method that learns both a q-function and a policy. DDPG constitutes the adaption of q-learning methods to continuous action spaces [**?**]. Another example of an off-policy algorithm is the SAC [**?**] method, which bridges stochastic policy optimization with the DDPG approach and has entropy regularization as one of its central features, which translates into training a policy that maximizes the expected return and entropy, a measure of randomness in the policy [**?**].

Lastly, with the advent of deep learning becoming one of the most ubiquitous techniques in machine learning, RL algorithms have evolved beyond the traditional tabular methods [**?**]. Traditional RL has evolved to **Deep Reinforcement Learning (DRL)**, which studies how to use deep neural networks in RL problems to leverage their generalization abilities for solving more complex problems.

2.3 Soft Actor-Critic

The **Soft Actor-Critic (SAC)** algorithm was first proposed in [**?**], and it is inserted within the category of model-free Reinforcement Learning (RL) algorithms. This Deep Reinforcement Learning (DRL) model leverages an off-policy method for learning a stochastic policy in continuous state and action spaces.

The most innovative and primary feature of SAC is **entropy regularization**, idealized to mitigate a central issue in model-free approaches related to sample efficiency [**?**, **?**]. As environments and problems become more complex, collecting representative samples of past interactions becomes harder. When traditional RL methods focus on maximizing the expected cumulative rewards, SAC considers a general maximum entropy objective that favours exploration, accelerates learning and prevents the policy from converging into bad local optimums [**?**].

Entropy is a measure of randomness for probability distributions [**?**]. The higher the entropy of a random variable, the more unpredictable the results. For instance, if a dice is loaded always to come up the same side, it has low entropy. If x is a random variable with a density function P, the entropy H of x can be computed with equation 2.16 [**?**].

$$H(p) = \mathbb{E}_{x \sim p}[-\log p(x)] \tag{2.16}$$

Entropy-regularised RL introduces a bonus reward at each time step dependent on the entropy of the policy at that step. The optimal policy π^* can thus far be calculated by equation 2.17 [**?**].

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t (r(s,a,s) - \alpha \log(\pi(\cdot|s))] \tag{2.17}$$

An additional temperature parameter $\alpha > 0$ controls the relative importance of this objective against the standard goal of cumulative reward maximization. This enables the formalization of new state and state-action value functions as defined by equations, 2.18 and 2.19 [**?**, **?**].

$$V_{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^t \left(R_{t+k+1} - \alpha \log\left(\pi(\cdot|S_{t+k})\right)\right) \middle| S_t = s\right] \tag{2.18}$$

$$Q_{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^t R_{t+k+1} - \alpha \sum_{k=1}^{\infty} \gamma^t \log(\pi(\cdot|S_{t+k}))\right) \middle| S_t = s,\ A_t = a\right] \tag{2.19}$$

Additionally, the bellman equation for Q_{π} can also be redefined as equation 2.20.

$$Q_{\pi}(s,a) = \mathbb{E}_{S_{t+1} \sim P}\left[R_t + \gamma V^{\pi}(S_{t+1}) \mid S_t = s,\ A_t = a\right] \tag{2.20}$$

To achieve its main goal, SAC uses function approximators for the policy and Q-function [**?**]. In this context, a parameterized state value function $V_{\psi}(\mathbf{s}_t)$, a soft Q-function $Q_{\theta}(\mathbf{s}_t, \mathbf{a}_t)$, and a tractable policy $\pi_{\phi}(\mathbf{a}_t|\mathbf{s}_t)$ are considered with parameters ψ, θ, and ϕ, respectively. Usually, the value functions are modelled as neural networks, and the policy as a Gaussian with a mean and covariance given by neural networks [**?**].

$$V_{\pi}(s) = \mathbb{E}_{A_t \sim \pi}\left[Q_{\pi}(S_t, A_t) - \alpha \log(\pi(A_t|S_t)) \middle| S_t = s\right] \tag{2.21}$$

Given that the state value function is related to the Q-function according to equation 2.21, there is no need to incorporate an additional function approximator for it [**?**]. However, including a separate approximator for the soft value can bring stability to the training process and additional convenience for training simultaneously with other networks [**?**]. In this context, the soft value function can be trained to minimise the squared residual error, as depicted in equation 2.22 [**?**].

$$J_V(\psi) = \mathbb{E}_{S_t \sim \mathcal{D}} \left[\frac{1}{2} \Big(V_\psi(S_t) - \mathbb{E}_{A_t \sim \pi_\phi}[Q_\theta(S_t, A_t) - \log \pi_\phi(A_t|S_t)] \Big)^2 \right] \tag{2.22}$$

Where $\mathcal{D}$ is a distribution, or a replay buffer, containing previously sampled states and actions. The gradient of equation 2.22 can be calculated using an unbiased estimator as observed in equation 2.23, where the actions are directly sampled from policy instead of the replay buffer [?].

$$\hat{\nabla} J_V(\psi) = \nabla_\psi V_\psi(s) \Big(V_\psi(s) - Q_\theta(s,a) + \log \pi_\phi(a|s) \Big) \tag{2.23}$$

The parameters of the soft Q-function are trained to minimize the soft bellman residual. The objective function can be illustrated by equation 2.25, and can be optimized with stochastic gradients through equation 2.25. SAC makes use of two independently trained Q-functions to mitigate positive bias in the policy improvement step. The minimum Q-value is then taken between the two Q-function approximators, which was found to accelarate training in particularly complex tasks [?, ?].

$$J_Q(\theta) = \mathbb{E}_{(S_t, A_t) \sim \mathcal{D}} \left[\frac{1}{2} \Big(Q_\theta(S_t, A_t) - \hat{Q}(S_t, A_t) \Big)^2 \right] \tag{2.24}$$

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s,a) \Big(Q_\theta(s,a) - r(s,a) - \gamma V_{\overline{\psi}}(s') \Big) \tag{2.25}$$

Where the target $\hat{Q}$ is given by equation 2.26.

$$\hat{Q}(s,a) = r(a,s) + \gamma\, \mathbb{E}_{S_{t+1} \sim p}[V_{\overline{\psi}}(S_{t+1}) \mid S_t = s] \tag{2.26}$$

Regarding the policy, a reparametrization trick is applied using a neural network transformation illustrated in equation 2.27, where ε_t is an input noise vector sampled from a fixed distribution $\mathcal{N}$ [?, ?]. In this manner, the equations for the policy objective and respective gradient estimation can be defined as the following equations 2.28 and 2.29 [?, ?].

$$a = f_\phi(\varepsilon_t; s) \tag{2.27}$$

$$J_\pi(\phi) = \mathbb{E}_{S_t \sim \mathcal{D}, \varepsilon_t \sim \mathcal{N}} \left[\log \pi_\phi(f_\phi(\varepsilon_t; S_t)|S_t) - Q_\theta(S_t, f_\phi(\varepsilon_t; S_t)) \right] \tag{2.28}$$

$$\hat{\nabla}_\theta J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a|s) + \Big(\nabla_a \log \pi_\phi(a|s) - \nabla_a Q(s,a)) \nabla_\phi f_\phi(\varepsilon_t; s) \tag{2.29}$$

The complete model can be described by algorithm 1. The general method cycles between collecting experience from the environment with the current policy and updating the approximators using the stochastic gradients from the minibatches sampled from the replay buffer [?].

Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\overline{\psi}$, $\theta$, $\phi$
for each iteration do
    for each environment step do
        $a \sim \pi_\phi(a, s)$
        $s' \sim p(s'|s, a)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, r(s, a), s')\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\overline{\psi} \leftarrow \tau\psi + (1-\tau)\overline{\psi}$
    end for
end for
```

2.4 Graph Representation Learning

Several objects and problems can be naturally expressed in the real world using graphs, such as social networks, power grids, transportation networks, recommendation systems or drug discovery. The usefulness of such representations is tied to how they instinctively represent the complex relationships between objects. However, graph data is often very sparse and complex, and their sophisticated structure is difficult to deal with [**?**, **?**].

Furthermore, the performance of machine learning models strongly relies not only on their design but also on good representations of the underlying information [**?**]. Ineffective representations, on the one hand, can lack important graph features and, on the other, can carry vast amounts of redundant information, affecting the algorithms' performance in leveraging the data for different analytical tasks [**?**, **?**].

In this context, **Graph Representation Learning** studies how to learn the underlying features of graphs to extract a minimal but sufficient representation of the graph attributes and structure [**?**, **?**, **?**]. Currently, the improvements in deep learning allow representation learning techniques consisting of the composition of multiple non-linear transformations that yield more abstract and, ultimately, more useful representations of graph data [**?**].

2.5 Graph Neural Networks

In the present, deep learning and ANNs have become one of the most prominent approaches in Artificial Intelligence research [**?**]. Approaches such as recurrent neural networks and convolutional networks have achieved remarkable results on Euclidean data, such as images or sequence data, such as text and signals [**?**]. Furthermore, techniques regarding deep learning applied to graphs have also experienced rising popularity among the research community, more specifically **Graph**

Neural Networks (GNNs) that became the most successful learning models for graph-related tasks across many application domains [**?**, **?**].

The main objective of GNNs is to update node representations with representations from their neighbourhood iteratively [**?**]. Starting at the first representation $H^0 = X$, each layer encompasses two important functions:

- **Aggregate**, in each node, the information from their neighbours
- **Combine** the aggregated information with the current node representations

The general framework of GNNs, outlined in [**?**], can be defined mathematically as follows:

Initialization: $H^0 = X$
For $k = 1, 2, \ldots, K$

$$a_v^k = \text{AGGREGATE}^k\{H_u^{k-1} : u \in N(v)\}$$
$$H_v^k = \text{COMBINE}^k\{H_u^{k-1}, a_v^k\}$$

Where $N(v)$ is the set of neighbours for the v-th node. The node representations H^K in the last layer can be treated as the final representations, which sequentially can be used for other downstream tasks [**?**].

2.5.1 Graph Convolutional Network

A **Graph Convolutional Network (GCN)** [**?**] is a popular architecture of GNNs praised by its simplicity and effectiveness in a variety of tasks [**?**, **?**]. In this model, the node representations in each layer are updated according to the following convolutional operation:

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^kW^k) \tag{2.30}$$

$\tilde{A} = A + I$ - Adjacency Matrix with self-connections

$I \in \mathbb{R}^{N \times N}$ - Identity Matrix

$\tilde{D}$ - Diagonal Matrix, with $\tilde{D}_{ii} = \sum_j {}_{ij}$

σ - Activation Function

$W^k \in \mathbb{R}^{F \times F'}$ - Laywise linear transformation matrix

F, F' - Dimensions of node representations in the k-th and $(k+1)$ layer, respectively

$W^k \in \mathbb{R}^{F \times F'}$ is a layerwise linear transformation matrix that is trained during optimization [**?**]. The previous equation 2.30 can be dissected further to understand the *AGGREGATE* and *COMBINE* function definitions in a GCN [**?**]. For a node i, the representation updating equation can be reformulated as:

$$H_i^k = \sigma(\sum_{j \in \{N(i) \cup i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) \tag{2.31}$$

$$H_i^k = \sigma(\sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) + \frac{1}{\tilde{D}_i} H^{k-1} W^k) \tag{2.32}$$

In the second equation, the *AGGREGATE* function can be observed as the weighted average of the neighbour node representations [?]. The weight of neighbour j is defined by the weight of the edge (i, j), more concretely, A_{ij} normalized by the degrees of the two nodes [?]. The *COMBINE* function consists of the summation of the aggregated information and the node representation itself, where the representation is normalized by its own degree [?].

Spectral Graph Convolutions

Regarding the connection between GCNs an spectral filters defined on graphs, spectral convolutions can be defined as the multiplication of a node-wise signal $x \in \mathbb{R}^N$ with a convolutional filter $g_\theta = diag(\theta)$ in the *Fourier domain* [?, ?], formally:

$$g_\theta \star \mathrm{x} = U_{g_\theta} U^T \mathrm{x} \tag{2.33}$$

$\theta \in \mathbb{R}^N$ - Filter parameter

U - Matrix of eigenvectors of the normalized graph Laplacian Matrix $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

The eigendecomposition of the Laplacian matrix can also be defined by $L = U \Lambda U^T$ with Λ serving as the diagonal matrix of eigenvalues and $U^T \mathrm{x}$ is the graph Fourier transform of the input signal x [?]. In a practical context, g_θ is the function of eigenvalues of the normalized graph Laplacian matrix L, that is $g^\theta(\Lambda)$ [?, ?]. Computing this is a problem of quadratic complexity to the number of nodes N, something that can be circumvented by approximating $g_\theta(\Lambda)$ with a truncated expansion of Chebyshev polynomials $T_k(x)$ up to K-th order [?, ?]:

$$g_{\theta'}(\Lambda) = \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{2.34}$$

$\tilde{\Lambda} = \frac{2}{\lambda_{\mathbf{max}}} \Lambda - \mathbf{I}$

$\lambda_{\mathbf{max}}$ - Largest eigenvalue of L

$\theta' \in \mathbb{R}^N$ - Vector of Chebyshev coefficients

$T_k(x)$ - Chebyshev polynomials

$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$

By combining this with the previous equation, the first can be reformulated as:

$$g_\theta \star \mathrm{x} = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\mathrm{x} \tag{2.35}$$

$\tilde{L} = \frac{2}{\lambda_{\max}} L - I$

From this equation, it can be observed that each node depends only on the information inside the K-th order neighbourhood and with this reformulation, the computation of the equation is reduced to $O(|\xi|)$, linear to the number of edges ξ in the original graph G.

To build a neural network with graph convolutions, it's sufficient to stack multiple layers defined according to the previous equation, each followed by a nonlinear transformation. However, the authors of GCN [?] proposed limiting the convolution number to $K = 1$ at each layer instead of limiting it to the explicit parametrization by the Chebyshev polynomials. This way, each level only defines a linear function over the Laplacian Matrix L, maintaining the possibility of handling complex convolution filter functions on graphs by stacking multiple layers [?, ?]. This means the model can alleviate the overfitting of local neighbourhood structures for graphs whose node degree distribution has a high variance [?, ?].

At each layer, it can further considered $\lambda_{\max} \approx 2$, which the neural network parameters could accommodate during training [?]. With these simplifications, the equation is transformed into:

$$g_{\theta'} \star \mathrm{x} \approx \theta'_0 \mathrm{x} + \theta'_1 \mathrm{x}(L - I_N)\mathrm{x} = \theta'_0 \mathrm{x} - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{2.36}$$

θ'_0, θ'_1 - Free parameters that can be shared over the entire graph

The number of parameters can, in practice, be further reduced, minimising overfitting and minimising the number of operations per layer as well [?] as equation 2.37 entails.

$$g_\theta \star \mathrm{x} \approx \theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})\mathrm{x} \tag{2.37}$$

$\theta = \theta'_0 = -\theta'_1$

One potential problem is the $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ matrix whose eigenvalues fall in the $[0,2]$ interval. In a deep GCN, the repeated utilization of the above function often leads to an exploding or vanishing gradient, translating into numerical instabilities [?, ?]. In this context, the matrix can be further renormalized by converting $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ into $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ [?, ?]. In this case, only the scenario where there is one feature channel and one filter is considered which can be then generalized to an input signal with C channels $X \in \mathbb{R}^{N \times C}$ and F filters (or hidden units) [?, ?]:

$$H = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X W \tag{2.38}$$

$W \in \mathbb{R}^{C \times F}$ - Matrix of filter parameters

H - Convolved Signal Matrix

2.5.2 Graph Attention Network

GAT [?] is another type of GNNs that focuses on leveraging an attention mechanism to learn the importance of a node's neighbours. In contrast, the GCN uses edge weight as importance, which may not always represent the true strength between two nodes [?, ?].

The Graph Attention Layer defines the process of transferring the hidden node representations at layer $k-1$ to the next node presentations at k. To ensure that sufficient expressive power is attained to allow the transformation of the lower-level node representations to higher-level ones, a linear transformation $W \in \mathbb{R}^{F \times F'}$ is applied to every node, followed by the self-attention mechanism, which measures the attention coefficients for any pair of nodes through a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ [?, ?]. In this context, relationship strength e_{ij} between two nodes i and j can be calculated by:

$$e_{ij} = a(WH_i^{k-1}, WH_j^{k-1}) \tag{2.39}$$

$H_i^{k-1} \in \mathbb{R}^{N \times F'}$ - Column-wise vector representation of node i at layer $k-1$ (N is the number of nodes and F the number of features per node)

$W \in \mathbb{R}^{F \times F'}$ - Shared linear transformation

$a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ - Attentional Mechanism

e_{ij} - Relationship Strength between nodes i and j

Theoretically, each node can attend to every other node on the graph, although it would ignore the graph's topological information in the process. A more reasonable solution is to only attend nodes in the neighbourhood [?, ?]. In practice, only first-order node neighbours are used, including the node itself, and to make the attention coefficients comparable across the various nodes, they are normalized with a *softmax* function:

$$\alpha_{ij} = \text{softmax}_j(\{e_{ij}\}) = \frac{exp(e_{ij})}{\sum_{l \in N(i)} exp(e_{il})}$$

Fundamentally, α_{ij} defines a multinomial distribution over the neighbours of node i, which can also be interpreted as a transition probability from node i to each node in its neighbourhood [?]. In the original work [?], the attention mechanism is defined as a single-layer Feedforward Neural Network that includes a linear transformation with weigh vector $W_2 \in \mathbb{R}^{1 \times 2F'}$ and a LeakyReLU nonlinear activation function with a negative input slope $\alpha = 0.2$ [?, ?]. More formally, the attention coefficients are calculated as follows:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(W_2[WH_i^{k-1} || WH_j^{k-1}]))}{\sum_{l \in N(i)} \exp(\text{LeakyReLU}(W_2[WH_i^{k-1} || WH_l^{k-1}]))} \tag{2.40}$$

$||$ - Vector concatenation operation

The novel node representation is a linear composition of the neighbouring representations with weights determined by the attention coefficients [**?**, **?**], formally:

$$H_i^k = \sigma(\sum_{j \in N(i)} \alpha_{ij} W H_j^{k-1}) \tag{2.41}$$

Multi-head Attention

Multi-head attention can be used instead of self-attention, determining a different similarity function over the nodes. An independent node representation can be obtained for each attention head according to the equation bellow [**?**, **?**]. The final representation is a concatenation of the node representations learned by different heads, formally:

$$H_i^k = \Big\|_{t=1}^{T} \sigma(\sum_{j \in N(i)} \alpha_{ij}^t W^t H_j^{k-1})$$

T - Number of attention heads

α_{ij}^t - attention coefficient computed from the t-th attention head

W^t - Linear transformation matrix of the t-th attention head

Lastly, the author also mentions that other pooling techniques can be used in the final layer for combining the node representations from different heads, for example, the average node representations from different attention heads [**?**, **?**].

$$H_i^k = \sigma(\frac{1}{T} \sum_{t=1}^{T} \sum_{j \in N(i)} \alpha_{ij}^t W^t H_j^{k-1}) \tag{2.42}$$

2.6 Smart Grid Services

Given the global ecological emergency and the increasing energetic crisis, there is a necessity for advancements in energy distribution and transmission systems now more than ever. To fulfil the need for energy sustainability, traditional centralized distribution grids must be adapted to, on the one hand, accommodate the rise of distributed renewable energy sources in corporate and domestic consumers and, on the other, to make more efficient and reliable distribution of energy resources [**?**, **?**].

The *Smart Grid* or the *Smart Power Grid* conceptualizes this modernization of the electricity network by leveraging the technological advancements in information technology and communication science to create intelligent systems that manage and monitor the distributed generation of energy [**?**, **?**]. Figure 2.5 describes the smart grid pyramid, which has asset management at its base. On this foundation, the foundation of the smart grid is laid out by the circuit topology, IT systems and telecommunications infrastructure, the basic ingredients for the emergence of fundamental applications such as smart meters and distribution automation [**?**]. In turn, these serve as building

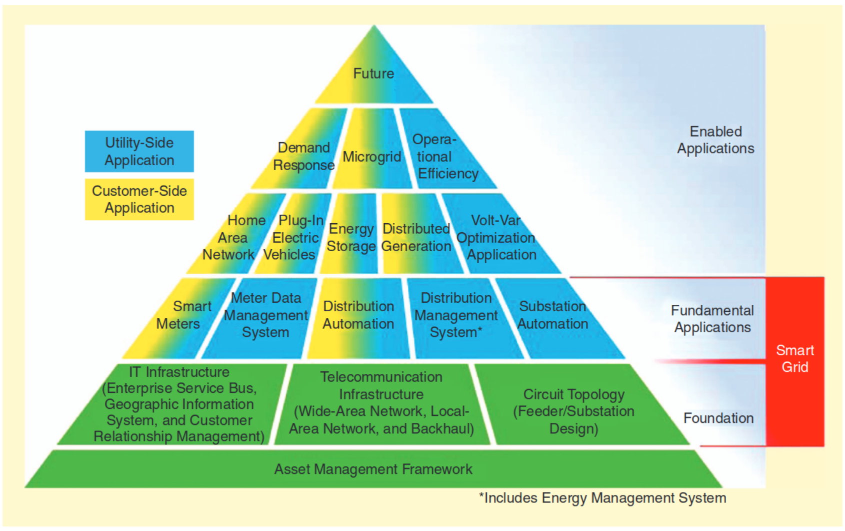
Future
Demand Response
Microgrid
Opera-tional Efficiency
Utility-Side Application
Customer-Side Application
Home Area Network
Plug-In Electric Vehicles
Energy Storage
Distributed Generation
Volt-Var Optimization Application
Smart Meters
Meter Data Management System
Distribution Automation
Distribution Management System*
Substation Automation
IT Infrastructure (Enterprise Service Bus, Geographic Information System, and Customer Relationship Management)
Telecommunication Infrastructure (Wide-Area Network, Local-Area Network, and Backhaul)
Circuit Topology (Feeder/Substation Design)
Asset Management Framework
Enabled Applications
Fundamental Applications
Foundation
Smart Grid
*Includes Energy Management System

Figure 2.5: Smart Grid Capabilities Pyramid [?]

blocks for creating more intelligent systems that leverage upper-layer applications, enabling the true smart grid capabilities [?].

Chapter 3

Literature Review

In this chapter, the literature review regarding GRL approaches and DED systems is documented. This unit is divided into three sections with the first exposing the research methodology and the others reviewing GRL approaches and the DED Systems, the second is further subdivided into plain GCN, attention-based and other approaches.

3.1 Research Methodology

This literature review focuses on analysing the existent literature around the main objective of this dissertation, which is improving GRL techniques in the context of this work's application domain, smart grid services. In this manner, the main research questions are presented as:

- RQ1 - *How can RL algorithms be adapted to effectively solve sequential decision-making problems on graph-based environments?*
  - RQ1.1 - *What are the existent GRL approaches?*
  - RQ1.2 - *What are their limitations?*
- RQ2 - *How can GRL algorithms improve smart grid services?*
  - RQ2.1 - *How can GRL algorithms improve Dynamic Economic Dispatch (DED) in power distribution grids?*

These interrogations aggregate the relevant topics we aim to address in this review. The main requirement on the analyzed literature was to only consider research from the past five years, with the exception of seminal works. We also exclusively considered literature published in scientific conferences and top-tier journals.

In this manner, the initial exploratory research was conducted, primarily using *Scopus* and *Web of Science* for searching the published literature and alternatively using *Google Scholar* for finding cross-references when necessary. The research questions were translated into fundamental search queries where the research process was based.

- `"Graph Reinforcement Learning" OR "Reinforcement Learning on Graphs"`
- `"Reinforcement Learning" AND "Power" AND "Dispatch"`
- `"Graph Reinforcement Learning" OR "Reinforcement Learning on Graphs" AND "Power" AND "Dispatch"`

The gathered literature was screened and the relevant works were thoroughly reviewed, the following sections present the main findings.

3.2 Graph Reinforcement Learning Approaches

GRL or Reinforcement Learning on Graphs is a relatively new area in the broader field of machine learning. GRL techniques have shown significant progress in solving problems with underlying graph-based representations such as power grid management [?, ?], smart transport [?, ?] or task offloading [?, ?]. In this work, the main focus lies on studying the development of GRL techniques and subsequent application to smart grid services such as dynamic economic energy dispatch systems [?, ?], residential electricity behaviour identification and energy management [?], or Volt-VAR regulation [?].

Research on this topic has significantly increased in the last few years with the improvements of DRL techniques and the developments in GNNs in the mid-2010s [?, ?, ?, ?]. GNNs became the state-of-the-art for solving numerous data mining tasks involving graph-structured data, excelling at classification, link prediction and representation learning [?, ?]. This advancement brought more sophisticated RL applications on graphs and the surge of a new field studying how to combine the improvements of graph mining and reinforcement learning techniques [?, ?].

3.2.1 Plain GCN-Based GRL

A common approach in Graph Reinforcement Learning model implementation is the use of graph convolutions with the GCNs architecture for leveraging graph-based structures to extract and aggregate the essential features of data in hand and improve the performance of RL agents in those environments. The techniques listed in this subsection constitute approaches that integrate a GCN with RL algorithms. The gathered literature can be observed in table 3.1.

[?] implements a GRL system to improve the decision quality of economic dispatch under high penetration of distributed energy generations. To accomplish this, a SAC system is employed with the main objective of finding the optimal action policy for minimizing generation cost with the appropriate reliability concerns. This problem is represented by an undirected graph with nodes describing the power grid elements with their respective attributes and edges describing the underlying energy connections between those units. To extract the structural features of the graph, this work implements a full connected layer to perform feature transformation with a two-layer GCN followed by three full connected layers for the non-linear mapping of state-to-action policy in both actor and critic modules. [?] develops a similar approach, with both concluding that it significantly reduces learning time for achieving better training results in comparison to plain SAC and showing significant improvement on economy and flexibility of the system on more complex and sparse state graphs. The use of GCNs enables the system to adapt to changes in the state space by leveraging the network's generalization ability.

In [**?**] a three-layer GCN is used to extract node feature and graph topology information and is integrated into a Rainbow-based [**?**] DRL algorithm for electric vehicle charging guidance. In this article, the testing results show promising performance in reducing operation costs for electric vehicle users, portraying a model with good generalization ability in untrained scenarios. This work and [**?**] further tested their proposed model's performance when inducing topology changes, yield promising results in the adaptability of the two GRL algorithms in scenarios where the environment suffers dynamic changes.

Another interesting implementation of this approach is [**?**], which studies and compares different solutions for optimizing autonomous exploration under uncertain environments. It analyses combinations of a single agent Deep Q-Network (DQN) and Advantageous Actor-Critic (A2C) with Graph Convolutional Networks, Gated Graph Recurrent Networks and Graph U-Nets. The algorithms are executed in test cases of various sizes and the paper reports that the GCN-DQN was the model that achieved the highest reward during policy training, followed by the GGNN-A2C model, although in the end, it concludes that the second showed improved scalability in relation to the first model. Other similar approaches include [**?**] for residential electricity behaviour identification and energy management with a behaviour correlation graph and [**?**] for line flow control in a power grid simulation.

The reviewed literature fails to consider methods for analysing the scalability of proposed models in relatively larger scenarios, except [**?**], with some proposing this as relevant future work [**?**, **?**]. Beyond that, only [**?**], [**?**] [**?**] defined methods for evaluating the performance of the algorithms under topology variations. This shows a gap for more complete studies in plain GCN-based approaches that also focus on studying the scalability of GRL to large scenarios and the adaptability of the models to dynamic topology variations. In most approaches, namely in [**?**, **?**, **?**, **?**], the presented results portray GRL techniques as significantly more effective than plain DRL models without a GCN already suggesting that these techniques are a potential solution for solving sequential decision-making problems with graph-based environments.

Table 3.1: GCN-Based GRL Literature

Reference	DRL Algorithm	GNN Algorithm	Application Domain
[?]	MDP	GCN	Interpret GNNs at model-level
[?]	DDPG	GCN	Automatic transistor sizing
[?]	A3C	GCN	Automatic Virtual Network Embeddings
[?]	Deep Q-Learning	GCN	Task Offloading in Edge Computing
[?]	Rainbow	GCN	Electrical Vehicle Charging Guidance
[?], [?]	SAC	GCN	Dynamic economic energy dispatch
[?]	SAC	GCN	Multi-access Edge Computing
[?]	DQN	GCN	Line flow control
[?]	DQN	GCN	Autonomous Exploration under uncertainty
[?]	DQN	GCN	Residential electricity behavior identification and energy management

3.2.2 Attention-based GRL

Another effective approach in extracting relevant topology and graph features relies on using attention mechanisms to weigh different nodes' contributions dynamically. While this encompasses techniques that use the GAT architecture, which is a GNN design with the attention mechanism at its core, various scholars propose GCN approaches integrated with attention mechanisms such as [?] and [?]. In the top part of table 3.2 the single-agent reviewed attention-based approaches can be observed, while at the bottom some relevant multi-agent approaches are listed.

[?] proposes a DDPG-based algorithm improved with a GAT block with three graph Attention Layers for extracting and learning the topology information for achieve real-time optimal scheduling for Active Distribution Networks (ADNs). This paper compares the obtained test results against a GCN-DDPG model and shows increased performance over the GCN method in reducing cost and power loss. Beyond this, the work demonstrates that the GAT's attention mechanism enables the algorithm to focus on more important nodes and improve the signal-to-noise ratio compared to its GCN counterpart. [?] and propose a multi-agent approach to the same domain but more focused on voltage regulation with a multi-agent SAC instead of a single-agent DDPG algorithm.

In [?], another model for the electric vehicle charging guidance is proposed, consisting of a bi-level approach of a Rainbow-based algorithm with a GAT block. The upper level focuses on the decision-making process regarding charging, while the lower level handles routing. The proposed model proved to be more effective than a shortest distance path-based [?] and a DRL-based [?] approach. [?] develops a similar approach with a Double-prioritized DQN for the same application domain. In [?] and [?], the sequential distribution system restoration problem is addressed with a multi-agent RL algorithm equipped with a GCN with an attention mechanism. In the first case, multi-head attention is used as the convolution kernel for the GCN with a DQN algorithm. In the second, self-attention is used for improving the centralized training of the used multi-agent actor-critic algorithm, more concretely, by embedding it in the critic networks. At the same time, the GCN is integrated into the actor networks for extracting the graph features. Both solutions proved more efficient than traditional RL techniques, with the first highlighting its solution generalization ability and the second showing increased scalability facing the non-GRL techniques. In general, the literature regarding attention-based approaches is a lot sparser and scarcer than GCN-based approaches. Only three relevant single-agent works were found [?, ?, ?] which might either suggest that scholars have a slight preference for multi-agent systems when implementing GRL with attention mechanisms or that these approaches in single-agent systems still require further research. Nevertheless, some of the works showed models with good adaptability to topology variations [?, ?, ?] and scalability to large scenarios [?, ?, ?, ?]. Notably, some works suggest directly embedding the GNN in th RL framework to boost the methodology computation performance and robustness [?, ?].

Table 3.2: Attention-Based GRL Literature

Reference	**DRL Algorithm**	**GNN Algorithm**	**Application Domain**
[?]	DDPG	GAT	Optimal Scheduling for ADNs
[?]	Rainbow	GAT	Electric Vehicle Charging Guidance
[?]	DQN	GAT	Electric Vehicle Charging Guidance
[?]	DQN	GCN	Multi-agent Sequential Distribution System Restoration
[?]	DQN	GCN	Active Power Rolling Dispatch
[?]	AC	GCN	Multi-agent Service Restoration
[?]	SAC	GAT	Multi-agent Voltage Regulation

3.2.3 Other Approaches

This subsection includes other relevant and promising GRL approaches that combine of other GNNs architectures with RL algorithms . In [?], a GraphSAGE network [?] is used with a

Deep Dueling Double Q-Network (D3QN) for emergency control of Undervoltage load shedding for power systems with various topologies. The author presents promising results for the GraphSAGE-D3QN model compared to a GCN-D3QN, achieving higher cumulative reward and faster voltage recovery speed, although it required longer decision times. The proposed model performed excellently in the application domain and successfully generalized the learned knowledge to new topology variation scenarios. Another approach that showed good performance with the GraphSAGE architecture was [?] in the context of the dynamic economic dispatch problem.

[?] focused on solving the Job shop scheduling problem through a priority dispatching rule with a Graph Isomorphism Network (GIN) [?] and an actor-critic PPO algorithm where the GIN is shared between actor and critic networks. The method showed superior performance against other traditional manually designed priority dispatching rule baselines, outperforming them by a large margin.

Table 3.3: Other GRL Approaches

Reference	**DRL Algorithm**	**GNN Algorithm**	**Application Domain**
[?]	DQN	GraphSAGE	Undervoltage Load Shedding
[?]	PPO	GraphSAGE	Dynamic Economic Dispatch
[?]	PPO	GIN	Dispatch for Job Shop Scheduling

3.3 Dynamic Economic Dispatch Systems

RL algorithms are already regarded as a well established potential solution for solving economic dispath problems [?]. In table 3.4 the relevant RL approaches to the problem, including GRL techniques, can be observed. In a general level, the reviewed literature regarding DED solutions take into account a wide range of considerations and constraints. The recent works show an almost ubiquitous study of Energy Storage Systems (ESSs) management and Renewable Energy Source (RES) curtailment, given the current global energetic issues and the relevance of these technologies for solving them.

Some systems [?, ?, ?] take further steps in ensuring grid stability by also considering voltage deviations while a notable paper considers battery degradation when calculating the cost of dispatch [?]. In general, GRL algorithms show superior performance in relation with plain DRL approaches [?, ?, ?] as also concluded in section 3.2.1, with studies successfully ensuring adaptability of the GRL models to variations in the scenario's topology [?, ?]. However, only [?] conducts tests in test cases of different sizes, which shows a gap for studies addressing the scalability limitations of the developed models.

Table 3.4: Dynamic Economic Dispatch RL Systems

Reference	Approach	Application
[?]	DDPG with Prioritized Experience Replay Mechanism and L2 Regularization	Integrated Energy Systems, Utility (Selling, Purchasing and Gas) and ESSs
[?]	DDPG	Thermal Power, Renewable Energy Sources and ESSs
[?]	SAC with Imitation learning	Thermal Power, Renewable Energy Sources and Voltage deviation
[?]	GCN-SAC with Replay Buffer	Thermal Power, Renewable Energy Sources, ESSs and Voltage Deviation
[?]	GCN-SAC	Utility (Time-of-Use), Thermal Power, Renewable Energy Sources and ESSs
[?]	GraphSAGE-PPO	Thermal Power, Renewable Energy Sources, ESSs and Voltage Deviation
[?]	Multi-Agent RL with Function Approximation and Diffusion Mechanism	Utility (Time-of-Use), Thermal Power (Diesel) and ESSs (Considering degradation)
[?]	Multi-Agent GAT-DQN	Thermal Power, Renewable Energy Sources and ESSs

3.4 Conclusions

In this chapter, we reviewed relevant literature for this dissertation's main research topic, GRL algorithms. GRL is very promising field, where several different applications and techniques were already studied. GNNs architectures such as GCN have been extensively applied with DRL algorithms for enabling feature extracting from graph-based state representations [?, ?]. Architectures such as the GraphSAGE and other attention-based have also been successfully applied with very promising results [?, ?] in comparison with GCNs. However, less research regarding their integration with DRL algorithms was discovered. This suggests that a possible improvement and research direction in the development of GRL techniques might be connected with exploring the use of different GNNs architectures and using the rising attention-based techniques.

- RQ1 - **How can RL algorithms be adapted to effectively solve sequential decision-making problems on graph-based environments?**
  - RQ1.1 - **What are the existent GRL approaches?** Existent approaches ubiquitously use DRLs with GNNs for efficiently extract graph features. They can be divided into plain GCN approaches, Attention-based approaches and others. GCNs compromise

a popular method while attention-based approaches showed to be less researched but more promising [**?**, **?**]. Tables 3.1, 3.2 and 3.3 depict the reviewed GCN, attention-based and other implementations, respectively.

  - RQ1.2 - **What are their limitations?** Research already depicts GRL techniques as better performing in relation to plain RL algorithms in graph-based environments, which portrays GRL is a potential solution to this problem [**?**, **?**, **?**, **?**, **?**, **?**]. The reviewed literature proved to be quite scarce and sparse, probably due to the novelty of the field. After a thorough review, we failed to find works that listed specific limitations of GRL algorithms. This suggests the existence of a research gap for works with a thorough and well-documented scientific process as well as comparative and systematic studies between the different approaches, highlighting models performance in large scenarios and under topology variation. Furthermore, some papers suggested the directly embedding of the GNN into the RL framework [**?**, **?**].

- RQ2 - **How can GRL algorithms improve smart grid services?**
  - RQ2.1 - **How can GRL algorithms improve Dynamic Economic Dispatch (DED) in power distribution grids?** RL algorithms are already regarded as a well established potential solution for solving economic dispath problems [**?**]. We were able to find evidence of GRL being successfully implemented in DED systems such as [**?**] and [**?**] in a power grid simulation context, showing efficient performance and scalability in comparison with plain DRL models in the same issue. However, as it was the case with GRL algorithms, the sparseness of different considerations, constraints and formalizations of the DED problem, highlighted in section 3.3 bring added complexity when comparing different approaches, since models are build with different DED requirements.

Chapter 4

Methodological Approach

In this chapter, the problem this work aims to address is formally stated, as well as the proposed solution. In section 4.1 the problem statement is exposed, section lists the solutions main functional, non-functional and structural requirements, section 4.6 addresses the architecture of the solution, section exposes the established methodology and section contains the proposed work plan.

4.1 Problem Statement

The reviewed literature addressing solutions to sequential decision-making problems in graph-based environments is sparse and scarce, leading to a research gap for comparative and systematic GRL approaches analysing scalability under different sized scenarios and adaptability to topology variation.

As addressed in the previous chapter, graphs are ubiquitous representations that can serve to instinctively represent several problems and their objects. In some network-oriented domains, these representations reveal underlying features that can't be naturally represented by plain Euclidean data. This problem becomes even more difficult considering that graph data is complex and sparse, something that brings the need for methods that efficiently extract representations.

By conducting a thorough literature review of the relevant studies in this context, we observed that current RL algorithms are not as efficient as GRL techniques in handling such complex environments, because of not considering and generalizing environment topology features in the decision-making process. This deeply affects the performance of decision systems inserted in network-oriented domains where the intricate relationships between the objects may be relevant for mapping the observable environment states to optimal action policies.

More and more GRL attracts the curiosity of academics, only increasing the relevance of this problem. With the recent advancements of GNNs, the popularity around GRL has risen because of their excellent efficiency in creating optimal graph representations and other graph machine learning problems. However, with GRL being a field whose research is still in initial phase, the gathered literature is very sparse, with a lack of works addressing the benefits, disadvantages and performance of the various proposed models. Moreover, the literature also highlights the importance of studying GRL models in scenarios under topology variations and of different sizes for analysing their scalability.

4.1.1 Scope

This dissertation will focus on studying this problem in the context of single-agent RL algorithms, given that multi-agent systems are significantly more complex to implement. Furthermore, in the context of the dissertation's application domain, which is smart grid services, the possible improvements in GRL techniques will be implemented to the Dynamic Economic Dispatch (DED) problem that studies solutions that optimize power generation cost while maintaining reliable grid stability. Additionally, the GRL proposed models may be also implemented to solve other smart grid systems such as Undervoltage Load Shedding and Volt-VAR Regulation.

4.2 Problem Formalization

In this section, the main problem addressed by this dissertation is formally introduced. Firstly, it's approached from an application domain perspective concerning the DED problem and its considered features and characteristics. In the second subsection 4.2.2, the DED problem is formalized as a dynamic sequential decision-making problem, and its characteristics are presented under the form of its corresponding MDP.

These two formalizations are crucial for giving the appropriate view on how the GRL algorithms and their environment will be approached and studied.

4.2.1 Dynamic Economic Dispatch

Since the early 20th century, the improvement of power distribution grids has greatly improved society's well-being and enabled countless technological advancements. The issue of optimally distributing and accommodating customers' load demands among available power generators in an economic and reliable manner has been studied early on by academics. [**?**]. The Dynamic Economic Dispatch (DED) problem addresses this concern, with a special focus on the economic factor of power systems operation. The main objective of this problem is to find the optimal dispatch strategy for the available generators that minimizes the total operating cost over a dispatch period while conforming to a particular set of constraints, such as fulfilling the total load demand of the power system. In the next subsections 4.2.1.1 and 4.2.1.2, the objective function and constraints considered for the DED problem are formally introduced and further discussed.

$F(t)$	Total Operational Cost at timestep t
$F_{\text{NRES}}(t)$	Total Non-Renewable Generators Operational Cost at timestep t
$F_{\text{RES}}(t)$	Total Renewable Generators Operational Cost at timestep t
T	Terminal Timestep
t	Timestep
c_i^{NRES}	Cost of conventional generator i in €/MWh
7 $P_i^{\text{NRES}}(t)$	Current generated power of non-renewable generator i at timestep t in MW
β_{RES}	RES wasted energy penalty term
$P_i^{\text{RES}}(t)$	Current generated power of renewable generator i at timestep t in MW
$\overline{P^{\text{NRES}}}$, $\underline{P^{\text{NRES}}}$ $\overline{P_i^{\text{RES}}}(t)$	Current maximum and minimum output power of non-renewable generator i Current maximum output power of renewable generator i at timestep t and before curtailment in MW
$P_i^{\text{LOAD}}(t)$	Active Power demand of load i at timestep t
$Q_i^{\text{LOAD}}(t)$	Reactive Power demand of load i at timestep t
F_i or $F_{i,j}$	Powerline i status (connected or disconnected)
$\text{rho}_i(t)$ or $\text{rho}_{i,j}(t)$	Relative flows of powerline i or with origin in substation i and destination in substation j. Ratio of the flow divided by its thermal limit.
$\overline{\eta_i}$, $\underline{\eta_i}$ $P_i^{\text{G}}(t)$	Maximum ramp up and down limits for non-renewable generators Current generated active power of generator i in MW
$Q_i^{\text{G}}(t)$	Current generated reactive power of generator i in MW
N	Number of non-renewable generators
M	Number of renewable generators
K	Number of loads
L	Number of powerlines
θ_{soft}, θ_{hard}	Soft and Hard overflow thresholds
T_{soft}	
$T_{\text{reconnect}}$	
$\overline{F_{\text{NRES}}}$	
$\overline{c^{\text{NRES}}}$	

4.2.1.1 Objective Function

As stated previously, the main goal of the DED problem is to minimise the total cost of a power grid operation while fulfilling the total load demand over a time period T. The operating cost of all dispatchable generators at a timestep t can be described by equation 4.1 and comprises the first component of the DED objective function.

$$F_{\text{NRES}}(t) = \sum_{i=0}^{N} c_i^{\text{NRES}} P_i^{\text{NRES}}(t)\Delta t \tag{4.1}$$

In this context, conventional generation cost is described as the total sum of the generators operating cost, which in turn is described as a linear function dependent on their output value $P_i^{\text{NRES}}(t)$ and static cost c_i^{NRES} in € per MWh. The reduction of the total operation cost of a power system is undoubtedly tied with the minimisation of $F_{\text{NRES}}(t)$, which describes a straightforward economic cost of the operation of a power grid.

However, more and more modern distribution grids include Renewable Energy Source (RES), which also enable the manipulation of their output levels through curtailment actions. These acts allow the power system to adapt to its dynamic needs associated with reliability, stability, and security by adjusting the maximum output of RES. Regardless of its contributions, curtailment does not come without its disadvantages, mainly connected with the hidden cost of curtailed (unused) RES energy that might be transferred to other generators' operations for security reasons, such as preventing line overflow.

In this manner, as a means of maximising the RES usage, a second component of the objective function is considered that focuses on expressing this cost, represented in equation 4.2. A penalty factor β_{RES} controls the influence of wasted RES energy on the overall objective. Additionally, this work considered another element representing the average cost $\overline{c_i^{\text{NRES}}}$ of conventional generators as the cost of the abandoned RES energy. This creates a more understandable and comparable definition of the total operating cost of renewable generators.

$$F_{\text{RES}}(t) = \beta_{\text{RES}} \sum_{i=0}^{M} (\overline{P_i^{\text{RES}}}(t) - P_i^{\text{RES}}(t))\Delta t \tag{4.2}$$

Finally, the primary objective function of DED can be defined by minimising the total sum of both components $F_{\text{NRES}}(t)$ and $F_{\text{RES}}(t)$ for the period T. This is clearly stated in equation 4.3.

$$\min F(t) = \min \sum_{t=1}^{T} (F_{\text{NRES}}(t) + F_{\text{RES}}(t) \tag{4.3}$$

4.2.1.2 Constraints

In order to maintain stability and reliable power distribution, the system must follow several operational constraints. These restrictions are related to generators, voltage, powerlines and overall stability.

The primary constraint considered in DED consists of compliance with power balance. This implies that, at all times, the sum of the production output of all generators must surpass the total system load demand, as portrayed by equation 4.4 [**?**].

$$\sum_{i}^{N} P_i^{\text{NRES}}(t) + \sum_{i}^{M} P_i^{\text{RES}}(t) = \sum_{i}^{L} P_i^{\text{LOAD}}(t) + \delta, \delta > 0 \tag{4.4}$$

Another equally important group of constraints is the Kirchoff's Laws of current and voltage. These fundamental principles establish the foundational rules that govern the behaviour of electrical circuits and power networks. The Kirchoff's voltage law, represented by equation 4.5, determines that the sum of all electrical potentials around a closed circuit loop must equal zero, ensuring that all voltages are consistent and feasible. On the other hand, the Kirchoff's current law, illustrated in equation 4.6, dictates that the sum of all currents entering and leaving any node in a power networks must equal zero to guarantee that the conservation of energy is respected.

$$\sum_{k=1}^{n} V_k = 0 \tag{4.5}$$

$$\sum_{k=1}^{n} I_k = 0 \tag{4.6}$$

Regarding non-renewable generators, their output value is bounded by absolute maximum and minimum values respective to each generator, $\underline{P_i^{\text{NRES}}}$ and $\overline{P_i^{\text{NRES}}}$, as depicted in equation 4.7. Furthermore, as illustrated by equation 4.8, the varying power between two different timesteps is also restricted by the maximum ramp-up and down limits, $\underline{\eta_i}$ and $\overline{\eta_i}$.

$$\forall t, i : \underline{P_i^{\text{NRES}}} \leq P_i^{\text{NRES}}(t) \leq \overline{P_i^{\text{NRES}}} \tag{4.7}$$

$$\forall t, i : \underline{\eta_i} \Delta t \leq P_i^{\text{NRES}}(t+1) - P_i^{\text{NRES}}(t) \leq \overline{\eta_i} \Delta t \tag{4.8}$$

In contrast with these static production limits, renewable generator production has a dynamic maximum output level and is not bounded by ramp rate limits. Depending on the type of RES, its maximum generation output is inherently tied to the environmental conditions that support its operation. These conditions, such as the availability of natural inputs, climate and other local dynamic factors, determine the efficiency and capacity at which the renewable sources operate. In such manner and as equation 4.9 portrays, the output value of renewable generations for any timestep t is bounded by a maximum production value, $\overline{P_i^{\text{RES}}}(t)$ and 0.

$$\forall t, i : 0 \leq P_i^{\text{RES}}(t) \leq \overline{P_i^{\text{RES}}}(t) \tag{4.9}$$

Regarding powerlines, there is several restrictions in their dynamic operation that aim to consider the real behaviour of power systems. There are two main thresholds associated with two real-world scenarios:

- **Hard Overflow** - In case the relative flow of any powerline surpasses the hard overflow threshold, θ_{hard}, it's instantly disconnected. This corresponds to a instantenous overcurrent and prevents the power system from operating in cases of extreme overflow;
- **Soft Overflow** - A line is allowed to surpass the soft overflow threshold, θ_{soft}, for T_{soft} timesteps before being disconnected for safety reasons. This consists in an time overcurrent

and describes the level at which a line is allowed to be in overflow for maximum amount of time;

Regardless of the scenario, when a powerline is disconnected for safety reasons, it's always affected by a reconnection cooldown, $T_{\text{reconnect}}$.

4.2.2 Markov Decision Process (MDP)

In this section, the DED problem is formalized as a sequential decision-making problem and its MDP is uncovered. This is necessary to approach DED as a RL problem and propose an appropriate solution.

Action The considered action space includes the change in conventional generator redispatching ΔP_i^{NRES} and the renewable energy curtailment P_i^{RES}. The first concerns only non-renewable dispatchable generators and is done in increments and decrements. At the same time, the second is the ratio of curtailment applied to the total amount of generation output of a renewable generator.

This second type of action refers to the energy discarded from RES for grid stability and security purposes. However, they're represented as the upper bound on a renewable generator's output level. In this manner, while positive dispatch actions are directly tied with higher energy output and, consequently, the operational cost of the system, curtailment actions have the opposite effect, given that they are represented as the upper bound on renewable generator output levels. Therefore, higher curtailment values reduce the wasted energy provided from RES, making these actions crucial for an optimal grid operation.

$$A = \{P_1^{\text{NRES}}, P_2^{\text{NRES}}, \ldots, P_N^{\text{NRES}}, P_1^{\text{RES}}, P_2^{\text{RES}}, \ldots, P_M^{\text{RES}}\} \tag{4.10}$$

Observation The observation space is composed of four components concerning Generators, RES, Loads and Powerlines. The relevant characteristics were included in the action space highlighted by the following equations:

$$o_1(t) = \begin{bmatrix} P_1^{\text{NRES}} & P_2^{\text{NRES}} & \ldots & P_N^{\text{NRES}} \end{bmatrix} \tag{4.11}$$

$$o_2(t) = \begin{bmatrix} \frac{P_1^{\text{RES}}}{\overline{P}_1^{\text{RES}}} & \frac{P_2^{\text{RES}}}{\overline{P}_2^{\text{RES}}} & \ldots & \frac{P_M^{\text{RES}}}{\overline{P}_M^{\text{RES}}} \end{bmatrix} \tag{4.12}$$

$$o_3(t) = \begin{bmatrix} P_1^{\text{LOAD}} & P_2^{\text{LOAD}} & \ldots & P_K^{\text{LOAD}} \\ Q_1^{\text{LOAD}} & Q_2^{\text{LOAD}} & \ldots & Q_K^{\text{LOAD}} \end{bmatrix} \tag{4.13}$$

$$o_4(t) = \begin{bmatrix} F_1 & F_2 & \ldots & F_L \\ \text{rho}_1 & \text{rho}_2 & \ldots & \text{rho}_L \end{bmatrix} \tag{4.14}$$

$$o(t) = \{o_1(t), o_2(t), o_3(t), o_4(t)\} \tag{4.15}$$

This tuple encompasses the main elements of the power grid and their characteristics without capturing their topology. This is the space considered for pure-DRL baselines, and it was taken from similar literature that addresses DED with equivalent considerations.

Concerning the network's topology, for GRL algorithms, this tuple was replaced by a graph structure representing the power grid configuration. This graph is defined by its adjacency, weight, and feature matrix and its feature tuple for node i at timestep t can be observed in equation 4.16.

$$o_i(t) = \{P_i^{\text{LOAD}}, Q_i^{\text{LOAD}}, P_i^{\text{NRES}}, P_i^{\text{RES}}, \overline{P_i^{\text{RES}}}, t\} \tag{4.16}$$

In this context, the features are aggregated along the existent nodes which may have several connected elements such as loads and generators. The adjacency matrix is derived from *line status* and the weight matrix corresponds to th *rho* value of each line. Finally, a graph representation is obtained from the initial equation 4.15, enabling their exploit and posterior study by GRL algorithms.

$$o(t) = G(\text{Adj}, W, X) \tag{4.17}$$

where,

$$\text{Adj}_{i,j} = F_{i,j} \tag{4.18}$$

$$W_{i,j} = \text{rho}_{i,j} \tag{4.19}$$

$$X_i = o_i \tag{4.20}$$

Reward Considering reward functions, three formalizations were considered. The initial implementation can be described by equation 4.21 and corresponds to the total amount of cost saved at time step t, with the maximum cost, $\overline{F_{\text{NRES}}}$, representing the total cost of running the power system for the current step with all generators running at maximum output.

$$\begin{aligned} r_1(t) &= \overline{F_{\text{NRES}}} - F_{\text{NRES}}(t) \\ &= \overline{F_{\text{NRES}}} - \sum_{i=0}^{N} c_i^{\text{NRES}} P_i^{\text{NRES}}(t)\Delta t \end{aligned} \tag{4.21}$$

This definition aims to encompass the main objective of DED, directly tying reward maximization and the reduction of operation cost. In this manner, the reward represents an estimate of the system expense over a timestep in €. However, it disregards a secondary objective concerning the maximization of RES generated energy.

In this context, another component is added to consider the cost of wasted RES energy, depicted in equation 4.22. This cost is calculated using the average cost per MW of all non-renewable generators of the system, $\overline{c^{\text{NRES}}}$, and a penalty factor, β_{RES}, to control the impact of wasted energy in reward value.

$$\begin{aligned} r_2(t) &= -F_{\text{RES}}(t) \\ &= \overline{c^{\text{NRES}}}\beta_{\text{RES}} \sum_{i=0}^{M} (\overline{P_i^{\text{RES}}}(t) - P_i^{\text{RES}}(t))\Delta t \end{aligned} \tag{4.22}$$

In addition, this component was also idealized as a bonus instead of a penalty. In contrast with the last definition, the maximization of renewable energy can also be considered through a positive reward for its utilization instead of penalizing its waste. This second version of the RES component can be observed in equation 4.23

$$r_2(t) = \overline{c^{\text{NRES}}}\beta_{\text{RES}} \sum_{i=0}^{M} P_i^{\text{RES}}(t)\Delta t \tag{4.23}$$

$$r(t) = r_1(t) + r_2(t) \tag{4.24}$$

Invalid Actions and Game Over In the context of the power system simulation, there are some considerations accounting to invalid actions and errors that cause a terminal state of the system or a *game over*. If at any timestep the power system reaches an infeasible state there is a game over and the episode ends. These happen any time there's no following action that conforms to the contraints of power balance and the Kirchoff's laws. In this case, the reward at the timestep it happens is minimum (-1).

At any other case when actions don't respect the other considered constraints related to the power output, ramp limits and powerlines, the action is considered invalid and the returned reward is zero.

Episodes and Steps Considering the time difference between the two steps, each timestep represents a five-minute period where the environmental characteristics remain constant. Furthermore, the length of an episode corresponds to a week of operation, which in turn translates into 2016 timesteps.

$$\Delta t = 300s \tag{4.25}$$

OS	Arch Linux
Kernel	6.10.6-zen1-1-zen
CPU	Intel i7-9750 @ 4.5 GHz
GPU	Nvidia GeForce GTX 1650 Mobile / Max-Q
GPU Memory	4 GB
Driver Version	555.58.02
CUDA Version	12.5
RAM	16 GB

Table 4.1: MSI Laptop Specifications

$$\text{Episodic Length} = 2016t \tag{4.26}$$

4.3 Device Specifications

The experiments were executed on two different machines, the author's own and a remote server made available by LIACC (*Laboratório de Inteligência Artificial e Ciência de Computadores*). Tables 4.1 and 4.2 highlight their hardware specifications.

4.4 Technologies

Beyond being a well-documented, fast, robust and modular language, *Python* is an ubiquitous technology for implementing machine learning algorithms. Furthermore, the main libraries used to implement the simulation environment and algorithm are native to this language.

In table x, the main requirements for the project can be observed, along with their respective version number and a short description of their purpose and functionality.

Grid2Op is a critical requirement used to simulate the power grid operation in different manageable scenarios. This is further discussed in the next subsection 4.5. Stable-Baselines3 was chosen for its performance, customization and descriptive documentation, while PyTorch Geometric was favoured because of the wide variety of GNNs Implementations included. Additionally, these libraries mix well because stable-baselines3 already uses PyTorch in its DRL implementations.

OS	Ubuntu 20.04.1
Kernel	5.15.0-117-generic
CPU	13th Gen Intel(R) Core i7-13700K @ 5.4 GHz
GPU	NVIDIA RTX A6000
GPU Memory	48 GB
Driver Version	535.183.01
CUDA Version	12.2
RAM	62 GB

Table 4.2: LIACC Server Specifications

Name	**Description**	**Version Number**
Python	Functional Programming Language	3.11.9
Numpy	Numerical Computing Library	1.24.3
Ray	Used for hyperparameter tuning	2.21.0
PyTorch	ML Library used by stable-baselines3	2.3.0
PyTorch Geometric	GNN Implementations	2.5.3
Grid2Op	Power Grid Simulation Platform	1.9.8
Pandas	Data Manipulation and Analysis	1.5.3
LightSim2Grid	Fast Grid2Op Backend	0.8.2
Gymnasium	RL Single-Agent API	0.29.1
Stable Baselines 3	DRL Implementations	2.3.2

Table 4.3: Project Requirements

Lastly, *Gymnasium* is used as the standard RL single-agent API in the entirety of the project, adding robustness, scalability and flexibility to agent-environment interaction process in the implementation.

4.5 Simulation Environment

The *Grid2Op* framework [**?**] will be used for modelling the sequential decision-making process on simulated power distribution grids. *Grid2Op* is designed by RTE (*Réseau de Transport d'Électricité*), the electricity transmission system operator of France, and is equipped with a variety of pre-defined scenarios used in coding competitions and based on real-world data [**?**].

This *python* library implements the creation of power grid environments that emulate a subset of the elements and physical constraints of real power systems. On one hand, the main objects and their interactions are captured realistically, while on the other, the abstraction level used to model the fundamental elements of a power grid facilitates the implementation of intelligent systems in safe and controllable scenarios.

Furthermore, *Grid2Op* models the dynamic topology of power systems as graph-structured data, which also eases the study of how this information can influence decision-making systems performing on these environments. In figure 4.1, a graphical plot of the environment state at an arbitrary timestep can be observed.

This framework is compatible with the *Gymnasium* framework [**?**], a widely used toolkit for developing RL algorithms, which will also will be used in this work together with the Grid2Op simulation environment.

- redispatch - change the production output of non-renewable generators in incrementing or decrementing intervals

- curtail - change the production setpoint of renewable generators bellow the current maximum available power [**?**]

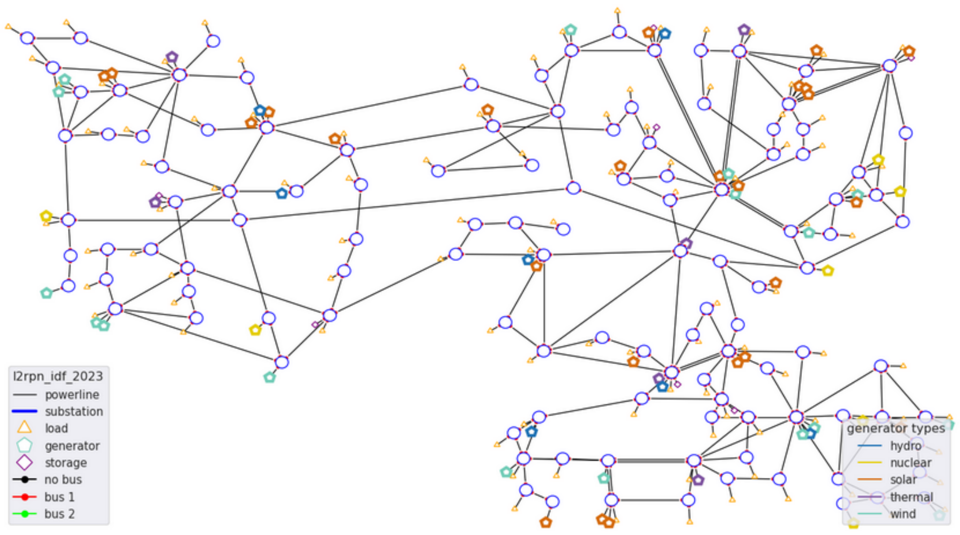

Figure 4.1: Grid2Op *l2rpn_idf_2023* 118-bus test case [**?**]

4.5.1 Elements

The main elements that are modelled by the simulation environment are presented in this subsection. The power grid is reduced to its fundamental components and their properties:

- **Buses** are the fundamental objects of the power grid, representing nodes where power sources, loads and other elements are connected[**?**].
- **Powerlines** represent edges in the power grid and connect the different buses together. They represent the physical transmission and distribution lines and allow power to flow from one part of the grid to another [**?**].
  - status
  - rho
- **Generators** are critical grid elements connected to buses whose main role is to produce power and maintain grid stability by balancing the energy supply and demand. They can be Conventional Thermal Generators, Wind Turbines or Photovoltaic Cells [**?**].
  - $P_i^{\text{NRES}}(t)$ - Power generation at non-renewable generator i
  - $P_i^{\text{RES}}(t)$ - Power generation at renewable generator i
  - $\overline{P_i^{\text{RES}}}(t)$ - Maximum power generation at renewable generator i
  - $\overline{P_i^{\text{NRES}}}, \underline{P_i^{\text{NRES}}}$ - Maximum/Minimum power generation at renewable generator i
- **Loads** consume power from the grid, simulating electricity use. They're also associated to an individual bus [**?**].
  - P_i^{LOAD} - Active power at load i

  - Q_i^{LOAD} - Reactive power at load i

- **Storage Units** can act as both consumers and producers. They're able to retain energy from the power grid when production surpasses demand for later injecting back power when convenient. Storage units are bound by a maximum energy storage capacity [**?**].

Beyond *grid2op* to build and run the test cases and *gymnasium* as the RL framework, this project will use *PyTorch* [**?**] with *PyTorch Geometric* [**?**] library for developing the GNN because of its extensive list of available implemented models. The different combinations of algorithms will be applied to a set of modified scenarios that fulfil the settled requirements. For Deep Reinforcement Learning algorithms, solutions with plain SAC, DDPG and PPO approaches and combined with the GCN, GAT and GIN architectures.

Concerning result analysis, it's also important to point out the use of quantitative methods for evaluating the different implemented models, a topic that is further explored in the following section 4.7.

4.5.2 Parameters

Grid2Op also defines a number of customizable environment parameters that control some very important grid properties and settle some bounds on the agent-envrionment interaction. As there is a significant number of parameters that consider details that are out of the scope of this project, only the relevant parameters are exposed in table 4.4.

Table 4.4: Grid2Op Relevant Parameters

Parameter	**Description**	**Default**
`NO_OVERFLOW_DISCONNECTION`	If `true`, the powerlines over their thermal limit will never be disconnected	`false`
`NB_TIMESTEP_OVERFLOW_ALLOWED`	Number of timesteps a powerline is allowed to be in a soft overflow before it gets disconnected by time overcurrent	2
`NB_TIMESTEP_RECONNECTION`	Number of timesteps that a line disconnected for security reasons will stay disconnected	10
`HARD_OVERFLOW_THRESHOLD`	If the power flow of a line is above the settled hard overflow threshold it's instantaneously and automatically disconnected	2.0
`SOFT_OVERFLOW_THRESHOLD`	If the power flow of a line is above the defined soft overflow threshold for over `NB_TIMESTEP_OVERFLOW_ALLOWED` steps, it's automatically disconnected	1.0
`ENV_DC`	If `true`, the environment will use direct current approximations instead of alternative current	`false`
`LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION`	If set to `true`, the environment will automatically limit curtailment (and storage) actions that otherwise would lead to an infeasible system state. Might help the training and learning process of the model and significantly increases the survivability of the agent	`false`

Most of the parameters, with the exception of one, were left at their default values depicted in table 4.4. This includes the overflow thresholds, all of the timestep limits mentioned and direct current flag, whose values were found to be sensible enough to maintain a realistic yet feasible model of the environment. The only parameter that was tampered was `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE` with the goal of improving the learning efficency of the algorithms during training.

4.5.3 Scenarios

Table 4.5: Test Case Sizes

	l2rpn_case14_sanbox	l2rpn_icaps_2021_large	l2rpn_idf_2023
Buses	14	36	118
Powerlines	20	59	186
Generators	6	22	62
Loads	11	37	99
Episodes	1004	2952	832

This work will use the pre-defined *grid2op l2rpn_case14_sandbox*, *l2rpn_icaps_2021_large* and *l2rpn_idf_2023* test environments. The scenarios define the properties and characteristics of loads and generators at each time step, as well as the grid layout [?] for weekly episodes. The data from the scenarios is non-continuous, with each episode being independent of the others in its original sequence (and they are also shuffled in practice). The test cases' characteristics can be further observed in table 4.5.

The first case was used for the primary implementation and development of the algorithms, given its small grid size. Motivated by its considerable grid size and amount of training data, the second test case was mainly used for the calibration of the model's main parameters. It's a subset of the original 118-bus system [?] with 50 years' worth of data divided into independent [1] weekly scenarios. Finally, the third scenario is based on the same original test case as the previous one, and it includes modifications that aim to accommodate the *possible energy mix* of France in 2035, containing 16 years of data [?]. It was mainly used to study the algorithm's scalability and performance in larger scenarios in comparison with the baseline plain-DRL model.

4.6 Solution Architecture

Considering the conclusions taken from the reviewed literature, in chapter 3, our proposed solution combines efficient Deep Reinforcement Learning (DRL) approaches with Graph Neural Networks (GNNs), the state-of-the-art approach for graph representation learning. Generally, the system receives graph-based representations of the environment and encodes them using the GNN algorithm. By also leveraging deep learning techniques, DRL maps the encoded embeddings to optimal action sequences with the goal of meeting the real-time load demand and reducing the operating cost of the power distribution grid. The general architecture of the solution can be observed in figure 4.2.

[1] non-consecutive

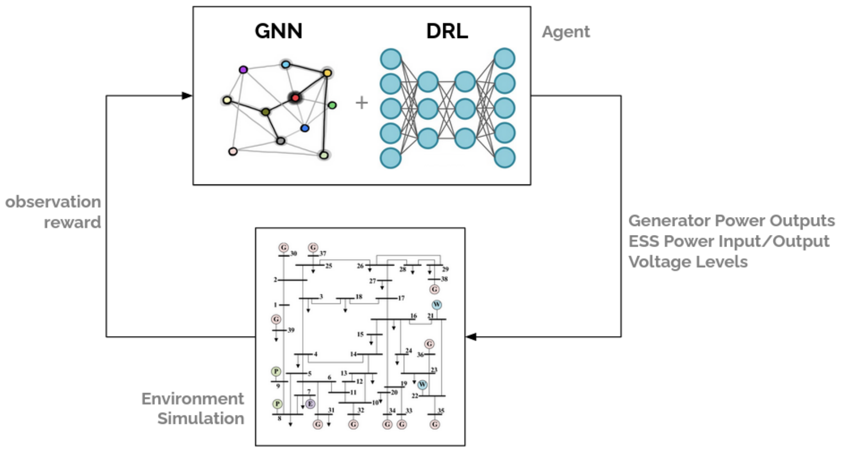

Figure 4.2: Solution Architecture

To fulfil this, the agent adjusts the generator power output and voltage levels and manages the ESS operation in real-time. The proposed solution is trained and tested in a power grid simulation that models the appropriate elements, properties, constraints and operation of the power distribution grid.

In order to achieve this purpose, the GRL models were implemented using *stable-baselines3* DRL implementations and *torch_geometric* GNN models. Beyond having several different implementations available for both their respective types of algorithms, utilizing these two technologies simplifies the integration of models from each, allowing for more straightforward testing of various combinations.

Although stable-baselines3 allows for the implementation of custom feature extractors, this method was not computationally optimal. DRL models of this library that use mini-batching have an increased complexity due to the extractor being called on each batch observation at every step. This issue is derived from another problem which is obviously the storage of raw observations in the replay buffer, where the data is batched from, instead of the extracted features.To solve this, the GNN feature extraction layer was included directly in the observation space which ensured the execution of module only once per step.

4.7 Evaluation Metrics

The solution described in the previous subsection 4.6 involves intricate operational mechanisms, something that calls for a sophisticated evaluative and test process. Furthermore, given this dissertation's comparative nature, the evaluation and analysis methods will be critical factors in studying and confronting the different combinations of GNNs and DRL, as well as possible improvements in integrating these techniques. In this manner, we define the six dimensions for evaluating and analysing the different GRL models:

Learning efficiency This dimension assesses how effectively the models learn and improve their decision-making process over time. It involves evaluating how quickly they converge to optimal or near-optimal dispatch strategies through the convergence rate. This is mainly assessed by analysing the evolution of episodic accumulative reward over the training process.

Dispatch Efficiency The performance of the GRL model in managing power distribution from the various generators will be mainly measured by the average operating cost (in *Euros*) derived from the agent's sequence.

Abandoned RES An important factor to take into consideration is the wasted RES energy. The models will be analysed on their RES penetration the average unused RES power.

Survivability Although the primary objective of DED focuses on the economic performance of a power system, a model must successfully meet the specified constraints and complete episodes in order to accurately determine optimal dispatch strategies and understand the correlations with environmental dynamics.

In this context, the survivability of models must be addressed and improved in order to maintain grid reliability, stability, and security beyond an economic operation.

This factor is evaluated through the survival rate metric, which represents the percentage of maximum episode length the model is able to survive.

Scalability The solution will be tested on scenarios of various sizes to analyse its scalability. This is a crucial factor for comparing the GRL and pure-DRL approaches, as the literature suggests that the former is more scalable and adaptable to changes in topology.

Computational Efficiency It is crucial that the solution is able to perform well in real-time execution. In this context, it's important to assess the solution's decision computation performance and measure the time necessary for the model offline training and the observed CPU/GPU resource utilisation.

To ensure an unbiased evaluation of the proposed models, the data from the scenarios is divided into training and testing samples with a 90-10 ratio. Given the complexity of the problem and the limited data available, this split allows for a sufficient number of episodes to effectively learn optimal policies during training while still providing a representative and robust, albeit small, test sample to evaluate the model's performance and generalisation capability. Additionally, cross-validation was considered, but its use was discarded because of the algorithms' slow performance and apparent reduced learning efficiency.

Chapter 5

Result Discussion

$$\forall i \in N_{\text{p}}, \text{rho}_i(t_1) > \theta_{\text{hard}} \implies \forall t \in [t_1, t_1 + T_{\text{reconnect}}], F_i(t) = 0 \tag{5.1}$$

$$\forall i \in N_{\text{p}}, t \in [t_1, t_1 + T_{\text{soft}}], \text{rho}_i(t) > \theta_{\text{soft}} \implies \forall t \in [t_1 + T_{\text{soft}}, t + T_{\text{reconnect}}], F_i(t) = 0 \tag{5.2}$$

In this chapter, the obtained results are thoroughly analysed and discussed. The performed experiments can be subdivided into 6 categories: Curtail Lower Limit (section 5.4.2), Limit Infeasible Curtailment Actions 5.4.1, Reward Tests 5.2, GNNs Hyperparameter Tuning, GNN Implementation Comparison 5.6 and Scability Tests 5.7.

5.1 Implementation Details

The development of GRL algorithms posed as a complex and difficult problem. As mentioned in the literature review and restated in the problem statement, there is a scarse availability of open-source implementations and frameworks that are implemented to accommodate the implementation of these techniques.

5.1.1 Reproducibility

A major concern and challenge brought with the technologies at hand was to implement a solution that brought reproducible results. This is critical to ensure appropriate analysis and calibration of the model as well as to allow other academics and interested readers to reproduce the results and follow the tuning process.

Initially, exploratory experiments with *stable-baselines3* revealed that the DRL algorithms were developed in a deterministic and reproducible nature but with the inclusion of the *torch_geometric* GNNs the results were not consistent.

In this context, the documentation of both libraries were thoroughly analysed and studied and it was ensured that all of the appropriate seeds were being defined. However, results in the same machine still differed. This was later found to be related with the non-deterministic nature of some *torch_geometric* functions in the GPU, namely *torch-scatter* operations [?] [?].

Although, at first, this issue seemed unsolvable, a specific parameter was found in the *PyTorch* Documentation [?] that successfully mitigated this problem and allowed for result reproducibility. Nevertheless, for different machines, the results still showed inconsistencies.

```
import torch
torch.use_deterministic_algorithms(True)
```

5.1.2 Flexibility

One of the objectives that was also a large focus of the implemented solution, was to allow different combinations of DRL and GNN algorithms. Since GRL still is a relatively recent field, it's important that research around this area builds foundations for other studies and this is facilitated with the developed flexible framework that enables experiments on other models that are currently unexplored.

With the advantages of having two popular frameworks for each type of algorithms with a consistent API for both made this goal easier to achieve than the former one associated with reproducibility.

5.2 Reward Function

Several implementations posed as relevant formulas to describe the reward function of the environment, namely three: Economic Reward, RESs Penalty Factor Reward and RESs Bonus Reward. The first considered only the saved cost in non-renewable generators operation, while the second and third accounted for RES maximization in a penalty and bonus factor, respectively.

Considering the new parameter introduced by both the Penalty and Bonus Factor rewards, β, the experiments took into account three distinct values: $\{0.4, 0.6, 0.8\}$.

It's important to note that when evaluating and comparing reward functions, the average cumulative reward values retain different meanings. Furthermore, although still relevant to analyse the convergence of the models, other metrics were favoured, such as daily operating cost, abandoned RES energy and the survival rate.

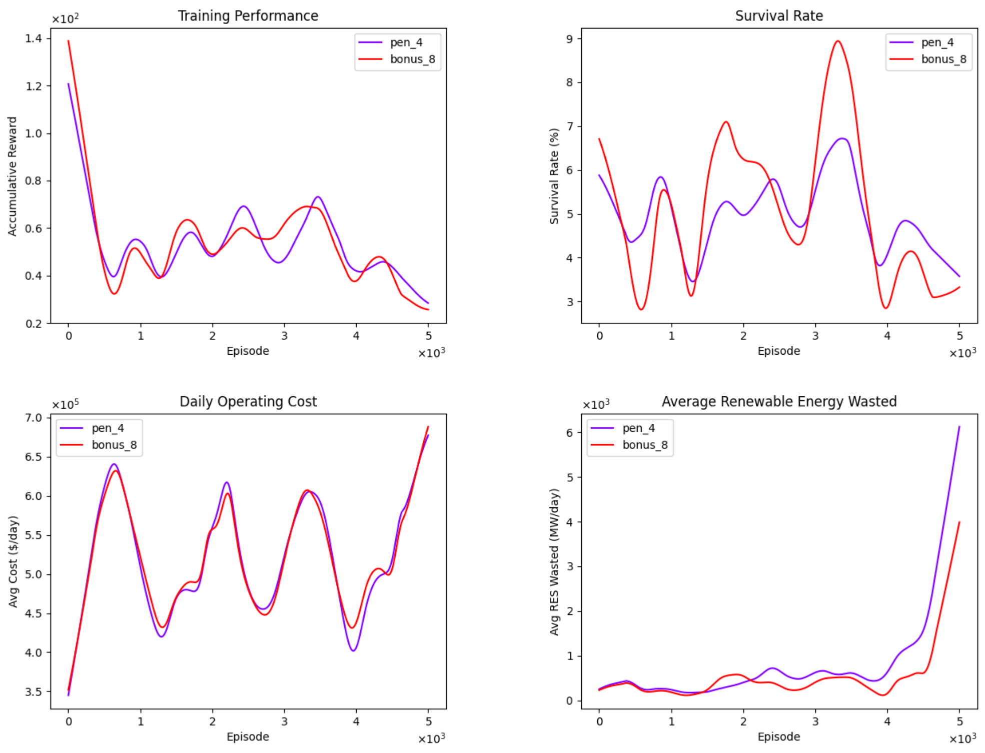

Figure 5.1: Training Results of the Experiments concerning Observation Space.

Table 5.1: Test Results for Reward Parameters Experiments

reward	β	Metrics			
		Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
bonus	0.8	40.88	96.43	560927.60	3115.18
penalty	0.4	40.57	80.75	565893.63	4100.20
bonus	0.4	40.13	76.23	565757.16	3723.72
penalty	0.8	38.22	76.35	555582.14	2612.34
penalty	0.6	37.54	81.40	558362.82	2570.71
economical	-	34.38	58.65	574981.17	2610.70
bonus	0.6	30.78	55.06	566786.07	2750.34

Results revealed that the proposed implementations were superior to the already defined Economic Reward both in survivability and overall performance of the models. The with the best results were observed in the Bonus Factor Reward with $\beta = 0.4$. Although

5.3 Observation Space

Regarding the observation space, two matters were found to be worth to analyse. Firstly, it was hypothesized that using time keys (minute, hour, and day of the week) instead of the current step might help the model find temporal patterns in the sequential process that could yield performance improvements.

Additionally, given the complexity of the problem at hand, a min-max scale was applied in particular keys also with the goal of increasing learning efficency and overall performance. The scaling was applied to $P_i^{\text{NRES}}(t)$, $P_i^{\text{RES}}(t)$, and $\overline{P_i^{\text{RES}}}(t)$.

$$x' = \frac{x - \min x}{\max x - \min x} \tag{5.3}$$

Table 5.2: Test Results for Observation Space Parameters Experiments

step	**scaled**	**Metrics**			
		Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
True	False	40.00	65.26	559046.91	2302.77
False	False	56.31	85.65	565029.79	2088.03
True	True	39.93	66.036	567707.16	2581.26
False	True	45.33	69.96	552337.96	2520.84

5.4 Action Space

The initial exploratory experiments and posterior training performance analysis showed significant instability and lack of convergence of DRL models. During the experimentation phase, the developed implementations, SAC and GCN-SAC, showed a significantly reduced survival rate. This posed a substantial problem since models failed to survive enough steps to appropriately learn economic policies, leading to relatively short results in their test phase.

Table 5.3: Test Results for `no_curtail` Environment Parameter Search

no_curtail	**Metrics**			
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
True	130.15	676.74	536423.98	0.0
False	22.13	37.14	619721.30	10315.47

Upon further research, the root of this problem was found to be related to the inclusion of curtailment actions, which made the action space more complex and, in turn, severely affected the models' performance. Table 5.3 clearly states this problem, by highlighting the test results of two models with and without curtailment actions. As observed, the inclusion of curtailment actions caused an average cumulative reward decrease of 83.00% in relation to the model without these type of actions. Additionally, the survivability of the worst model was also significantly inferior, with an average episode length of 37.14 steps.

It was found that during the exploratory phase, the abrupt changes in curtailment caused infeasible states of the system and, consequently, an early episode truncation. This caused the model to perform very poorly and, as a consequence, severely affected the learning efficiency, since the algorithms were not managing to survive enough steps to appropriately learn optimal dispatch policies.

With the purpose of further analysing and solving this problem, two methods described in the following subsections 5.4.1 and 5.4.2 were experimented to mitigate it.

5.4.1 Limit Curtailment Infeasible States

The first method that was analysing the effect of using the grid2op `LIMIT_INFEASIBLE_CURTAILMENT_STORA` parameter, which according to its own long name and documentation limits infeasible system states derivative from curtailment (and storage) actions.

For this purpose, another model was trained with this flag set to `True`.

The training results confirmed the initial suspicions, demonstrating that while performance is significantly and negatively affected when curtailment actions are included, by setting the `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION` the model was able to achieve higher results than without the parameter. As observed in table **??**, the usage of the parameter translated into an improvement of 346.63 % in average cumulative reward and 1100.02 % in survival rate compared with the worst model.

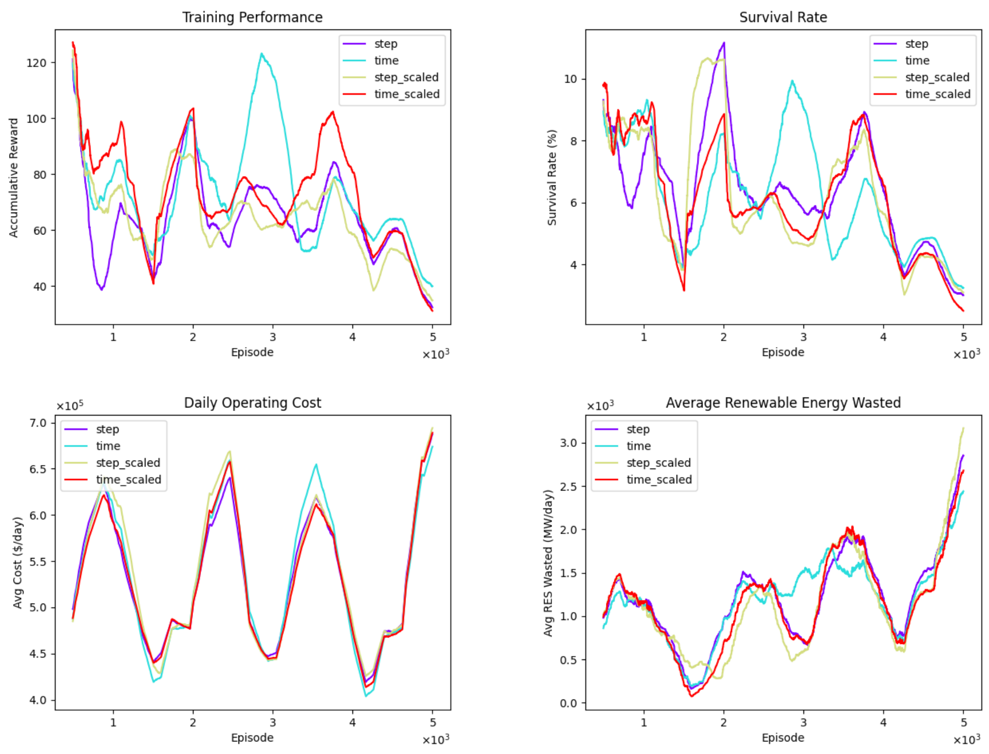

Figure 5.2: Training Results of the Experiments concerning Observation Space.

Although its usage has an apparent positive effect on model performance, it still did not mitigate the concerns related to the introduced complexity of curtailment actions. The model without

Table 5.4: Test Results for `limit_inf` *Grid2Op* Parameter Experiments

no_curtail	**limit_inf**	**Metrics**			
		Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
True	False	130.15	676.74	536423.98	0.00
False	True	98.84	445.77	588610.57	8039.41
False	False	22.13	37.14	619721.30	10315.47

these actions still outperformed the others by a large margin, outlining a difference of 31.31 reward units on average to the second-best model. The proposed method failed to mitigate the advantage brought by the behaviour of the model without curtailment actions, which by default uses all of the available renewable energy at each timestep, resulting in 0.0 MW of wasted RES energy.

5.4.2 Curtailment Lower Limit

Another possible solution to the problem of added intricacy advent from including curtailment actions is the definition of a lower bound for curtailment actions that restricts the model operation in this regard.

$$y = 1 - \frac{(1 - \underline{P}^{\text{RES}})E_{\text{train}}}{E_{\text{decay\_end}}} \tag{5.4}$$

$$y = (1 - \underline{P}^{\text{RES}}) \sqrt[\alpha]{\frac{E_{\text{decay\_end}} - E_{\text{train}}}{E_{\text{train}}}} \tag{5.5}$$

Reflecting on the secondary goal of maximising the usage of generated renewable energy, the idealisation of a lower limit to confine the curtailment action space became a feasible solution to aid in the model's learning efficiency. In this manner, this condition was included in the implementation along with three possible strategies applied solely in the training stage:

- Fixed lower bound
- Linear decreasing lower bound
- Square Root decreasing lower bound

An experiment compared the performance of the proposed strategies and the impact of this method on the former one introduced in subsection 5.4.1. The test results are observable in table 5.5.

Results proved that the introduced curtailment lower bounds considerably increased the model's training performance and convergence, with the square root decay method achieving the overall highest average cumulative reward during the test phase. The models with a fixed lower bound and linear decay also exceeded the method presented in the former subsection without outperforming the model without curtailment. The best model outperformed the one without curtailment actions

Table 5.5: Test Results for `decay_type` GCN Parameter Experiments

decay	**Metrics**			
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
sqrt	153.77	628.19	579978.88	7356.86
fixed	122.58	461.28	541391.22	7480.09
linear	111.52	289.36	550832.62	7092.56
None	98.84	445.77	588610.57	8039.41

in average cumulative reward by 18.15 %, which showed the relevance of including these actions to learning efficiency and justified their usage.

Nevertheless, the model with no curtailment reached a slightly higher survival rate without curtailment actions, as observable in table 5.6.

Table 5.6: Test Results for `decay_type` GCN Parameter Experiments

no_curtail	**limit_inf**	**decay**	**Metrics**			
			Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
False	True	sqrt	153.77	628.19	579978.88	7356.86
False	False	sqrt	153.77	628.19	579978.88	7356.86
True	False	None	130.15	676.74	536423.98	0.00
False	True	None	98.84	445.77	588610.57	8039.41
False	False	None	22.13	37.14	619721.30	10315.47

Interestingly, the combination of both proposed methods had no performance improvement, with the most significant being observed in the lower bound method. This clearly shows the superiority of this solution over the other, which, on its own, did not justify the usage of curtailment actions. However, given that the usage of the `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION` parameter also had a positive effect on performance and the author was not sure that the simultaneous usage of both methods always translated into the same learning efficiency as using only the lower bound method, a combination of both solutions was considered as the optimal procedure.

5.5 GNNs Hyperparameter Tuning

As a key area of this research work, the GNN component of the proposed algorithm was given a special emphasis on what accounts for hyperparameter tuning. Concrete experiments were devised to assess the performance of different parameter combinations, mainly focusing on the GCN architecture. This section is organised as follows: in this first subsection 5.5.1, the performance of the several available aggregation schemes is unveiled and analysed; in subsection 5.5.2 the focus is shifted towards the number of GNN layers; in the following subsection 5.5.4 the effect of other specific GCN parameters is analysed, and, lastly, in the last subsection 5.5.5 the same process is performed for the GAT architecture.

5.5.1 Aggregate Function

The aggregation function is a crucial component of GNN algorithms, and because of this fact, it was the first parameter investigated. An experiment was conducted to ascertain the best-performing aggregation function of the GCN architecture, and the available schemes include the following:

- `sum`
- `max`
- `min`
- `mean`
- `mul`

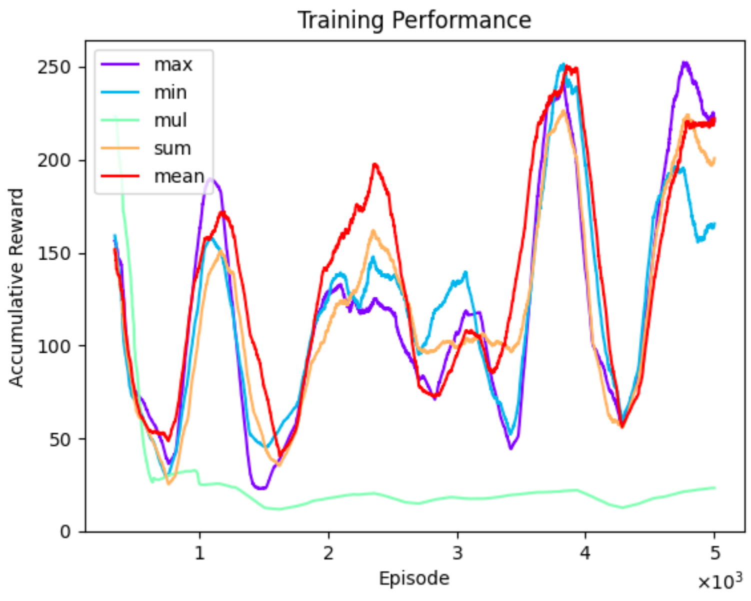

Figure 5.3: Training Performance per Aggregation Scheme

As observed in figure 5.3, training performance only revealed an apparently poor performing multiplication scheme, with other schemes achieving somewhat similar results. However, this aggregation function ended up achieving a significantly higher survival rate during training, which is verified in the test results depicted in table **??**.

As observed in figure 5.3, training performance revealed a poorly performing multiplication scheme compared to the other tested functions. However, the multiplication function yielded a significantly high average length of 1104.06 in tests, over 43% of the model with the second-best survivability (sum), while at the same time achieving only 15.76 of average cumulative reward, which is a significantly lower value compared with the model with the second-to-last reward of 80.89. As observed in the test results, this case is primarily due to a high average daily operating

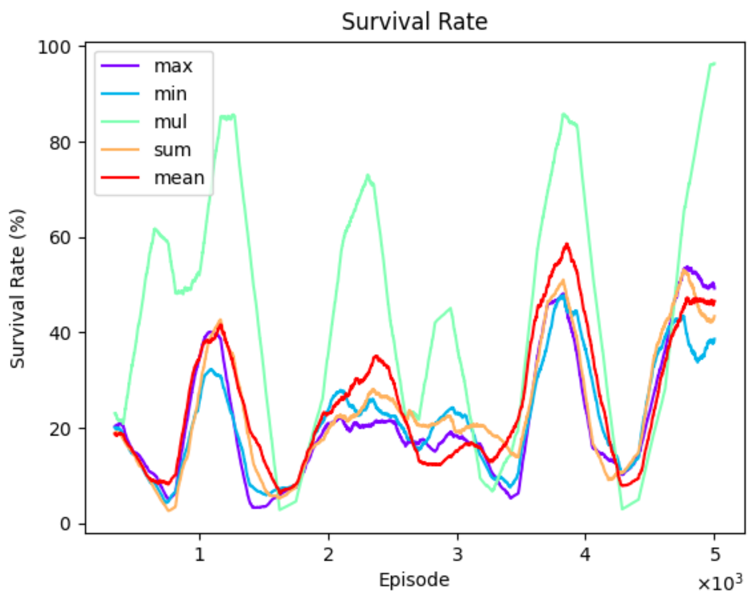

Figure 5.4: Survival Rate per Aggregation Scheme

cost, close to four thousand euros more than any other model. Beyond that, this model did not perform optimally when analysing the average energy from RES wasted, with over three thousand megawatts of unused renewables in relation to every other model.

Table 5.7: Test Results for `aggr` GCN Parameter Experiments

aggr	**Metrics**			
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
max	119.16	677.75	560219.45	6564.49
sum	114.58	768.58	569952.37	7268.28
mean	92.90	551.24	557637.84	6268.06
min	80.89	495.70	559033.63	6687.94
mul	15.76	1104.06	608319.66	10305.96

The test results also showed that the function with the best average cumulative reward was `max`, outperforming the second-placed `sum` by almost five reward points. Furthermore, this model managed to do better than average in most metrics while simultaneously being outperformed in all of them. This shows the balance between the importance of cost saving, the maximisation of renewable energy and model survivability that the algorithm must equate for optimally solving the problem of DED.

In this manner, although the *max* function yielded the best performance only by a small margin, the author still chose it as the optimal aggregation scheme.

5.5.2 Layers

Another critical characteristic of GNN architecture is the number of layers. Another separate experiment was devised to assess the optimal value for this parameter, considering the `max` aggregation scheme, which, as exposed in the last subsection 5.5.1, yielded the best results. The models had one to six GCN layers, and the results are observable in table 5.8.

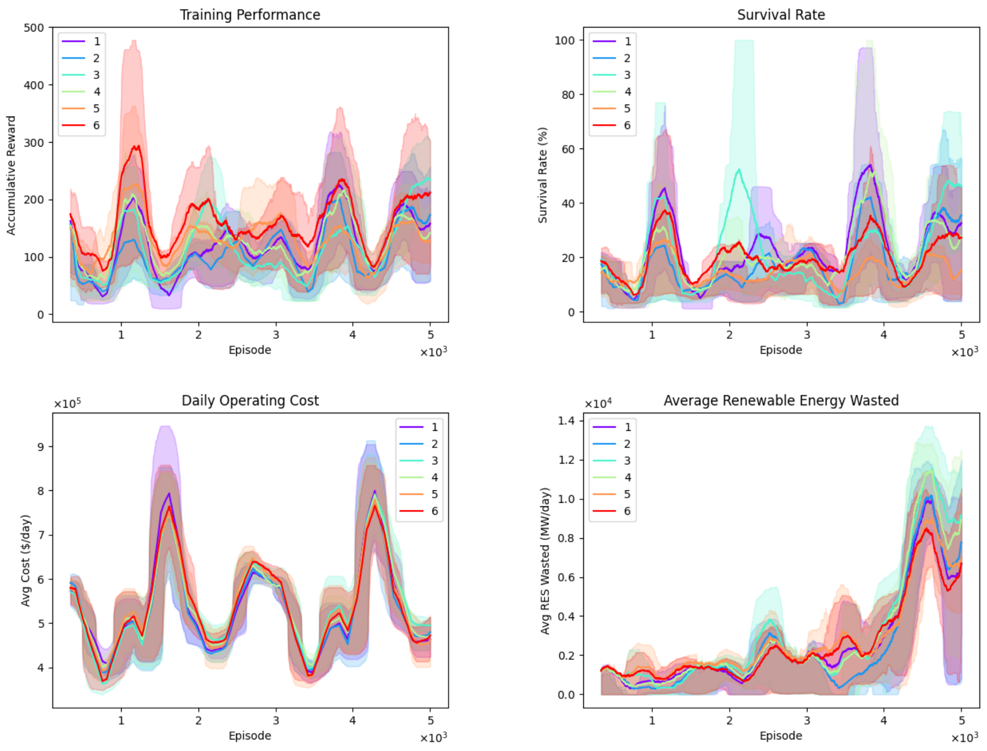

Figure 5.5: Training Results of models with 1, 2, 3, 4, 5, and 6 GNN Layers

Regarding the study conducted around this GNN characteristic, a configuration of six layers stands out from the rest, reaching an average cumulative reward of 135.75 reward units. This value represents a 13.92% improvement regarding the second-best model in this metric, which was the one-layer network.

Table 5.8: Test Results for `num_layers` GCN Parameter Experiments

num_layers	**Metrics**			
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
6	135.75	487.09	560147.77	5862.49
1	119.16	677.75	560219.45	6564.49
3	110.73	736.32	567510.58	7116.24
4	98.67	451.51	577464.90	8641.05
5	92.97	286.65	550205.21	6397.41
2	91.58	694.67	575093.64	5989.95

Generally, the calibration process has failed to reach ideal levels of survivability. Once again, the top-performing model in cumulative performance did not correspond to the best metric, the three-layered network, with an average survival rate of 36.52%. This value corresponds to an average survivability of 2.56 days per weekly episode, which, in a real-world scenario, models would break the power grid constraints after only a few days of operation.

5.5.3 Output Features

Table 5.9: Test Results for `out_channels` GCN Parameter Experiments

out_channels	**Metrics**			
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
3				
6				
18				
36				

5.5.4 GCN Parameters

Some particular GNN parameters not covered in the last subsections were also deemed relevant to analyse. This examination was motivated by an exploratory investigation that revealed the potential performance increase of using these parameters.

The `act_first` parameter controls if the GNN activation function, which is ReLU, is applied before the computation of the symmetric normalisation coefficients. The other parameter analysed was the `improved` flag, which defines whether the model computes $mathbf\hat{A}$ as $\mathbf{A} + 2\mathbf{I}$.

In this context, a grid search was conducted around these two parameters, whose test results are observable in table 5.10.

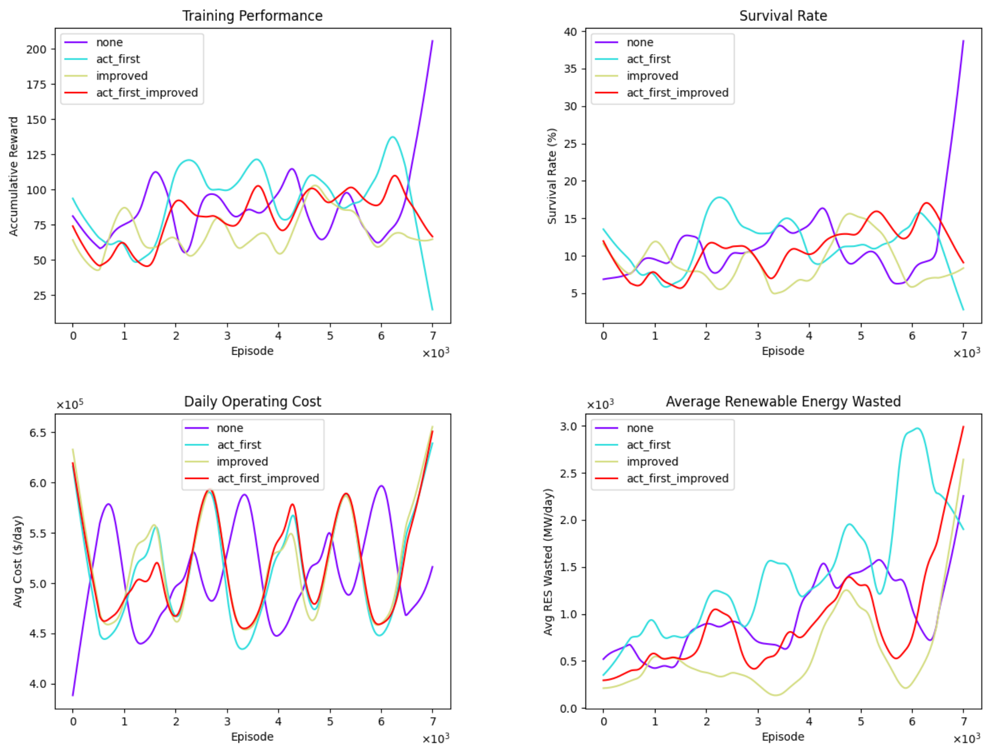

Figure 5.6: Training Results of models with 2, 3, 4 and 5 GNN Layers

Table 5.10: Test Results for `act_first` and `improved` GCN Parameter Search

act_first	improved	Metrics			
		Avg. CR	Avg. Length	Avg. DOC	Avg. REW
True	False	118.97	640.40	564201.23	7167.22
True	True	101.26	738.60	546592.78	6297.46
False	False	78.62	759.82	575798.57	7640.76
False	True	75.29	328.78	548378.53	5043.04

The use of `act_first` positively affected performance, considering the two best results. In contrast, the models that performed the activation after normalisation failed to surpass 100 reward units. Furthermore, the models benefited from not defining `improved`, translating into a cumulative reward improvement in both cases. In this manner, the chosen optimal configurations for these parameters were `act_first = True` and `improved = False`, corresponding to the combination used for the best-performing model.

5.5.5 GAT Parameters

As another prominent state-of-the-art architecture, the GAT network was also target to some tuning in specific parameters. This seemed appropriate given the tuning process the GCN network was particularly subject to, intending to also leverage the specific attributes of the GAT architecture.

The optimal combination of `aggr`, `num_layers`, and `out_channels` observable for the GCN model was reused in this architecture. While this was not an optimal approach, this decision was taken given the timing constraints of the project and the often significant amount of training time.

In this manner, an experiment was conducted around the three most relevant parameters of this model: `act_first`, `heads` and `v2`. The first, as seen in the previous subsection, positively affected the model's test results and learning efficiency, which encouraged further analysis.

The second and third parameters are specific to the GAT network, with the first representing the number of attention heads used in the model and the second consisting of a flag that controls the use of the GATv2 architecture proposed in [**?**].

This study used a random search of these parameters to ascertain the optimal combination, training 10 models for 3000 episodes each using the *Ray Tune* library. The models were trained for 1, 2, and 3 heads and the test results are observable in the table 5.11.

Table 5.11: Test Results for the `act_first`, `heads` and `v2` GAT Parameter Search

act_first	**heads**	**v2**	**Metrics**			
			Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**
True	3	True	117.09	258.93	582144.59	7216.87
False	3	False	111.41	726.20	569354.23	7403.57
True	1	True	94.46	594.57	571177.15	6785.37
True	2	True	92.50	650.37	562035.66	7349.15
False	3	True	91.18	197.57	552279.16	8988.24
True	2	False	88.77	611.40	565936.09	7250.76
True	3	False	80.03	579.84	556860.38	6723.43
True	1	False	76.94	522.30	574402.26	8366.63
False	1	False	76.06	476.15	549382.92	7971.11
False	1	True	59.17	273.63	574194.39	5665.66

Furthermore, the outcomes of the test phase revealed that the model with the best average cumulative reward was the one with `act_first` and `heads` set and three `heads` with 117.09 reward units. However, this model showed a significantly low survival rate. In contrast, the best-performing model in survival rate, which was also the second-best model in cumulative reward, had the two parameters unset and three `heads` as well, which directed the choice of attention heads to this value.

With regards to using the second version of the GAT architecture, the results show that out of the top five models in cumulative reward, four use GATv2 implementation. The test results displayed that this choice almost consistently outperformed the other.

Lastly, the `act_first` parameter also showed acceptable performance, with three models in the top five best-performing and the being present best model. This conclusion closes the optimal combination of these parameters considered in the following comparison and scalability experiments.

5.6 GNN Implementation Comparison

Table 5.12: Test Results for 36-bus Scenario

Model	**Metrics**					
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	342.84	1118.24	601896.45	5443.65	0.0029	47.76
GCN-SAC	153.77	628.19	579978.88	7356.86	0.0097	31.11
GAT-SAC	105.75	231.88	564739.25	7070.78	0.0071	12.52

5.7 Scalability Tests

Table 5.13: Test Results for the 118-bus Scenario

Model	**Metrics**					
	Avg. CR	**Avg. Length**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	81.98	361.92	2788763.38	4740.03	0.0067	14.82
GCN-SAC	67.85	389.82	2901439.28	5323.58	0.0096	13.67
GAT-SAC	54.31	240.28	2591733.60	4709.37	0.012	11.39

Chapter 6

Conclusions

In conclusion, this report establishes a solid foundation for implementing and proposing concrete improvements to GRL techniques and efficiently solve the problem of DED in graph-based environments.

We concluded in the literature review, that GRL is already a promising solution for DED, yielding better performance than other approaches. In general, GNNs are an ubiquitous method, throughout the gathered literature, for extracting efficient environment topological representations. The identified limitations of existent implementations lie on their lack of: (1) scalability to larger environments, resulting in a significant decrease in computation performance; (2) a seamless integration between GNN and DRL algorithms and (3) adaptability to real-time topology changes.

Moreover, we aim to study how to tackle these limitations by implementing different GRL approaches for solving the DED problem and conducting a comparative study between the different techniques. The implementations will be compromised of various combinations of GNNs, for efficiently extracting the relevant environment features, and DRL algorithms for mapping the extracted representations into optimal action sequences. Case studies will be carried out within different size modified IEEE bus systems for testing the various models and confront their results. Lastly, the models will be analysed considering their training efficiency, computational performance, dispatch efficiency, scalability to large scenarios and adaptability to topological changes. Based on the conclusions drawn from the comparative study, we will propose a GRL that implements concrete enhancements.

6.1 Main Contributions

In this section, we list the expected contributions derived from this work on a Scientific, Technological and Application levels:

Scientific A systematic and comparative study of different GRL approaches. This fills the research gap for systematic studies comparing different proposed techniques. Furthermore, this work will bring a clearer insight regarding the best practices on implementing GRL models

Technological A model resulting from this study with concrete improvements over the current GRL techniques proposed so far and tackling their limitations. Beyond this,

Application An efficient solution for the DED problem with GRL algorithms, compromising significant contribution to the research on DED systems in complex scenarios

6.2 Future Work

While this work extensively explored potential improvements in GRL techniques and their applications to the DED problem, the scope of the issue is vast, and several areas for improvement were left unaddressed due to time and planning constraints. The main future concerns are aligned with improving the performance of the model and further investigating other potential enhancements to the GRL techniques.

Firstly, it would be valuable to explore additional DRL and GNN implementations to compare with the results obtained from the studied models. For example, alternative combinations of GNN models like GraphSAGE and DRL algorithms such as PPO and DDPG might yield better performance than those tested in this study. Additionally, the author is keen to compare the studied single-agent GRL technique with multi-agent systems in the DED problem. Given their distinct perspectives on the issue, such a comparison could provide valuable insights into the effectiveness of each approach.

Given the complexity and diversity of modern power systems, it would also be worthwhile to consider additional elements, such as Energy Storage System (ESS) storage, which were excluded to avoid overly complicating the environmental dynamics. Moreover, investigating how voltage control could enhance the agent's reliability and, consequently, the survivability of models appears to be a relevant area for further study.

Finally and most importantly, further experimentation and calibration, especially with larger scenarios, might be necessary to understand the failures of this work's approach on GRL techniques. There is still promising evidence from the literature of their performance on graph-based scenarios, but the study and analysis that was conducted failed to reach that conclusion.

Appendix A

Hyperparameters

A.1 SAC Parameters

A.2 GNN Parameters

A.3 Environment Parameters

Table A.1: SAC Parameters Description

Name	**Description**
`policy`	The policy model to use
`learning_rate`	Learning rate applied in Q-Values, Actor and Value functions
`buffer_size`	Size of the replay buffer
`learning_starts`	Number of transitions to collect before the start of training
`batch_size`	Minibatch size for each gradient update
`tau`	Soft update coefficient or Polyak Update
`gamma`	Discount factor
`train_freq`	Update frequency in training
`gradient_steps`	Number of gradient steps per rollout
`action_noise`	Type of action noise
`ent_coef`	Entropy regularization coefficient
`target_update_interval`	Step period to update the target network
`target_entropy`	Target entropy during learning of `ent_coef`
`use_sde`	Use generalized State Dependent Exploration (gSDE)
`sde_sample_freq`	Step period to sample new noise matrix for gSDE
`use_sde_at_warmup`	Use gSDE instead of uniform sampling before learning starts
`optimize_memory_usage`	Enable a memory efficient variant of the replay buffer at a cost of more complexity
`replay_buffer_class`	Class of replay buffer
`replay_buffer_kwargs`	Additional keyword arguments to pass to replay buffer
`stats_window_size`	Size of statistics output window
`tensorboard_log`	Log location for tensorboard
`verbose`	Verbosity level
`seed`	Seed for the pseudo random generators
`device`	Device which the code will be executed in
`_init_setup_model`	Wether to build the network at the creation of the instance
`env`	Gymnasium Environment
`policy_kwargs`	Additional arguments for policy

Table A.2: SAC Policy Parameters Description

Name	**Description**
`net_arch`	Architecture of policy and value networks
`optimizer_class`	Optimizer to use
`optimizer_kwargs`	Additional keyword arguments for optimizer
`activation_fn`	Activation function
`n_critics`	Number of critic networks
`features_extractor_class`	Features extractor for the minibatches
`features_extractor_kwargs`	Additional keyword arguments for feature extractor

Table A.3: GNN Parameters description

Name	**Description**
`in_channels`	Input sample size
`hidden_channels`	Hidden sample size
`num_layers`	Number of message passing layers
`out_channels`	Output size
`dropout`	Dropout probability
`act`	Non-linear activation function
`act_first`	Apply activation before normalization
`act_kwargs`	Additional keyword arguments for activation function
`norm`	Normalization function
`norm_kwargs`	Additional arguments passed to normalization function
`jk`	Jumping knowledge mode
`aggr`	Aggregation scheme
`aggr_kwargs`	Additional keyword arguments for aggregation scheme
`flow`	Message passing flow direction
`node_dim`	The axis along which to propagate
`decomposed_layers`	Number of feature decomposition layers
`improved`	If `True`, layer computes $\hat{\mathbf{A}}$ as $\mathbf{A}+2\mathbf{I}$
`cached`	If `True`, layer will cache $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ (only for transductive scenarios)
`add_self_loops`	Add self-loops to the input graph
`normalize`	Add self-loops and compute symmetric normalization coefficients
`bias`	Learn additive bias
`heads`	Number of heads for multi-head attention
`v2`	Use *GATv2Conv* instead of *GATConv*
`concat`	Concatenate multi-head attentions instead of averaging
`negative_slope`	LeakyReLU angle of the negative slope
`edge_dim`	Edge feature dimensionality
`fill_value`	How to fill edge features of self-loops

Table A.4: GCN-Specific Parameters description

Name	**Description**
`improved`	If `True`, layer computes $\hat{\mathbf{A}}$ as $\mathbf{A}+2\mathbf{I}$
`cached`	If `True`, layer will cache $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ (only for transductive scenarios)
`add_self_loops`	Add self-loops to the input graph
`bias`	Learn additive bias

Table A.5: GAT-Specific Parameters description

Name	**Description**
`heads`	Number of heads for multi-head attention
`bias`	Learn additive bias
`add_self_loops`	Add self-loops to the input graph
`v2`	Use *GATv2Conv* instead of *GATConv*
`concat`	Concatenate multi-head attentions instead of averaging
`negative_slope`	LeakyReLU angle of the negative slope
`edge_dim`	Edge feature dimensionality
`fill_value`	How to fill edge features of self-loops

Table A.6: GNN Parameters description

Name	**Description**
`env_path`	
`reward`	
`obs_scaled`	
`obs_step`	
`act_no_curtail`	
`act_limit_inf`	
`climit_type`	
`climit_end`	
`climit_low`	
`climit_factor`	

Appendix B

Experiment Results

B.1 Reward Experiments

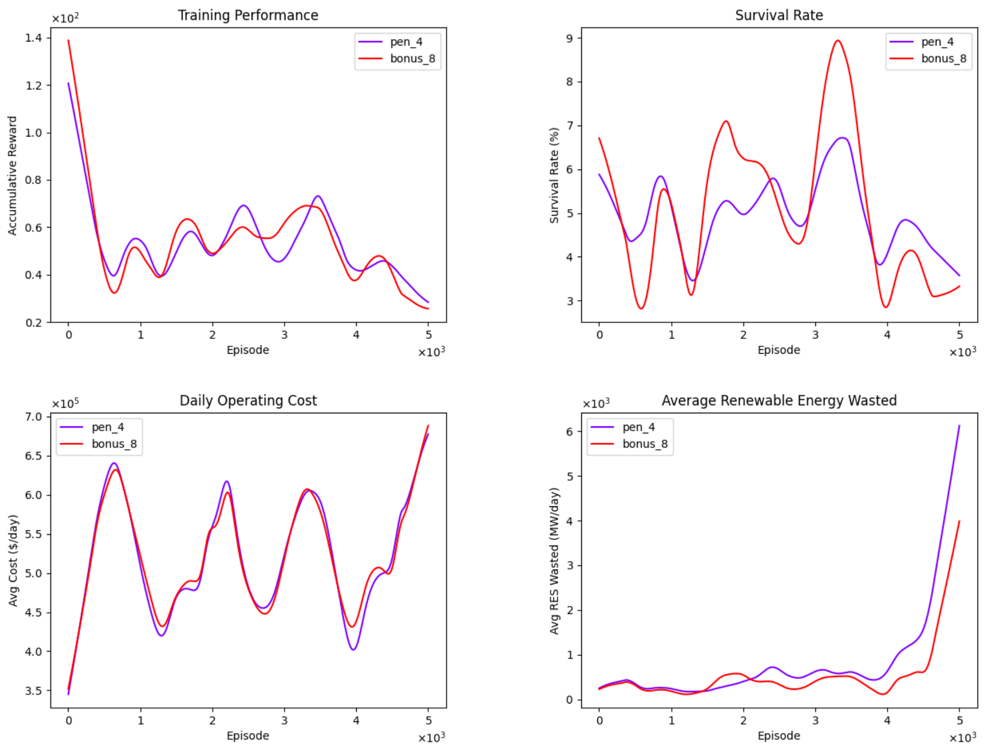

Figure B.1: Training Results of the best Penalty and Bonus Factor Rewards.

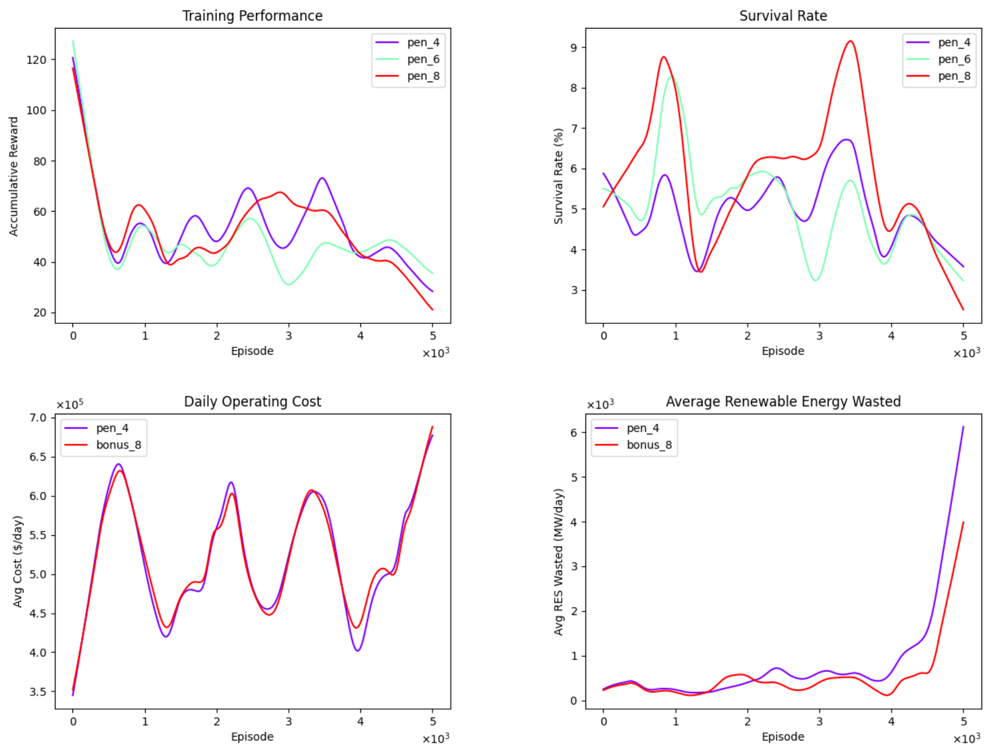

Figure B.2: Training Results of the Penalty Factor Rewards.

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
pen_4	40.57	80.75	565893.63	4100.20	230.99
pen_6	37.54	81.40	558362.82	2570.71	231.09
pen_8	38.22	76.35	555582.14	2612.34	222.58
bon_4	40.13	76.23	565757.16	3723.72	223.90
bon_6	30.78	55.06	566786.07	2750.34	189.01
bon_8	40.88	96.43	560927.60	3115.18	256.59

Table B.1: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.

B.2 Limit Infeasible Curtail Action Experiment Results

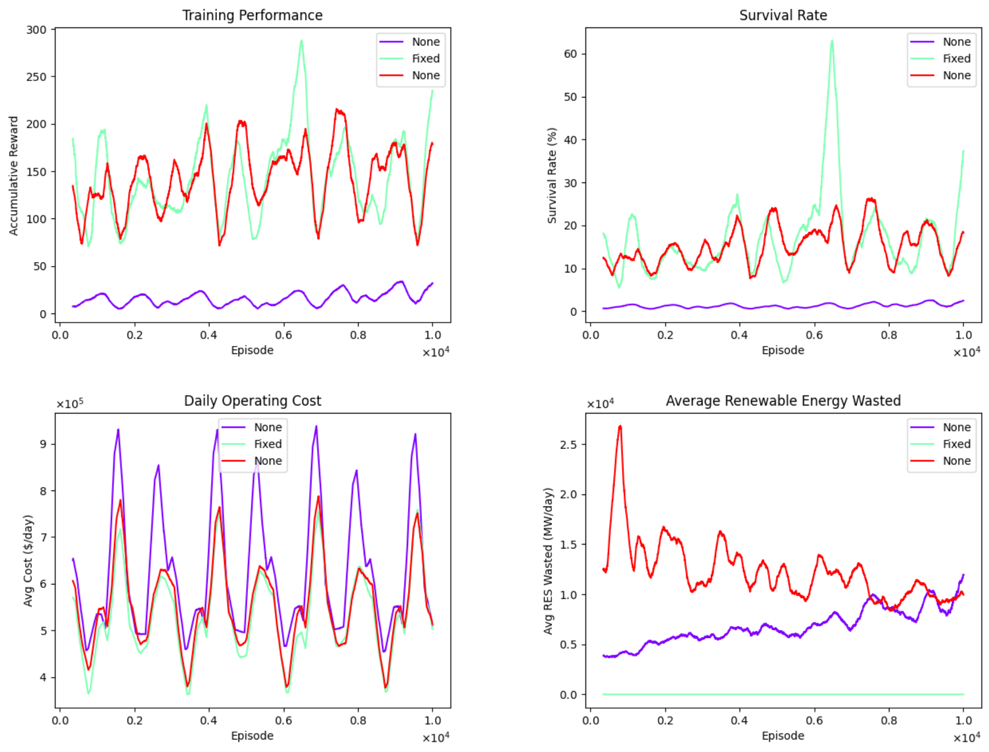

Figure B.3: Training Results of the Experiments concerning Limit Infeasible Curtail Actions.

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
no_curtail	133.564	515.807	531470.223	0.0	692.832
no_limit	128.173	403.515	585853.240	6564.428	569.376
limit_curtail	127.951	492.732	572004.947	8877.733	678.717

Table B.2: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.

B.3 Curtailment Lower Limit Smoothing

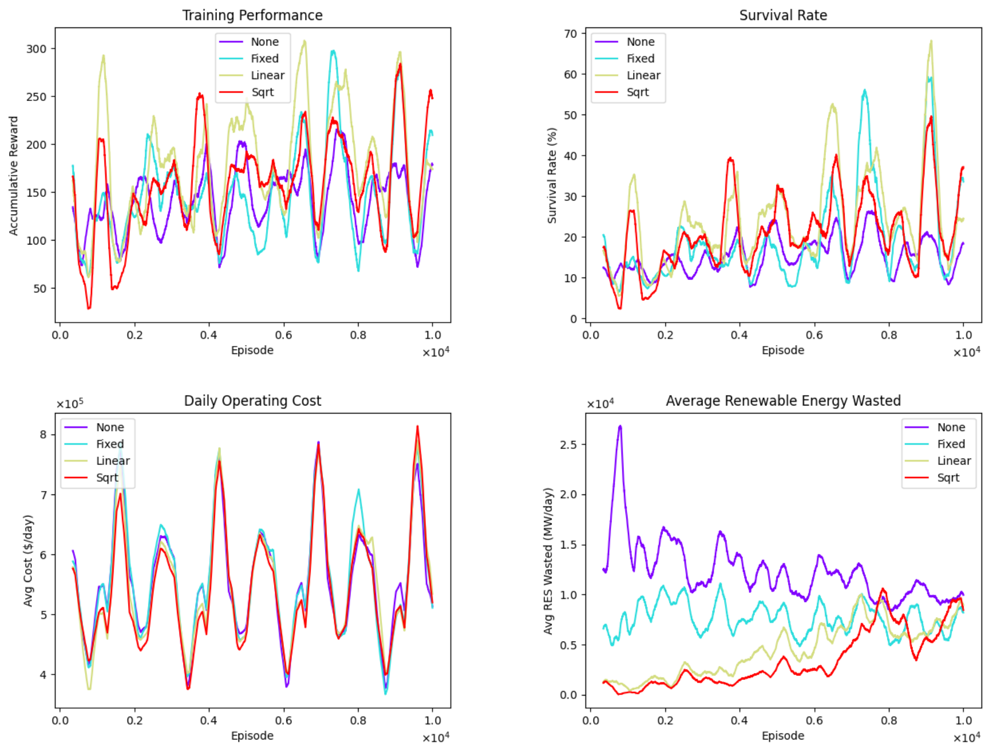

Figure B.4: Training Results of the Experiments concerning Curtailment Lower Limit Smoothing.

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
none	111.495	544.295	577960.425	12111.066	730.146
fixed	185.698	761.281	574382.241	7081.859	1007.278
linear	176.919	806.186	569758.273	5400.686	1052.257
sqrt	127.951	492.732	572004.947	8877.733	678.717

Table B.3: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.

Parameter	**Values**
Aggregation Function	{'sum','mean', 'min', 'max', 'mul'}
Number of Layers	{1,2,3,4,5}
Hidden Channels	{6, 12, 18, 24, 36}
Ouput Channels	{3, 6, 12, 18, 24, 36}
Dropout Rate	[0.1, 0.4]
Activation First	True, False
Heads	1,2,3,6
GATv2	True, False

Table B.4: General GNN Parameters Tuned

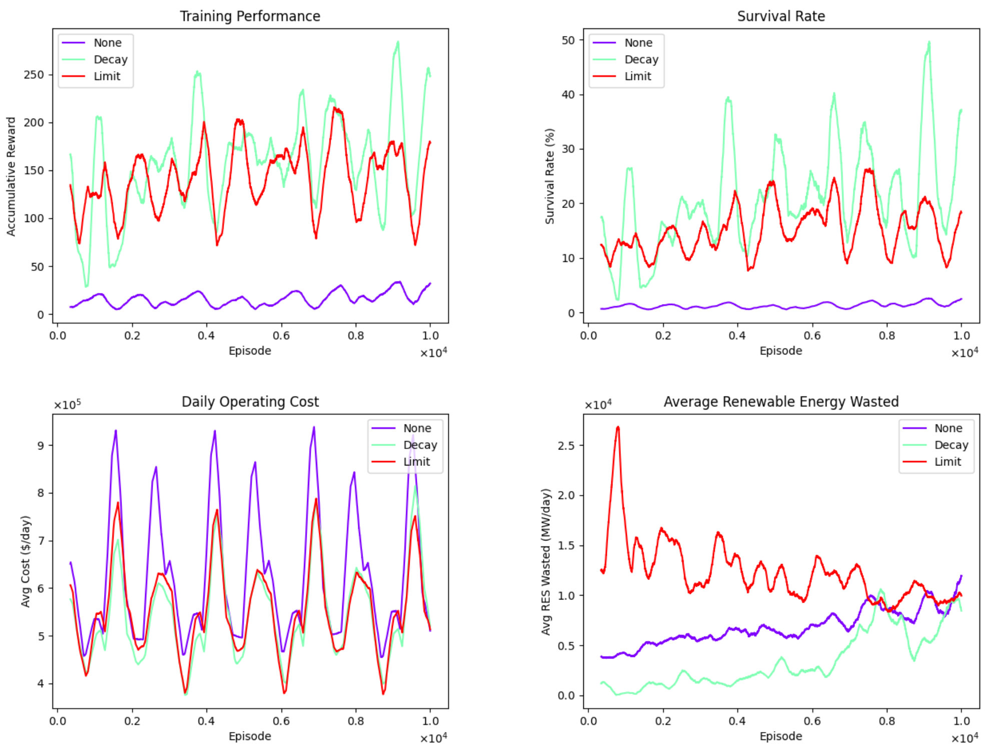

Figure B.5: Training Results of the Experiments concerning Curtailment Lower Limit Smoothing.

Implementation	**Parameter**	**Values**
GCN	Improved	True, False
GAT	Heads	1,2,3,6
GAT	GATv2	True, False

Table B.5: Model-Specific Parameters Tuned

Parameter	**Values**
Aggregation Function	{'sum','mean', 'min', 'max', 'mul'}
Number of Layers	1
Hidden Channels	18
Ouput Channels	6
Dropout Rate	0.1
Activation First	True

Table B.6: Parameters of Aggregation Function Experiment

B.4 Experiments on the GNN Tuning

B.5 Experiments on the GCN Aggregation Function

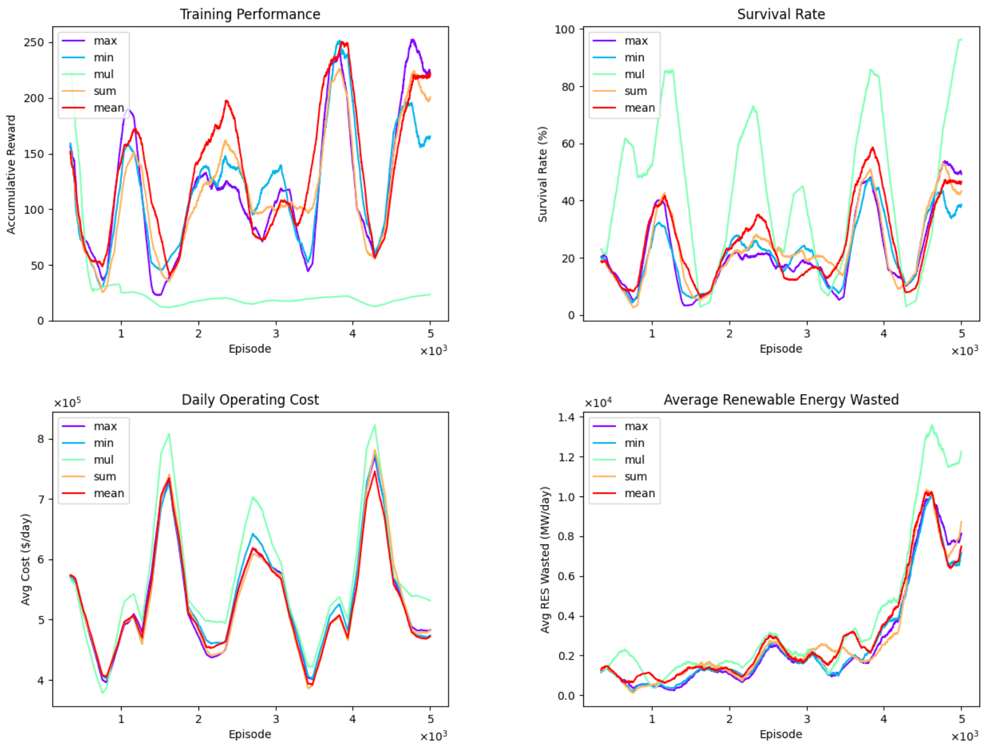

Figure B.6: Training Results of different types of GNN aggregation functions

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
max	119.16	677.75	560219.45	6564.49	552.51
sum	114.58	768.58	569952.37	7268.28	606.28
mean	92.90	551.24	557637.84	6268.06	454.24
min	80.89	495.70	559033.63	6687.94	416.63
mul	15.76	1104.06	608319.66	10305.96	846.43

Table B.7: Validation Results of different types of GNN aggregation functions.

B.6 Experiments on the number of GCN layers

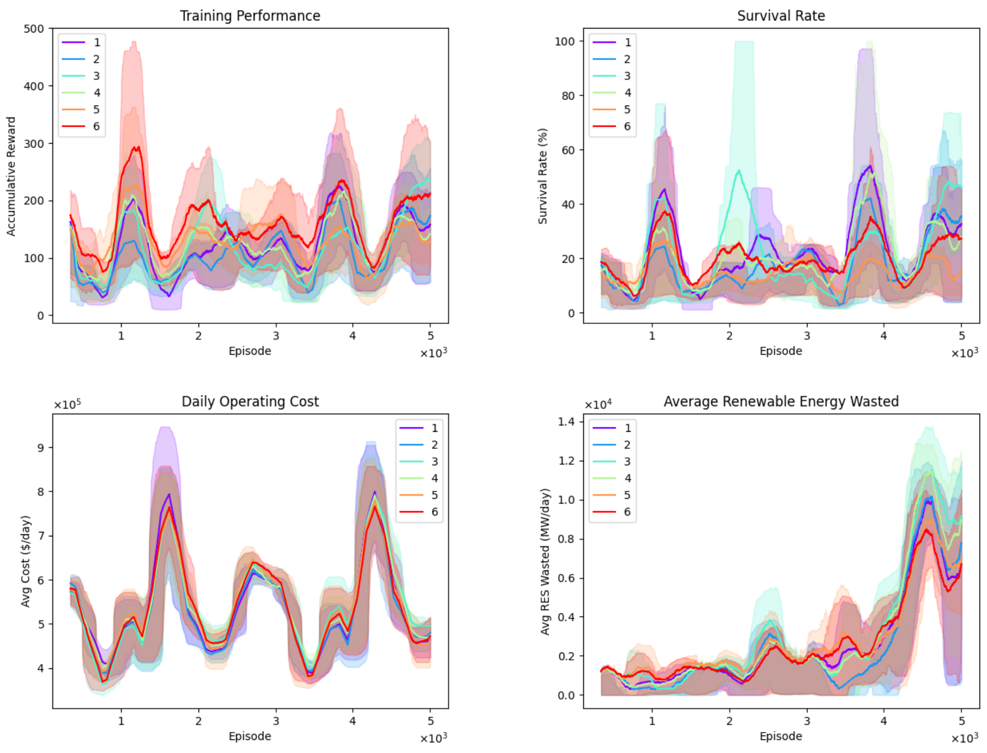

Figure B.7: Training Results of models with 2, 3, 4 and 5 GNN Layers

Parameter	**Values**
Aggregation Function	'max'
Number of Layers	[1, 6]
Hidden Channels	18
Ouput Channels	6
Dropout Rate	0.1
Activation First	True

Table B.8: General GNN Parameters Tuned

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
6	135.75	487.09	560147.77	5862.49	655.44
3	110.73	736.32	567510.58	7116.24	744.50
1	101.24	709.76	565865.82	6855.58	573.30
4	98.67	451.51	577464.90	8641.05	529.76
5	92.97	286.65	550205.21	6397.41	386.19
2	91.58	694.67	575093.64	5989.95	638.11

Table B.9: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.

B.7 Experiments on the GNN Parameters *act_first* and *improved*

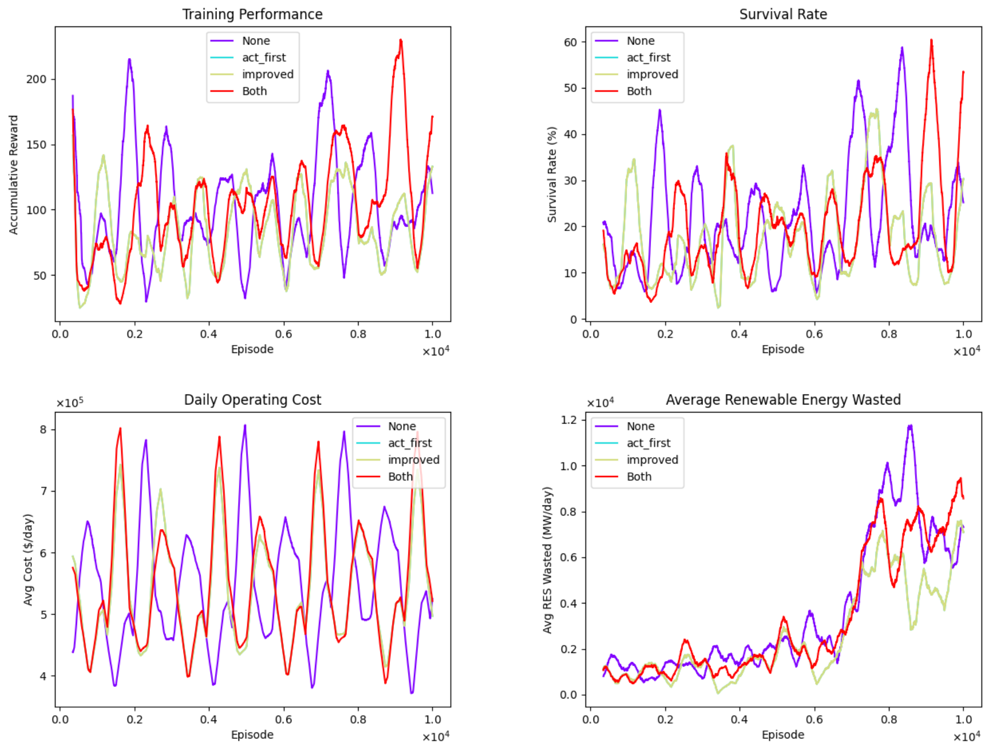

Figure B.8: Training Results of the act_first and improved Parameter Tests.

B.8 Experiments on the number of GAT Heads

B.9 GNNs vs. SAC Experiments

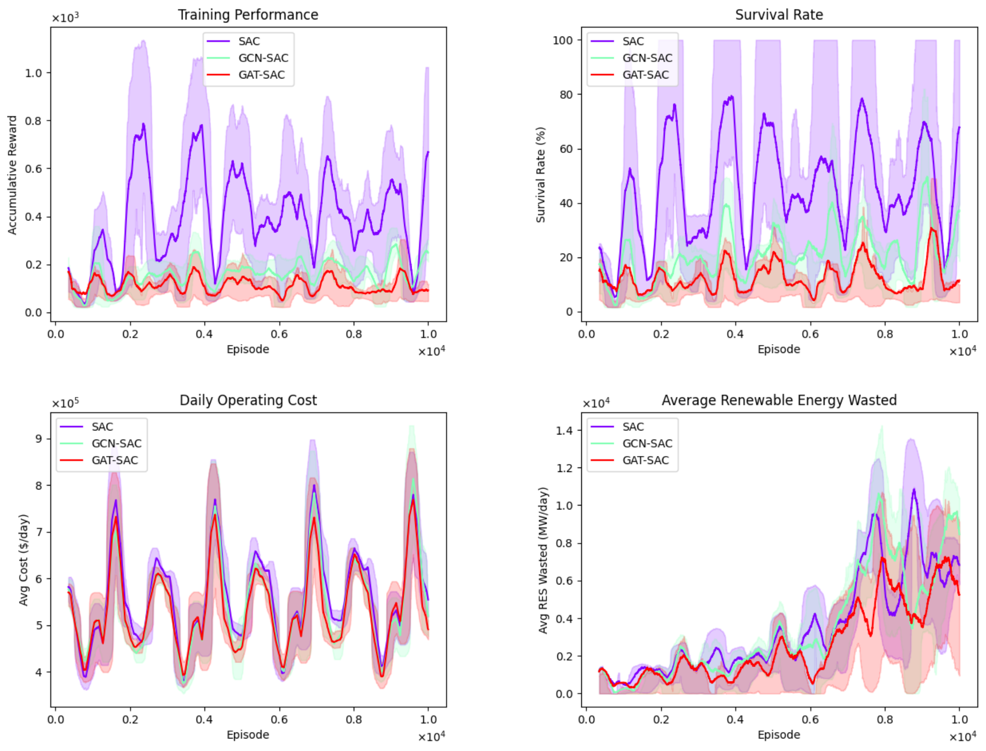

Figure B.9: Training Results of models with 1, 2 and 3 GAT Heads

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
sac	342.84	1118.24	601896.44	5443.64	741.23
gcn_sac	23.38	573.73	549592.13	9885.84	467.29
gat_sac	55.80	481.72	553388.36	6064.43	603.91

Table B.10: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.

B.10 Experiments on 118-bus scenario

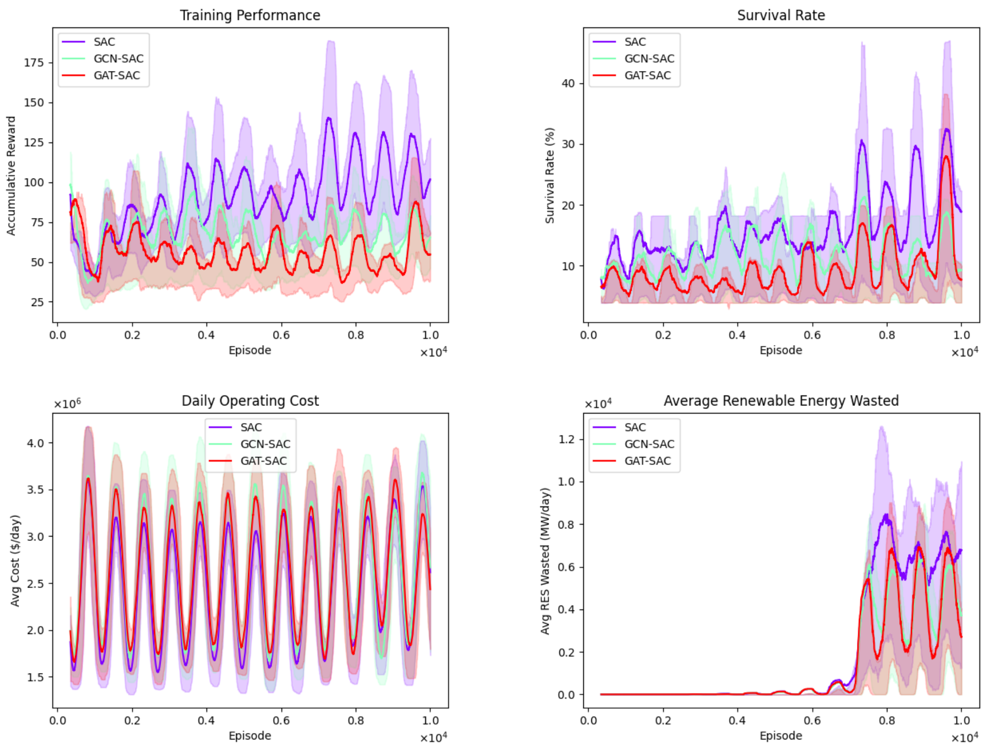

Figure B.10: Training Results of models with 1, 2 and 3 GAT Heads

Model	**Avg. Accumulative Reward**	**Avg. Length (Steps)**	**Avg Daily Operating Cost (€)**	**Avg. Renewables Wasted (MW/day)**	**Total Time (Seconds)**
sac	342.838	1118.241	601896.445	5443.649	756.142
gcn_sac	11.74	815.64	2679050.39	6567.22	318.94

Table B.11: Validation Results of the Experiments concerning Limit Infeasible Curtail Actions.