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

September 29, 2024

Graph Reinforcement Learning for Improving Smart Grid Services

António Bernardo Linhares Oliveira

Mestrado em Engenharia Informática e Computação

September 29, 2024

Resumo

A Aprendizagem por Reforço de Grafos (GRL) é um tema que tem merecido grande atenção por parte dos académicos nos últimos anos. Ao permitir que as técnicas de Aprendizagem por Reforço aprendam e otimizem processos de decisão sequenciais em ambientes baseados em grafos, os sistemas podem ser melhorados e ganhar a capacidade de aproveitar as caraterísticas da topologia dos grafos em domínios de aplicação orientados para as redes. Com os avanços no final da década de 2010 nas Redes Neuronais de Grafos na aprendizagem de como extrair representações eficientes de grafos de um determinado cenário, foram propostos métodos mais sofisticados de GRL e o tópico começou a atrair a curiosidade dos académicos. Embora, nos últimos anos, muito trabalho tenha sido feito nessa área, a pesquisa sobre as técnicas ainda é considerada em estágio inicial.

Além disso, considerando os atuais desafios globais associados à sustentabilidade e aos sistemas energéticos, há uma necessidade crescente de avanços em sistemas inteligentes centrados na energia para modernizar as atuais redes elétricas. 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 na Aprendizagem Automática, os sistemas podem ser adaptados à descentralização da produção de energia e gerir eficientemente a monitorização, a distribuição e a transmissão de sistemas de energia. Neste trabalho, a ênfase principal reside na melhoria dos algoritmos GRL que serão aplicados para resolver o problema de despacho económico dinâmico de energia como o seu principal domínio de aplicação, considerando fontes de energia renováveis e sistemas de armazenamento de energia.

Desta forma, esta dissertação pretende fazer avançar a investigação existente sobre técnicas de Aprendizagem por Reforço em Grafos mediante os objectivos seguintes: (1) realizar uma revisão exaustiva da literatura recente relativa às várias abordagens de GRL propostas e aos Sistemas de Despacho Económico Dinâmico, de modo a obter uma perspetiva global das técnicas mais recentes e das suas limitações; (2) implementar e calibrar as duas abordagens mais proeminentes, resultando nos modelos GCN-SAC e GAT-SAC (3) realizar um estudo empírico comparativo e sistemático das duas abordagens de GRL propostas face ao algoritmo SAC no problema de Despacho Económico Dinâmico de Energia.

Os resultados acabaram por favorecer o algoritmo SAC de última geração relativamente às implementações propostas. As capacidades de extração de caraterísticas dos modelos propostos não conseguiram ultrapassar a eficiência da amostra e a estabilidade do SAC, com o primeiro modelo a apresentar um melhor desempenho em ambos os cenários considerados. No entanto, as abordagens GRL, particularmente o GCN-SAC, mostraram resultados promissores no cenário maior, retratando uma boa escalabilidade para ambientes mais complexos.

Abstract

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 in 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 these 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) implementing and calibrating the two most prominent approaches, resulting in the models of GCN-SAC and GAT-SAC (3) performing a comparative and systematic empirical study on both proposed GRL approaches against the SAC algorithm on the Dynamic Economic Power Dispatch problem.

Results ended up favouring the state-of-the-art SAC algorithm over the proposed implementations. The proposed models' feature extraction abilities failed to overcome the sample efficiency and stability of SAC, with the former model yielding a better performance in both scenarios. However, the GRL approaches, particularly GCN-SAC, showed promising results in the larger scenario, portraying good scalability to more complex environments.

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

ACM Classification: Computing Methodologies → Machine Learning → Learning Paradigms → Reinforcement Learning

Acknowledgements

To my supervisors for all the guidance, teachings, and patience 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

ADN Active Distribution Network

ANN Artificial Neural Network

CNN Convolutional Neural Network

DDPG Deep Deterministic Policy Gradient

DED Dynamic Economic Dispatch

DQN Deep Q-Network

DRL Deep Reinforcement Learning

ESS Energy Storage System

GAT Graph Attention Network

GCN Graph Convolutional Network

GIN Graph Isomorphism Network

GNN Graph Neural Network

GRL Graph Reinforcement Learning

IT Information Technology

LIACC Artificial Intelligence and Computer Science Laboratory

MDP Markov Decision Process

ML Machine Learning

MLP Multilayer Perceptron

PPO Proximal Policy Optimization

RES Renewable Energy Source

RL Reinforcement Learning

SAC Soft Actor-Critic

Chapter 1

Introduction

In this introductory chapter, the context and motivation regarding this dissertation, as well as its key objectives, are presented. In section 1.1, we contextualize this work into the relevant background and conditions in which it was performed. After the main context is exposed, the primary motivations are uncovered in section 1.2, followed by the objectives of this project 1.3, respectively. Lastly, the last section 1.4 contains a quick word on the structure of this document and its main components.

1.1 Context

Several real-world problems and their objects can be instinctively represented by graph structures. These representations capture not only 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 of 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's topological features [10, 67, 68, 78]. 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.

Additionally, this dissertation is written in the context of 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 economical 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 manage power distribution efficiently [10, 34], voltage regulation, system restoration [77], grid reliability [45] or other associated processes. Currently, renewable energy sources play a major role in reducing the reliance on fossil fuels [50], 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 [53], resulting in improvements in storage capacity and approximating the current solutions to the average consumer [36].

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 aim to develop a proposed solution with an architecture that is adaptable and applicable to other smart grid problems, resulting in a significant contribution to these services.

Furthermore, the main reasons for addressing GRL algorithms lie in 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 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 their 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 analyze and confront the gathered results. Lastly, we hope accomplishing these first two goals enables the implementation of a state-of-the-art GRL model and specific improvements to these techniques.

1.4 Document Structure

This document 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; (4) the problem formulation and proposed solution architecture; (5) the discussion and analysis of results; and, lastly, (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 provides an important context for the rest of this work by explaining its background concepts.

This chapter is divided into section 2.1 addressing Artificial Neural Networks, section 2.2 regarding the Reinforcement Learning (RL) problem and its prominent algorithms, 2.3 explaining Graph Representation Learning, 2.4 exposing the current GNN approaches and 2.5 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 [20, 23]. 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 [20, 23]. 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 [20, 23]. 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 output data y, the main goal of an ANN is to approximate to the considered function knowing x and y [8, 23]. The new model of the relation between both samples can then be further applied to new input data x' with the goal of predicting y'.

The simplest and most fundamental form of an ANN is a **Feedforward Neural Network** and is further described in the next subsection.

2.1.1 Feedforward Neural Networks

A **Feedforward Neural Network**, or a Multilayer Perceptron (MLP), defines a mapping $y = f(x;\theta)$ and learns the best composition of parameters θ to approximate it to the unknown model

f^* [8, 23]. The MLP serves as a fundamental part of developing the other more complex types and modern implementations of neural networks [8].

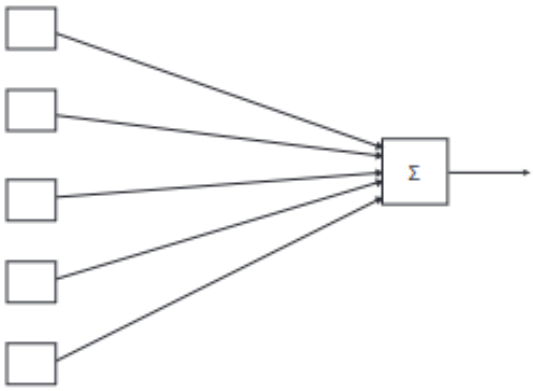

Figure 2.1: The Perceptron. A single-layer neural network unit used in supervised learning. This diagram shows the process where weighted inputs are combined, followed by applying an activation function to produce an output. The perceptron forms the basis of more complex neural network architectures [8].

The main building block of a MLP is the **Perceptron**, pictured in figure 2.1, a simple computational model initially designed as a binary classificator that mimics biological neurons' behaviour [8]. A neuron has one or more inputs x and 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 [8].

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

Functions that compute $b + w \cdot x > 0$ are called *linear units* and are identified with Σ [8, 23]. 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 [23].

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 output value [8]. The dimensionality of the hidden layers determines the overall width of the model, and the greater the number of layers, the higher the *depth* of the network [8, 23]. This last notion, together with the process of how biological neurons interact, also motivated the conceptualization of **Deep Neural Networks**, which consist of feedforward neural networks with a considerable amount of hidden layers. By increasing network depth, these models can learn highly complex patterns and abstractions in data that *shallow* networks cannot capture [23].

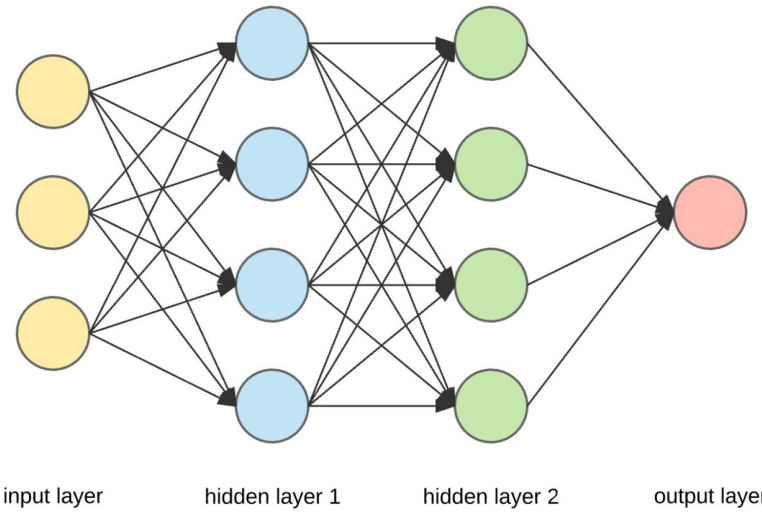

Figure 2.2: Feedforward Neural Network Architecture. This diagram illustrates a basic feedforward neural network architecture, consisting of an input layer, two hidden layers, and an output layer. The arrows represent the flow of information between neurons across layers. The network's structure enables the transformation of input features into learned representations for a specific task through multiple layers of non-linear functions [1].

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, as described by figure 2.2 [23]. 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 involves minimizing or maximizing another function that consists in an *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 [23]. 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 [8, 23].

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 [15]. The main objective is to maximize the received cumulative reward by trying between the available actions and discovering which ones yield the most reward [55]. 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 [55]. 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 [55]. The classic formalization 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 [55]. 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 [55]. 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 must 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 [55].

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 [55, 42]. 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 [55].

The central component of MDPs is the agent, which acts as a decision-maker and learns from interactions with the environment. In a continuous process, the agent takes actions that affect the environment's state, which in turn presents new situations [55]. The environment also responds with the reward signals, which the agent aims to maximize over time through its decision process.

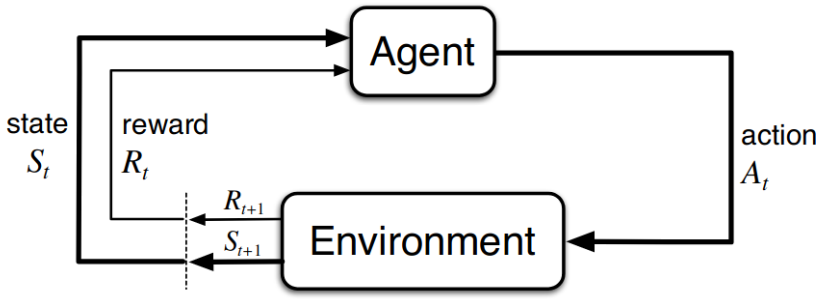

Figure 2.3: Agent-Environment Interaction in a MDP. The agent takes actions based on its current policy, which leads to state transitions in the environment. The environment responds by providing feedback through rewards and new states. The agent uses this feedback to update its policy to maximize cumulative rewards over time [55].

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 [55, 42]. 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} [55]. 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* [42].

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

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* [55]. This results in S_t and R_t having well-defined discrete probability distributions in function of the preceding state and chosen action [55]. Therefore, the probability of receiving a particular reward and state given the previous state and selected action, which characterizes a finite MDPs dynamics, may be characterized by the 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 definition. 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} [55]. 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* [55].

From function p in equation 2.3, the state-transition probabilities, also called the *transition function*, can be computed as described by equation 2.4 [55, 42].

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

In addition, the expected rewards can be calculated for state-action pairs (equation 2.5) or state-action-next-action triples (equation 2.6) [55, 42].

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

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

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 [55]. 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 [55], 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). [55]

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 [55].

Formally, the cumulative reward received over time is also called the **expected return**, G_t, and can be described by equation 2.7 [55].

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

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* [55]. Each episode ends in a special state, called the terminal state, restarting a new sequence of states and actions completely independent of the previous episode [55]. In this context, episodes can be considered to end in the same terminal state, with different accumulated rewards for the different outcomes [55].

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 [55]. 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 [55].

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

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 [55]. 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 [15].

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

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 [55, 42]. 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 [55]. 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 [55].

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

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

The utility of such functions relies on the possibility of estimating them considering the past experience of the agent [55]. 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 [55]. These equations are called **Bellman equations**, and they characterize the relationship between the value of current state or 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 a 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

Regarding the taxonomy of RL techniques, algorithms can be divided into **model-free** and **model-based** approaches [44]. 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 [42, 44]. 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 [52]. 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 generalize the model to real-world environments so that the bias is not over-exploited becomes a very complex task [44]. Model-free algorithms can be also further divided into Q-learning and Policy Approximation techniques. The first category consists of models that estimate and optimize a Q-function to determine the expected reward of the several available action pathways. In turn, policy approximation methods focus on directly optimizing the policy and adjusting their strategies throughout execution based on the feedback from the environment. The former class of approaches can handle more complex situations, such as probabilistic or continuous action spaces. The full taxonomy of modern RL approaches can be observed in figure 2.4.

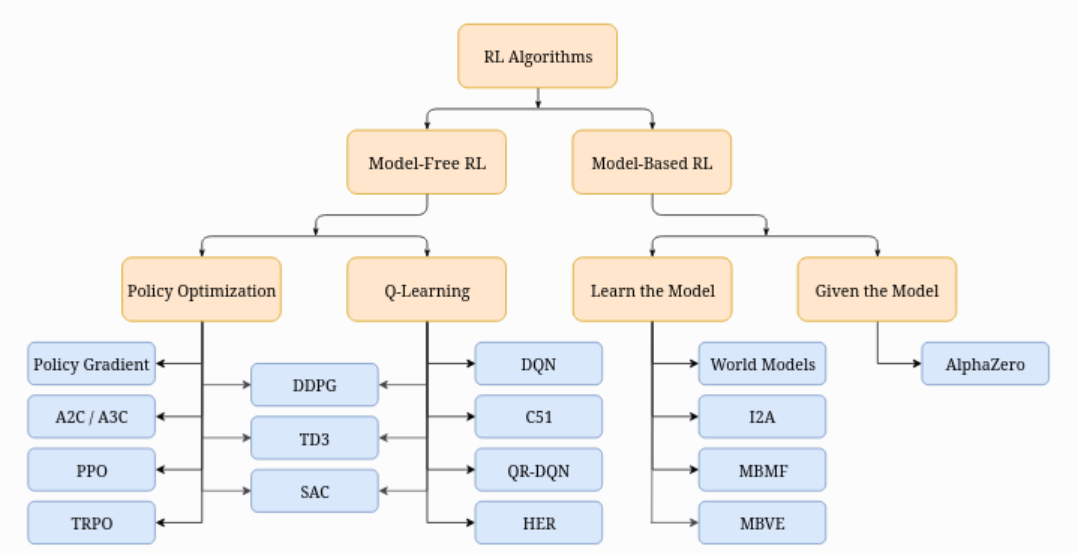

Figure 2.4: Taxonomy of Reinforcement Learning algorithms, categorized into Model-Free and Model-Based approaches. Model-free RL includes two subcategories: Policy Optimization and Q-learning. In Model-Based RL, algorithms are divided into those that Learn the Model and those that assume the model is given [44].

Furthermore, algorithms can also be subdivided into **on-policy** and **off-policy** methods. [42] On-policy algorithms evaluate and improve a single policy used to determine the agent's behaviour [42]. The methods under the policy optimization category, such as A2C and A3C [40] or the Proximal Policy Optimization (PPO) [51], usually 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 [42]. Such approaches include Q-Learning algorithms such as Deep Q-Networks (DQNs) [41, 44]. Deep Deterministic Policy Gradient (DDPG) [37] 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 [44]. Another example of an off-policy algorithm is the SAC [24] 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 [44].

Lastly, with the advent of deep learning, which became intrinsically tied with the several paradigms of machine learning,RL algorithms too have evolved beyond the traditional tabular methods [42]. Traditional RL has evolved into **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.

2.2.5 Soft Actor-Critic

The **Soft Actor-Critic (SAC)** algorithm was first proposed in [24], and it is inserted within the category of model-free Reinforcement Learning (RL) algorithms. This Deep Reinforcement Learn-

ing (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 [44, 24]. 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 [24].

Entropy is a measure of randomness for probability distributions [44]. The higher the entropy of a random variable, the more unpredictable the results. For instance, if a die 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 [44].

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

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 [44].

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

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 [44, 24].

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

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

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

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

To achieve its main goal, SAC uses function approximators for the policy and Q-function [24]. 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 is Gaussian with a mean and covariance given by neural networks [24].

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

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 [24]. 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 [24]. In this context, the soft value function can be trained to minimize the squared residual error, as depicted in equation 2.22 [24].

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

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 [24].

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

The parameters of the soft Q-function are trained to minimize the soft bellman residual. The objective function can be illustrated by equation 2.24, and can be optimized with stochastic gradients through equation 2.25. SAC uses 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 accelerate training in particularly complex tasks [44, 24].

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

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

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 $\mathcal{N}$ [44, 24]. 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 [44, 24].

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

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

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

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 mini-batches sampled from the replay buffer [24].

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\overline{\psi}$, θ, ϕ
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

2.3 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 [39, 76].

Furthermore, the performance of machine learning models strongly relies not only on their design but also on good representations of the underlying information [39]. 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 [39, 62].

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 [25, 76, 14]. 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 [14].

2.4 Graph Neural Networks

In the present, deep learning and ANNs have become one of the most prominent approaches in Artificial Intelligence research [14]. Techniques 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 [63]. 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 [14, 63].

The main objective of GNNs is to update node representations with representations from their neighbourhood iteratively [56]. 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, as outlined in [56], 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 [56].

2.4.1 Graph Convolutional Network

A **Graph Convolutional Network (GCN)** [30] is a popular architecture of GNNs praised by its simplicity and effectiveness in a variety of tasks [39, 56]. In this model, the node representations in each layer are updated according to the convolutional operation, described in equation 2.30.

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

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

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

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

σ - Activation Function

$W^k \in \mathbb{R}^{F \times F'}$ - Layerwise 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 [56]. The previous equation 2.30 can be dissected further to understand the *AGGREGATE* and *COMBINE* function definitions in a GCN [56]. For a node i, the representation updating equation can be reformulated as:

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

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

In the second equation, the *AGGREGATE* function can be observed as the weighted average of the neighbour node representations [56]. 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 [56]. 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 [56].

Spectral Graph Convolutions

Regarding the connection between GCNs and spectral filters defined on graphs, spectral convolutions can be utilized 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* [39, 56], as formally stated in equation 2.33.

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

$\theta \in \mathbb{R}^N$ - 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 [56]. In a practical context, g_θ is the function of eigenvalues of the normalized graph Laplacian matrix L, that is $g^\theta(\Lambda)$ [39, 56]. 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, according to equation 2.34 [39, 56].

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

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

$\lambda_{\mathbf{max}}$ - 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 equation 2.34 with equation 2.33, the second can be reformulated as 2.35 [39].

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

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

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 [39, 56].

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 [30] 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 [30, 56]. This means the model can alleviate the over-fitting of local neighbourhood structures for graphs whose node degree distribution has a high variance [30, 56].

At each layer, it can further be considered that $\lambda_{\max} \approx 2$, which the neural network parameters could accommodate during training [56]. With these simplifications, the main 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, minimizing over-fitting and the number of operations per layer as well [56], 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 [39, 56]. In this context, the matrix can be further re-normalized by converting $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ into $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ [39, 56]. 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) [39, 56]:

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

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

H - Convolved Signal Matrix

2.4.2 Graph Attention Network

GAT [58] 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 [56, 58].

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 $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ [56, 58]. 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 its topological information in the process. A more reasonable solution is to only attend nodes in the neighbourhood [58, 56]. 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 a node i, which can also be interpreted as a transition probability from node i to each node in its neighbourhood [56]. In the original work [58], 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 $\alpha = 0.2$ [56, 58]. 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 [58, 56], 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 below [58, 56]. 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 [58, 56].

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

2.5 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 [18, 60].

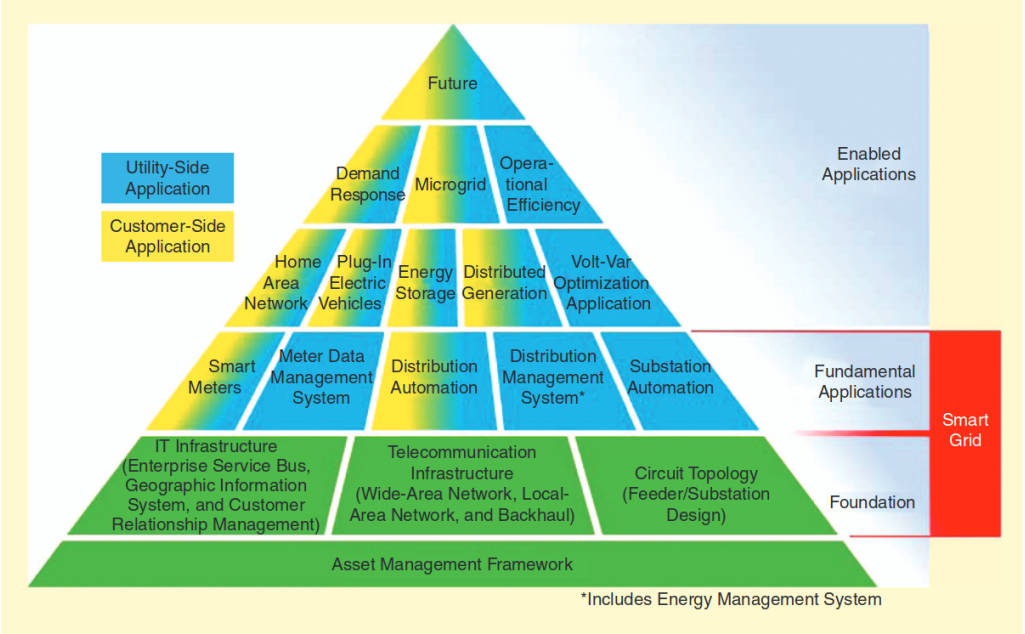

Figure 2.5: Smart Grid Infrastructure Pyramid. Depicts the hierarchical relationship between foundational layers and advanced applications. The base represents essential technological components like asset management frameworks, while the top indicates future applications, emphasizing how core infrastructure supports advanced grid functionality and efficiency improvements [18].

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 [6, 18]. Figure 2.5 describes the smart grid pyramid, which has asset management at its base. As observed, 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 [18]. In turn, these serve as building blocks for creating more intelligent systems that leverage upper-layer applications, enabling the true smart grid capabilities [18].

Chapter 3

Literature Review

This chapter documents the literature review regarding GRL approaches and DED systems. The structure of this unit is subdivided into sections that contain several aspects of the review process, with section 3.1 exposing the methodology for this literature analysis.

Sections 3.2 and 3.3 document the actual findings of the review process, with the first illustrating GRL approaches in general and the second focusing on RL techniques for solving the Dynamic Economic Dispatch (DED) problem specifically. Section 3.2 is also subdivided into the two most prominent and found approaches, which are GCN-bases techniques, in section 3.2.1, and attention-based techniques, in section 3.2.2.

Finally, in the last section 3.4, the main considerations taken from this literature review are uncovered and reflected upon.

3.1 Research Methodology

This literature review focuses on analysing the existing works 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 of the analysed literature was to only consider research from the past five years, except

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 [34, 11], smart transport [67, 3] or task offloading [22, 33]. 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 [10, 65], residential electricity behaviour identification and energy management [11], or Volt-VAR regulation [29].

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 [30, 58, 35, 21]. 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 [69, 43]. 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 [59, 43].

3.2.1 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.

[34] 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 fully connected layer to perform feature transformation with a two-layer GCN followed by three fully connected layers for the non-linear mapping of state-to-action policy in both actor and critic modules. [10] 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 [68], a three-layer GCN is used to extract node feature and graph topology information and is integrated into a Rainbow-based [28] 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 [10] further tested their proposed model's performance when inducing topology changes, yielding 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 [9], 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 [11] for residential electricity behaviour identification and energy management with a behaviour correlation graph and [70] 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 [9], with some proposing this as relevant future work [10, 68]. Beyond that, only [68], [11] [10] 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 [68, 10, 34, 70], 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	GNN	Application Domain
[74]	MDP	GCN	Interpret GNNs at model-level
[61]	DDPG	GCN	Automatic transistor sizing
[72]	A3C	GCN	Automatic Virtual Network Embeddings
[57]	Deep Q-Learning	GCN	Task Offloading in Edge Computing
[68]	Rainbow	GCN	Electrical Vehicle Charging Guidance
[34, 10]	SAC	GCN	Dynamic economic energy dispatch
[32]	SAC	GCN	Multi-access Edge Computing
[70]	DQN	GCN	Line flow control
[9]	DQN	GCN	Autonomous Exploration under uncertainty
[11]	DQN	GCN	Residential electricity behaviour 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 [77] and [16]. 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.

[65] proposes a DDPG-based algorithm improved with a GAT block with three graph Attention Layers for extracting and learning the topology information to 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. [12] 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 [67], 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 [66] and a DRL-based [48] approach. [71] develops a similar approach with a Double-prioritized DQN for the same application domain. In [77] and [16], 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 to improve the centralized training of the 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 [65, 67, 71], 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 works showed models with good adaptability to topology variations [65, 5, 12] and scalability to large scenarios [77, 65, 5, 12]. Notably, some works suggest directly embedding the GNN in th RL framework to boost the methodology computation performance and robustness [65, 65].

Table 3.2: Attention-based GRL literature.

Reference	DRL	GNN	Application Domain
[65]	DDPG	GAT	Optimal Scheduling for ADNs
[67]	Rainbow	GAT	Electric Vehicle Charging Guidance
[71]	DQN	GAT	Electric Vehicle Charging Guidance
[77]	DQN	GCN	Multi-agent Sequential Distribution System Restoration
[5]	DQN	GCN	Active Power Rolling Dispatch
[16]	AC	GCN	Multi-agent Service Restoration
[12]	SAC	GAT	Multi-agent Voltage Regulation

3.2.3 Other Approaches

This subsection includes other relevant and promising GRL approaches that combine other GNNs architectures with RL algorithms. These techniques include research on promising approaches that do not fit the main categories of GCN-based and Attention-based GRL. In this context, the list of reviewed literature and its main applications can be observed in table 3.3.

In [45], a GraphSAGE network [26] 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 [78] in the context of the dynamic economic dispatch problem.

[75] focused on solving the Job shop scheduling problem through a priority dispatching rule with a Graph Isomorphism Network (GIN) [69] 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 in literature.

Reference	DRL Algorithm	GNN Algorithm	Application Domain
[45]	DQN	GraphSAGE	Undervoltage Load Shedding
[78]	PPO	GraphSAGE	Dynamic Economic Dispatch
[75]	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 dispatch problems [46]. In table 3.4, the relevant RL approaches to the problem, including GRL techniques, can be observed. On a general level, the reviewed literature regarding DED solutions takes into account a wide range of considerations and constraints. The recent works seem to favour a focus around Energy Storage Systems (ESSs) management and RES curtailment, given the current global energetic issues and the relevance of these technologies for solving them.

Some systems [10, 27, 78] 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 [38]. In general, GRL algorithms show superior performance in relation with plain DRL approaches [10, 34, 78] as also concluded in section 3.2.1, with studies successfully ensuring the adaptability of the GRL models to variations in the scenario's topology [10, 78]. However, only [5] 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 literature.

Reference	Approach	Application
[73]	DDPG with Prioritized Experience Replay Mechanism and L2 Regularization	Integrated Energy Systems, Utility (Selling, Purchasing and Gas) and ESSs
[31]	DDPG	Thermal Power, Renewable Energy Sources and ESSs
[27]	SAC with Imitation learning	Thermal Power, Renewable Energy Sources and Voltage deviation
[10, 34]	GCN-SAC	Utility (Time-of-Use), Thermal Power, Renewable Energy Sources and ESSs
[78]	GraphSAGE-PPO	Thermal Power, Renewable Energy Sources, ESSs and Voltage Deviation
[38]	Multi-Agent RL with Function Approximation and Diffusion Mechanism	Utility (Time-of-Use), Thermal Power (Diesel) and ESSs (Considering degradation)
[5]	Multi-Agent GAT-DQN	Thermal Power, Renewable Energy Sources and ESSs

The main considerations regarding action space and observation space found in literature can be observed in tables 3.5 and 3.6, respectively. Regarding the first, the literature doesn't deviate much from including dispatch actions on non-renewable generators in the form of increments or decrements of the power output. Those intervals are restricted by maximum ramp rate and maximum and minimum output limits [34, 10]. In some works [78], the problem is simplified by using discretization to reduce the complex dimensionality of DED. These use output changes of fixed intervals, considering only actions that constitute a multiple of that interval. Furthermore, other already mentioned features can be included in action space, such as ESS power output set-point [78, 34, 10, 38, 73, 65] Renewable Energy Source (RES) curtailment [78, 34, 10, 65], and voltage control [27, 65]. The reviewed literature favoured the inclusion of ESS storage set-point in the action space, along with its respective output value in the observation space, with some recent works applying a specialized focus on this element, such as [38] which also accounts for battery health and its degradation.

Table 3.5: Action space of DED systems.

Reference	Model	Action Space				
		Dispatch	**Curtail**	**ESS**	**Voltage**	**Others**
[78]	SAGE-PPO	x	x	x		
[34, 10]	GCN-SAC	x	x	x		
[27]	LM-SAC & IL-SAC	x			x	
[38]	Cooperative Multi-Agent	x		x		
[31]	FH-DDPG and FH-RDPG	x				
[73]	Improved DDPG			x		(CHP + GB)
[65]	GAT-DDPG		x	x	(SVC)	(FL)

With respect to the observation space, the choices were sparser. The literature mainly encompasses the active power of both loads and generators, but there was a division into works that included reactive power and those that did not. Beyond that, [27] is the only paper that considered the voltage of generators and loads, which is sensible given that it also considers voltage control actions. [78] also considers voltage levels but only for buses and not for the individual elements, including their voltage angle.

Lastly, [27] opted to include immediate forecasts of the active power demand of loads and renewable generator output before curtailment actions. In turn, [38, 73] used time-varying electricity prices from utility grids, with the former also accounting for gas prices.

Table 3.6: Observation space of DED systems.

Reference	Model	Observation Space							
		Generators	**Loads**	**ESS**	**Nodes**	**Lines**	**Topology**	**Timestep**	**Others**
[78]	SAGE-PPO	P and Q	P and Q		P, Q, v, and θ	P and F	x		
[34, 10]	GCN-SAC	P	P and Q	SOC			x	x	
[27]	LM-SAC & IL-SAC	P, Q, and v	P, Q, and v			rho and F			Forecast of P^{LOAD} and $\overline{P^{\text{RES}}}$
[38]	Cooperative Multi-Agent	P	P	SOC and SOH					
[31]	FH-DDPG & FH-RDPG	P^{RES}	P	SOC					
[73]	Improved DDPG		P	SOC					Price of purchasing energy and gas
[65]	GAT-DDPG	P and Q	P and Q				x		

3.4 Conclusions

In this chapter, we reviewed relevant literature for this dissertation's main research topic, GRL algorithms. GRL is a very promising field where several different applications and techniques have already been studied. GNNs architectures such as GCN have been extensively applied with DRL algorithms for enabling feature extracting from graph-based state representations [10, 9]. Architectures such as the GraphSAGE and other attention-based have also been successfully applied with very promising results [45, 65] 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 to 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 [65, 77]. 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 concerning plain RL algorithms in graph-based environments, which portrays GRL is a potential solution to this problem [10, 34, 78, 67, 68, 70]. The reviewed literature was quite scarce and sparse, probably due to the field's novelty. 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 direct embedding of the GNN into the RL framework [65, 67].

- 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 dispatch problems [46]. We were able to find evidence of GRL being successfully implemented in DED systems such as [10] and [34] in a power grid simulation context, showing efficient performance and scalability in comparison with plain DRL models in the same issue. However, as 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 built with different DED requirements.

Chapter 4

Methodological Approach

In this chapter, we formally state the problem this work aims to address and the proposed solution. Firstly, in section 4.1, the problem statement is exposed, and in section 4.2, the formal definition and considerations for the DED problem and its translation into an MDP, are characterized. Sections 4.3 and 4.4 delve into more technical aspects of this work, highlighting the specifications of the machines used to conduct the experiments and the main technological requirements, respectively. Section 4.5 further extends on the main simulation environment features and attributes, followed by section 4.6, which explains the general architecture of the proposed solution. Lastly, the evaluation process and other relevant considerations are disclosed in the last section 4.7. In table 4.1, the mathematical notation used from this chapter onwards can be consulted.

Table 4.1: Notation used throughout the methodological approach and onwards.

Symbol	Definition
$F(t)$	Total Operational Cost at time step t
$F_{\text{NRES}}(t)$	Total non-renewable generators operational cost at time step t
$F_{\text{RES}}(t)$	Total renewable generators operational cost at time step t
T	Terminal time step
t	Time step
c_i^{NRES}	Cost of conventional generator i in €/MWh
$P_i^{\text{NRES}}(t)$	Current generated power of non-renewable generator i at time step t in MW
β_{RES}	RES wasted energy penalty term
$P_i^{\text{RES}}(t)$	Current generated power of renewable generator i at time step t in MW
$\overline{P^{\text{NRES}}}$, $\underline{P^{\text{NRES}}}$	Current maximum and minimum output power of non-renewable generator i
$\overline{P_i^{\text{RES}}}(t)$	Current maximum output power of renewable generator i at time step t and before curtailment in MW
$P_i^{\text{LOAD}}(t)$	Active Power demand of load i at time step t
$Q_i^{\text{LOAD}}(t)$	Reactive Power demand of load i at time step t
F_i or $F_{i,j}$	Power line i status (connected or disconnected)
$\text{rho}_i(t)$ or $\text{rho}_{i,j}(t)$	Relative flows of power line i or with origin in substation i and destination in substation j. The ratio of the flow divided by its thermal limit
$\overline{\eta_i}$, $\underline{\eta_i}$	Maximum ramp up and down limits for non-renewable generators
$P_i^{\text{G}}(t)$	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 power lines
θ_{soft}, θ_{hard}	Soft and Hard overflow thresholds
T_{soft}	Interval of time the agent is allowed to be in soft overflow in the same line
$T_{\text{reconnect}}$	Cool-down on power line reconnection after an overflow-related disconnection
$\overline{F_{\text{NRES}}}$	Maximum generator output possible in the power grid
$\overline{c^{\text{NRES}}}$	Average cost in in €/MWh of connected non-renewable generators

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 instinctively represent several problems and their objects. In some network-oriented domains, these representations reveal underlying features that plain Euclidean data can't naturally represent. This problem becomes even more difficult considering that graph data is complex and sparse, which requires methods that efficiently extract representations from underlying data.

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 profoundly 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 an initial phase, the gathered literature is sparse, lacking 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 explore the problem within the scope of single-agent reinforcement learning (RL) algorithms, acknowledging that multi-agent systems present considerably more complex challenges for implementation. Specifically, within the smart grid services application domain, potential enhancements in GRL techniques will be applied to address the DED problem. This involves optimizing power generation costs while ensuring grid stability. Additionally, the proposed GRL models could be adapted to address other smart grid challenges, such as Undervoltage Load Shedding and Volt-VAR Regulation. However, they are considered out of the direct scope of this study.

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 in 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, improving 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 economically and reliably has been studied early on by academics [64]. The Dynamic Economic Dispatch (DED) problem addresses this concern, with a particular 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 following 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.

4.2.1.1 Objective Function

As stated previously, the main goal of the DED problem is to minimize 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 time step t can be described by equation 4.1 and comprises the first component of the DED objective function. The chosen DED formulation is heavily inspired by the works of [10, 34, 34] without the considerations relative to ESS.

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

In this context, conventional generation cost is described as the total sum of the generator's operating cost, which in turn is defined 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 depreciation of $F_{\text{NRES}}(t)$, which describes a straightforward economic cost of the operation of a power grid.

However, modern distribution grids increasingly 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 disadvantages, mainly concerning the hidden cost of unused RES energy abandoned due to security reasons, such as preventing line overflow. In practice, This discarded energy ends up being compensated with other generators' power.

In this manner, to maximize the RES usage, a second component of the objective function 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, also used in literature [10, 34, 34]. 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 minimizing 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.

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

4.2.1.2 Constraints

The system must follow several operational constraints to maintain stability and reliable power distribution. These restrictions are related to generators, voltage, power lines and overall system 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 [10, 34, 31].

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

Another equally important group of constraints is Kirchoff's Laws of current and voltage. These fundamental principles establish the foundational rules that govern the behaviour of electrical circuits and power networks. 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, Kirchoff's current law, illustrated in equation 4.6, dictates that the sum of all currents entering and leaving any node in a power network 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 [10, 34, 38]. Furthermore, as illustrated by equation 4.8, the varying power between two different time steps is also restricted by the maximum ramp-up and down limits, $\underline{\eta_i}$ and $\overline{\eta_i}$ [10, 34].

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

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

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 time step t is bounded by a maximum production value, $\overline{P_i^{\text{RES}}}(t)$ and 0 [10, 34].

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

Regarding power lines, there are 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 power line surpasses the hard overflow threshold, θ_{hard}, it's instantly disconnected. This corresponds to an instantaneous overcurrent and prevents the power system from operating in cases of extreme overflow [4].
- **Soft Overflow** - A line is allowed to surpass the soft overflow threshold, θ_{soft}, for T_{soft} time steps before being disconnected for safety reasons. This consists of a time overcurrent and prevents power lines from being in overflow for more than a given time frame. [4].

Regardless of the scenario, when a power line is disconnected for safety reasons, it's always affected by a reconnection cool-down, $T_{\text{reconnect}}$.

4.2.2 Markov Decision Process (MDP)

In this section, DED 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}, as illustrated in equation 4.10. 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 is the standard option throughout literature, used in [78, 34, 10, 27] together with the ESS power set-point. In this case, we don't consider the inclusion of ESS elements to simplify the action space and scale both types of action to $[-1, 1]$ as the implementation requires.

This second type of actions refers to the energy discarded from RES for grid stability and security purposes. However, they're defined 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 from RES, making these actions crucial for an optimal grid operation.

$$a(t) = \{P_1^{\text{NRES}}, P_2^{\text{NRES}}, \ldots, P_N^{\text{NRES}}, P_1^{\text{RES}}, P_2^{\text{RES}}, \ldots, P_M^{\text{RES}}\} \tag{4.10}$$

Observation The observation space comprises four components concerning renewable and non-renewable generators, loads and power lines. The segments blocks that compose the observation space are depicted in equations 4.11, 4.12, 4.13, and 4.14.

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

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

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

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

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

This tuple encompasses the main elements of the power grid and their characteristics without capturing their topology, similar to the features included in the observation space of [27] without forecast keys. Additionally, reactive power was also only included for the loads, and voltage was not considered for any element as observed in most literature [34, 10, 38, 31, 73, 65]. This space is considered for pure DRL algorithms.

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 time step t can be observed in

equation 4.16. This formulation is once again similar to the works of [34, 10] but with the difference of aggregating non-renewable generated power and renewable generated power in two separate keys and without considering ESS power set-point.

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

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 the *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. This is similar to the implementation used in [34, 10], with the distinction that in this case, it's scaled to $[-1, 1]$. This addition allows better comparison along different-sized scenarios, which facilitates scalability analysis.

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

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 time step 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 price 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 considerations that account for invalid actions and errors that cause a terminal state of the system or a *game over*. If, at any time step, the power system reaches an infeasible state, there is a game over, and the episode ends. These may happen at any time when no subsequent action conforms to the constraints of power balance and Kirchoff's laws. In this case, the reward at the time step in which 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 power lines, the action is considered invalid and the returned reward is zero.

Episodes and Steps Considering the time difference between the two steps, each time step 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 time steps.

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

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

4.3 Device Specifications

The experiments were executed on two different machines, the author's own and a remote server made available by Artificial Intelligence and Computer Science Laboratory (LIACC). Tables 4.2 and 4.3 highlight their hardware specifications.

Table 4.2: MSI laptop specifications.

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 4.3: LIACC server specifications.

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

4.4 Technologies

Beyond being a well-documented, fast, robust and modular language, *Python* is a 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 4.4, the main requirements for the project can be observed, along with their respective version number and a short description of their purpose and functionality.

Table 4.4: Project technological requirements.

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

Grid2Op is a critical requirement used to simulate the power grid operation in different manageable scenarios, further discussed in the following subsection 4.5. *Stable-Baselines3* [49] was chosen for its performance, customization and descriptive documentation, while *PyTorch Geometric* [19] 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.

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

4.5 Simulation Environment

The *Grid2Op* framework [4] 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 predefined scenarios used in coding competitions and based on real-world data [4].

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 the 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 in these environments. Figure 4.1 shows a graphical plot of the environment state at an arbitrary time step.

This framework is compatible with the *Gymnasium* framework [17], a widely used toolkit for developing RL algorithms, which will also 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 set-point of renewable generators below the current maximum available power [4]

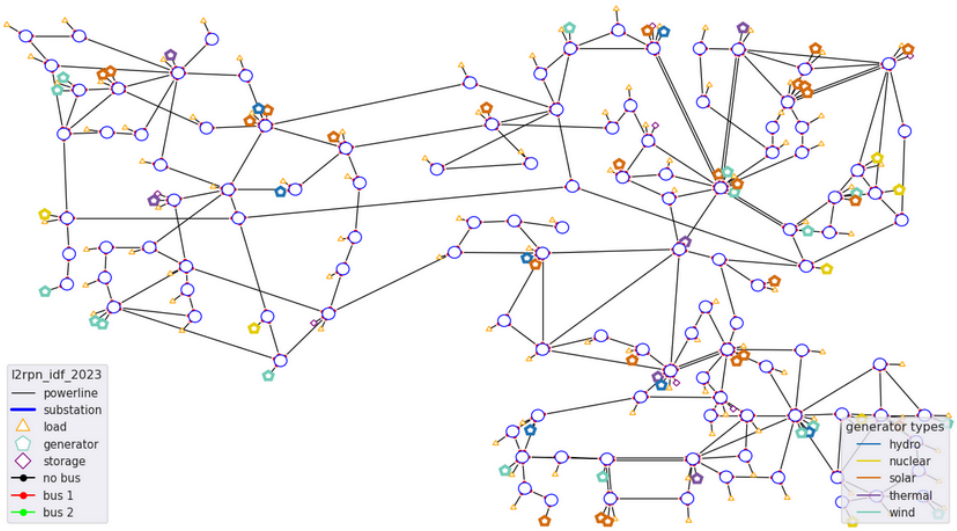

Figure 4.1: *Grid2Op* `l2rpn_idf_2023` 118-bus test case [4]. This graphical diagram is produced by the already implemented logic of *Grid2Op* that allows the visualization of the current states of the power grid.

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[4].
- **Power Lines** represent edges in the power grid and connect the different buses. They represent the physical transmission and distribution lines and allow power to flow from one part of the grid to another [4].
  - 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 [4].
  - $P_i^{\text{NRES}}(t)$ - 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 with an individual bus [4].
  - P_i^{LOAD} - Active power at load i
  - Q_i^{LOAD} - Reactive power at load i

Beyond *Grid2Op* to build and run the test cases and *gymnasium* as the RL framework, this project will use *PyTorch* [47] with *PyTorch Geometric* [19] 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 several customizable environment parameters that control important grid properties and settle some bounds on the agent-environment interaction. As a significant number of parameters consider details that are out of this project's scope, only the relevant parameters are exposed in table 4.5.

Table 4.5: *Grid2Op* relevant parameters [4].

Parameter	Description	Default
`NO_OVERFLOW_DISCONNECTION`	If `true`, the power lines over their thermal limit will never be disconnected	`false`
`NB_TIMESTEP_OVERFLOW_ALLOWED`	Number of time steps a power line is allowed to be in a soft overflow before it gets disconnected by time overcurrent	2
`NB_TIMESTEP_RECONNECTION`	Number of time steps 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. It might help the training and learning process of the model and increase the survivability of the agent	`false`

Except for one, most of the parameters were left at their default values, depicted in table 4.5. This includes the overflow thresholds, all of the time step limits mentioned and the direct current flag, whose values were found to be sensible enough to maintain a realistic yet feasible model of the environment. The only parameter tampered with was `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION` to improve the learning efficiency of the algorithms during training.

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 to meet the real-time load demand and reduce the operating cost of the power distribution grid. The general architecture of the solution can be observed in figure 4.2.

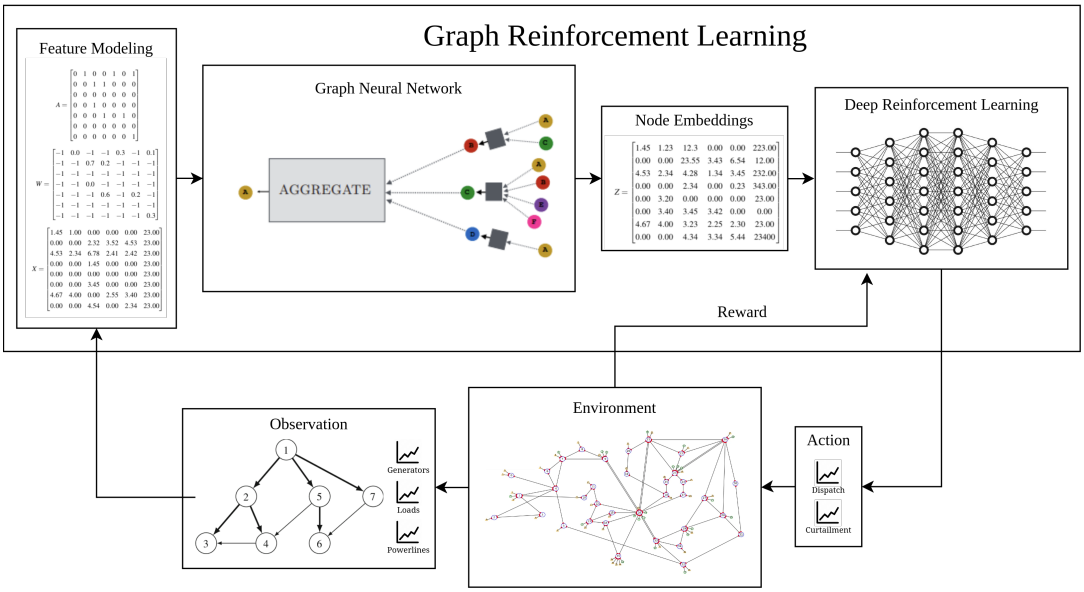

Figure 4.2: General Architecture of the Proposed Solution. This implementation uses a GNN algorithm to extract relevant node embeddings from each state and consequently train a DRL method with the new and more representative depictions.

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.

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 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 implementing 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: the storage of raw observations in the replay buffer, where the data is batched from instead of directly saving the extracted features. To solve this, the GNN feature extraction layer was included directly in the observation space, which ensured the execution of the module only once per step.

4.7 Evaluation Metrics

The solution described in the previous subsection 4.6 involves intricate operational mechanisms that call 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 and 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 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 to maintain grid reliability, stability, and security beyond an economic operation. A high level of survivability reflects a solid ability to manage environmental constraints, leading to improved power system availability.

This factor is evaluated through the survival rate metric, which represents the percentage of maximum episode length the model remains operational.

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 can perform well in real-time execution. In this context, assessing the solution's decision computation performance and measuring the time necessary for the model offline training and the observed CPU/GPU resource utilization is essential.

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 generalization capability. Additionally, cross-validation was considered, but its use was discarded because of the algorithms' slow performance and apparent reduced learning efficiency.

Finally, the defined process considered a training of 5000 episodes for the intermediate calibration process and, 10000 episodes for the final comparative analysis of the algorithms. These values were mainly motivated by the options taken in similar literature [10, 34, 27] and by the timing restrictions that bound this work.

Chapter 5

Result Discussion

In this chapter, the obtained results are thoroughly analyzed and discussed. The performed research can be subdivided into intermediate calibration and final evaluation experiments. The first section 5.1 highlights some important implementation features and is followed by the intermediate experiments.

The former category of studies is composed of experiments that tackle: the reward function in section 5.2; action space, in section 5.3; and GNN tuning, in section 5.4.

Lastly, the final results of the best models in the 36-bus and 118-bus environment are exposed in sections 5.5 and 5.6, respectively.

5.1 Experimental Setup

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

5.1.1 Algorithms

The implementation of the proposed GRL implies choosing the DRL and GNN algorithms that compose it. Given that it is the main object of study, a priority was given to testing different GNN models, which introduce the most prominent changes within the proposed architecture. Because of its well-established position as a state-of-the-art solution in similar problems [34, 10, 32, 27] and the timing restrictions, the SAC algorithm was privileged among the available DRL implementations. These components were then combined to develop the proposed GCN-SAC and GAT-SAC implementations.

5.1.2 Parameters

Considering the choice of parameters, the timing constraints again prevented a more specialized calibration process of the DRL hyperparameters. Similarly to the selection of the used algorithms, the primary focus of the calibration experiments was around the GNN parameters, specifically GCN. The main choices in SAC hyperparameters were inspired by the parameters used in [27] and in [34, 10], which implemented the model as a solution to DED in a similar perspective.

Beyond that, the relevant environment and GNN parameters were targeted to thoroughly analyze their performance impact, with the rest being left at their default values. In the context of the first category, other possible action space and reward function configurations were studied, while in the second, the aggregation scheme, number of layers, and output features received a distinct focus. Regarding other GNN features, some specific GCN and GAT parameters were also studied when appropriate.

5.1.3 Scenarios

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 [4]. 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 5.1.

Table 5.1: *Grid2Op* test case sizes [4].

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

Given its small grid size, the first case was used for the primary implementation and development of the algorithms. Motivated by its considerable grid size and amount of training data, the second test case was mainly used to calibrate the model's main parameters. It's a subset of the original 118-bus system [13] with 50 years' worth of data divided into independent [1] weekly scenarios. The hierarchical relation between these two scenarios and their configuration can be observed in figure 5.1.

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 [4]. It was mainly used to study the algorithm's scalability and performance in larger scenarios compared to the baseline plain-DRL model.

[1] non-consecutive

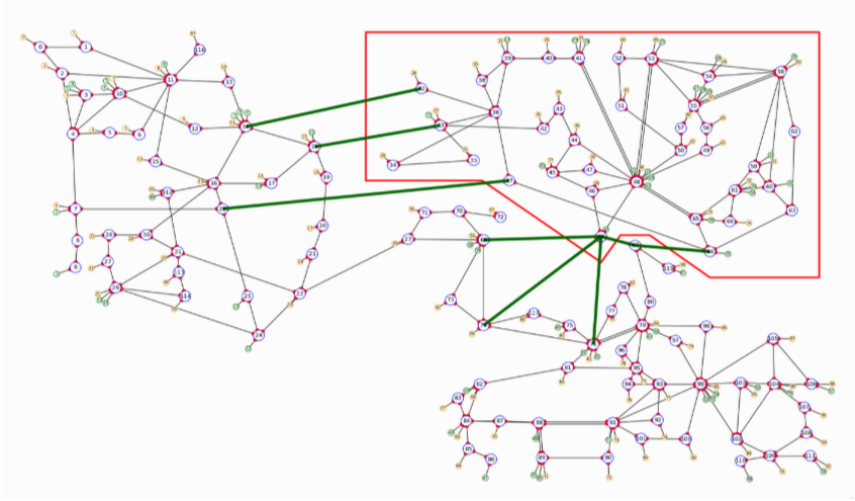

Figure 5.1: Hierarchical relation between the utilized 36-bus and 118-bus scenarios, with the first being observed in the red rectangle. The green pentagons illustrate the existent generators, while the red triangles represent the loads [4].

5.1.4 Reproducibility

A major concern and challenge brought by the technologies at hand was implementing a solution that brought reproducible results. This is critical to ensure appropriate analysis and calibration of the model and 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 was thoroughly analyzed and studied, ensuring that all appropriate seeds were being defined. However, the results for the same machine still differed. This was later found to be related to the non-deterministic nature of some *torch_geometric* functions in the GPU, namely *torch-scatter* operations [54] [2].

Although, at first, this issue seemed unsolvable, a specific parameter was found in the *PyTorch* Documentation [2] that successfully mitigated this problem and allowed for result reproducibility. Nevertheless, for different machines, the results still showed inconsistencies.

```
1  import torch
2
3    SEED = 1111111
4    torch.seed_everything(SEED)
5  torch.use_deterministic_algorithms(True)
```

5.1.5 Flexibility

One of the objectives that was also a significant focus of the implemented solution was to allow different combinations of DRL and GNN algorithms. Since GRL still is a relatively recent field, research around this area must build foundations for other studies, and this is facilitated by the developed flexible framework that enables experiments on other currently unexplored models.

Using two popular frameworks for each type of algorithm (DRL and GNNs) with a consistent API made this goal more straightforward 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 the Penalty and Bonus Factor rewards, β, the experiments considered three distinct values: $\{0.2, 0.4, 0.6\}$.

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

As depicted in figure 5.2, for the reward bonus strategy, the best model was the one with $\beta = 0.6$, outperforming the other alternatives in both average accumulative reward and survival rate during test.

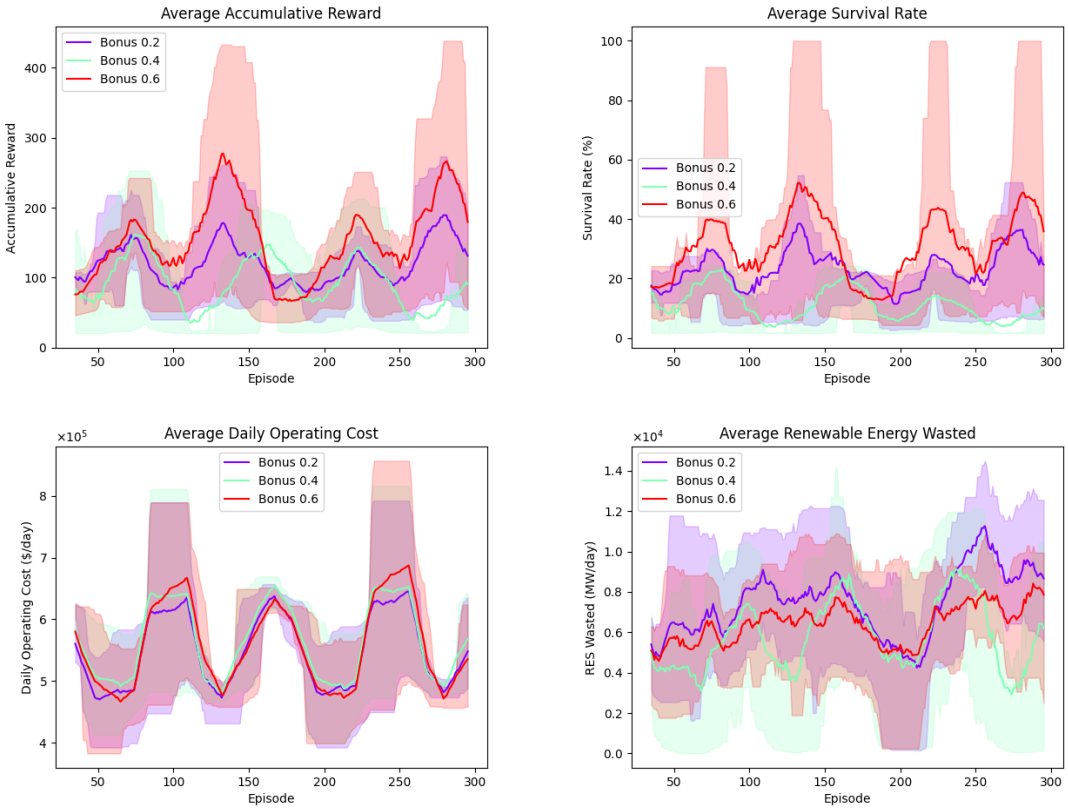

Figure 5.2: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a bonus reward for used renewable energy during test with bonus factors of 0.2, 0.4 and 0.6. The reward function with $\beta = 0.6$ outperformed the other models in the first two metrics, showing a good balance among the others.

Nevertheless, when analyzing the other dimensions directly related to the DED, the $\beta = 0.6$ bonus factor achieved a good balance between the average daily operating cost and wasted renewable energy. A factor of 0.2 resulted in a smaller cost but with increased wasted RES, while 0.4 was able to perform significantly better in terms of maximization of RES but with a lower survival rate and higher daily operating cost.

Regarding the penalty strategy, the model with a penalty factor of $\beta = 0.4$ was superior, closely followed by the model with $\beta = 0.2$, which achieved higher but less consistent results in accumulative reward. Furthermore, the first surpassed the second in survival rate by a large margin and a significantly lower cost and abandoned RES than the first.

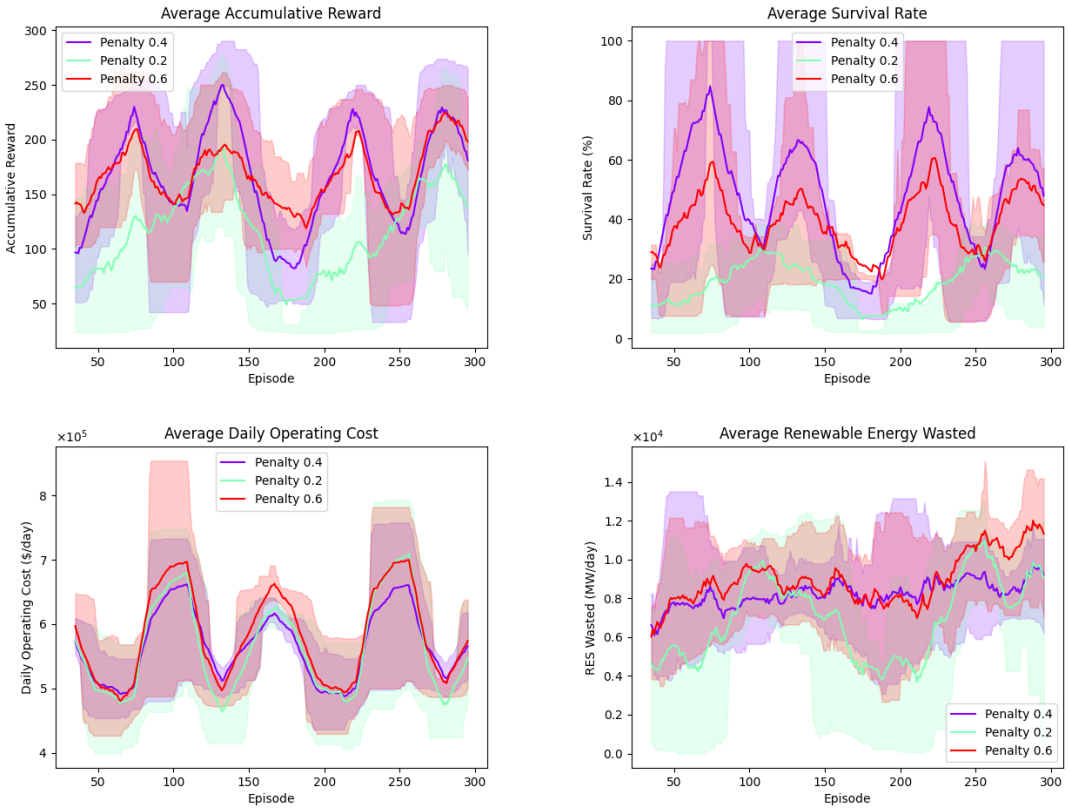

Figure 5.3: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a reward penalty for used renewable energy during test with penalty factors of 0.2, 0.4 and 0.6. Although the reward function with $\beta = 0.6$ achieved higher results in some cases, the model with $\beta = 0.4$ was the most consistent in all dimensions.

The average of the metrics analyzed in the context of this experiment can be observed in table 5.2. The best model overall was the one that introduced a penalty factor of $\beta = 0.4$ on wasted renewable energy. This model achieved the best trade-off between survival rate, average cost and average wasted renewables, which motivated its choice as the optimal configuration for the reward function. Additionally, the default economical reward function was also trained, achieving a solid middle position in terms of overall performance, showing the potential losses of using the two idealized strategies (bonus and penalty) with non-optimized factors.

Table 5.2: Average Accumulative Reward, Survival rate, Daily Operating Cost, and Wasted Renewable Energy of the different studied reward configurations during test.

reward	β	**Metrics**			
		Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
Penalty	0.6	166.67	38.50	586562.08	8926.40
Penalty	0.4	159.83	44.95	568533.92	8187.96
Bonus	0.6	148.70	30.16	562174.43	6484.24
Economical	-	127.20	24.71	562149.32	8353.61
Bonus	0.2	118.37	22.77	553932.52	7420.42
Penalty	0.2	113.29	18.49	566361.78	6954.43
Bonus	0.4	93.86	11.70	569404.33	5819.33

5.3 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: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with and without curtailment actions during test.

no_curtail	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	130.15	33.57	536423.98	0.00
False	22.13	1.84	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 this type of action. 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 learn optimal dispatch policies appropriately.

With the purpose of further analyzing and solving this problem, two methods described in the following subsections 5.3.1 and 5.3.2 were experimented to mitigate it.

5.3.1 Limit Curtailment Infeasible States

The first method analyzed the effect of using the *Grid2Op* `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION` 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`.

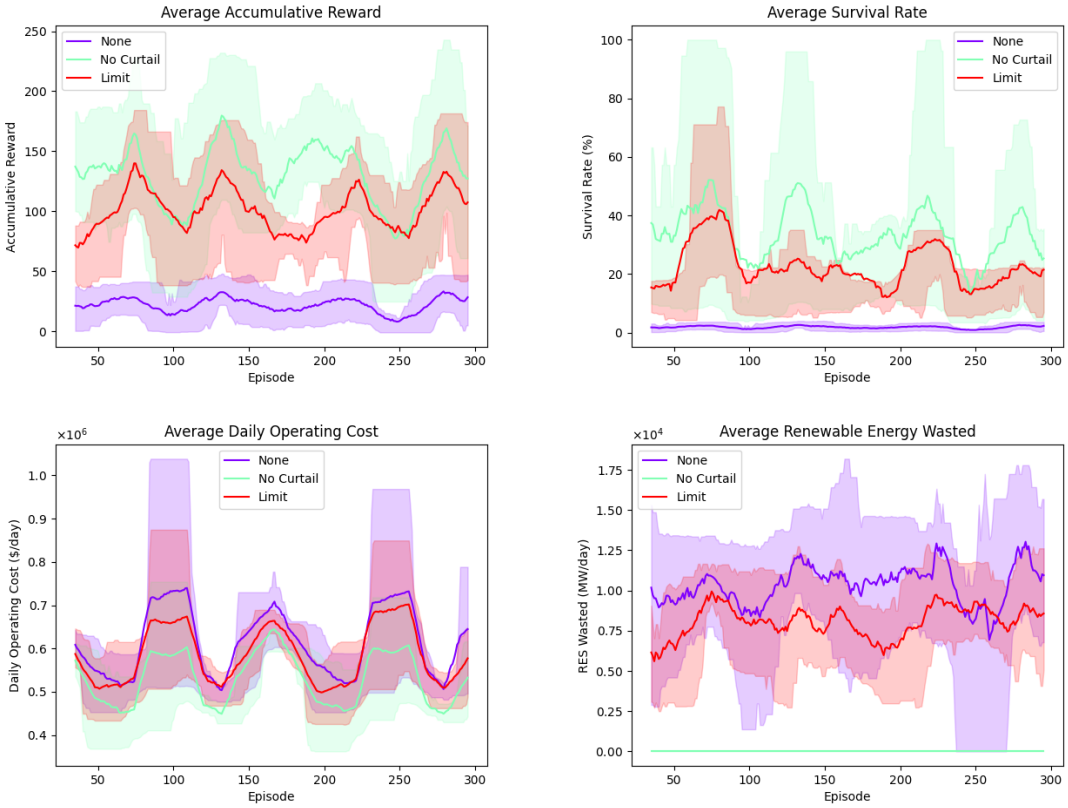

Figure 5.4: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with and without the `limit_inf` parameter against a model without curtailment actions during test. The inclusion of this parameter positively affected performance but was not enough to surpass the performance of the model with no curtailment actions.

The results in figure 5.4 confirmed the initial suspicions, demonstrating that while performance is significantly 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 5.4, 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.

Table 5.4: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with and without the `limit_inf` parameter against a model without curtailment actions during test.

no_curtail	**limit_inf**	**Metrics**			
		Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	False	130.15	33.57	536423.98	0.00
False	True	98.84	22.11	588610.57	8039.41
False	False	22.13	1.84	619721.30	10315.47

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 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 model's behaviour without curtailment actions, which by default uses all of the available renewable energy at each time step, resulting in 0.0 MW of wasted RES energy.

5.3.2 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. Additionally, we use a decay function to amplify the freedom of the model as it goes further into the training phase. This was expected to enable the algorithm to achieve a longer survival rate in the first episodes, which could translate into a better learning performance. In this context, two decay strategies were devised, corresponding to linear and square root decay and represented in equations 5.1 and 5.2, respectively, where $E_{\text{decay\_end}}$ is the episode where the model reaches the final limit, E_{train} the amount of training episodes, $\underline{P^{\text{RES}}}$ is the final curtailment lower bound, and α is a factor that controls the rate at which the bound decays.

$$y = 1 - \frac{(1 - \underline{P^{\text{RES}}})E_{\text{train}}}{E_{\text{decay\_end}}} \tag{5.1}$$

$$y = (1 - \underline{P^{\text{RES}}}) \sqrt[\alpha]{\frac{E_{\text{decay\_end}} - E_{\text{train}}}{E_{\text{train}}}} \tag{5.2}$$

Reflecting on the secondary goal of maximizing the usage of generated renewable energy, the idealization 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:

- No bound;
- 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 5.3.1. The proposed techniques were studied with the `limit_inf` parameter, and its train and test results are observable in figure 5.5 and table 5.5, respectively.

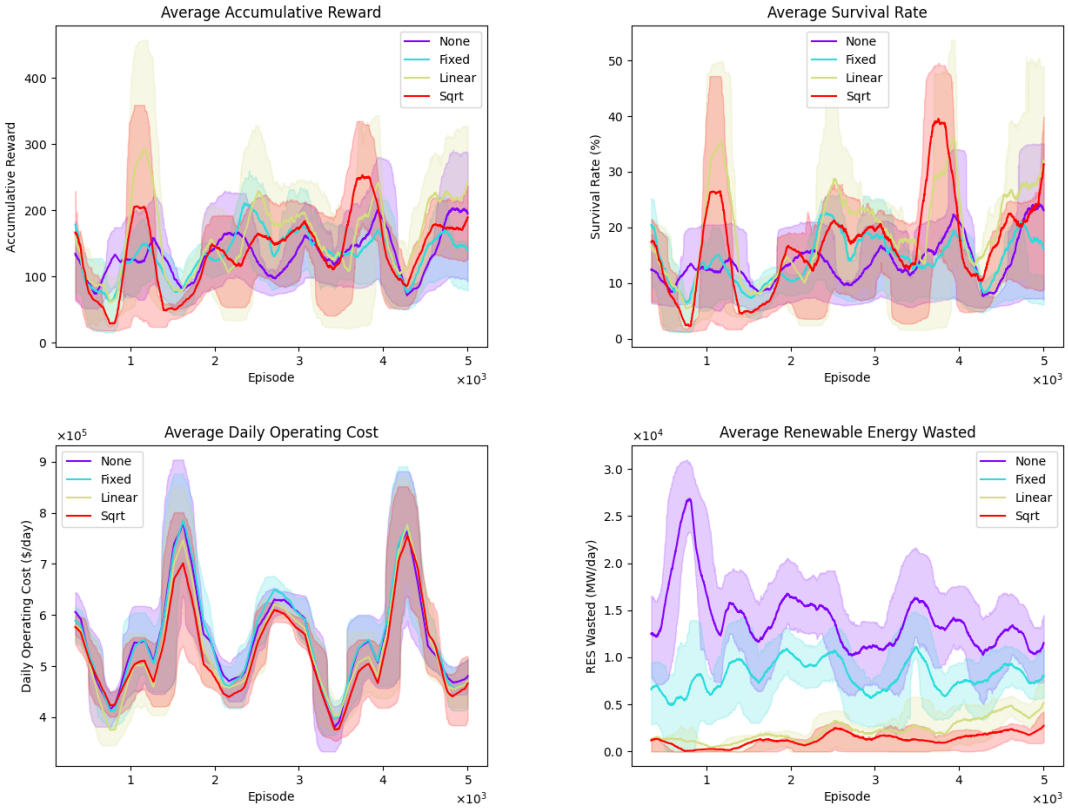

Figure 5.5: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models without a curtailment lower bound, with a fixed limit, and with linear and square root limit decay during training. The square root decay method showed the best overall performance. The methods mainly affected the wasted RES, with the square root decay showing the lowest results during training and the model with no lower bound the highest.

The training performance results show that using the proposed lower bound strategies significantly affected the amount of abandoned RES energy. This implies that using a lower bound successfully promotes the maximization of renewable energy, with models benefiting from utilizing the decay functions to allocate more freedom gradually within their action space. However,

this correlation was not as evident in the test results, in table 5.5, with the methods showing a slight performance improvement in the same metric.

Table 5.5: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models without a curtailment lower bound, with a fixed limit, and with linear and square root limit decay during test.

decay	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
sqrt	153.77	31.16	579978.88	7356.86
Fixed	122.58	22.88	541391.22	7480.09
Linear	111.52	14.35	550832.62	7092.56
None	98.84	22.11	588610.57	8039.41

Furthermore, results proved that the introduced curtailment lower bounds considerably increased the model's average accumulative reward and survivability, with the square root decay method achieving the overall highest average cumulative reward in the test phase. The models with a fixed lower bound and linear decay also exceeded the technique presented in the former subsection without outperforming the model without curtailment. The best model outperformed the one without curtailment actions 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: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with square root curtailment limit decay, the `limit_inf` parameter and without curtailment actions during test.

no_curtail	**limit_inf**	**decay**	**Metrics**			
			Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
False	True	sqrt	153.77	31.16	579978.88	7356.86
False	False	sqrt	153.77	31.16	579978.88	7356.86
True	False	None	130.15	33.57	536423.98	0.00
False	True	None	98.84	22.11	588610.57	8039.41
False	False	None	22.13	1.84	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.

5.4 GNN Hyperparameter Tuning

As a key area of this research work, the GNN component of the proposed algorithm was given a particular 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 organized as follows: in this first subsection 5.4.1, the performance of the several available aggregation schemes is unveiled and analyzed; in subsection 5.4.2, the focus is shifted towards the number of GNN layers; in the following subsection 5.4.4 the effect of other specific GCN parameters is analyzed, and, lastly, in the last subsection 5.4.5 the same process is performed for the GAT architecture.

5.4.1 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 `sum`, `max`, `min`, `mean` and `mul`.

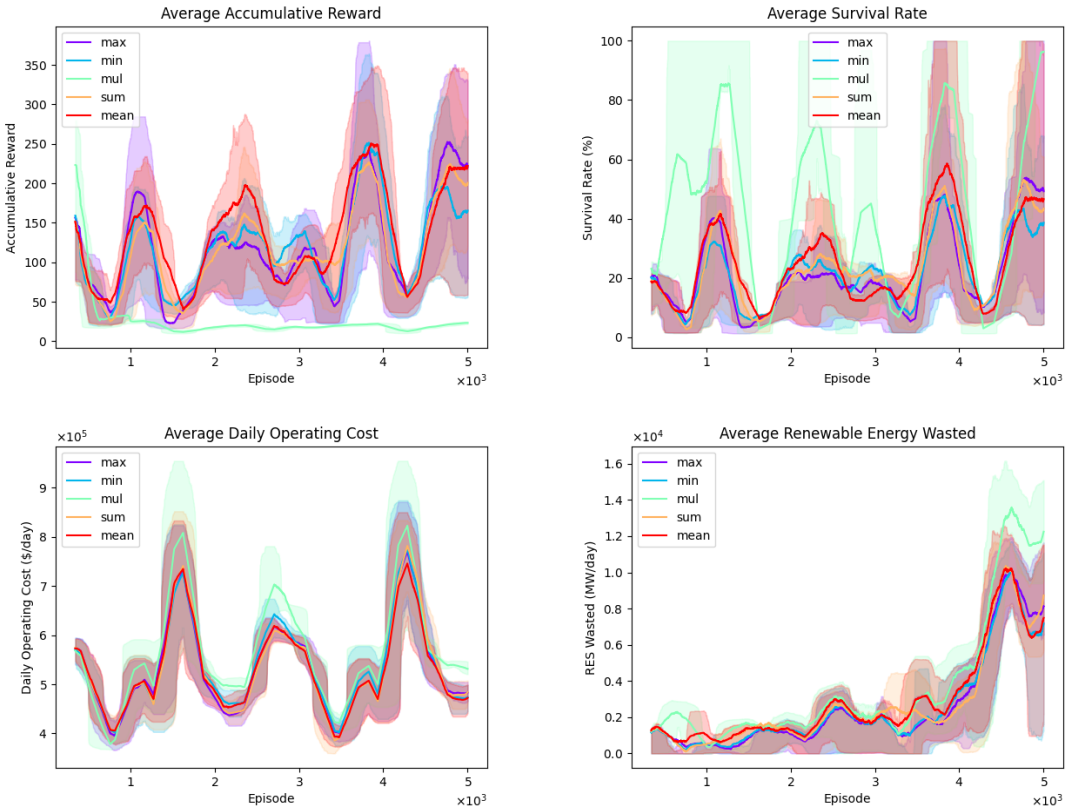

Figure 5.6: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during training. The `max` function showed the best overall performance, with the `mul` showing a significantly low average accumulative reward while also yielding the best survival rate by a large margin.

As observed in figure 5.6, training performance revealed a poorly performing `mul` scheme compared to the other tested functions. However, the `mul` function yielded a significantly higher average survival rate of 54.77 % in tests, over 43 % of the model with the second-best survivability, 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 in figure 5.7 and table 5.7, this discrepancy is primarily due to a high average daily operating cost, close to four thousand euros more than any other model. Beyond that, this model did not perform optimally when analyzing the average energy from RES wasted, with over three thousand megawatts of unused renewables in relation to every other model.

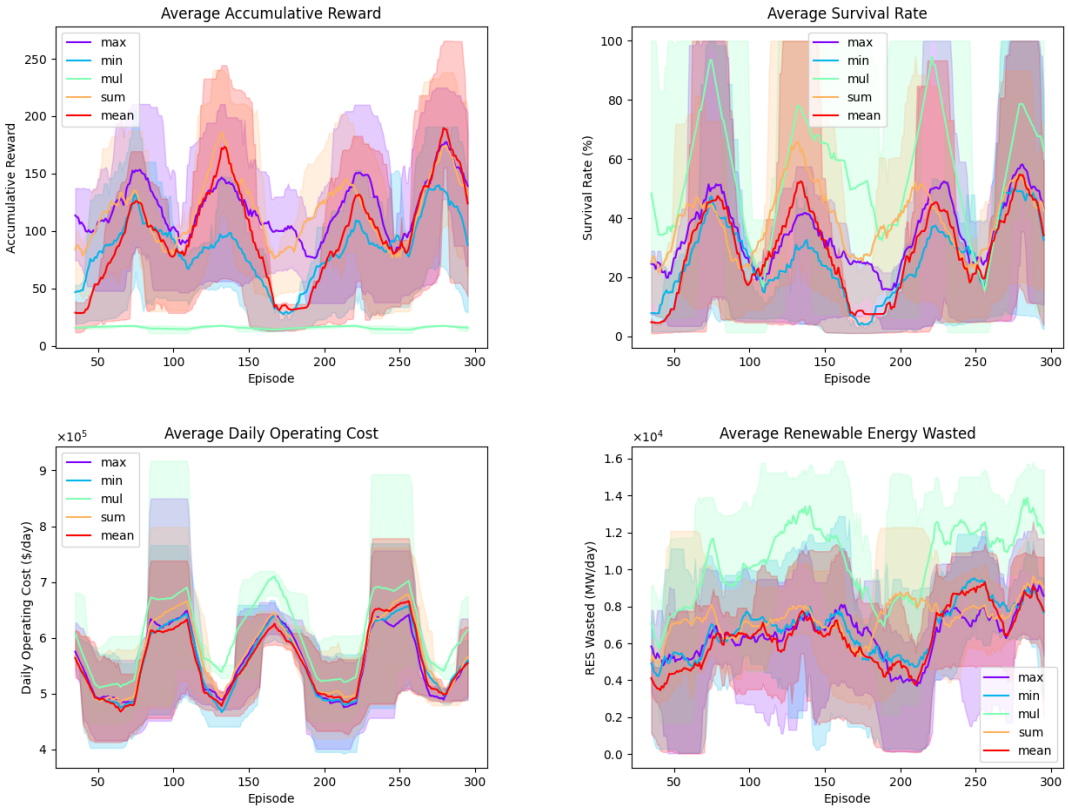

Figure 5.7: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during test. The multiplication function achieved the lowest average accumulative reward while also yielding the best survival rate, a difference justified by its considerably negative results in average cost and average abandoned RES.

Table 5.7: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during test.

aggr	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
max	119.16	33.62	560219.45	6564.49
sum	114.58	38.12	569952.37	7268.28
mean	92.90	27.34	557637.84	6268.06
min	80.89	24.59	559033.63	6687.94
mul	15.76	54.77	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 maximization 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, it was still chosen as the optimal aggregation scheme.

5.4.2 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 5.4.1, yielded the best results. The models had one to six GCN layers, and the test results are observable in figure 5.8.

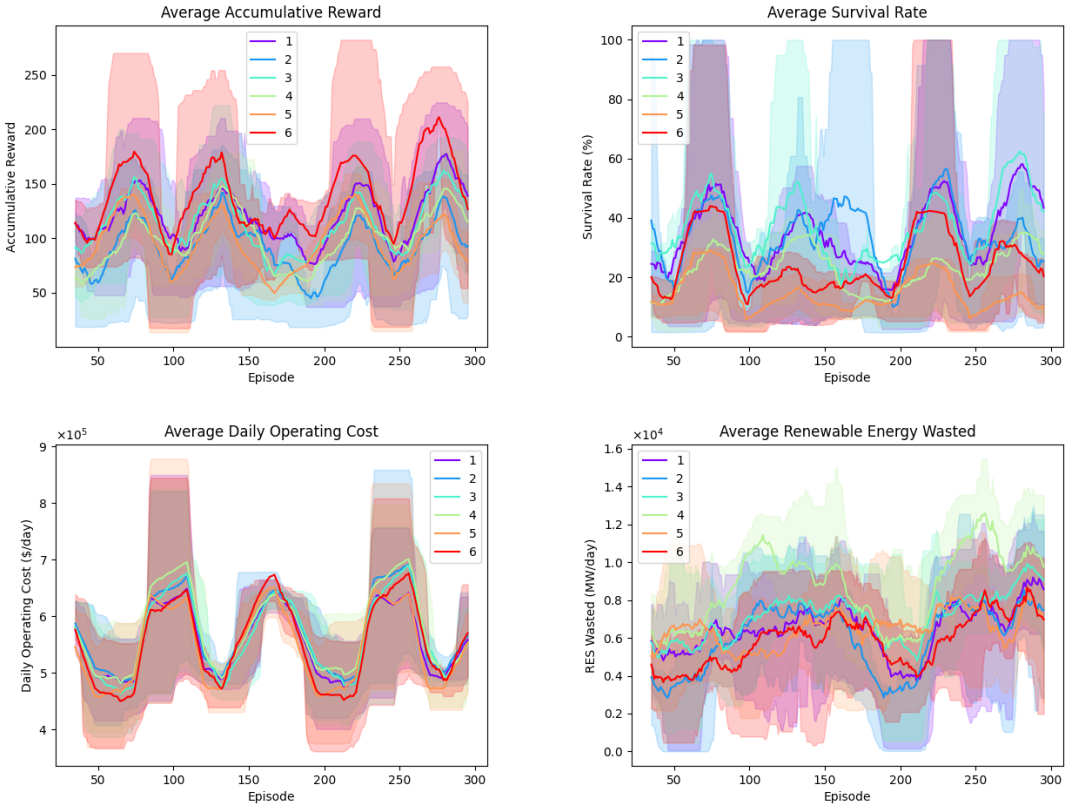

Figure 5.8: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 1, 2, 3, 4, 5, and 6 GCN layers during test. The six-layer GCN achieved the highest performance in average accumulative reward and the best balance concerning the other metrics.

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, as observed in table 5.8. This value represents a 13.92 % improvement regarding the second-best model in this metric, which was the one-layer network.

Table 5.8: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 1, 2, 3, 4, 5, and 6 GCN layers during test.

num_layers	Metrics			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
6	135.75	24.16	560147.77	5862.49
1	119.16	33.62	560219.45	6564.49
3	110.73	36.52	567510.58	7116.24
4	98.66	22.40	577464.90	8641.05
5	92.97	14.22	550205.21	6397.41
2	91.58	34.46	575093.64	5989.95

Generally, the calibration process has failed to reach ideal levels of survivability so far. Once again, the top-performing model in this metric did not correspond to the best average accumulative reward and yielded 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. In this case, the low reward is tied with the second-worst amount of abandoned RES energy.

5.4.3 Output Features

Concerning output features, values that were smaller, similar, and greater than the number of input features were considered. Four models were compared with 3, 6, 18, and 36 output features and the optimal parameters as the other arguments. These considerations aimed to cover both cases where the GNN summarizes the graph features, outputting fewer node attributes, and situations where it extends on them by adding more features to the output.

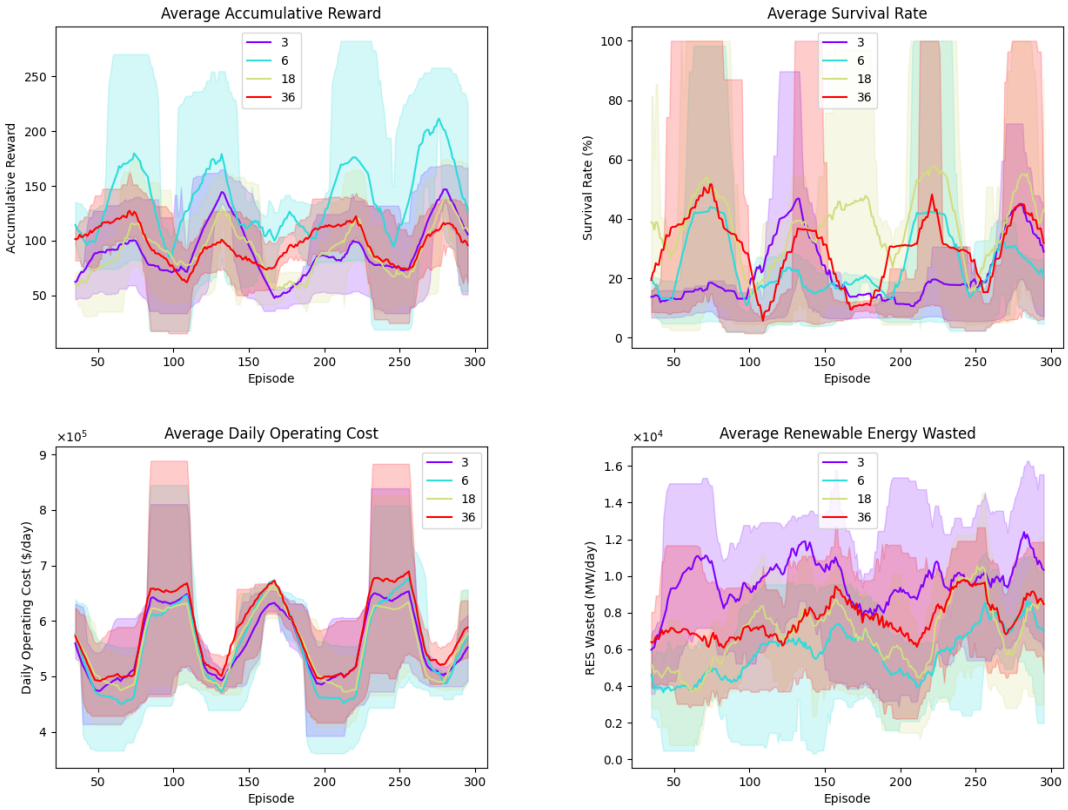

Figure 5.9: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 3, 6, 18, and 36 GCN output features during test. The model with 6 output features outperformed the others in all metrics except for survival rate.

The main test results, depicted in figure 5.9, outlined an optimal performance when using 6 output features, corresponding to the same number of input features. This model was able to outperform all of the others in all metrics with the exception of survival rate, as observable in table 5.9. The best model surpassed the second-best by over 41.52 % in average accumulative reward, while the third-placed model with 18 output features yielded the best survival rate of 38.58 %.

Table 5.9: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 3, 6, 18, and 36 GCN output features during training.

out_channels	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
6	135.75	24.16	560147.77	5862.49
36	95.92	28.35	582984.75	7575.34
18	88.47	38.58	562226.73	6874.71
3	87.68	21.25	563134.93	9497.97

In general, although only one model was trained with fewer features than the input, the others

were able to reach superior levels of performance. In this context, we concluded that the proposed GRL approach favoured a similar or greater amount of output features, with the best results being achieved with the same number of input attributes.

5.4.4 GCN Parameters

Some particular GNN parameters not covered in the last subsections were also deemed relevant to analyze. 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 normalization coefficients. The other parameter analyzed was the `improved` flag, which defines whether the model computes $\hat{\mathbf{A}}$ 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.

Table 5.10: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with different configurations for the GCN `act_first` and **improved!** parameters during training.

act_first	improved	Metrics			
		Avg. CR	Avg. SR	Avg. DOC	Avg. REW
True	False	118.97	31.77	564201.23	7167.22
True	True	101.26	36.64	546592.78	6297.46
False	False	78.62	37.69	575798.57	7640.76
False	True	75.29	16.31	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 normalization 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.

5.4.5 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 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`. As seen in the previous subsection, the first 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 [7].

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: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for a random search of GAT `heads`, `act_first`, and `v2` parameters during test.

act_first	**heads**	**v2**	**Metrics**			
			Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	3	True	117.09	12.84	582144.59	7216.87
False	3	False	111.41	36.02	569354.23	7403.57
True	1	True	94.46	29.49	571177.15	6785.37
True	2	True	92.50	32.26	562035.66	7349.15
False	3	True	91.18	9.80	552279.16	8988.24
True	2	False	88.77	30.33	565936.09	7250.76
True	3	False	80.03	28.76	556860.38	6723.43
True	1	False	76.94	25.91	574402.26	8366.63
False	1	False	76.06	23.62	549382.92	7971.11
False	1	True	59.17	13.57	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.

Regarding the second version of the GAT architecture, the results show that four of the top five models in cumulative reward 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 in the following comparison and scalability experiments.

5.5 36-Bus Scenario Analysis

This section performs a final performance analysis on the GCN-SAC and GAT-SAC algorithms, comparing them with the baseline SAC algorithm. The intermediate experiments allowed a deeper understanding of the relevant parameters of the proposed models, which was used to determine the optimal combination of parameters for both.

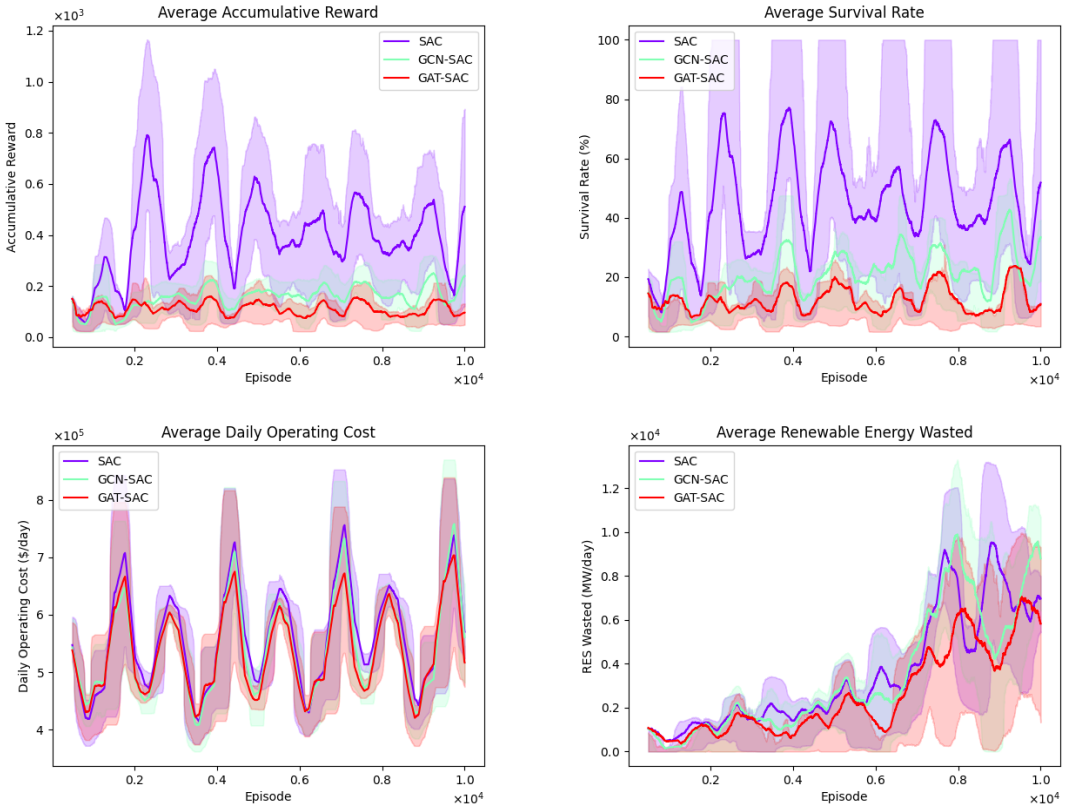

Figure 5.10: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 36-bus environment. The SAC model outperformed the other models, reaching a higher convergence and yielding a better learning efficiency.

The overall results indicate a significant superiority of the SAC algorithm over the proposed implementations. Figures 5.10 and 5.11 portray the performance of the models during training and test, respectively, highlighting this dominance. This model was able to outperform the proposed GCN-SAC algorithm by over 122.16 %, as observed in table 5.12, which depicts the test results. These results are only exacerbated because this model didn't have a focused calibration process as the proposed implementations. However, a trend observed particularly in this model and in the others is the high variability and instability of results, with standard deviations almost always surpassing 50 % of the average values for all metrics except for daily operating cost.

Figure 5.11: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 36-bus environment. The SAC model was superior in all metrics except for daily operating cost, surpassing the proposed implementations by a large margin and achieving significantly higher levels of average accumulative reward and survival rate. All models, especially the best, showed a great level of variance and instability in results.

In the case of the proposed GRL techniques, the calibration process failed to improve on performance stability. As observed in the graph in figure 5.10, this instability is also present during training and shows that the models fail to generalize the learned information to specific new scenarios. The baseline SAC algorithm saw the highest variability of accumulative reward during both training and test phases, which is expected given the absence of a specialized tuning process for this algorithm. We believe the sample efficiency and inherent stability of SAC gave it additional leverage in this optimization problem.

Table 5.12: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 36-bus environment.

Model	Metrics					
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	342.84	55.47	601896.45	5443.65	0.0029	47.76
GCN-SAC	153.77	31.16	579978.88	7356.86	0.0097	31.11
GAT-SAC	105.75	11.50	564739.25	7070.78	0.0071	12.52

Considering the average step processing time of the model, the SAC algorithm is also remarkably superior to the proposed models, with a processing time of 29 milliseconds in this scenario, against the average times of 97 and 71 milliseconds of GCN-SAC and GAT-SAC, respectively. The GRL techniques were only able to surpass the SAC model in training time, but even in this case, the results are not positive since this was mostly due to this algorithm surviving on average a significantly higher amount of steps than the other implementations. This caused the model to train for a larger amount of steps than the GRL techniques, which also demonstrates the importance of this metric for learning efficiency. By surviving longer into each episode, the SAC algorithm was able to face more diverse situations and leverage its features into finding more relevant and generalizable patterns in them.

5.6 118-Bus Scenario Analysis

To ascertain the scalability of the several implementations is finally analyzed by comparing their performance on the 118-bus scenario. The insights taken from relevant literature, beyond suggesting a promising performance of GRL, showed the advantages of these techniques in larger and more complex scenarios. Although the first prospect was not verified in the last subsection, we still hoped that the feature extraction capabilities of the GRL implementations were exacerbated by larger and more complex scenarios.

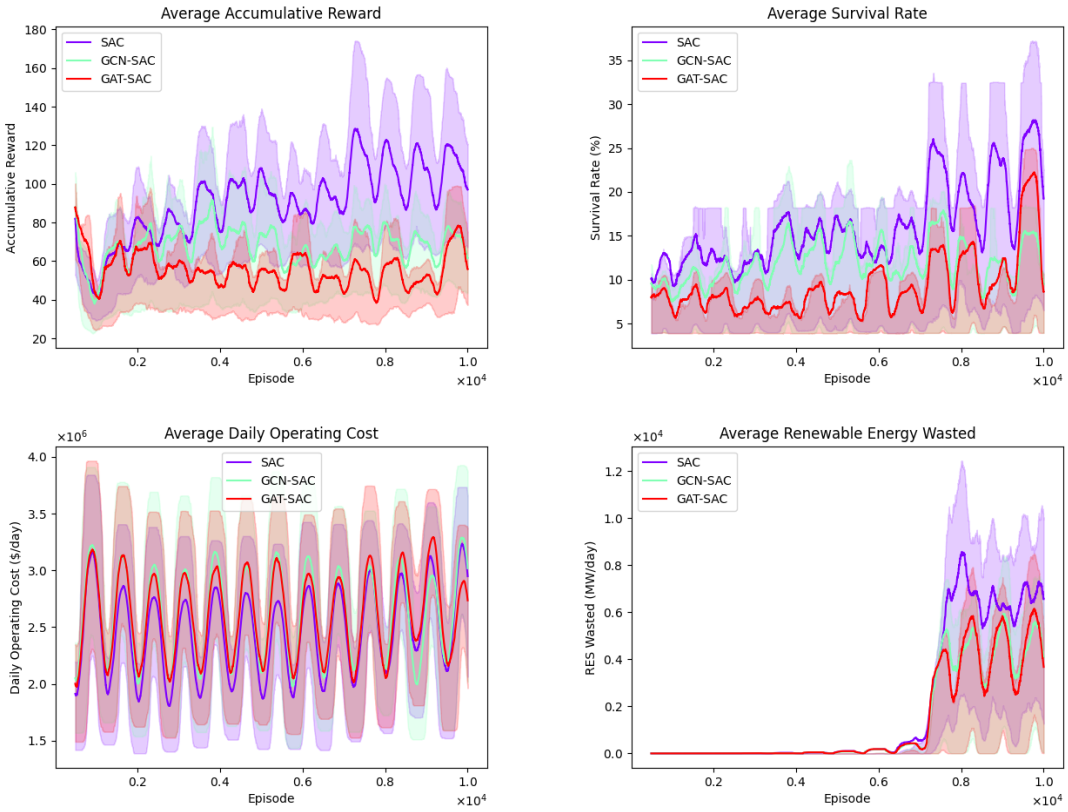

Figure 5.12: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 118-bus environment. The SAC algorithm showed the best training performance overall in all metrics.

Once more, the proposed implementations did not exceed the SAC algorithm in overall performance, as demonstrated by table 5.13. This model obtained the best average reward per episode and the best convergence, observable in figure 5.12 but at the expense of a slightly shorter survivability than the second-placed GCN-SAC. In this context, the proposed implementations' performance was a lot closer to the baseline, with the GCN-SAC model even showing the best average survival rate in the test phase, as shown by the graph in figure 5.13. However, the results of all models in this metric were generally bad, and this larger scenario aggravated this trend, which is reflected in the best survival rate of only 19.34 %. The GAT-SAC model was best regarding average renewable energy wasted and average daily operating cost. Nevertheless, its low average cumulative reward is mostly explained by its low survival rate.

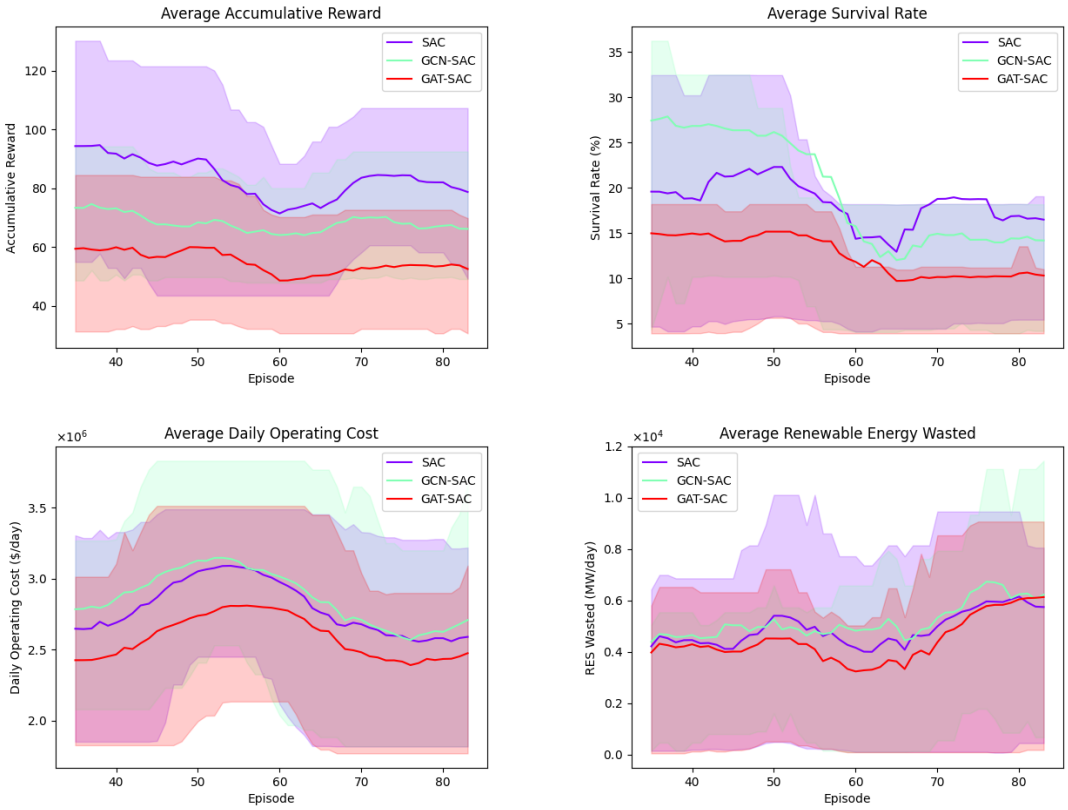

Figure 5.13: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 118-bus environment. The SAC model was again superior in performance by an inferior margin. The GCN-SAC model achieved higher but less consistent results in survival rate.

The connection between bad survivability and higher times is also verified in this case, although it is not as apparent as in the last scenario. The GCN-SAC model had the highest average survival rate during the test stage and also achieved a better training time than the SAC algorithm. However, the former model did surpass the GRL implementation in average survival rate during training, which explains the higher training time of the SAC algorithm.

Table 5.13: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 118-bus environment.

Model	**Metrics**					
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	81.98	17.95	2788763.38	4740.03	0.0067	14.82
GCN-SAC	67.85	19.34	2901439.28	5323.58	0.0096	13.67
GAT-SAC	54.31	11.92	2591733.60	4709.37	0.0122	11.39

Lastly, our proposed GCN-SAC was able to maintain a similar step processing time in both scenarios, which shows great scalability of this implementation. The SAC Model was still superior again, but its step time more than doubled compared to the results observed in the 36-bus scenario. GAT-SAC wasn't as consistent as GCN-SAC in this metric, with its processing time considerably increasing.

Chapter 6

Conclusions

In this dissertation, we document the study performed around the improvement of GRL techniques to solve the DED problem. After a thorough study of the relevant literature and a complete overview of relevant knowledge, the results showed that GRL approaches demonstrated confident results facing the modern pure-DRL algorithms. Beyond that, the literature was particularly optimistic about the scalability and adaptability of these techniques in larger grids with real-time topological changes.

It's crucial to highlight the complexity of the DED problem, which was not fully explored in the context of this work. The literature showed a diverse set of constraints, datasets, action, and observation spaces tied to this problem. However, the conducted review failed to find relevant works with appropriately and openly distributed sources, which severely complicates the replication of results.

The initial experiments and the calibration process unveiled several concerns, mainly related to the action space complexity and the GNN tuning. Despite the thorough tuning process, the proposed GCN-SAC and GAT-SAC implementations failed to surpass the sample efficiency and stability of the state-of-the-art SAC algorithm, with the latter showing superior performance in training convergence and overall results. Nonetheless, our results demonstrate a slight advantage of the GCN-SAC model on scalability, which showed consistent results on processing time in a larger scenario and performance closer to the state-of-the-art model.

Beyond that, a test framework resulted from this work, enabling anyone to replicate the performance of the models implemented. The framework allows any combination of DRL and GNN algorithm (implemented by *stable-baselines3* or *torch_geometric*) on several available power grid environments, as well as performing training, test and visual analysis on the models. This enables similar studies on other algorithms and more specialized tuning processes, further advancing research and knowledge around the field of GRL. Finally, in a last reflection on the results achieved, the author still believes in the promising results and solid foundation established in the literature review for the novel GRL techniques. Although this was not verified in this study, there are still a lot of practical insights from model implementation that can be gathered in order to improve the model's computational performance and learning efficiency. Additionally, the proposed

implementations could benefit from a more extensive tuning process and subsequent analysis to substantiate the potential superiority of these models.

6.1 Main Contributions

In this section, we highlight the key contributions of this work at the scientific, technological, and application levels:

Scientific This dissertation offers a comprehensive and systematic comparative analysis of the two most prominent Graph Reinforcement Learning (GRL) approaches, addressing a notable gap in the research concerning the comparison of different proposed techniques. Additionally, it introduces a method that enables the experimentation with diverse DRL and GNNs models. Through this analysis, clearer insights into the best practices for implementing GRL models are achieved. Moreover, the empirical evaluation conducted on two proposed GRL approaches provides valuable information regarding their performance in different contexts.

Technological From a technological perspective, this study results in the development of an enhanced GRL model that builds upon and improves the existing techniques, addressing their key limitations. In addition, a flexible framework is introduced, allowing experimentation with various power system challenges. This framework is adaptable to different problems and provides a robust environment for testing and refining GRL models.

Application On the application level, this work presents an efficient and innovative GRL-based solution for the Dynamic Economic Dispatch (DED) problem, specifically designed to handle complex scenarios within smart grid systems. This solution contributes significantly to the advancement of research in DED systems, offering practical improvements in optimizing power distribution while managing dynamic demands.

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.

References

[1] Feedforward Neural Network. https://orgs.mines.edu/daa/wp-content/uploads/sites/38/2019/08/1_Gh5PS4R_A5drl5ebd_gNrg@2x.jpg.

[2] Reproducibility — PyTorch 2.4 documentation. https://pytorch.org/docs/stable/notes/randomness.html.

[3] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Computer Communications*, 196:184–194, December 2022.

[4] B. Donnot. Grid2Op - A testbed platform to model sequential decision making in power systems. https://grid2op.readthedocs.io/en/latest/index.html.

[5] Yuyang Bai, Siyuan Chen, Jun Zhang, Jian Xu, Tianlu Gao, Xiaohui Wang, and David Wenzhong Gao. An adaptive active power rolling dispatch strategy for high proportion of renewable energy based on distributed deep reinforcement learning. *Applied Energy*, 330:120294, January 2023.

[6] R. Bayindir, I. Colak, G. Fulli, and K. Demirtas. Smart grid technologies and applications. *Renewable and Sustainable Energy Reviews*, 66:499–516, December 2016.

[7] Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks? In *ICLR 2022 - 10th International Conference on Learning Representations*. arXiv, January 2022.

[8] Eugene Charniak. *Introduction to Deep Learning*. The MIT Press, Cambridge, Massachusetts London, England, 2018.

[9] Fanfei Chen, John D. Martin, Yewei Huang, Jinkun Wang, and Brendan Englot. Autonomous Exploration Under Uncertainty via Deep Reinforcement Learning on Graphs. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6140–6147, October 2020.

[10] Junbin Chen, Tao Yu, Zhenning Pan, Mengyue Zhang, and Bairong Deng. A scalable graph reinforcement learning algorithm based stochastic dynamic dispatch of power system under high penetration of renewable energy. *International Journal of Electrical Power & Energy Systems*, 152:109212, October 2023.

[11] Xinpei Chen, Tao Yu, Zhenning Pan, Zihao Wang, and Shengchun Yang. Graph representation learning-based residential electricity behavior identification and energy management. *Protection and Control of Modern Power Systems*, 8(1):1–13, December 2023.

[12] Yongdong Chen, Youbo Liu, Junbo Zhao, Gao Qiu, Hang Yin, and Zhengbo Li. Physical-assisted multi-agent graph reinforcement learning enabled fast voltage regulation for PV-rich active distribution network. *Applied Energy*, 351:121743, December 2023.

[13] Rich Christie. Power Systems Test Case Archive. https://labs.ece.uw.edu/pstca/pf118/pg_tca118bus.htm.

[14] Peng Cui, Lingfei Wu, Jian Pei, Liang Zhao, and Xiao Wang. Graph Representation Learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 17–26. Springer Nature, Singapore, 2022.

[15] Emma Brunskill. CS234: Reinforcement Learning Winter 2023. https://web.stanford.edu/class/cs234/.

[16] Bangji Fan, Xinghua Liu, Gaoxi Xiao, Yu Kang, Dianhui Wang, and Peng Wang. Attention-Based Multi-Agent Graph Reinforcement Learning for Service Restoration. *IEEE Transactions on Artificial Intelligence*, pages 1–15, 2023.

[17] Farama Foundation. Gymnasium Documentation. https://gymnasium.farama.org/index.html.

[18] Hassan Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, January 2010.

[19] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, April 2019.

[20] Aleksandr I. Galuškin. *Neural Networks Theory*. Springer, Berlin Heidelberg, 2007.

[21] Hongyang Gao and Shuiwang Ji. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2083–2092. PMLR, May 2019.

[22] Zhen Gao, Lei Yang, and Yu Dai. Fast Adaptive Task Offloading and Resource Allocation in Large-Scale MEC Systems via Multi-Agent Graph Reinforcement Learning. *IEEE Internet of Things Journal*, pages 1–1, 2023.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, 2016.

[24] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870. PMLR, July 2018.

[25] William L Hamilton. *Graph Representation Learning*. Morgan & Claypool Publishers, September 2020.

[26] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs, September 2018.

[27] Xiaoyun Han, Chaoxu Mu, Jun Yan, and Zeyuan Niu. An autonomous control technology based on deep reinforcement learning for optimal active power dispatch. *International Journal of Electrical Power & Energy Systems*, 145:108686, February 2023.

[28] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.

[29] Daner Hu, Zichen Li, Zhenhui Ye, Yonggang Peng, Wei Xi, and Tiantian Cai. Multi-agent graph reinforcement learning for decentralized Volt-VAR control in power distribution systems. *International Journal of Electrical Power & Energy Systems*, 155:109531, January 2024.

[30] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017.

[31] Lei Lei, Yue Tan, Glenn Dahlenburg, Wei Xiang, and Kan Zheng. Dynamic Energy Dispatch Based on Deep Reinforcement Learning in IoT-Driven Smart Isolated Microgrids. *IEEE Internet of Things Journal*, 8(10):7938–7953, May 2021.

[32] Lixiong Leng, Jingchen Li, Haobin Shi, and Yi'an Zhu. Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing. *Multimedia Tools and Applications*, 80(19):29163–29175, August 2021.

[33] Nan Li, Alexandros Iosifidis, and Qi Zhang. Graph Reinforcement Learning-based CNN Inference Offloading in Dynamic Edge Computing. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 982–987, December 2022.

[34] Peng Li, Wenqi Huang, Zhen Dai, Jiaxuan Hou, Shang Cao, Jiayu Zhang, and Junbin Chen. A Novel Graph Reinforcement Learning Approach for Stochastic Dynamic Economic Dispatch under High Penetration of Renewable Energy. In *2022 4th Asia Energy and Electrical Engineering Symposium (AEEES)*, pages 498–503, March 2022.

[35] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[36] Paul Lienert. GM takes on Tesla in home and commercial energy storage, management. https://www.reuters.com/business/autos-transportation/gm-takes-tesla-home-commercial-energy-storage-management-2022-10-11/, October 2022.

[37] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

[38] Weirong Liu, Peng Zhuang, Hao Liang, Jun Peng, and Zhiwu Huang. Distributed Economic Dispatch in Microgrids Based on Cooperative Reinforcement Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2192–2203, June 2018.

[39] Zhiyuan Liu and Jie Zhou. *Introduction to Graph Neural Networks*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer International Publishing, Cham, 2020.

[40] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937. PMLR, June 2016.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[42] Miguel Morales. *Grokking Deep Reinforcement Learning*. Manning, Shelter Island [New York], 2020.

[43] Mingshuo Nie, Dongming Chen, and Dongqi Wang. Reinforcement Learning on Graphs: A Survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):1065–1082, August 2023.

[44] OpenAI. Spinning Up Documentation. https://spinningup.openai.com/en/latest/index.html.

[45] Yangzhou Pei, Jun Yang, Jundong Wang, Peidong Xu, Ting Zhou, and Fuzhang Wu. An emergency control strategy for undervoltage load shedding of power system: A graph deep reinforcement learning method. *IET Generation, Transmission & Distribution*, 17(9):2130–2141, May 2023.

[46] A. T. D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, March 2021.

[47] PyTorch Foundation. PyTorch 2.3 Documentation. https://pytorch.org/docs/2.3/.

[48] Tao Qian, Chengcheng Shao, Xiuli Wang, and Mohammad Shahidehpour. Deep Reinforcement Learning for EV Charging Navigation by Coordinating Smart Grid and Intelligent Transportation System. *IEEE Transactions on Smart Grid*, 11(2):1714–1723, March 2020.

[49] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[50] Reuters. Renewables supplied 88% of Portugal's electricity consumption in January. https://www.reuters.com/world/americas/renewables-supplied-88-portugals-electricity-consumption-january-2023-02-01/, February 2023.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

[52] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017.

[53] Liam Stoker. Energy storage outranks solar in company investment plans. https://www.reuters.com/business/energy/energy-storage-outranks-solar-company-investment-plans-2023-12-11/, December 2023.

[54] sunfanyunn. Issue #92 · pyg-team/pytorch_geometric. https://github.com/pyg-team/pytorch_geometric/issues/92.

[55] Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 2014.

[56] Jian Tang and Renjie Liao. Graph Neural Networks for Node Classification. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 41–61. Springer Nature, Singapore, 2022.

[57] Zhiqing Tang, Jiong Lou, Fuming Zhang, and Weijia Jia. Dependent Task Offloading for Multiple Jobs in Edge Computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, August 2020.

[58] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, February 2018.

[59] N. Vesselinova, R. Steinert, D.F. Perez-Ramirez, and M. Boman. Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking. *IEEE access : practical innovations, open solutions*, 8:120388–120416, 2020.

[60] Tamilmaran Vijayapriya and Dwarkadas Pralhadas Kothari. Smart Grid: An Overview. *Smart Grid and Renewable Energy*, 02(04):305–311, 2011.

[61] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, July 2020.

[62] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Nature Singapore, Singapore, 2022.

[63] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph Neural Networks. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 27–37. Springer Nature, Singapore, 2022.

[64] X. Xia and A. M. Elaiw. Optimal dynamic economic dispatch of generation: A review. *Electric Power Systems Research*, 80(8):975–986, August 2010.

[65] Qiang Xing, Zhong Chen, Tian Zhang, Xu Li, and KeHui Sun. Real-time optimal scheduling for active distribution networks: A graph reinforcement learning method. *International Journal of Electrical Power & Energy Systems*, 145:108637, February 2023.

[66] Qiang Xing, Zhong Chen, Ziqi Zhang, Ruisheng Wang, and Tian Zhang. Modelling driving and charging behaviours of electric vehicles using a data-driven approach combined with behavioural economics theory. *Journal of Cleaner Production*, 324:129243, November 2021.

[67] Qiang Xing, Yan Xu, and Zhong Chen. A Bilevel Graph Reinforcement Learning Method for Electric Vehicle Fleet Charging Guidance. *IEEE Transactions on Smart Grid*, 14(4):3309–3312, July 2023.

[68] Qiang Xing, Yan Xu, Zhong Chen, Ziqi Zhang, and Zhao Shi. A Graph Reinforcement Learning-Based Decision-Making Platform for Real-Time Charging Navigation of Urban Electric Vehicles. *IEEE Transactions on Industrial Informatics*, 19(3):3284–3295, March 2023.

[69] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks?, February 2019.

[70] PeiDong Xu, YangZhou Pei, Xinhu Zheng, and Jun Zhang. A Simulation-Constraint Graph Reinforcement Learning Method for Line Flow Control. In *2020 IEEE 4th Conference on Energy Internet and Energy System Integration (EI2)*, pages 319–324, October 2020.

[71] Peidong Xu, Jun Zhang, Tianlu Gao, Siyuan Chen, Xiaohui Wang, Huaiguang Jiang, and Wenzhong Gao. Real-time fast charging station recommendation for electric vehicles in coupled power-transportation networks: A graph reinforcement learning method. *International Journal of Electrical Power & Energy Systems*, 141:108030, October 2022.

[72] Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li. Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1040–1057, June 2020.

[73] Ting Yang, Liyuan Zhao, Wei Li, and Albert Y. Zomaya. Dynamic energy dispatch strategy for integrated energy system based on improved deep reinforcement learning. *Energy*, 235:121377, November 2021.

[74] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 430–438, New York, NY, USA, August 2020. Association for Computing Machinery.

[75] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 1621–1632. Curran Associates, Inc., 2020.

[76] Liang Zhao, Lingfei Wu, Peng Cui, and Jian Pei. Representation Learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 3–15. Springer Nature, Singapore, 2022.

[77] Tianqiao Zhao and Jianhui Wang. Learning Sequential Distribution System Restoration via Graph-Reinforcement Learning. *IEEE Transactions on Power Systems*, 37(2):1601–1611, March 2022.

[78] Yu Zhao, Jun Liu, Xiaoming Liu, Keyu Yuan, Kezheng Ren, and Mengqin Yang. A Graph-based Deep Reinforcement Learning Framework for Autonomous Power Dispatch on Power Systems with Changing Topologies. In *2022 IEEE Sustainable Power and Energy Conference (iSPEC)*, pages 1–5, December 2022.

Appendix A

Code Availability

The source code developed for this dissertation is available in a public GitHub repository to ensure transparency, reproducibility, and further research. The repository contains all the necessary scripts, data processing tools, and implementation details for the Graph Reinforcement Learning (GRL) models and algorithms used throughout this work.

The repository can be accessed at the following URL:

- https://github.com/antonoliv/grl-ded

Follow the setup instructions in the README file, which provides detailed steps for installing dependencies and running the experiments.

A.1 License and Reuse

The code is released under the MIT License, allowing reuse, modification, and distribution for both commercial and non-commercial purposes, as long as the original license and copyright notice are retained. For more details, refer to the LICENSE file in the repository.

Appendix B

Hyperparameters

B.1 Environment Parameters

Table B.1: Description of Environment Parameters.

Name	Description
`env_path`	Path to the grid2op dataset
`reward`	Reward function
`obs_scaled`	If `True`, observation keys P_i^{NRES}, P_i^{RES}, and $\overline{P_i^{\text{RES}}}$ will be scaled
`obs_step`	If `True`, observation will use the current step instead of minute, hour and day of the week
`act_no_curtail`	If `True`, action space will not include curtailment actions
`act_limit_infeasible`	If `True`, the parameter `LIMIT_INFEASIBLE_CURTAILMENT_STORAGE_ACTION` is set
`cbound_type`	Type of curtailment lower bound, can be `None`, `fixed`, `linear`, and `sqrt`
`cbound_low`	Curtailment lower bound
`decay_end`	End of lower bound decay, if type is `linear` or `sqrt`
`decay_factor`	For if `sqrt` type, controls aggressiveness of decay

B.2 SAC Parameters

Table B.2: Description of SAC Parameters.

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 the 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 B.3: Description of SAC Policy Parameters.

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 mini-batches
features_extractor_kwargs	Additional keyword arguments for feature extractor

B.3 GNN Parameters

Table B.4: Description of General GNN Parameters.

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
normalize	Add self-loops and compute symmetric normalization coefficients

Table B.5: Description of GCN-Specific Parameters.

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 B.6: Description of GAT-Specific Parameters.

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 attention 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

B.4 Optimal Parameters

Table B.7: Optimal Environment Parameters. For `decay_end`, a value of 4200 for the intermediate experiments, while in the final results for the 36-bus and 118-bus systems, a value of 7200 was used.

Name	Value
`env_path`	`'l2rpn_icaps_2021_large'`
`reward`	`RESPenaltyReward(0.4)`
`obs_scaled`	`False`
`obs_step`	`True`
`act_no_curtail`	`False`
`act_limit_infeasible`	`True`
`cbound_type`	`'sqrt'`
`cbound_low`	`0.4`
`decay_end`	`4200/7200`
`decay_factor`	`3`
`machine`	`LIACC Server`

Table B.8: Used SAC Hyperparameters.

Name	**Value**
policy	'MultiInputPolicy'
learning_rate	1e-4
buffer_size	1e6
learning_starts	1e3
batch_size	256
tau	0.001
gamma	0.85
train_freq	1
gradient_steps	1
action_noise	None
ent_coef	'auto'
target_update_interval	1
target_entropy	'auto'
use_sde	False
optimize_memory_usage	False
replay_buffer_class	DictReplayBuffer
seed	123433334
device	'cuda'
_init_setup_model	True
policy_kwargs['net_arch']	6 * [128]
policy_kwargs['optimizer_class']	Adam
policy_kwargs['activation_fn']	ReLU
policy_kwargs['n_critics']	2
policy_kwargs['features_extractor_class']	GraphExtractor
policy_kwargs['features_extractor_kwargs']	None

Table B.9: Optimal GCN Parameters.

Name	**Value**
in_channels	6
hidden_channels	18
num_layers	6
out_channels	6
dropout	0.1
act	'relu'
act_first	True
norm	None
jk	None
aggr	'max'
flow	'source_to_target'
node_dim	-2
decomposed_layers	1
normalize	True
improved	False
cached	False
add_self_loops	None
bias	True

Table B.10: Optimal GAT Parameters.

Name	Value
in_channels	6
hidden_channels	18
num_layers	6
out_channels	6
dropout	0.1
act	'relu'
act_first	True
norm	None
jk	None
aggr	'max'
flow	'source_to_target'
node_dim	-2
decomposed_layers	1
normalize	True
heads	3
v2	True
concat	True
negative_slope	0.2
add_self_loops	True
edge_dim	None
fill_value	'mean'
bias	True

Appendix C

Experiment Results

1

C.1 Reward Functions

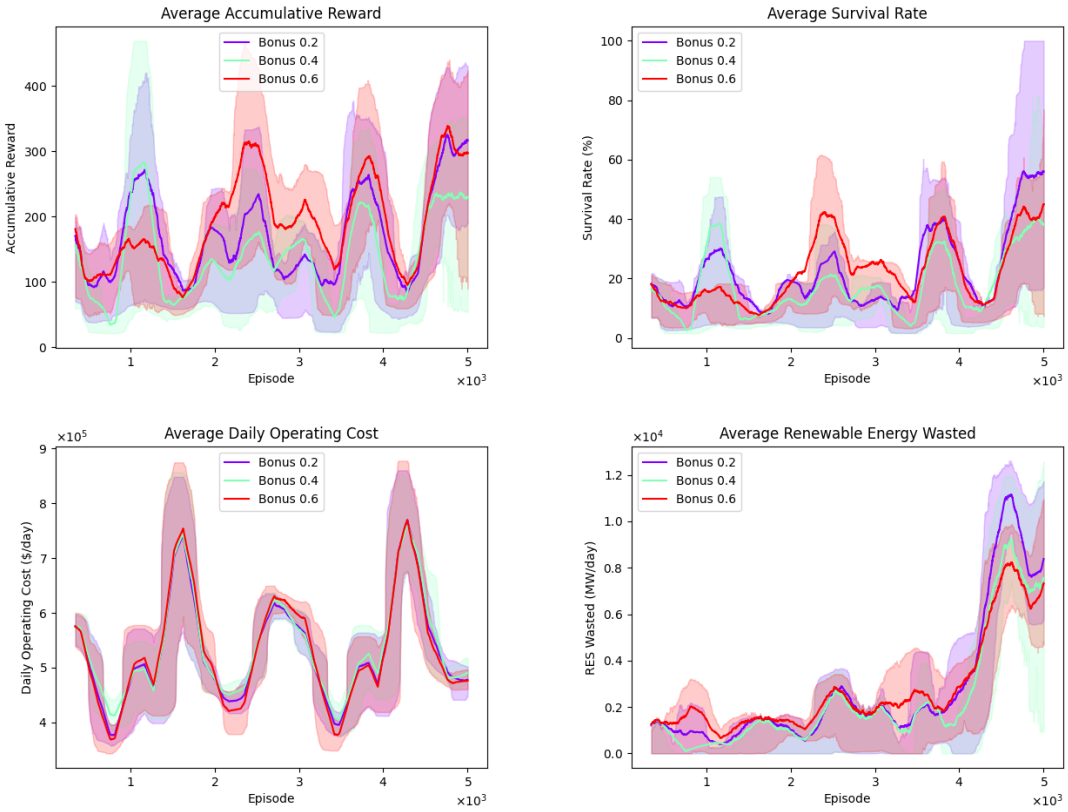

Figure C.1: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a bonus reward for used renewable energy during training with bonus factors of 0.2, 0.4 and 0.6.

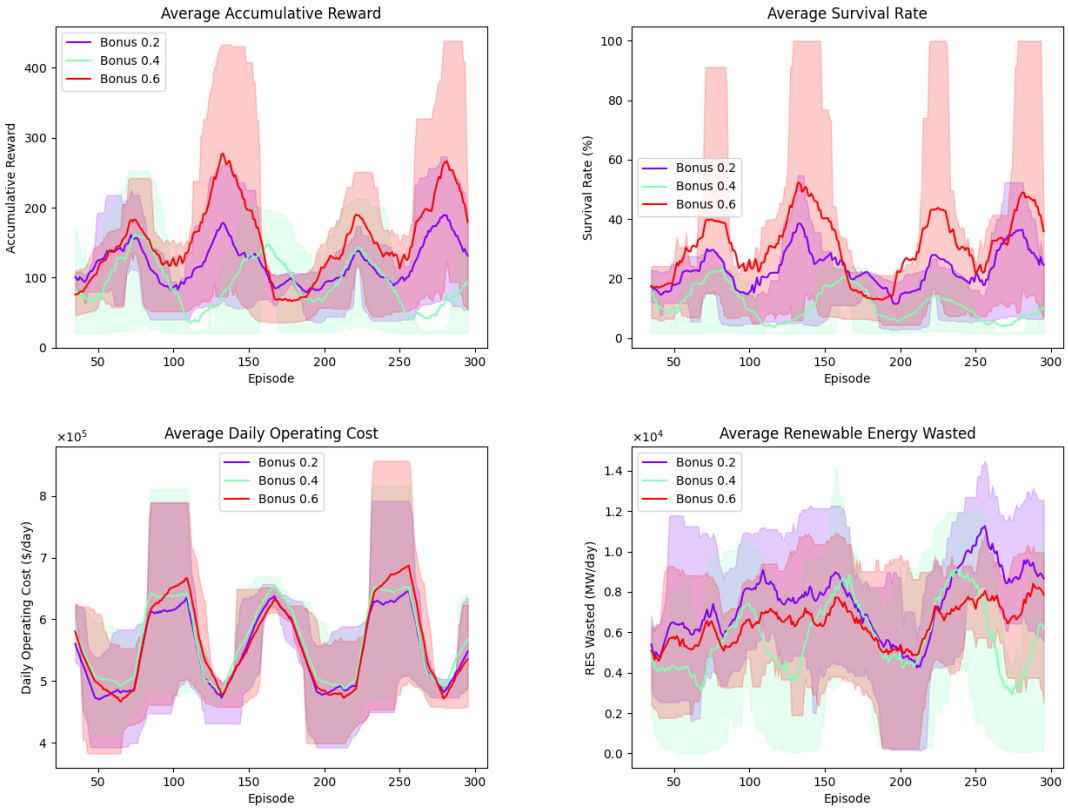

Figure C.2: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a bonus reward for used renewable energy during test with bonus factors of 0.2, 0.4 and 0.6.

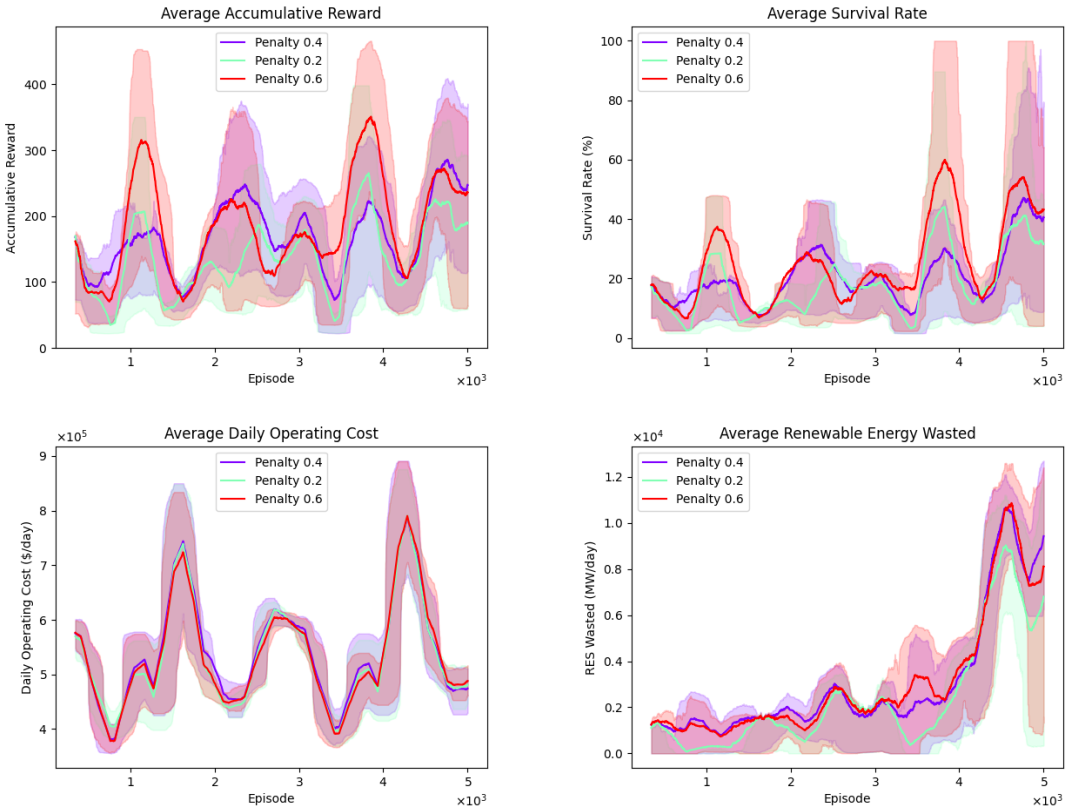

Figure C.3: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a reward penalty for used renewable energy during training with penalty factors of 0.2, 0.4 and 0.6.

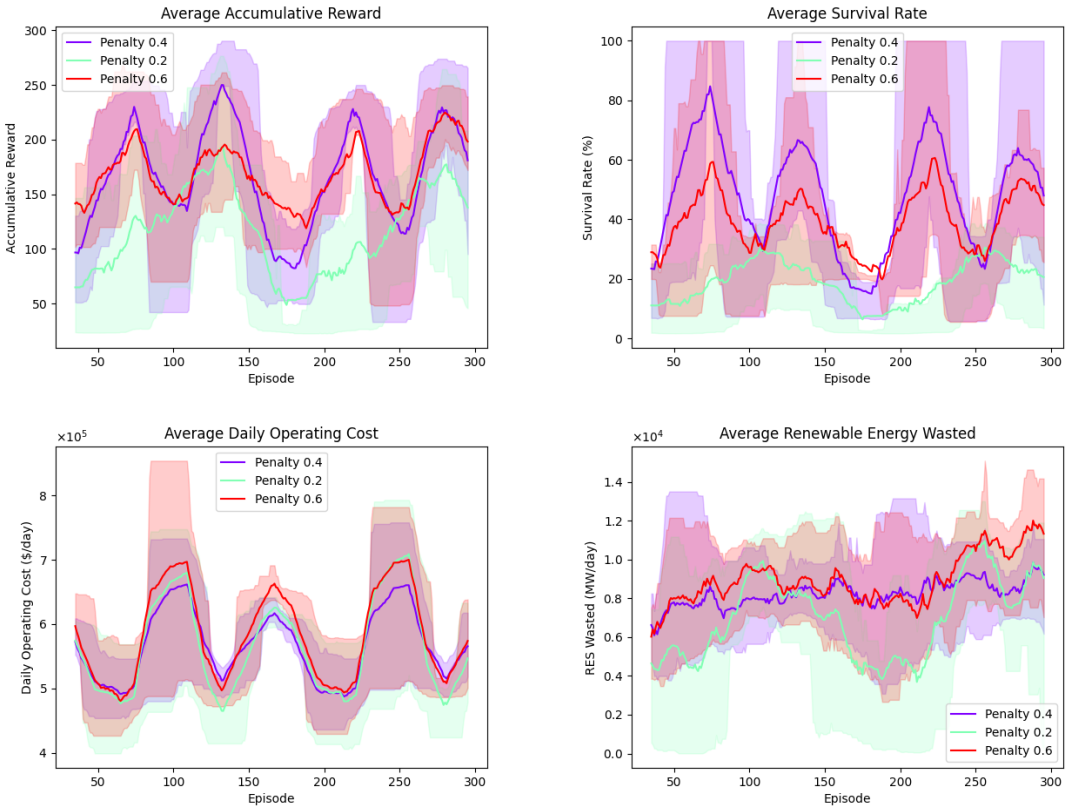

Figure C.4: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with a reward penalty for used renewable energy during test with penalty factors of 0.2, 0.4 and 0.6.

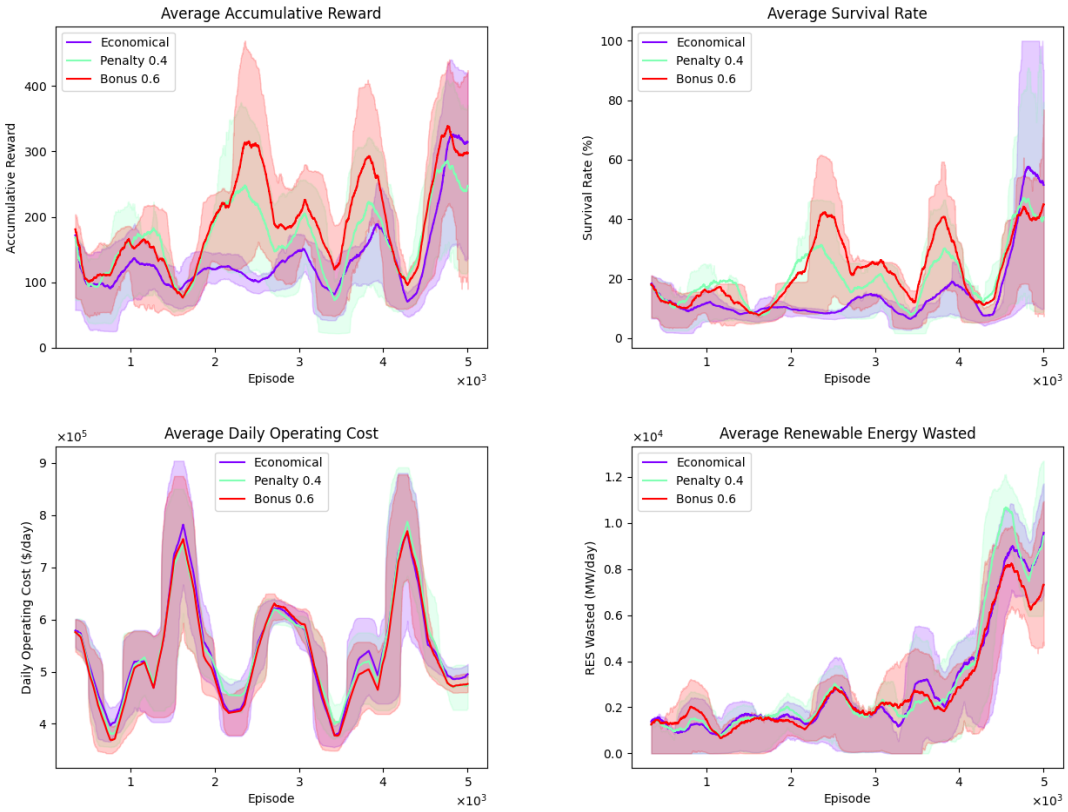

Figure C.5: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of the best models of each reward strategy against the performance of the default `economical` reward during training.

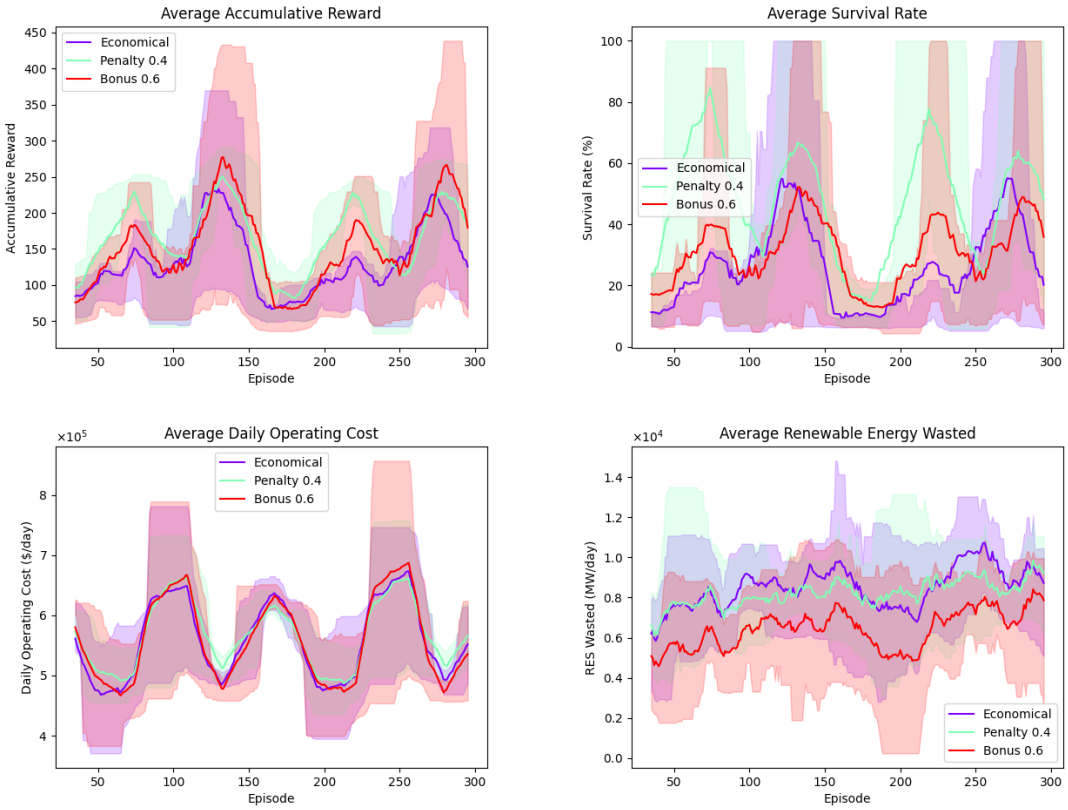

Figure C.6: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of the best models of each reward strategy against the performance of the default `economical` reward during test.

Table C.1: Average Accumulative Reward, Survival rate, Daily Operating Cost, and Wasted Renewable Energy of the different studied reward configurations during test.

reward	β	**Metrics**			
		Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
Penalty	0.6	166.67	38.50	586562.08	8926.40
Penalty	0.4	159.83	44.95	568533.92	8187.96
Bonus	0.6	148.70	30.16	562174.43	6484.24
Economical	-	127.20	24.71	562149.32	8353.61
Bonus	0.2	118.37	22.77	553932.52	7420.42
Penalty	0.2	113.29	18.49	566361.78	6954.43
Bonus	0.4	93.86	11.70	569404.33	5819.33

C.2 Limit Infeasible Curtail Actions

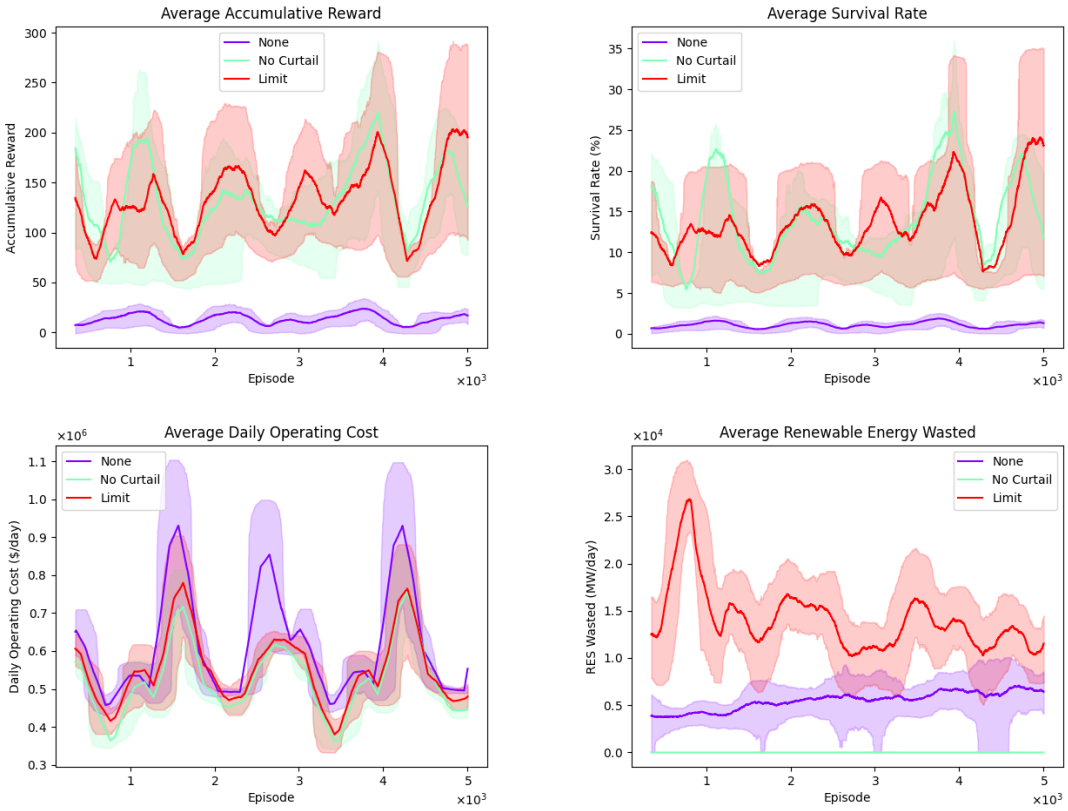

Figure C.7: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with and without the `limit_inf` parameter against a model without curtailment actions during training.

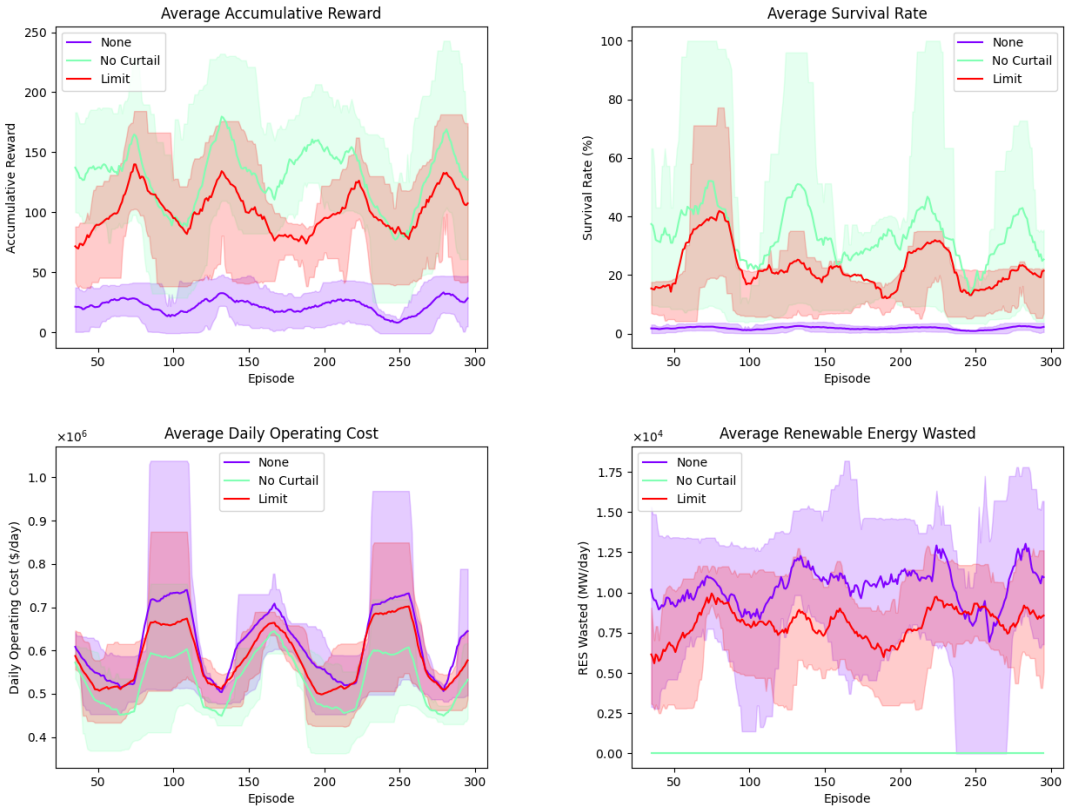

Figure C.8: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy of models with and without the `limit_inf` parameter against a model without curtailment actions during test.

Table C.2: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with and without the `limit_inf` parameter against a model without curtailment actions during test.

no_curtail	**limit_inf**	**Metrics**			
		Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	False	130.15	33.57	536423.98	0.00
False	True	98.84	22.11	588610.57	8039.41
False	False	22.13	1.84	619721.30	10315.47

C.3 Curtailment Lower Limit

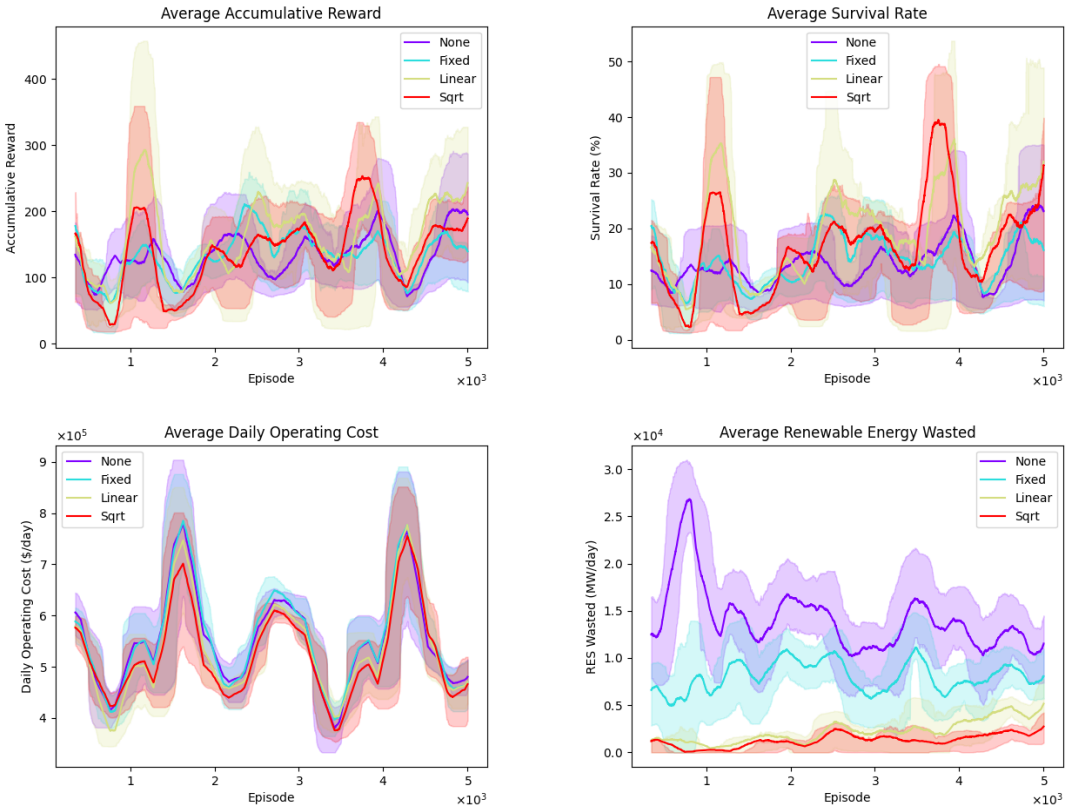

Figure C.9: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models without a curtailment lower bound, with a fixed limit, and with linear and square root limit decay during training.

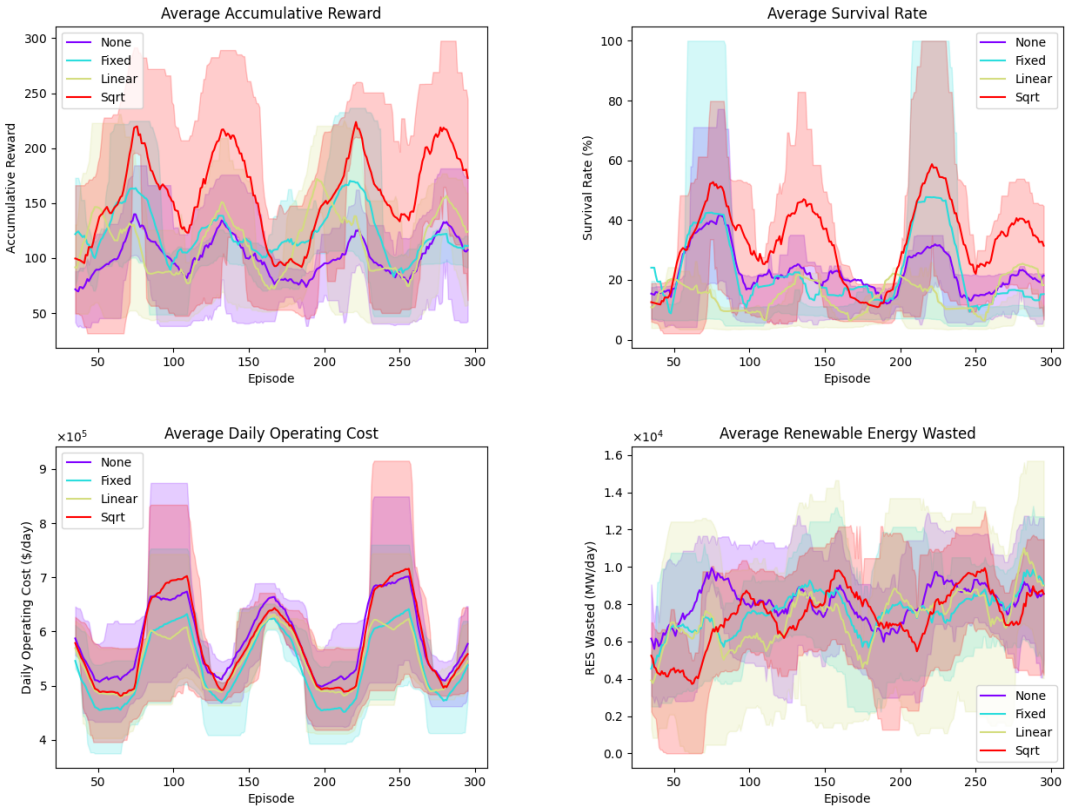

Figure C.10: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models without a curtailment lower bound, with a fixed limit, and with linear and square root limit decay during test.

Table C.3: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models without a curtailment lower bound, with a fixed limit, and with linear and square root limit decay during test.

decay	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
sqrt	153.77	31.16	579978.88	7356.86
Fixed	122.58	22.88	541391.22	7480.09
Linear	111.52	14.35	550832.62	7092.56
None	98.84	22.11	588610.57	8039.41

C.4 Lower Bound vs. Limit Infeasible Actions

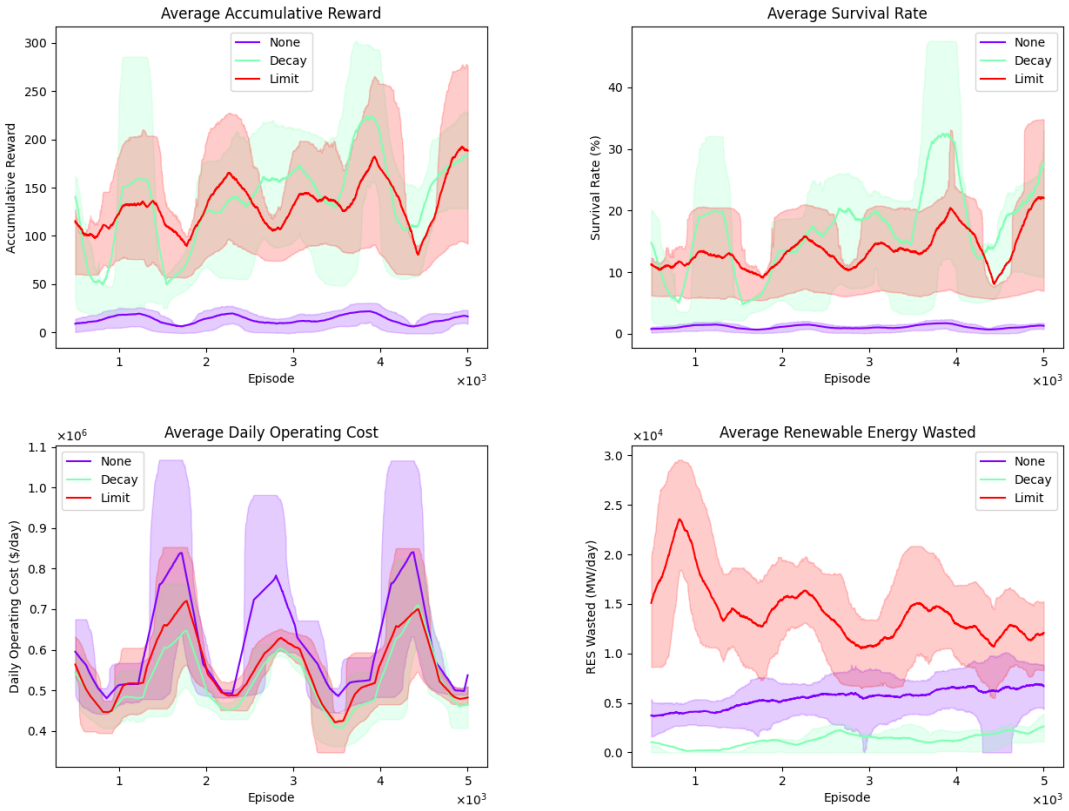

Figure C.11: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with square root curtailment limit decay, the `limit_inf` parameter and without curtailment actions during training.

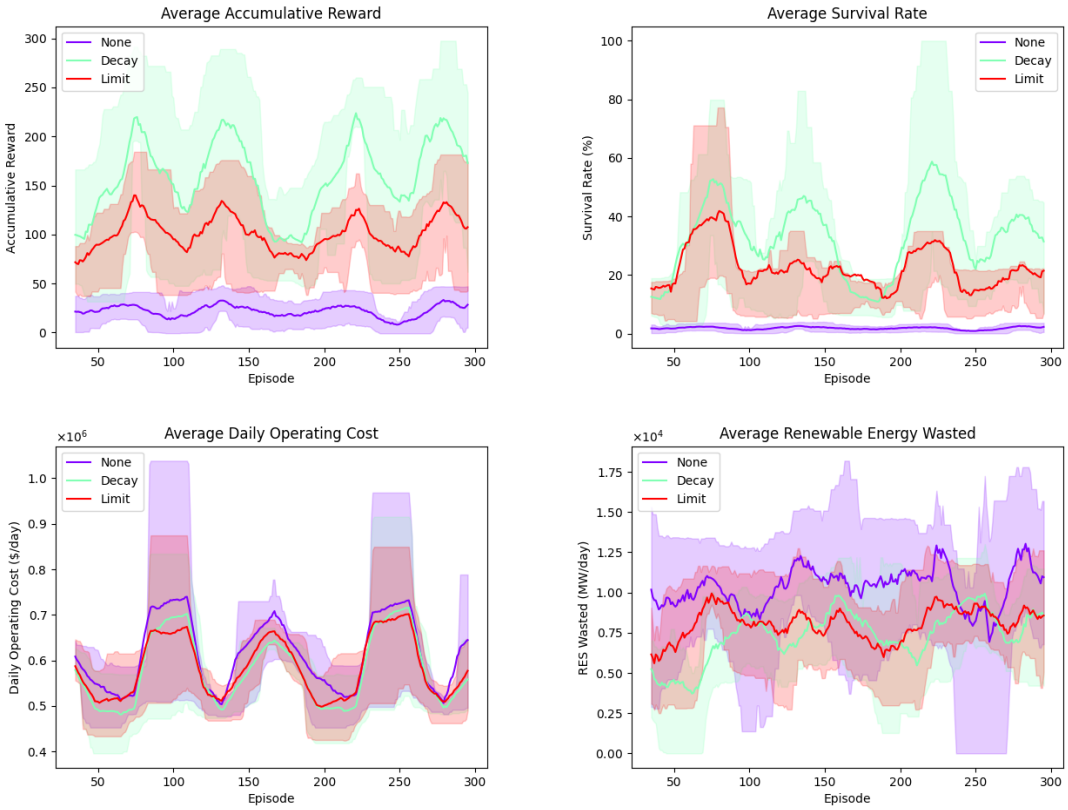

Figure C.12: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with square root curtailment limit decay, the `limit_inf` parameter and without curtailment actions during test.

Table C.4: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with square root curtailment limit decay, the `limit_inf` parameter and without curtailment actions during test.

no_curtail	**limit_inf**	**decay**	**Metrics**			
			Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
False	True	sqrt	153.77	31.16	579978.88	7356.86
False	False	sqrt	153.77	31.16	579978.88	7356.86
True	False	None	130.15	33.57	536423.98	0.00
False	True	None	98.84	22.11	588610.57	8039.41
False	False	None	22.13	1.84	619721.30	10315.47

C.5 GCN Aggregation Function

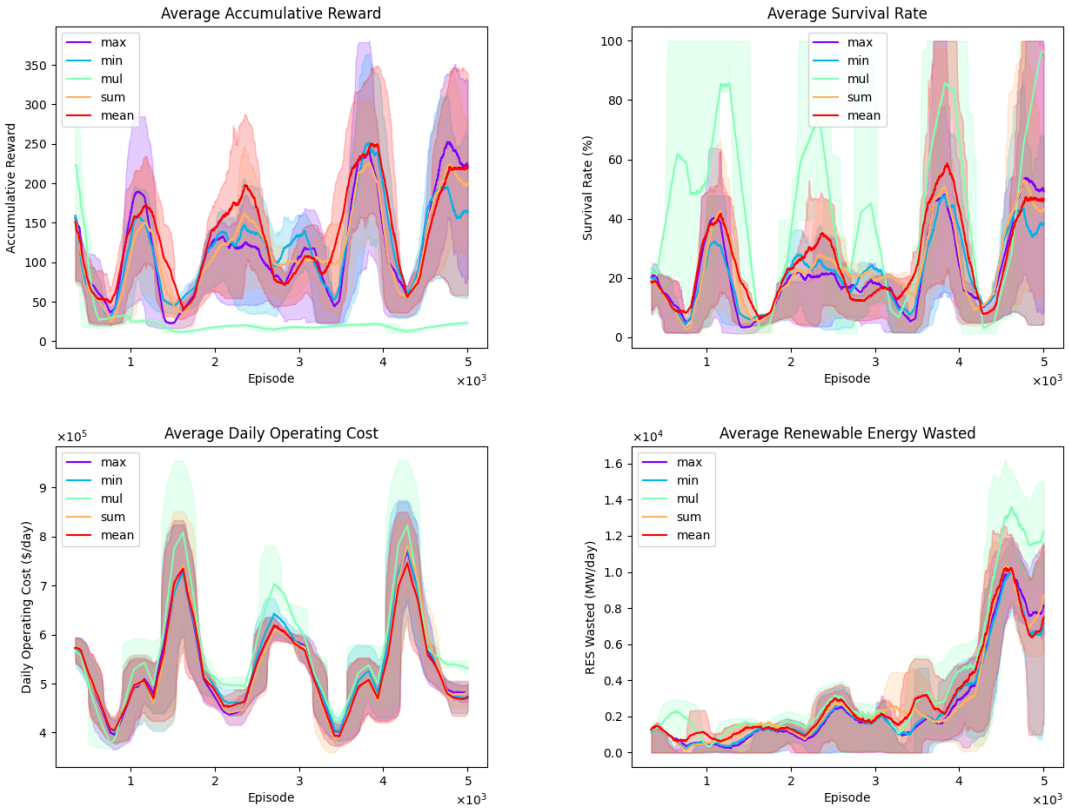

Figure C.13: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during training.

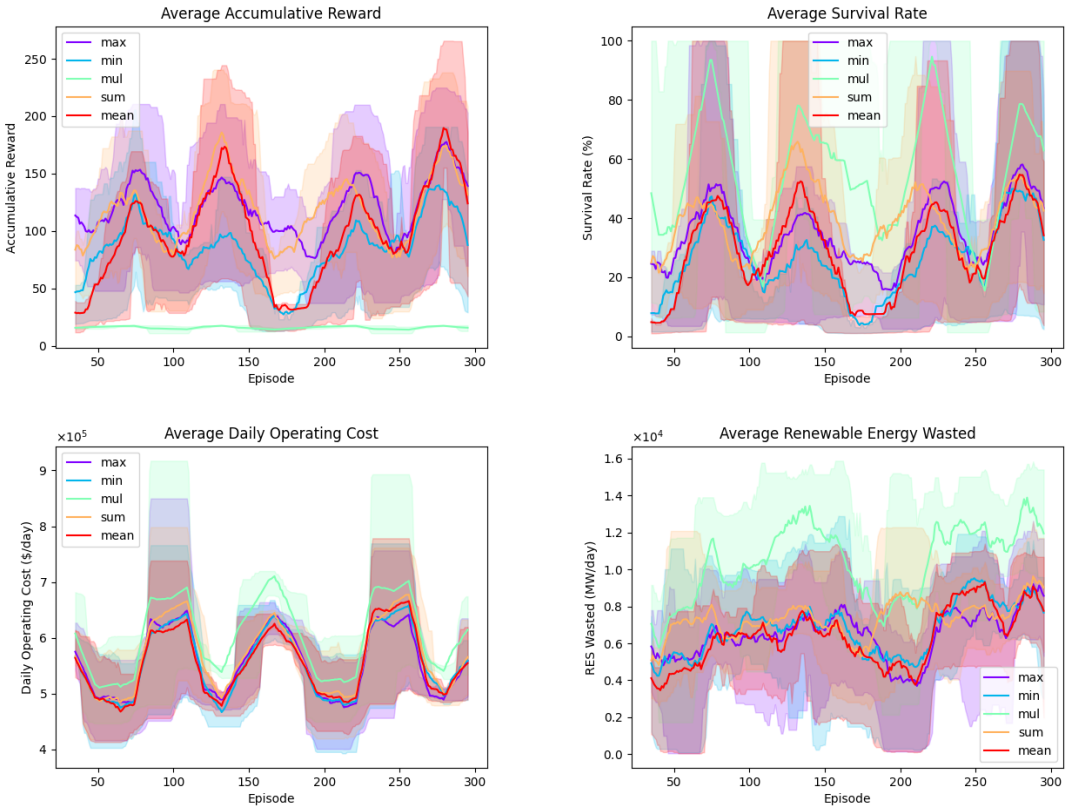

Figure C.14: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during test.

Table C.5: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with the `max`, `min`, `mean`, `sum`, and `mul` GCN aggregation schemes during test.

aggr	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
max	119.16	33.62	560219.45	6564.49
sum	114.58	38.12	569952.37	7268.28
mean	92.90	27.34	557637.84	6268.06
min	80.89	24.59	559033.63	6687.94
mul	15.76	54.77	608319.66	10305.96

C.6 GCN Layers

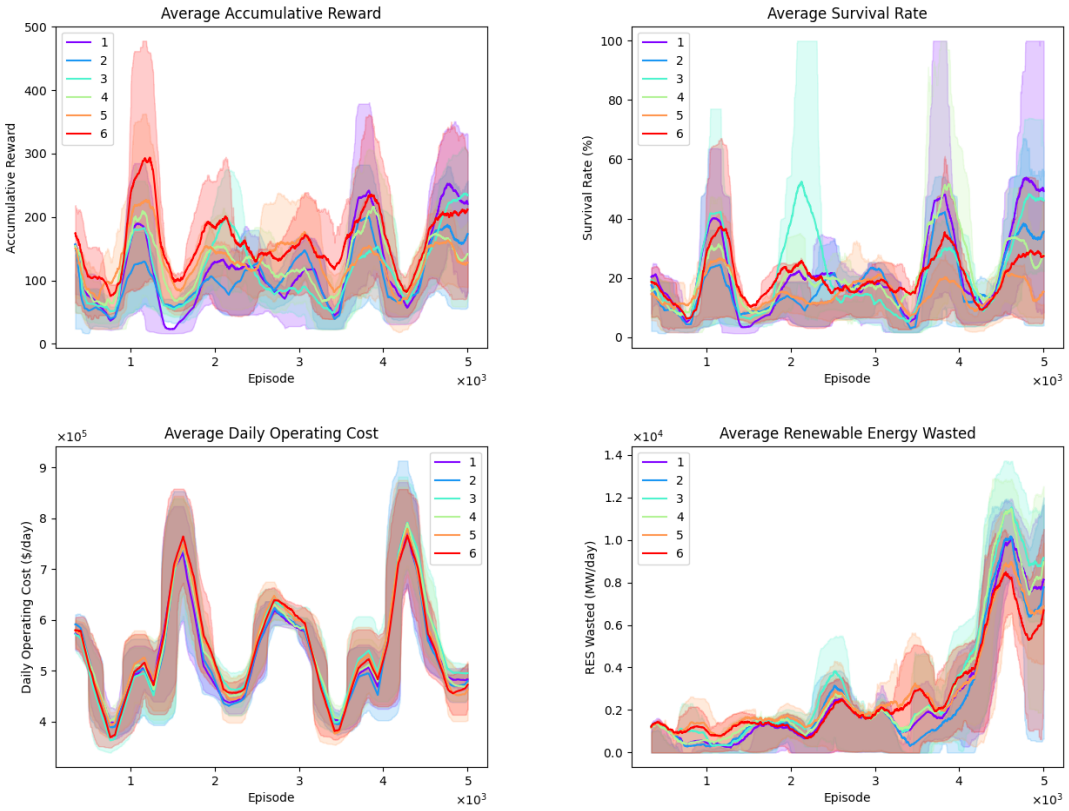

Figure C.15: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 1, 2, 3, 4, 5, and 6 GCN layers during training.

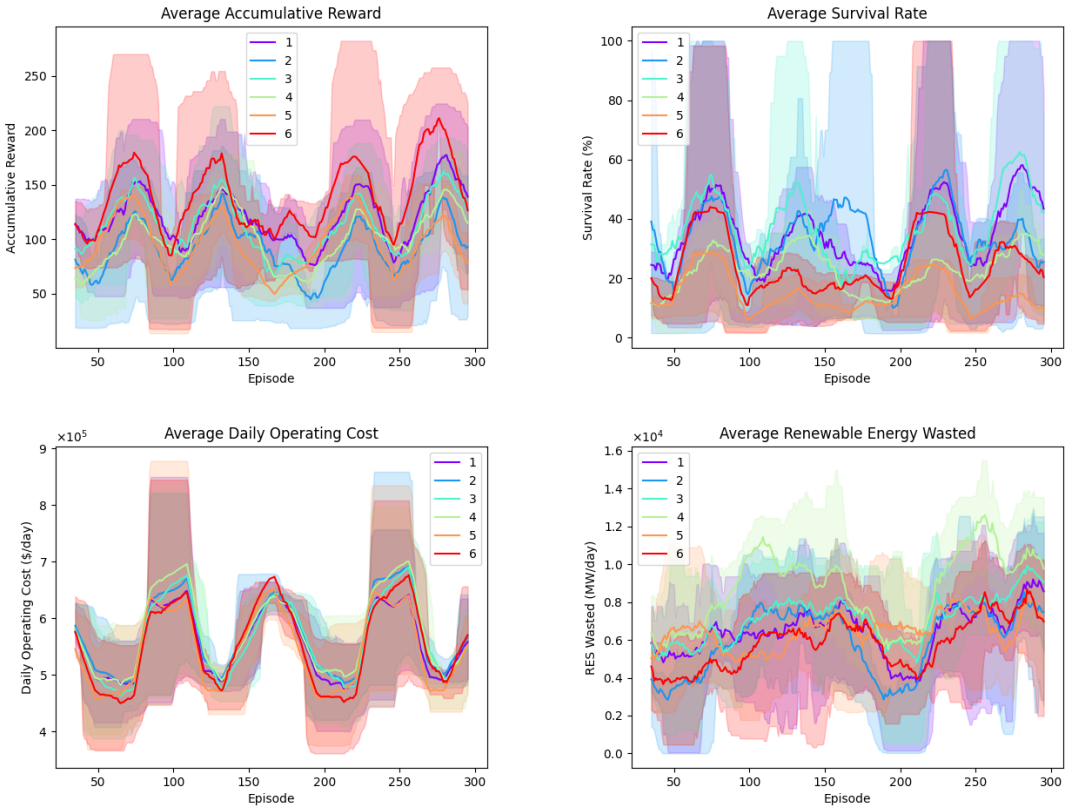

Figure C.16: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 1, 2, 3, 4, 5, and 6 GCN layers during test.

Table C.6: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with with 1, 2, 3, 4, 5, and 6 GCN layers during test.

num_layers	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
6	135.75	24.16	560147.77	5862.49
1	119.16	33.62	560219.45	6564.49
3	110.73	36.52	567510.58	7116.24
4	98.66	22.40	577464.90	8641.05
5	92.97	14.22	550205.21	6397.41
2	91.58	34.46	575093.64	5989.95

C.7 GCN Output Features

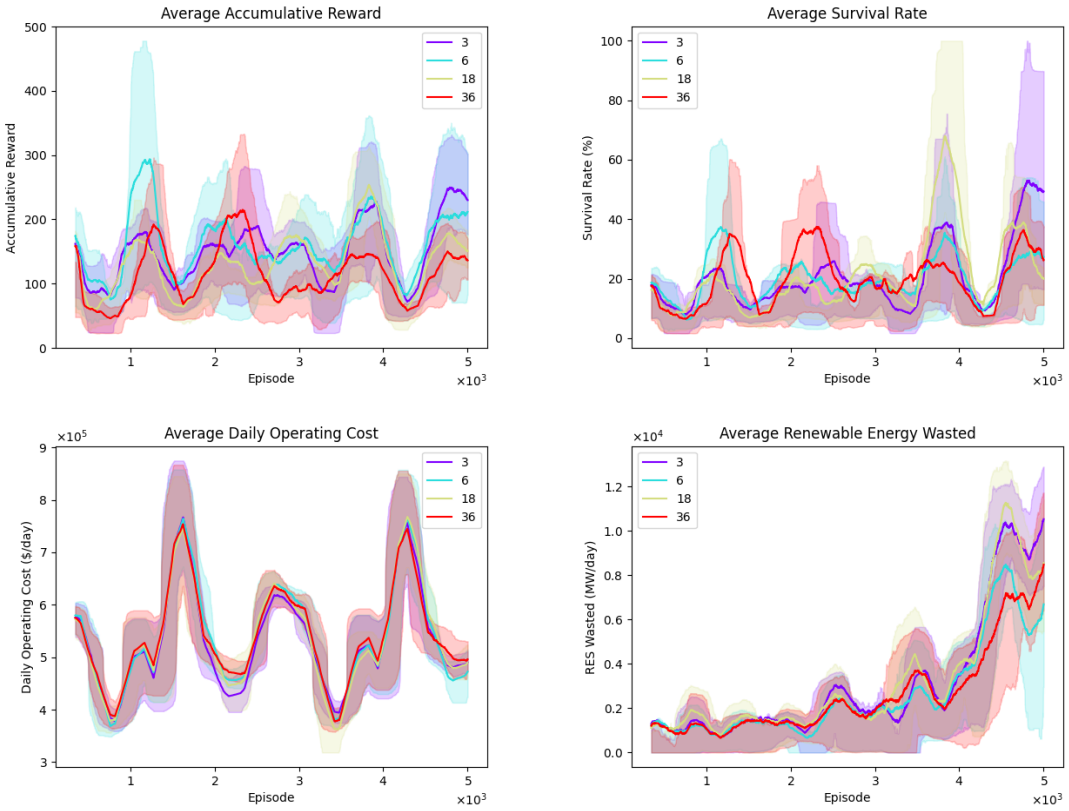

Figure C.17: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 3, 6, 18, and 36 GCN output features during training.

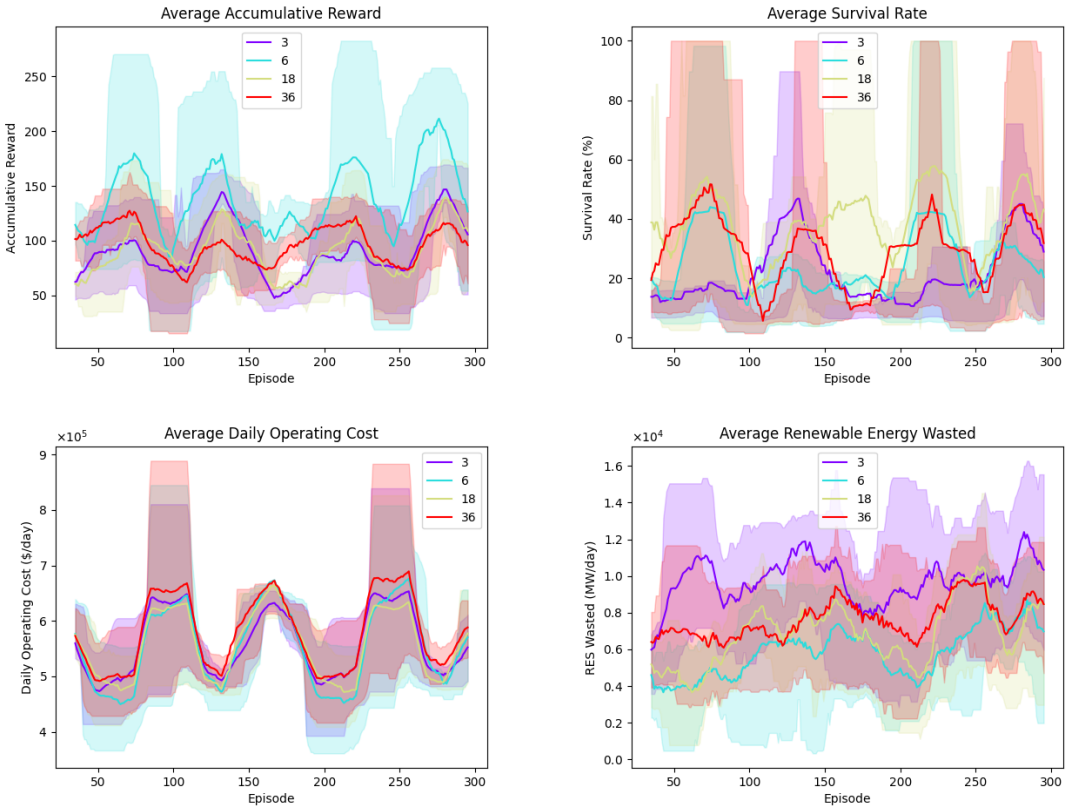

Figure C.18: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 3, 6, 18, and 36 GCN output features during test.

Table C.7: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with 3, 6, 18, and 36 GCN output features during training.

out_channels	**Metrics**			
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
6	135.75	24.16	560147.77	5862.49
36	95.92	28.35	582984.75	7575.34
18	88.47	38.58	562226.73	6874.71
3	87.68	21.25	563134.93	9497.97

C.8 GCN `act_first` and `improved` Parameters

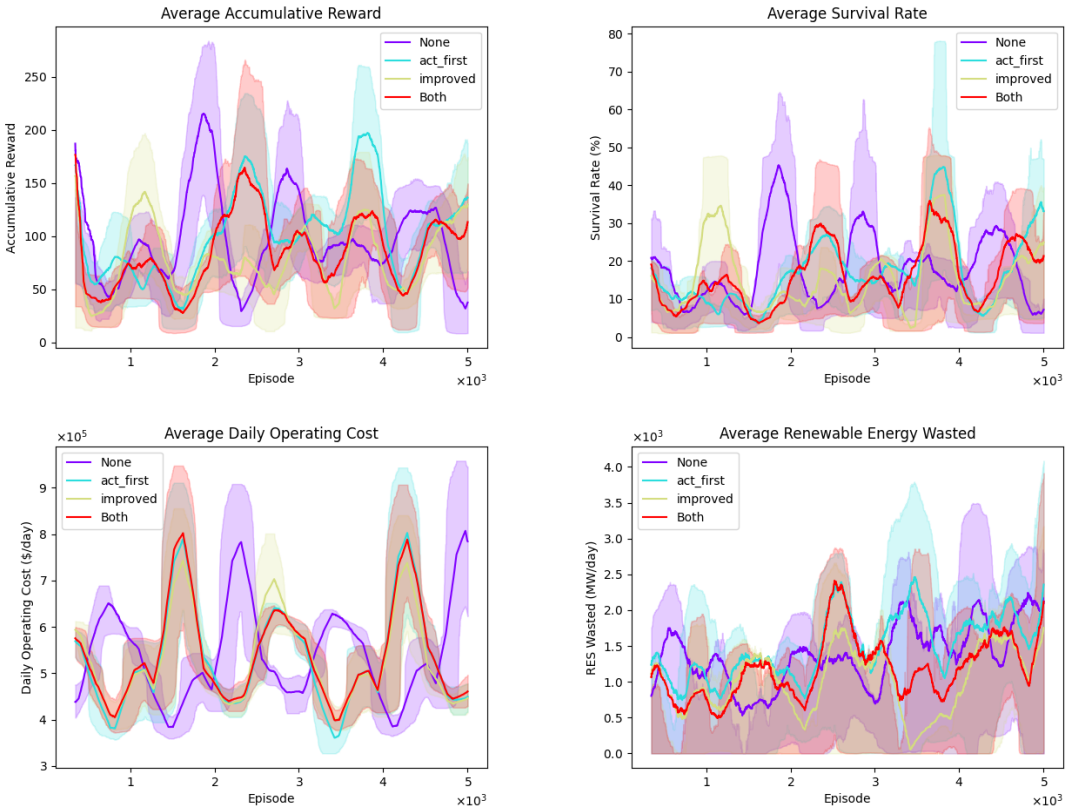

Figure C.19: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with different configurations for the GCN `act_first` and `improved` parameters during training.

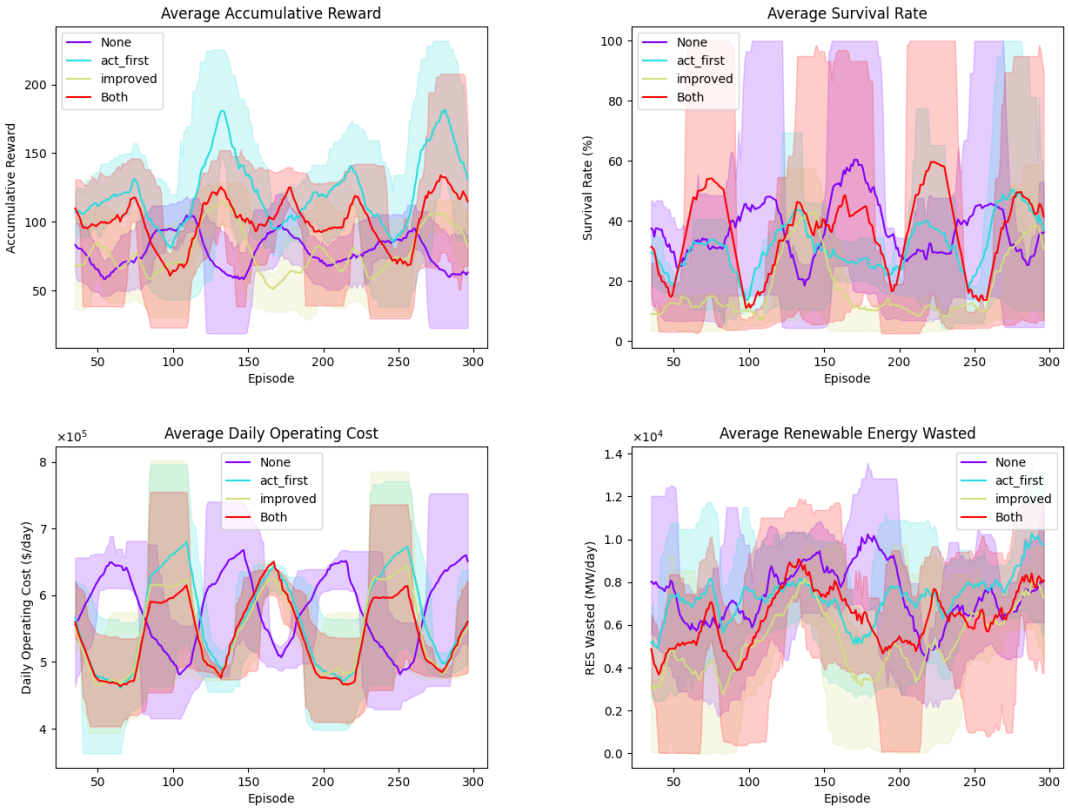

Figure C.20: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with different configurations for the GCN `act_first` and `improved` parameters during test.

Table C.8: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for models with different configurations for the GCN `act_first` and `improved` parameters during training.

act_first	**improved**	**Metrics**			
		Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	False	118.97	31.77	564201.23	7167.22
True	True	101.26	36.64	546592.78	6297.46
False	False	78.62	37.69	575798.57	7640.76
False	True	75.29	16.31	548378.53	5043.04

C.9 GAT `heads`, `v2`, and `act_first` Parameters

Table C.9: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for random search of GAT `heads`, `act_first`, and `v2` parameters during test.

act_first	heads	v2	Metrics			
			Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**
True	3	True	117.09	12.84	582144.59	7216.87
False	3	False	111.41	36.02	569354.23	7403.57
True	1	True	94.46	29.49	571177.15	6785.37
True	2	True	92.50	32.26	562035.66	7349.15
False	3	True	91.18	9.80	552279.16	8988.24
True	2	False	88.77	30.33	565936.09	7250.76
True	3	False	80.03	28.76	556860.38	6723.43
True	1	False	76.94	25.91	574402.26	8366.63
False	1	False	76.06	23.62	549382.92	7971.11
False	1	True	59.17	13.57	574194.39	5665.66

C.10 36-Bus Scenario

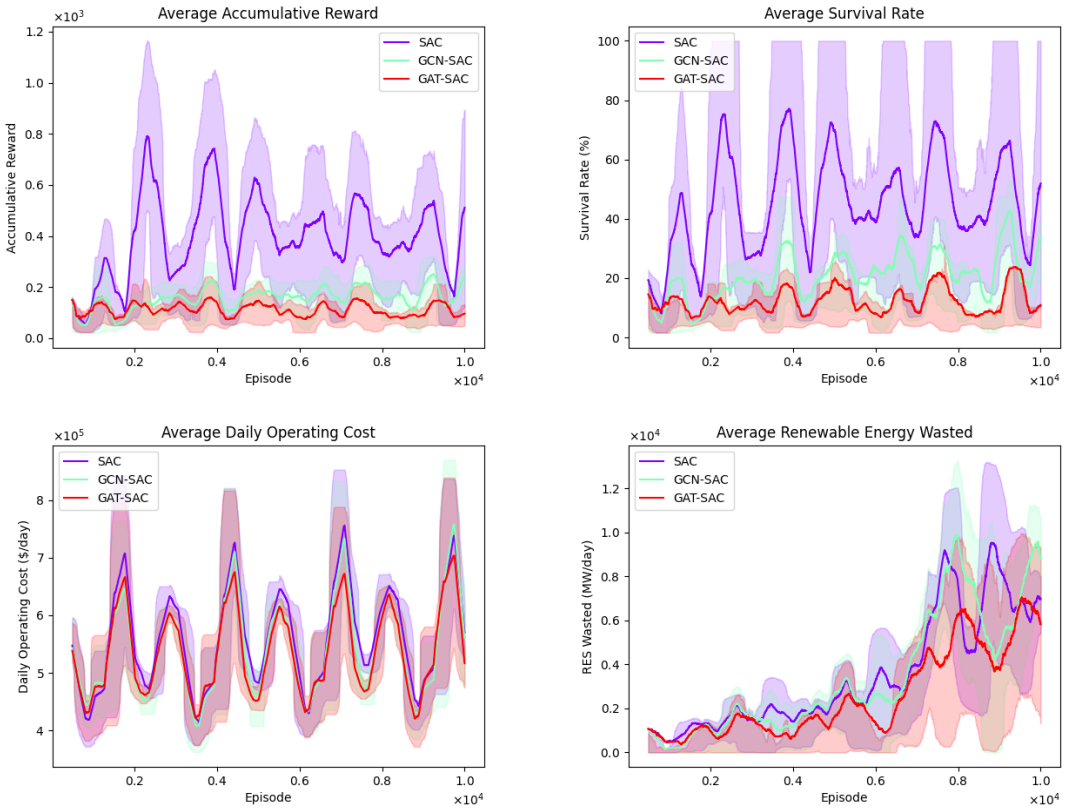

Figure C.21: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 36-bus environment.

Figure C.22: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 36-bus environment.

Table C.10: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 36-bus environment.

Model	**Metrics**					
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	342.84	55.47	601896.45	5443.65	0.0029	47.76
GCN-SAC	153.77	31.16	579978.88	7356.86	0.0097	31.11
GAT-SAC	105.75	11.50	564739.25	7070.78	0.0071	12.52

C.11 118-Bus scenario

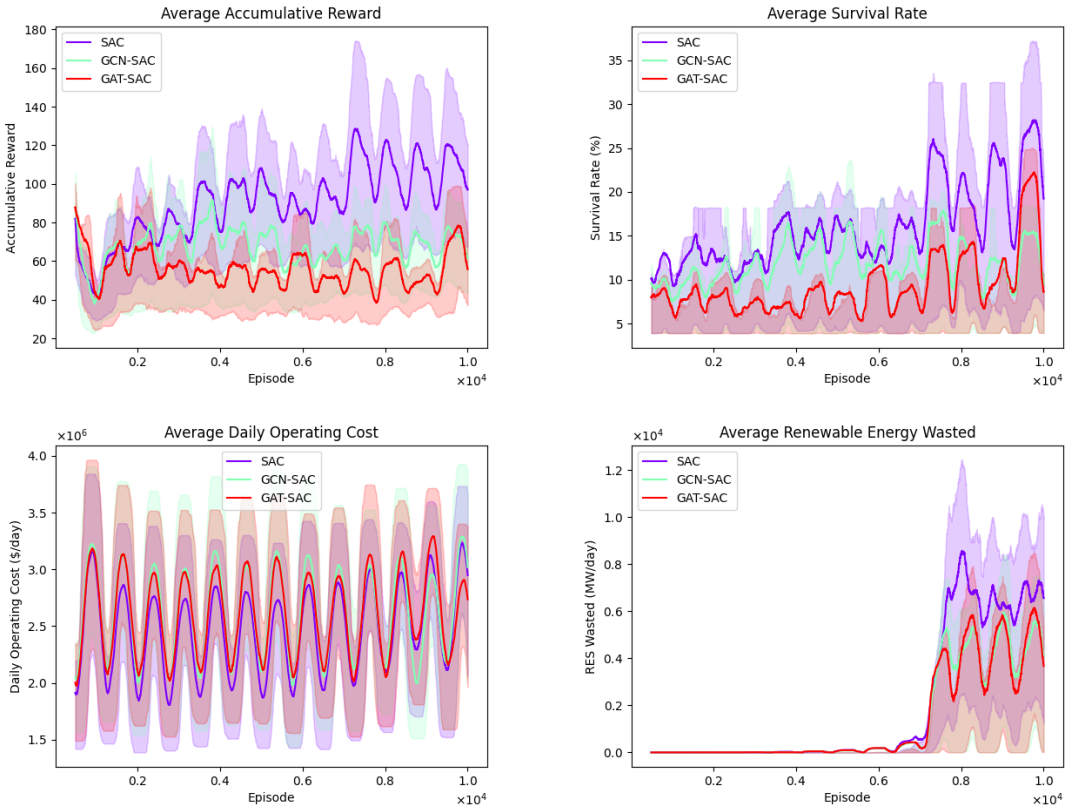

Figure C.23: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during training in the 118-bus environment.

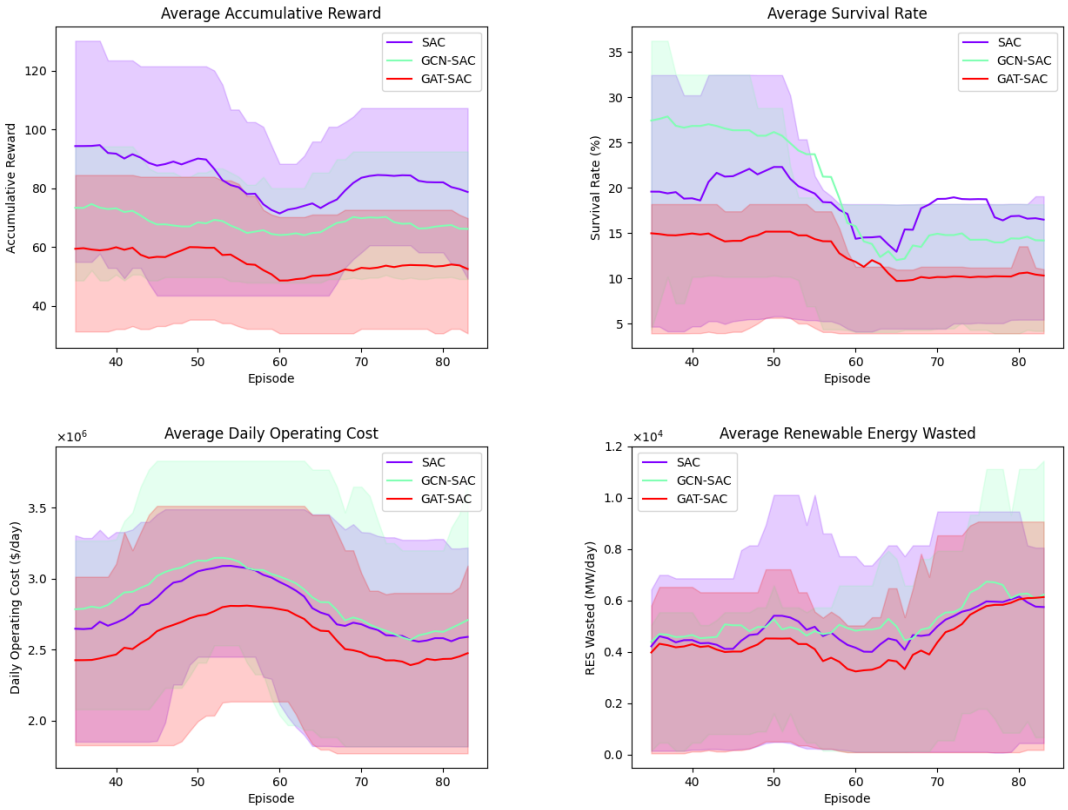

Figure C.24: Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 118-bus environment.

Table C.11: Average Accumulative Reward, Survival Rate, Daily Operating Cost, and Wasted Renewable Energy for the SAC, GCN-SAC and GAT-SAC models during test in the 118-bus environment.

Model	**Metrics**					
	Avg. CR	**Avg. SR**	**Avg. DOC**	**Avg. REW**	**Avg. ST**	**TT**
SAC	81.98	17.95	2788763.38	4740.03	0.0067	14.82
GCN-SAC	67.85	19.34	2901439.28	5323.58	0.0096	13.67
GAT-SAC	54.31	11.92	2591733.60	4709.37	0.0122	11.39