Graph-Based Multi-Agent Reinforcement Learning for Power Grid Control

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Carlo Fabrizio**

Student ID: 10747982
Advisor: Prof. Marcello Restelli
Co-advisors: Marco Mussi, Gianvito Losapio, Alberto Maria Metelli
Academic Year: 2024-25

Abstract

Climate change, which urges the rapid decarbonisation of energy-intensive industries, and the emergence of new power-hungry sectors, such as data centers, are significantly transforming traditional power grids. In particular, the necessary integration of renewable energy sources, combined with the expanding scale of power networks, presents significant challenges in controlling modern power grids. Traditional control systems, which are human and optimization-based, struggle to adapt and to scale in such evolving context, motivating the exploration of more dynamic and distributed control strategies. This thesis advances a *graph-based multi-agent reinforcement learning* (MARL) framework for real-time, scalable grid management. The proposed architecture consists of a network of distributed low-level agents acting on individual powerlines and coordinated by a high-level manager agent. A Graph Neural Network (GNN) is employed to encode network's topological information within the single low-level agent's observation. To accelerate convergence and enhance learning stability, the framework integrates Imitation Learning and potential-based reward shaping. In contrast to conventional decentralized approaches that decompose only the action space while relying on global observations, this method also decomposes the observation space. Each low-level agent acts based on a structured and informative local view of the environment constructed through the Graph Neural Network. Experiments on the Grid2Op simulation environment show that the proposed method outperforms the standard baseline, commonly employed in such contexts. A qualitative analysis of the learned GNN weights reveals the model's ability to capture structural patterns within power grids' observations, indicating strong generalization capabilities across different topologies. As a minor contribution, a novel learning-based method for decomposing power grids into independent subgrids was implemented, potentially enabling even more scalable multi-layer extensions of the proposed architecture.

Keywords: Power Grids, Reinforcement Learning, Multi-Agent Learning, Graph Neural Networks, Sequential Data Learning, Clustering

Abstract in lingua italiana

Il rapido processo di decarbonizzazione dell'industria energivora imposto dal cambiamento climatico e la nascita di nuovi settori ad alta intensità energetica, come i data centers, stanno profondamente trasformando le reti elettriche tradizionali. Nello specifico, l'introduzione di fonti di energia rinnovabili, unita all'aumento delle dimensioni delle reti elettriche, rende la gestione delle reti moderne, una sfida rilevante. I sistemi di controllo tradizionali, basati sull'intervento umano e su tecniche di ottimizzazione, fanno fatica a scalare in un contesto così mutevole, motivando lo sviluppo di soluzioni che siano dinamiche e distribuite. Questa tesi propone una procedura di controllo a tempo-reale e scalabile, basata sull'Apprendimento per Rinforzo in un contesto Multi Agente (MARL) e sull'apprendimento sui Grafi. L'architettura prevede una rete di agenti distribuiti di basso livello che agiscono sulle singole linee elettriche, coordinati da un agente di alto livello. Per includere le informazioni topologiche della rete all'interno delle osservazioni dei singoli agenti di basso livello, viene impiegata una Rete Neurale per Grafi. Per accelerare la convergenza e migliorare la stabilità, la procedura proposta integra techniche di imitazione dell'esperto e di modellamento del segnale di ricompensa tramite potenziale. A differenza degli approcci distribuiti classici per la gestione di reti elettriche, che si concentrano esclusivamente sulla scomposizione dello spazio delle azioni ma che mantengono un'osservazione globale, la soluzione proposta suddivide anche lo spazio delle osservazioni. Ciascun agente di basso livello prende decisioni basandosi su una visione locale e strutturata dell'ambiente, generata da una rete neurale per grafi. Gli esperimenti effettuati sul simulatore Grid2Op, mostrano come la procedura proposta superi in prestazioni la soluzione di riferimento comunemente adottata nell'ambito del controllo di reti elettriche. Un'analisi qualitativa sui parametri della rete neurale per grafi evidenzia l'abilità del modello di riconoscere il pattern strutturale dell'osservazione che riceve in ingresso, suggerendo buone capacità di generalizzazione. Come contributo aggiuntivo, è stata implementata una nuova procedura basata sull'apprendimento, per la decomposizione di reti elettriche in sottoreti indipendenti, aprendo la strada a possibili estensioni multi-livello della procedura proposta con poteziale aumento della scalabilità.

Parole chiave: Reti Elettriche, Apprendimento per Rinforzo, Apprendimento Multi-Agente, Reti Neurali per Grafi, Apprendimento su Dati Sequenziali, Analisi dei Gruppi

Contents

1 | Introduction

Modern power grids are undergoing a significant transformation due to the introduction of renewable energy sources and the increasing size of controlled grids, as highlighted in [35, 46].

Unlike conventional non-renewable generators, renewable ones (such as solar and wind) introduce high variability and limited dispatchability in the energy production [15]. Their energy production heavily depends on the atmospheric conditions. In addition, the electrification of large energy demanding sectors like transport, industry and large computer facilities adds further load and stress to power grids [35].

In parallel, the growing size of power grids, i.e., the increasing number of controllable elements and complex interconnections, results in a combinatorial explosion of potential control configurations.

Traditional power grid control systems, strongly reliant on human-in-the-loop and optimization-based algorithms, are unable to cope with the increasing complexity of the power grid control problem. Moreover, the integration of modern renewable energy sources into power grid systems, which are fundamentally different from non-renewable ones, requires the development of novel, and often more sophisticated, approaches to effectively transition from traditional non-renewable energy production to more sustainable alternatives.

The recent large-scale blackout that impacted Spain and Portugal clearly highlights the critical importance of developing smart and adaptable power grid control systems. According to The Guardian [15], the cause of the April 28, 2025 blackout in Spain and Portugal was likely linked to the instability of renewable generators and use of "legacy" management equipment. More precisely, a rare atmospheric phenomenon triggered oscillations in high-voltage transmission lines, which subsequently led to a widespread synchronization failure across the Iberian power grid. This cascading event caused an abrupt loss of approximately 15 gigawatts of power – around 60% of Spain's electricity supply – in just a few seconds. While the exact sequence of failures is still under investigation, this event underscores the fragility of highly interconnected power systems involving a large number of renewable generators.

To investigate the potential use of Artificial Intelligence methods within next-generation power grid controllers, the French electricity network management company RTE (Réseau de Transport d'Électricité) developed a series of "Learning to Run a Power Network" (L2RPN) challenges [35]. These challenges were designed to simulate realistic sequential decision-making environments where agents must maintain the power grid operative, under different conditions of uncertainty (the specific details will be given in Section 3.1). The challenges were offered over the Grid2Op framework [9], developed by RTE. Specifically, the Grid2Op framework enables realistic simulations of power grids and supports a wide range of control actions and interventions.

Problem Overview

This thesis focuses on topology-based control actions, which are particularly relevant in power grid operations due to their zero operational cost. These actions operate by altering the electrical connectivity of controllable components within the grid. Unlike interventions on generators – which incur in financial costs – topology reconfigurations offer a cost-free alternative for redirecting power flows and maintaining grid stability. However, the complexity of such actions lies in their discrete, combinatorial nature and their non-local effects on the grid. Therefore, the main challenge, within modern power grid control systems, is the huge and combinatorial nature of the topological action space. In realistic environments, the number of possible actions grows with the number of controllable elements, quickly leading to an unmanageable action space.

To address this issue, many competitive Reinforcement Learning (RL) methods have been developed over recent years. While effective in small or medium-sized environments, these solutions become impractical and computationally infeasible as the size of the power grid increases. Their applicability is heavily affected by the increasing dimensionality of both the observation and action spaces, resulting in poor scalability and high computational cost. This limitation becomes particularly severe when trying to control real-world power systems with thousands of elements and complex topologies.

As a response, a new research trend has emerged toward scalable control architectures. Several recent works proposed multi-agent solutions [8, 31, 52] that aim to decompose the action space across several agents, allowing each agent to independently control a specific portion of the grid. Despite a correct decomposition of the action space, these methods still rely on full or near-complete observation of the global grid state, which limits their scalability.

This thesis aims to go one step further by addressing both observation and action space dimensions of the scalability challenge. It proposes a multi-agent framework in which not only the action space but also the observation space is decomposed.

Objective

The objective of this thesis is to develop an intelligent topology-based control system for large-scale power grids using RL, with a particular focus on scalability. The goal is to design a distributed agent architecture, that can efficiently manage the increasing complexity of modern power grids by exploiting their graph-based structure.

More specifically, the thesis aims to:

- Design a multi-agent reinforcement learning framework in which individual agents control local subparts of the grid, while a higher-level controller coordinates their intervention.
- Encode grid's topological information into agents' observations, enabling better generalization and scalability.
- Implement and evaluate the proposed solution on the Grid2Op environment, comparing its performance to standard baselines and assessing its ability to maintain grid stability under uncertainty.
- Introduce a method to dynamically decompose power grids into independent subgrids.

These contributions mean to advance the development of scalable and adaptive control strategies for complex power grids using Reinforcement Learning and graph-based learning.

Contributions

This thesis proposes a scalable Graph Reinforcement Learning architecture for large-scale power grids. The main contributions are as follows:

- **Graph-based representation:** Development of an effective homogeneous graph representation of the power grid, enabling informative encoding of the system's topology.
- **Modular control architecture:** Design and implementation of a two-layer distributed architecture, consisting of low-level agents acting on powerlines and a high-level controller coordinating their behavior, promoting modularity and scalability.

- **Observation space decomposition:** Integration of Graph Neural Networks (GNNs) to perform feature extraction and implicitly decompose the observation space, allowing agents to operate on partial but informative views of the grid.
- **Imitation learning for training acceleration:** Implementation of Deep Q-Learning from Demonstrations (DQfD) [18] to inject expert behavior into the training process, significantly accelerating convergence.
- **Reward decomposition:** Design of a bootstrapped potential-based reward shaping strategy [1] aimed at improving credit assignment and learning efficiency, particularly in large-scale power grid scenarios.
- **Experimental evaluation:** Empirical validation of the proposed method on the Grid2Op simulation environment, achieving competitive performance.

Thesis Structure

The thesis is organized as follows:

- **Chapter 2 – Background:** This chapter introduces the fundamental concepts and tools relevant to the thesis. It begins with a general overview of Reinforcement Learning (RL), followed by a detailed discussion of the specific RL algorithms employed in this work. Subsequently, it touches few relevant aspects of Multi-Agent Reinforcement Learning (MARL), and essential concepts related to Graph Neural Networks (GNNs). Finally, it concludes by presenting the main principles underlying sequential data learning and Clustering.
- **Chapter 3 – Problem Description:** Presents the context and formal definition of the control task addressed in this thesis. It discusses the challenges related to scalability and outlines the research direction pursued. Finally, it provides a detailed description and formal formulation of the problem.
- **Chapter 4 – Related Work:** Reviews key literature on RL applied to power grid control, with a focus on research trends aligned with this thesis—namely, MARL approaches and methods employing GNNs. It discusses these research directions, highlighting some limitations of the existing solutions.
- **Chapter 5 – Proposed Solution:** Details the architecture designed in this work, including the graph representation of the power grid, the two-layer distributed control scheme, the use of GNNs, imitation learning, and the potential-based reward shaping mechanism.

- **Chapter 6 – Experiments and Results:** Describes the experimental setup, implementation details, and evaluation metrics. It reports the performance of the proposed method in the Grid2Op environment, comparing it to baseline approaches. In addition, a qualitative analysis of the GNN and an ablation study are conducted to gain deeper insight into the model's behavior. Finally, a comprehensive overview of the computational costs associated with the proposed solution is presented.
- **Chapter 7 – Dynamic Power Grid Clustering:** Explains the novel learning-based power grid decomposition method, along with the corresponding experimental results.
- **Chapter 8 – Conclusions and Future Works:** Summarizes the main findings of the thesis, discusses limitations, and proposes potential directions for future research.

2 | Theoretical Background

2.1. Reinforcement Learning

In the context of Artificial Intelligence and Machine Learning, Reinforcement Learning (RL) is the subtopic that deals with learning from interaction to solve sequential decision-making problems. The latter can be seen as a process in which a learner repeatedly takes actions in an environment, which in turn returns a reward. The learner's goal is to discover how to act in order to maximize the cumulative reward obtained from the environment. An intuitive description of Reinforcement Learning is provided by Sutton and Barto in [49], where they define it as:

"Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal."

This idea of goal-based learning from interaction is mathematically and computationally studied within the context of Reinforcement Learning [5, 49].

In general, sequential decision-making problems are formalized through Markov Decision Processes (MDPs) [42], in which actions influence not only the immediate reward, but also subsequent states and thus future rewards.

2.1.1. Markov Decision Processes

There are several variants of sequential decision-making problems, distinguished by the agent's capabilities to observe the environment, the structure of the state and action spaces, and the nature of the agent-environment interaction. In terms of observability, Fully Observable MDPs (simply called MDPs) assume that the agent has complete access to the true environment state, whereas Partially Observable MDPs (POMDPs) [47] model scenarios in which the agent can observe only a partial representation of the environment's state. With respect to space structure, Finite (or Countable) MDPs involve countable sets of states and actions, while Infinite MDPs allow for uncountably large or continuous state and/or action spaces. Concerning the nature of the agent-environment interaction, it can occur over discrete time steps—in which case we refer to discrete-time MDPs—or over continuous time, leading to continuous-time MDPs.

For the sake of simplicity and theoretical clarity, we restrict our attention to discrete-time countable MDPs and POMDPs. However, many of the results and algorithms discussed can be extended, under appropriate technical considerations, to discrete-time Infinite MDPs and POMDPs.

Fully Observable MDPs

The simplest case of a sequential decision-making problem occurs when the agent can fully observe the environment's state at each time step. In this case the agent-environment interaction loop occurs as described in Fig. 2.1, and assumes that the agent can access the complete state's information at each time step.

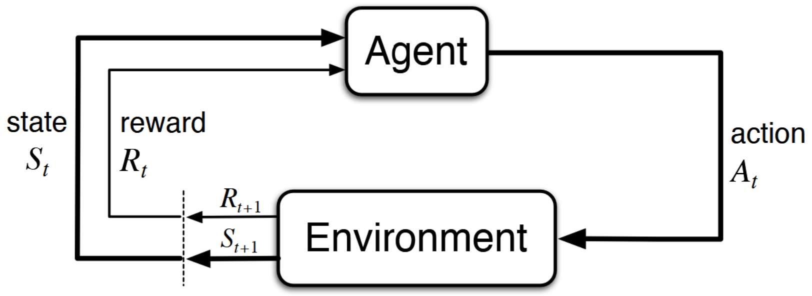


Figure 2.1: The agent-environment interaction in a MDP. Image taken from [49].

Inspired by the definition provided in [47] and in [42], a Fully Observable MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mu, \gamma)$ where:

- $\mathcal{S}$ is a set named state space.
- $\mathcal{A}$ is the set of possible actions, referred to as the action space. In general, not all actions may be valid in every state; to formalize this, one can define $A(s) \subseteq \mathcal{A}$ as the set of actions available in state s. However, for the sake of simplicity, we assume $A(s) = \mathcal{A}$ for all $s \in \mathcal{S}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function that specifies the immediate reward.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition model. $\mathcal{T}(s, a, s')$ specifies the probability of ending up in state s' after doing action a in state s.

- $\mu : \mathcal{S} \to [0, 1]$ is a probability distribution over the initial states.
- γ is the discount factor, that serves to simplify the mathematical analysis of continuing task MDP and will be explained in Section 2.1.2.

It is worth noting that, for practical purposes, it is generally assumed that the environment satisfies the Markov property:

$$Pr\{s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0\} = Pr\{s_{t+1} \mid s_t, a_t\} = \mathcal{T}(s_t, a_t, s_{t+1}), \tag{2.1}$$

even though, in reality, this assumption is almost never fully satisfied.

Partially Observable Markov Decision Processes (POMDP)

Partially Observable Markov Decision Processes relax the assumption that the agent is able to fully observe the state of the environment, which is still represented by a random variable S_t taking values in a finite state space $\mathcal{S}$ [47]. The agent instead, is able to perceive, at time t, an observation $o_t \in \Omega$, where Ω is the observation space.

Formally, a POMDP is a tuple $< \mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, O, \mathcal{R}, \mu, \gamma >$ where:

- $\mathcal{S}$ is a set named state space *(identical to MDPs)*.
- $\mathcal{A}$ is a set named action space *(identical to MDPs)*.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function that specifies the immediate reward *(identical to MDPs)*.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition model of the POMDP *(identical to MDPs)*.
- Ω is the observation space, set of observable features.
- $O : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ is the observation function, a probability distribution over the observation space.
- $\mu : \mathcal{S} \to [0, 1]$ is a probability distribution over the initial states *(identical to Fully Observable MDP)*.
- γ is the discount factor, that serves to simplify the mathematical analysis of continuing task MDP and will be explained in Section 2.1.2 *(identical to MDPs)*.

The only difference with respect to Fully Observable MDPs is the introduction of Ω and $O(o|s', a)$. The latter describes the probability of observing $o \in \Omega$ given that the agent is in state $s' \in \mathcal{S}$ after having played action $a \in \mathcal{A}$. The agent-environment interaction loop in this case is described in Fig. 2.2.

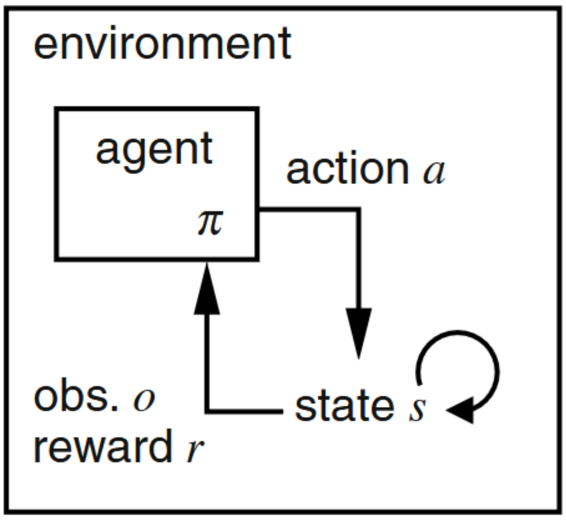


Figure 2.2: The agent-environment interaction in POMDP. Image taken from [47].

2.1.2. Goal and Reward

As discussed in Section 2.1, a RL agent is a goal-oriented agent that aims to maximize the expected cumulative reward, namely, the expected sum of the rewards received over time, denoted as the sequence $r_0, r_1, r_2, \ldots$. This concept can be generalized by defining the agent's objective as the maximization of the expected value of G_t (commonly referred to as the return), which represents a well-defined function of the reward sequence. The two definitions are equivalent if we consider $G_t = \sum_{t=0,1,2,\ldots} r_t$.
The mathematical formulation of the reward defines the specific goal the agent has to accomplish.
In general, a sequential decision-making problem can be categorized into two types based on the task duration: episodic tasks and continuing tasks.

Episodic and Continuing Task

- *Episodic task*: the sequential game the agent is playing encompasses several episodes of T_i rounds each. Practically speaking, the agent interacts with the environment through several matches that does not last forever. Each match starts in an initial state and ends in a terminal state, after a finite amount of rounds. Therefore, the reward sequence is finite and G_t is a function of T_i elements.

- *Continuing Task*: the interaction with the environment last forever, the agent plays a single never ending match with the environment. In this type of tasks, the reward sequence is infinite thus the return G_t is a function of infinite elements. If we use the plain cumulative reward as G_t, it becomes an infinite sum of elements: $\sum_{t=0}^{\infty} r_t$, making the problem mathematically complex. Therefore, it is convenient to consider the discounted return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} \,, \tag{2.2}$$

where γ is called "discount factor" so that, if the reward is non-zero, G_t converges for $\gamma < 1$. In this case, if the reward is bounded by $\hat{R}$, the discounted return is at most $\hat{R} \cdot \frac{1}{1-\gamma}$.

It is worth remarking that it is possible to turn an episode of an episodic task scenario to an infinite task scenario by considering $R_t = 0 \quad \forall t > T_i$. Leveraging this idea, it is possible to turn an episodic task scenario into an infinite task scenario gaining the advantage of using a single unified notation for both cases.
An important property of the return is that it satisfies the following recursive relationship:

$$\begin{aligned} G_t &\doteq r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots = r_t + \gamma \cdot \big(r_{t+1} + \gamma \cdot r_{t+2} + \dots\big) \\ &= r_t + \gamma \cdot G_{t+1} \,. \end{aligned} \tag{2.3}$$

2.1.3. Policy and Value Function

The final goal of an RL agent is to understand what is the best action to play in each situation with the aim of maximizing the expected return. More precisely, the RL agent wants to learn a policy $\pi(a|s) = Pr\{A_t = a|S_t = s\}$ that maximizes the expected return. In order to compare two policies, it is useful to define the *state-value function* $v_\pi(s)$ for a state s under policy π:

$$v_\pi(s) \doteq \mathbb{E}\big[G_t|S_t = s\big] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} \middle| S_t = s\right], \quad \forall s \in \mathcal{S} \,. \tag{2.4}$$

Therefore, given two policies π and π', if $v_\pi(s) \geq v_{\pi'}(s)$, $\forall s \in \mathcal{S}$, then we can say that $\pi \geq \pi'$. To evaluate the effect of a single-action deviation from a policy π, i.e., playing always policy π except in state s_t, where one plays action a, we can define the *action-value* *function*:

$$q_\pi(s,a) \doteq \mathbb{E}\big[G_t|S_t = s, A_t = a\big] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k}\middle|S_t = s, A_t = a\right], \quad \forall\big(s \in \mathcal{S} \wedge a \in \mathcal{A}\big). \tag{2.5}$$

The two quantities are related through

$$v_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[q_\pi(s,a)\right] . \tag{2.6}$$

Both the state-value function and the action-value function satisfy a recursive relation called *Bellman Equation* that derives from Eq. (2.3). The recursive equation for the state-value function is:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_t + \gamma v_\pi(s_{t+1}) \mid s_t = s\right] \\ &= \sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}} \pi(a|s)\mathcal{T}(s,a,s')(\mathcal{R}(s,a,s') + \gamma v_\pi(s')) \end{aligned} \tag{2.7}$$

The latter formula is useful if we want to evaluate the state-value function for a specific policy, given that we know exactly the transition model of the MDP. In other words, given a policy $\pi(a|s)$, if we know exactly the transition model of the MDP, namely $\mathcal{T}(s,a,s')$, $\forall(s',a,s)$, we can in principle compute exactly the state value function $v_\pi(s)$, $\forall s \in \mathcal{S}$, by solving the following system of equations for the $|\mathcal{S}|$ variables $v_\pi(s_i)$, $i = 1, ..., |\mathcal{S}|$:

$$\begin{cases} v_\pi(s_1) = \sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}} \pi(a|s_1)\cdot\mathcal{T}(s_1,a,s')\cdot\big(\mathcal{R}(s_1,a,s') + \gamma\cdot v_\pi(s')\big)\,, \\ v_\pi(s_2) = \sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}} \pi(a|s_2)\cdot\mathcal{T}(s_2,a,s')\cdot\big(\mathcal{R}(s_2,a,s') + \gamma\cdot v_\pi(s')\big)\,, \\ \quad\vdots \\ v_\pi\left(s_{|\mathcal{S}|}\right) = \sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}} \pi(a|s_{|\mathcal{S}|})\cdot\mathcal{T}(s_{|\mathcal{S}|},a,s')\cdot\big(\mathcal{R}(s_{|\mathcal{S}|},a,s') + \gamma\cdot v_\pi(s')\big)\,. \end{cases} \tag{2.8}$$

This is a linear system of $|\mathcal{S}|$ equations in $|\mathcal{S}|$ variables and it admits a unique solution if and only if termination is guaranteed for each initial state or $\gamma < 1$.
In practice, solving exactly this system of equations is too expensive, thus an approximate solution via iterative methods is generally computed (random initialization and iterative updates until convergence).

As for the action-value function, the corresponding equation is given below, its derivation follows a similar reasoning to that of the state-value function,

$$\begin{aligned} q_\pi(s,a) =& \mathbb{E}\left[r_t + \gamma \cdot v_\pi(s_{t+1}) \middle| s_t = s, a_t = a\right] \\ =& \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \Big(\mathcal{R}(s,a,s') + \gamma V_\pi(s') \Big) \end{aligned} \tag{2.9}$$

Optimal Policies and Optimal Value Functions

Even though there could be multiple optimal policies, we denote all of them with π_*. They all share the same state-value v_* function, called optimal state-value function and defined as:

$$v_*(s) \doteq \max_\pi \big(v_\pi(s)\big), \quad \forall s \in \mathcal{S}. \tag{2.10}$$

Optimal policies also share the same action-value function, denoted q_* and defined as:

$$q_* \doteq \max_\pi \big(q_\pi(s,a)\big), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.11}$$

Notably, q_* and v_* are related by the following expression:

$$v_*(s) = \max_a (q_*(s,a)) \quad \forall s \in \mathcal{S}. \tag{2.12}$$

It is worth noticing that there exists methods of Dynamic Programming that allow computing the optimal policies. However, they require the complete knowledge of the environment dynamics as a MDP. In other words, it is necessary to know $\mathcal{T}(s,a,s')$ so that we can exactly or approximately solve the System of Equations (2.8), and iteratively compute the optimal policy by greedily improving the current policy.
To overcome these limitations of Dynamic Programming methods, which are still useful for theoretical purposes, methods based on Temporal Difference (TD) learning have been developed.

2.1.4. Temporal Difference Learning

Rather than computing exactly $v_\pi(\cdot)$, the goal is to estimate it ($\hat{v}_\pi$) based on its difference with respect to a "target" (y) that ideally reflects the real $v_\pi(\cdot)$. In essence, the aim is to reduce the TD error, defined as:

$$TD = y - \hat{v}_\pi(\cdot)\,. \tag{2.13}$$

More precisely, the target can be defined as a truncated return, representing a partial estimate of the full return over a limited horizon.

- One option is to wait T time steps, or until the end of the episode, to obtain a highly accurate target, as done in the *Constant-α Monte Carlo* approach:

  $$y_t = \sum_{k=0}^{T} \gamma^k r_{t+k} \, .$$

  Although providing highly accurate estimates, this approach yields only a limited number of data points within a given time frame, thereby restricting the agent's learning capabilities.

- A more efficient alternative is to update the estimate at each time step, following the *TD(0)* method:

  $$y_t = r_t + \gamma \cdot \hat{v}\big(s_{t+1}\big) \, , \tag{2.14}$$

  where $\hat{v}(\cdot)$ denotes the current approximation of the state-value function. This approach enables more frequent updates and improves data efficiency, at the cost of reduced estimate accuracy.

- A compromise between the previous two approaches is offered by the *TD(n)* method:

  $$y_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} R_{t+n-1} + \gamma^n \hat{v}\big(s_{t+n}\big) \, . \tag{2.15}$$

In general, standard reinforcement learning algorithms employ the *TD(0)* target. Two main categories of methods – *"On-Policy"* and *"Off-Policy"* – are defined based on how the estimate $\hat{v}_\pi(s_{t+1})$ is computed within the *TD(0)* framework. This distinction naturally extends to the more general *TD(n)* setting as well.

- Off-Policy: the estimate of $\hat{v}_\pi(s_{t+1})$ is computed using a policy different from the one used to generate the data – typically a greedy or improved version of the current policy. Q-Learning [59] is a canonical example of an off-policy algorithm and its target is defined as:

  $$y_t = r_t + \gamma \max_a \big(\hat{Q}(s_{t+1}, a)\big), \tag{2.16}$$

  the max operator makes this method an Off-Policy algorithm.

- On-Policy: the estimate of $\hat{v}_\pi(s_{t+1})$ is derived using the same policy that the agent follows during interaction with the environment. SARSA [49] is a representative on-policy algorithm and its target is defined as:

$$y_t = r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}).$$

2.1.5. Reward Shaping

In RL, the reward function plays a central role as it defines the learning objective of the agent. The design of the reward function is therefore crucial: a poorly designed reward may lead to suboptimal, unstable, or even unintended behaviors, while a well-designed reward can significantly accelerate learning and improve performance.

However, in many real-world applications, especially those involving sparse or delayed rewards, the raw environmental reward signal may be insufficient to support efficient learning. In these cases, *reward shaping* techniques are often introduced to provide additional guidance, helping agents discover effective policies more quickly.

A particularly robust and widely studied method is *Potential-Based Reward Shaping* (PBRS), introduced by Ng et al. [38]. PBRS modifies the reward signal by adding the difference of a potential function Φ evaluated at successive states:

$$\widetilde{\mathcal{R}}(s, a, s') = \mathcal{R}(s, a, s') + \gamma\Phi(s') - \Phi(s). \tag{2.17}$$

This additive shaping term preserves the optimal policy by design, as the contributions of the potential function cancel out in a telescopic manner. Wiewiora [60] later showed that PBRS is equivalent to initializing the Q-function for Q-learning agents, establishing a strong theoretical connection between reward shaping and value-based methods. Traditionally, static heuristic functions, such as the Manhattan distance to a goal, are used as potential functions. While effective in simple goal-oriented MDPs, this approach becomes impractical in complex environments where task-specific knowledge is unavailable or difficult to define.

To address the challenge of designing informative potential functions in high-dimensional or complex environments, Adamczyk et al. [1] proposed the *Bootstrapped Reward Shaping* (BSRS). Instead of relying on handcrafted static potentials, BSRS uses the agent's current estimate of the state-value function as the potential. This dynamic, self-adaptive approach preserves the theoretical guarantees of PBRS while improving sample efficiency in both tabular and deep RL settings. Practically speaking, the potential function at timestep n

becomes:

$$\phi^n(s) = \eta \max_a \big(Q^n(s,a)\big) = \eta V^n(s), \tag{2.18}$$

where η is a parameters to be tuned that must be in the range $[-1, (1-\gamma)/(1+\gamma)]$ to ensure theoretical convergence. In this way, using a Q-learning algorithm, the target computation simply becomes:

$$\begin{aligned} y_t &= \hat{r}_t + \gamma \max_a \big(Q(s_{t+1}, a)\big) \\ &= \Big[r_t + \gamma\eta \max_a \big(Q(s_{t+1}, a)\big) - \eta \max_a \big(Q(s_t, a)\big)\Big] + \gamma \max_a \big(Q(s_{t+1}, a)\big) \\ &= \Big[r_t - \eta \max_a \big(Q(s_t, a)\big)\Big] + \gamma(1+\eta) \max_a \big(Q(s_{t+1}, a)\big) . \end{aligned} \tag{2.19}$$

In Multi-Agent RL settings with a shared global reward, the impact of a single agent's action on the overall reward can be highly diluted, making it difficult for agents to discern their individual contributions. This challenge is further exacerbated by the inherent non-stationarity of the environment, caused by simultaneous updates from multiple learning agents. In this thesis, we explore the use of Bootstrapped Reward Shaping (BSRS) in collaborative MARL scenarios to address this issue. The goal is to enable each agent to implement BSRS to speed-up the convergence.

2.2. Deep Reinforcement Learning

Considering those scenarios in which the state space $\mathcal{S}$ and/or the action space $\mathcal{A}$ are extremely large or the model of the environment is highly complex, standard RL methods are unfeasible. In such cases, approximate methods are required to reduce the complexity of the task. Deep neural networks act as powerful function approximators, making them well-suited for addressing these complexities.
Although numerous Deep RL methods have been developed and explored, this thesis specifically focuses on Deep Q-Learning approaches.
The reason why the main focus is on Deep Q-Learning approaches is because they are well suited and state-of-the-art in relatively small discrete action spaces [37], which is the case of this thesis's task.

2.2.1. DQN

In general, Q-Learning based methods aim to estimate the so called "Q-table", an $|\mathcal{S}| \times |\mathcal{A}|$ matrix with entries $q_\pi(s,a)$, $\forall(s,a)$. However, in same cases the dimension of the Q-table makes the learning task unpractical, like, for instance, in any continuous observation space

task.
The goal of Deep Q-learning [37] is turning the estimation of the Q table from "instance-based" to "model-based". In other words, instead of storing the entire Q-table, model the q-values by means of a function approximator (neural network) driven by parameters θ. The goal is learning an approximate action-value function $Q(s, a|\theta)$. To this end, the learning objective (loss function) is based on the Bellman Equation 2.9.
The target is defined as a TD(0):

$$y_t = r_t + \gamma \cdot \max_{a'} \left(Q(s_{t+1}, a'|\theta^-) \right), \tag{2.20}$$

where θ^- are the frozen parameters of the previous iteration.
The loss function is the MSE between the target and the estimate of the Q values:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1})} \left[(Q(s_t, a_t|\theta) - y_t)^2 \right]. \tag{2.21}$$

The gradient of the loss is:

$$\nabla \mathcal{L}(\theta) \propto \mathbb{E}_{(s_t, a_t, s_{t+1})} \left[(Q(s_t, a_t|\theta) - y_t) \cdot \frac{\partial Q(s_t, a_t|\theta)}{\partial \theta} \right]. \tag{2.22}$$

By using stochastic gradient descent, the expectation becomes a single observation, while, by using a mini-batch approach, the expectation become an average over the mini-batch losses.
To compute the loss, a set of tuples $< s_t, a_t, r_t, s_{t+1} >$ is required, commonly referred to as *"experiences"*. In essence, training involves interacting with the environment to gather experiences, and then learning from them to gradually improve the agent's performance. This latter point clearly illustrates the classic exploration-exploitation trade-off in RL: should the agent explore the action space by trying novel actions in the hope of collecting valuable new experiences, or should it rather exploits its current knowledge by selecting actions it currently believes to be optimal, aiming to maximize reward and refining their value estimates?
The vanilla Deep Q-Learning algorithm from [37] is shown in Alg. 2.1.

Algorithm 2.1 Deep Q-Learning with Experience Replay

```
1: Initialize replay memory $\mathcal{D}$ to capacity $N$
2: Initialize action-value function $Q$ with random weights
3: for episode = 1 to $M$ do
4:     Initialize sequence $s_1 = \{x_1\}$ and preprocess $\phi_1 = \phi(s_1)$
5:     for $t = 1$ to $T$ do
6:         With probability $\epsilon$, select a random action $a_t$
7:         Otherwise, select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
8:         Execute action $a_t$ and observe reward $r_t$ and next image $x_{t+1}$
9:         Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10:        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
11:        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
12:        Set target:
               $$y_j = \begin{cases} r_j & \text{if terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$$
13:        Perform gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
14:    end for
15: end for
```

Notice that, the pseudocode develops a DQN agent with ϵ-Greedy policy and works with a transformation of the state $\phi(s_t)$ rather than with the plain state s_t. The function $\phi(\cdot)$ represents a pre-processing step that is applied to the state before passing it to the agent. The algorithm maintains two networks, a main network and a target network. Those networks are initialized in the same way and synchronized every C steps. Only the main network is trainable; the target network is used to provide the estimations for the target y_t. The use of a target network is crucial for the success of DQN as proven by [11].

2.2.2. Double DQN

Every off-policy RL algorithm, such as Q-learning or Deep Q-learning, is affected by the problem of *maximization bias* or *overestimation* [49]. This issue arises when the same network is used both to select the best action and to evaluate its value during the computation of the target, see Eq. (2.16).

To address this problem, [54] proposes the use of *double learning*, which involves maintaining two separate networks, denoted as $Q_1(\cdot)$ and $Q_2(\cdot)$. At each iteration, one of the two networks is randomly selected and used to choose the greedy action, while the other evaluates its value.

Specifically, if $Q_1(\cdot)$ is chosen to select the action, then the Q-learning update for that iteration is performed according to the following formula:

$$\begin{aligned} Q_1(s_t, a_t) \leftarrow & Q_1(s_t, a_t) \\ & + \alpha \cdot \left[r_t + \gamma \cdot Q_2 \left(s_{t+1}, \arg\max_a \left(Q_1(s_{t+1}, a) \right) \right) - Q_1(s_t, a_t) \right], \end{aligned} \tag{2.23}$$

while, if the network $Q_2(\cdot)$ is selected, then the update rule becomes:

$$\begin{aligned} Q_2(s_t, a_t) \leftarrow & Q_2(s_t, a_t) \\ & + \alpha \cdot \left[r_t + \gamma \cdot Q_1 \left(s_{t+1}, \arg\max_a \left(Q_2(s_{t+1}, a) \right) \right) - Q_2(s_t, a_t) \right]. \end{aligned} \tag{2.24}$$

The concept of double learning can be naturally extended to the Deep Q-learning framework [55] by modifying the target computation used in the standard DQN formulation, see Eq. (2.20), as follows:

$$y_t^{double} = r_t + \gamma \cdot Q \left(s_{t+1}, \arg\max_a \left(Q(s_{t+1}, a | \theta) \right) | \theta^- \right), \tag{2.25}$$

which essentially consists in using the main network to select the optimal actions, whereas their values are evaluated through the target network.

2.2.3. Dueling DQN

Let us introduce a new quantity called *advantage function*, that relates the state-value and the action-value functions:

$$A_\pi(s, a) \doteq Q_\pi(s, a) - V_\pi(s). \tag{2.26}$$

Intuitively, the advantage function gives a relative measure of the importance of each action.
By rearranging the definition, we obtain the following equation, which forms the basis of the dueling network architecture:

$$Q_\pi(s, a) = V_\pi(s) + A_\pi(s, a). \tag{2.27}$$

The dueling network architecture computes the Q values by forcing the latter relation during the learning phase [58]. In particular, the network produces two streams: the value stream and the advantage stream, which are finally summed to obtain Q value estimations, as shown in Fig. 2.3. Trivially, the dueling network architecture learns which states are good independently of the action, potentially enhancing learning efficiency and generalization capabilities.

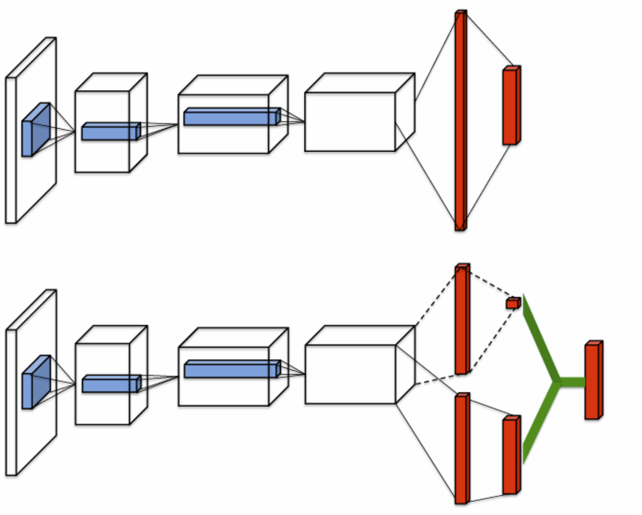


Figure 2.3: Single stream of the vanilla DQN (**top**) and a double stream of the Dueling DQN (**bottom**). Image taken from [58].

From a mathematical point of view, the Q value approximator is parametrized as:

$$Q(s, a|\theta, \alpha, \beta) = V(s|\theta, \beta) + A(s, a|\theta, \alpha). \tag{2.28}$$

It is important to keep in mind that $V(s|\theta, \beta)$ and $A(s, a|\theta, \alpha)$ are not good estimators for the state-value function and the advantage function, respectively. In other words, we can not recover A and V uniquely, given an estimation of Q from a dueling network that implements Equation (2.28). To see this, consider the following equation:

$$Q(s, a|\theta, \alpha, \beta) = \Big(V(s|\theta, \beta) + C\Big) + \Big(A(s, a|\theta, \alpha) - C\Big); \tag{2.29}$$

the estimates of the Q values are untouched but we obtain two different estimates of V and A. Therefore, directly using Equation (2.28) to implement the dueling network could result in poor performance.
To overcome this lack of identifiability, additional information and constraints must be

used to better characterize the estimated quantities. Note that, for a deterministic policy $a^* = \arg\max_{a' \in \mathcal{A}} \big(Q(s, a')\big)$, it follows that $Q(s, a^*) = V(s)$ and thus $A(s, a^*) = 0$. Therefore, good estimators of A and V are obtained by forcing the advantage function to have zero advantage at the chosen best action:

$$Q(s, a|\theta, \alpha, \beta) = V(s|\theta, \beta) + \Big(A(s, a|\theta, \alpha) - \max_{a' \in \mathcal{A}} \big(A(s, a'|\theta, \alpha)\big)\Big). \tag{2.30}$$

In order to increase the stability of the optimization, the max operator can be replaced with a mean:

$$Q(s, a|\theta, \alpha, \beta) = V(s|\theta, \beta) + \left(A(s, a|\theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'|\theta, \alpha)\right). \tag{2.31}$$

2.2.4. Prioritized Experience Replay Buffer (PERB)

The vanilla implementation of DQN, shown in Alg. 2.1, uniformly stores and samples experiences from the buffer, with no particular or additional criteria. Clearly, the storing and sampling procedure offers many potential improvements for a more efficient learning. Intuitively, we would like our agent to learn more accurately those past environment interactions that were difficult to estimate. To this end, a plain Experience Replay Buffer is replaced by a Prioritized Experience Replay Buffer (PERB) [44], which prioritizes the sampling of experiences that were difficult to be learned.
To measure how much an experience was surprising/unexpected for the agent, the absolute value of the TD error is used. An intuitive approach to implement a PERB could be: sample experience i with probability $P(i)$ proportional to the absolute value of the TD error $|\delta_i|$. However, this implementation introduces a non-negligible bias, negatively affecting the convergence of the model. To correct this bias, *weighted Importance Sampling (weighted IS)* is adopted when performing the network's updates. The sample weights are computed as:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}. \tag{2.32}$$

Therefore, an experience having a high probability of being sampled $(P(i))$, will have a reduced impact in the network update (small w_i). The exponent β controls the magnitude of the correction, total correction when $\beta = 1$ and no correction when $\beta = 0$. Since in typical RL the correction is most important near convergence, at the end of training, the value of β is increased over time from its initial value β_0 to 1. The complete pseudocode to implement Double DQN with proportional prioritization is taken from the original paper

[44] and provided in Alg. 2.2.

Algorithm 2.2 Double DQN with Proportional Prioritization

```
1: **Input:** minibatch $k$, step-size $\eta$, replay period $K$, replay size $N$, exponents $\alpha$, $\beta$, budget $T$
2: Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
3: Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
4: for $t = 1$ to $T$ do
5:     Observe $S_t$, $R_t$, $\gamma_t$
6:     Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with priority $p_t = \max_{i<t} p_i$
7:     if $t \equiv 0 \mod K$ then
8:         for $j = 1$ to $k$ do
9:             Sample transition $j \sim P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$
10:            Compute importance-sampling weight: $w_j = \frac{(N \cdot P(j))^{-\beta}}{\max_i w_i}$
11:            Compute TD-error: $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
12:            Update transition priority $p_j \leftarrow |\delta_j|$
13:            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
14:        end for
15:        Update weights: $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
16:        Copy weights into target network: $\theta_{\text{target}} \leftarrow \theta$
17:    end if
18:    Choose action $A_t \sim \pi_\theta(S_t)$
19: end for
```

Following Schaul et al. [44], the Prioritized Experience Replay Buffer is implemented through a Sum-Tree data structure to achieve $\mathcal{O}(\log N)$ complexity for both storage and sampling operations, where N is the buffer size. This design significantly reduces computational overhead compared to naive approaches. Implementation details are provided in Appendix A.

2.2.5. Deep Q-Learning from Demonstrations (DQfD)

In some scenarios, the RL agent is able to access data from previous control of the system. In such cases, the agent can leverage the "expert" experiences to massively accelerate the learning process through a technique known as *Imitation Learning* (IL). To correctly implement IL for a Deep Q-learning agent, Deep Q-learning from Demonstrations [18]

(DQfD) was proposed by DeepMind. DQfD consists in filling the agent's buffer with expert experiences and pre-train the agent by learning those experiences. After the pre-training phase, when the agent collects new experiences interacting with the environment, the Prioritized Experience Replay Buffer automatically balances expert demonstrations and new stored experiences.

More precisely, the DQfD algorithm comprises two phases: a pre-training phase in which the agent learns to imitate the expert behavior but does not interact with the environment and a training phase in which the agent actually plays on the environment.
During the pre-training phase the agent is trained to optimize a combination of four different losses: the 1-step double Q-learning loss, an n-step double Q-learning loss, a supervised large margin classification loss and a L2 regularization loss on network weights and biases.

- 1-step double Q-learning loss:

$$J_{DQ}(\theta) = r_t + \gamma \cdot Q\left(s_{t+1}, \arg\max_a \left(Q(s_{t+1}, a|\theta)\right)|\theta^-\right) - Q(s_t, a_t|\theta). \tag{2.33}$$

- n-step double Q-learning loss:

$$\begin{aligned} J_n(\theta) = \Big[& r_t + \gamma \cdot r_{t+1} + \cdots + \gamma^{n-1} \cdot r_{t+n-1} \\ & + \gamma^n \cdot Q(s_{t+n}, \arg\max_a \left(Q(s_{t+n}, a|\theta)|\theta^-)\right)\Big] - Q(s_t, a_t|\theta). \end{aligned} \tag{2.34}$$

- Supervised large margin classification loss:

$$J_E(\theta) = \max_{a \in \mathcal{A}} \left[Q(s, a|\theta) + l(a_E, a)\right] - Q(s, a_E|\theta)\,, \tag{2.35}$$

  where $l(a, a_E)$ is a margin function that is 0 when $a = a_E$ and positive otherwise.

The final loss is:

$$J(\theta) = J_{DQ}(\theta) + \lambda_1 J_n(\theta) + \lambda_2 J_E(\theta) + \lambda_3 J_{L2}(\theta)\,. \tag{2.36}$$

The supervised loss is essential because expert demonstrations typically cover only a limited subset of all possible state-action pairs. This limited coverage can result in poor Q-value estimates for unseen state-action pairs. Since the standard Q-learning update rule computes the target based on the best action for the next state, optimistic initializations of the Q values can lead to unreliable target estimates, failing the training process. Therefore,

the goal is to learn simultaneously that expert-selected state-action pairs are promising, and that actions not taken by the expert are likely mediocre.
The supervised loss forces the Q-value of the not taken actions to be at least a margin lower than the expert selected action.
The complete algorithm from the original DQfD paper is shown in Alg. 2.3.

Algorithm 2.3 Deep Q-learning from Demonstrations

```
 1: Inputs: $\mathcal{D}^{replay}$: initialized with demonstration dataset; $\theta$: behavior network weights
    (random); $\theta'$: target network weights (random); $\tau$: update frequency for target net-
    work; $k$: number of pre-training steps
 2: for steps $t \in \{1, 2, \ldots, k\}$ do
 3:     Sample mini-batch of $n$ transitions from $\mathcal{D}^{replay}$ with prioritization
 4:     Calculate loss $J(Q)$ using target network
 5:     Perform gradient descent step to update $\theta$
 6:     if $t \bmod \tau = 0$ then
 7:         $\theta' \leftarrow \theta$
 8:     end if
 9: end for
10: for steps $t \in \{1, 2, \ldots\}$ do
11:     Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
12:     Play action $a$ and observe $(s', r)$
13:     Store $(s, a, r, s')$ into $\mathcal{D}^{replay}$, overwriting oldest self-generated transition if over ca-
        pacity
14:     Sample mini-batch of $n$ transitions from $\mathcal{D}^{replay}$ with prioritization
15:     Calculate loss $J(Q)$ using target network
16:     Perform gradient descent step to update $\theta$
17:     if $t \bmod \tau = 0$ then
18:         $\theta' \leftarrow \theta$
19:     end if
20:     $s \leftarrow s'$
21: end for
```

2.3. Multi-Agent learning

Real-world environments often involve multiple autonomous decision-makers, like for instance self-driving cars sharing the road, robots collaborating in a warehouse, or software agents bidding in an online auction. Multi-Agent Learning (MAL) investigates how such

agents can effectively learn how to act together (or against each other) when they share an environment. Multi-agent learning systems can be informally defined as multiple learning agents interacting in the same shared environment, either in a competitive or cooperative setting.
Multi-Agent Reinforcement Learning (MARL) systems can be categorized according to the following attributes [40]:

- Control:
    1. Centralized: a central unit takes the decision for each agent in each time step.
    2. Decentralized: each agent takes the decision for itself.
- Observability:
    1. Partial Observability: each agent can observe only a local portion of the environment (POMDP).
    2. Fully Observability: each agent can observe the entire environment's state at each time step (MDP).
- Setting:
    1. Competitive (zero-sum games, for example).
    2. Cooperative: agents share a common goal.

Regarding the centralized versus decentralized control attribute in MARL settings, one can further categorize approaches based on whether training and/or execution are managed in a centralized or decentralized manner:

1. Centralized
    - Training: all the agents are trained together, typically under a shared objective. The key feature is that agents can access other agents information during training (observations, actions, policies,...). The latter information may not be available during inference.
    - Execution: all agents are able to share information during execution (observations, action proposals, policies,...).
2. Decentralized
    - Training: agents are trained without sharing information among each other. Each agent can only access its own local information.

- Execution: agents execute actions by accessing only their local information.

Within a cooperative MARL setting (as the one studied in the thesis), the approaches can be classified into five categories [40]:

- Independent Learners [36].
- Fully observable critic [21, 30].
- Value function factorization [48].
- Consensus [22].
- Learn to communicate [12].

We remark that in the Grid2Op simulation environment (the environment used in this work) only one agent can act at a given time step, therefore, most of the techniques used in MARL settings are useless as their goal is making the agents learning how to act simultaneously.
Nevertheless, some interesting insights of Independent Learners approaches can be useful.

2.3.1. Independent Learners

In Independent Learners approaches, each agent has at its disposal only its local observation and reward. Other agents are part of the environment. For a single agent the environment is a POMDP.
The problem of this approach is the non-stationarity (same observations and same actions sequences may results in very different cumulative reward). Therefore, the problem that Independent Learners approaches aim to solve is the learning in a highly non stationary scenario.
In general, learning in a POMDP is a complex task. The main solutions that have been proposed in literature to solve this problem are all based on the same idea: try to infer the non-observable part of the environment. Usually, to do this, recurrent mechanisms are introduced in the RL model; the agent can observe the evolution over time of its local observation and try to estimate the entire environment's state [17, 39].
However, the latter approach introduces additional and non-negligible model complexity that must be considered carefully.
In some specific type of environments, such as Grid2Op [9] and Traffic Signal Control (SUMO) [28], the specific structure of the environment (graphs in these cases) allows exploiting structural information to reduce the partial observability of a local agent. The idea is to enrich the informativeness of a single observation by including structural en-

vironmental information into the local observation. To exploit the graph environmental structure, several competitive solutions based on Graph Neural Networks (GNNs) have been proposed, both for Grid2Op [7, 16] and for SUMO [63].

2.4. Graph Neural Networks

2.4.1. Graph definition and notation

A graph is a pair (V, E) where V is the set of nodes and $E = \{(i, j) \mid i, j \in V; i \neq j\}$ is the set of edges.
The set of neighbors of a node i will be denoted by $N(i) = \{j \in V \mid (j, i) \in E\}$.
The connection structure of the graph is described by the adjacency matrix A, a $|V| \times |V|$ matrix where:

$$a_{i,j} = \begin{cases} 1 & \text{if node } i \text{ and node } j \text{ are directly connected,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.37}$$

In general, both nodes and edges can be associated to feature vectors. For the purpose of the thesis, only nodes will be associated to feature vectors.
Let $x_i \in \mathbb{R}^d$ be the feature vector of node i and let $\mathbf{x} \in \mathbb{R}^{|V| \times d}$ be the node features matrix of the graph, where row i-th represents the feature vector of node i. Thus, the complete graph, with features and edges, can be represented by a tuple $(\mathbf{x}, A)$. Graphs can be classified in many classes depending on several characteristics. The classes of our interest are:

- Directed Graphs: E contains ordered pairs of nodes,
- Undirected Graphs: the edges in E are unordered pairs, meaning that the order of the vertices in each edge is irrelevant,

and

- Static Graphs: V, E and $\mathbf{x}$ do not change over time,
- Dynamic Graphs: at least one among V, E and $\mathbf{x}$ changes over time. For example, if V changes, it means that nodes are created and deleted over time,

as well as

- Homogeneous Graphs: all nodes represent the same "entity", for example, all the nodes represent users of a social network,

- Heterogeneous Graphs: nodes represent different "entities", for example, some nodes represent users and others represent items.

The thesis focuses on Undirected Dynamic Homogeneous graphs.

2.4.2. Neural Message Passing

The basic operation of most Graph Neural Networks is the Neural Message Passing [13]. The message passing operation is repeated T times and aggregates the information from neighboring nodes into each individual node:

$$\begin{aligned} m_v^{t+1} &= \sum_{w \in N(v)} M_t\big(h_v^t, h_w^t, e_{vw}\big) , \\ h_v^{t+1} &= U_t\big(h_v^t, m_v^{t+1}\big) , \end{aligned} \tag{2.38}$$

where, h_j^t and e_{ij} represent, respectively, the embedding of node j and the embedding of edge (i, j) at time t, while $U_t(\cdot)$ and $M_t(\cdot)$ are learnable differentiable functions.
Different choices for the functions U_t and M_t yield distinct Graph Neural Network models.

2.4.3. Convolutional Graph Neural Network

Convolutional Graph Neural Networks (GCN) [24] perform the message passing by considering :

$$H^{t+1} = \sigma\big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^tW^t\big) , \tag{2.39}$$

where $\tilde{A} = A + I_{|V|}$ is the adjacency matrix with added self-connection, $I_{|V|}$ being the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ represents the number of connections of node i, and H^t and W^t are, respectively, the embedding of nodes and the layer-specific learnable parameter matrix at message passing step t ($H^0 = \mathbf{x}$). The activation function σ is any differentiable function, usually a ReLu($\cdot$)=max($0, \cdot$) is used.

2.4.4. Graph Attention Neural Network

Inspired by the advancement of attention mechanisms in deep learning [2, 56], Graph Attention Networks (GATs) [57] incorporate a self-attention mechanism that allows each node to assign different importance weights, referred to as attention coefficients, to its neighbors. This enables the model to compute a "relevance score" for each neighbor, ensuring that messages are aggregated in a non-uniform manner during the message passing process.

In particular, the attention coefficients e_{ij}^t at message passing step t are computed as follows [57]:

$$e_{ij}^t = a\big(W^t \cdot h_i^t, W^t \cdot h_j^t | \theta^t\big)\,, \tag{2.40}$$

where $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is an attention function (single layer feed-forward neural network with learnable parameters θ^t), W^t is the matrix of learnable parameters at step t, while h_i^t and h_j^t are, respectively, the embeddings of node i and node j at time step t. Note that the attention coefficients are computed only for the neighbors of a node, including the node itself.
Coefficients get normalized node-wise by a softmax function:

$$\alpha_{ij}^t = \text{softmax}_j\left(e_{ij}^t\right) = \frac{\exp\left(e_{ij}^t\right)}{\sum\limits_{k \in N(i) \cup i} \exp\left(e_{ik}^t\right)}\,. \tag{2.41}$$

More precisely, given the actual formulation of function $a(\cdot)$, the complete procedure to compute the attention coefficients of a single layer can be expressed as:

$$\alpha_{ij}^t = \frac{\exp\Big(\text{LeakyReLU}(\theta^{t^T} \cdot [W^t h_i^t || W^t h_j^t]\Big)}{\sum\limits_{k \in N(i) \cup i} \exp\Big(\text{LeakyReLU}(\theta^{t^T} \cdot [W^t h_i^t || W^t h_k^t]\Big)}\,, \tag{2.42}$$

where the operation T represents the transpose. The update of a node embedding is then computed as:

$$h_i^{t+1} = \sigma\left(\sum_{j \in N(i) \cup i} \alpha_{ij}^t \, W^t h_j^t\right). \tag{2.43}$$

In order to stabilize the learning, K independent attention mechanisms are performed, and the results are averaged:

$$h_i^{t+1} = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N(i) \cup i} \alpha_{ij}^{t,k} \, W^{t,k} \, h_j^t\right). \tag{2.44}$$

2.4.5. Graph Attention Neural Network v2

As suggested by [6], standard Graph Attention Networks lack in expressiveness. This work shows that basic GAT computes a type of attention that is called "static attention" and is fundamentally different from the famous "dynamic attention" mechanism of the original paper [2]. More specifically, it is shown that the ranking (i.e., `argsort`) of attention coefficients is identical across all nodes in the graph, which limits the expressive power of the GAT model. In other words, the model assigns the same ordering of importance to the neighbors of every node. The main problem arises from the definition of the function $a(\cdot)$ that is:

$$e_{ij}^t = \text{LeakyReLU}\left(\theta^{t^T} \cdot [W^t h_i^t || W^t h_j^t]\right). \tag{2.45}$$

The transformations θ^t and W are applied consecutively, thus the entire transformation can be collapsed into a single linear operation. To solve this problem it is sufficient to change the order of the operations, turning eq. (2.45) into:

$$e_{ij}^t = \theta^{t^T} \text{LeakyReLU}\left(W^t [h_i^t || h_j^t]\right). \tag{2.46}$$

2.5. Sequential Data Learning

Many real-world data sources, such as time series, natural language, and control signals in dynamical systems, are inherently sequential. Learning from sequential data requires models that can effectively capture temporal patterns and account for the ordering of events. To this end, sequential data learning models must incorporate some sort of memory mechanisms: a hidden state that summarizes the relevant information seen so far. There exists different types of Sequential Data Problems depending on the nature of the input and the output, as shown in Fig. 2.4.

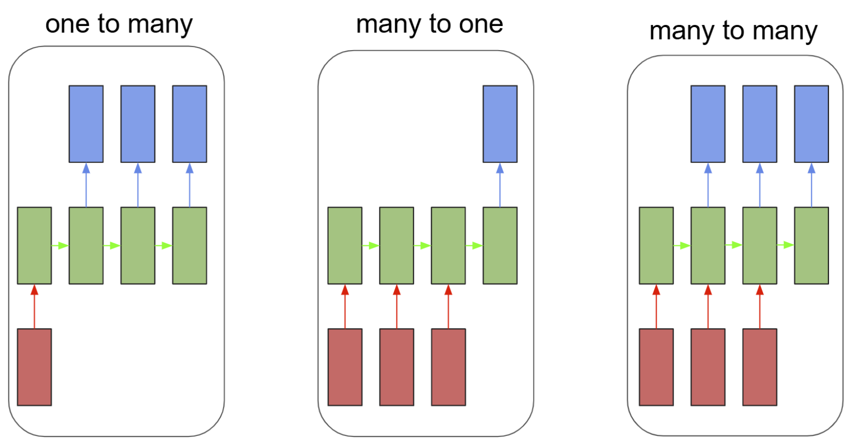


Figure 2.4: Types of Sequential Data Problems. The cells in red represent the input, those in blue the output, and the cells in green the memory components or hidden states.

Traditional models like Recurrent Neural Networks (RNNs) [43], Long Short-Term Memory (LSTM) [19] networks, and Gated Recurrent Units (GRUs) have been widely used in this domain due to their recursive structures, which process one element of the sequence at a time [14]. However, their sequential nature makes them computationally expensive and often limited in modeling long-range dependencies due to vanishing gradients.

2.5.1. Transformer architecture

To massively speed-up the training process and to better capture long-range dependencies in sequences, the Transformer architecture was introduced by Vaswani et al. in the paper "Attention is All You Need" [56]. The classic and widely adopted Transformer architecture is the one introduced in the original paper and depicted in Fig. 2.5.

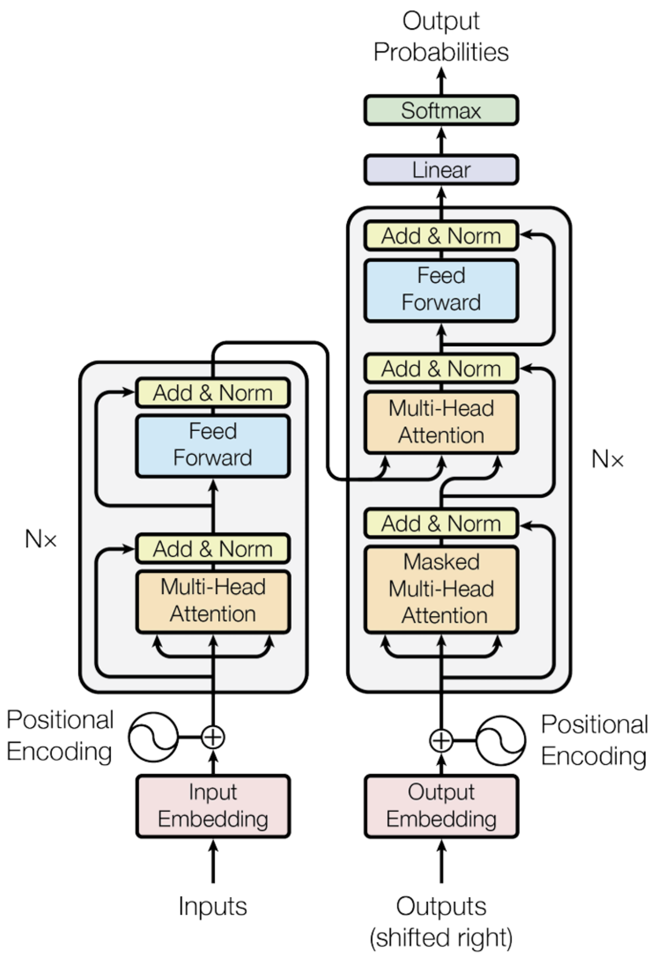


Figure 2.5: Original Transformer architecture. Image taken from [56].

Unlike recurrent models, Transformers leverage attention mechanisms to process the en-

tire input sequence in parallel during training, thereby removing the need for recurrence and enabling significantly faster training. Attention mechanism is a function that maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The elements of the output are computed as a weighted sum of the elements of the value vector, whose weights come from a "compatibility" function between the query and key vectors.
The attention mechanism used in the original paper is called *"Scaled Dot Product Attention"*. The keys matrix K and the queries matrix Q have dimension $n \times d_k$ while the values matrix V has dimension $n \times d_v$. The operation returns an output vector of shape $n \times d_v$ and is defined as:

$$Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V\,. \tag{2.47}$$

The *"Multi-Head Attention"* operation is defined as:

$$\begin{gathered} MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h) \cdot W^O \\ \text{where } head_i = Attention(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \end{gathered} \tag{2.48}$$

this operation consists of applying the standard attention mechanism multiple times in parallel, each with its own set of learned projection weights, to increase the learning capabilities.

The Transformer architecture introduced by [56] implements *Multi-Head Attention* by using the same input matrix for the queries (Q), keys (K), and values (V), resulting in what is known as the *Self-Attention* mechanism. It is worth noting that the self-attention mechanism is inherently permutation equivariant, meaning that permuting the input sequence results in the output being permuted in the same way. This property makes the vanilla Transformer architecture unsuitable for modeling sequential data on its own. To overcome this limitation, a mechanism known as Positional Encoding (PE) is applied to the input sequence: a matrix encoding positional information is added to the input sequence. There exists several PE strategies, the original Transformer architecture [56] employs a fixed, sinusoidal positional encoding defined as:

$$\mathrm{PE}(pos, , 2\ell) = \sin\left(\frac{pos}{10000^{\frac{2\ell}{d_\mathrm{v}}}}\right), \quad \mathrm{PE}(pos, 2\ell + 1) = \cos\left(\frac{pos}{10000^{\frac{2\ell}{d_\mathrm{v}}}}\right), \tag{2.49}$$

where:

- *pos* is the position in the sequence,
- ℓ is the dimension index,
- d_{v} is the dimensionality of the embedding vector.

The remaining components of the Transformer architecture (the Feed-Forward layers, Add & Norm operations, Linear projections, and Softmax) are standard modules commonly used in deep learning models and are not discussed in detail here.

Transformers have become the state-of-the-art model for sequential data in many domains, including natural language processing and time series forecasting.

In this work, Transformer-based components are considered in the context of learning an embedding representation of a time series.

2.6. Clustering

Clustering is the task of partitioning a set of data points into natural groups called clusters, based on some notion of similarity or distance [65]. It is desirable to have data points within the same cluster to be more similar to each other than to those in different clusters. Based on the type of data considered and the desired cluster characteristics, there are several cluster paradigms: representative-based, hierarchical and density-based. The thesis focus is on hierarchical clustering and density-based (DBSCAN) methods. Hierarchical clustering methods suffer from an high computational complexity ($\mathcal{O}(n^2 \log n)$), however, the specific task consists of clustering few data points, thus, the computational complexity is not a limit. Density-based methods such as DBSCAN are not limited to convex or spherical cluster shapes. This makes them particularly suitable for latent embedding spaces, which likely presents irregular structures of data points.

2.6.1. Hierarchical clustering

Hierarchical clustering methods can be classified into two main types: agglomerative and divisive. Agglomerative clustering begins with each data point in its own cluster and iteratively merges the closest clusters until a single, overarching cluster is formed. In contrast, divisive clustering starts with all data points grouped into a single cluster and progressively splits them into smaller clusters until each point resides in its own individual cluster. The thesis will focus on hierarchical agglomerative clustering. The basic pseudocode from [65] of an agglomerative hierarchical clustering method is provided in Alg. 2.4.

Algorithm 2.4 Agglomerative Hierarchical Clustering

```
1: **Input:** Dataset $\mathcal{D}$ with $n$ elements, target number of clusters $k$
2: Initialize clusters $\mathcal{C} \leftarrow \{C_i = \{\mathbf{x}_i\} \mid \mathbf{x}_i \in \mathcal{D}\}$ {Each point in a separate cluster}
3: Compute initial distance matrix $\Delta \leftarrow \{\|\mathbf{x}_i - \mathbf{x}_j\| \mid \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}\}$
4: **repeat**
5:     Find closest pair of clusters $C_i, C_j \in \mathcal{C}$
6:     Merge clusters: $C_{ij} \leftarrow C_i \cup C_j$
7:     Update clusters: $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{C_i, C_j\}) \cup \{C_{ij}\}$
8:     Update distance matrix $\Delta$ to reflect new clustering
9: **until** $|\mathcal{C}| = k$
```

Agglomerative Hierarchical clustering methods are distinguished by the specific definition of distance they use to measure the similarity between clusters, during the merging process. The most common similarity measures among clusters are illustrated in Fig. 2.6.

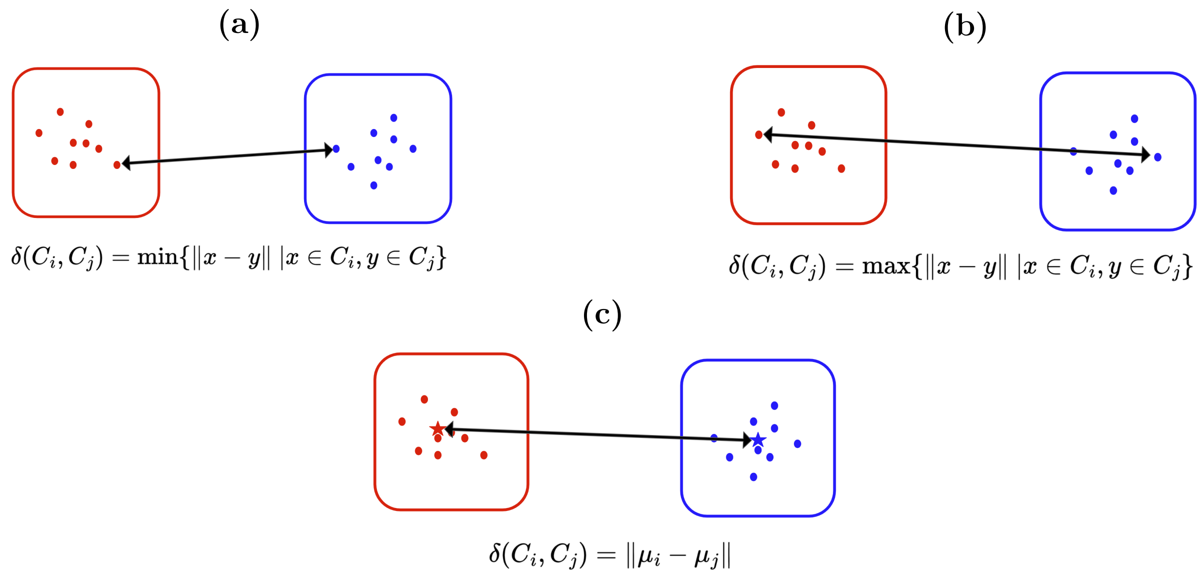

(a)
(b)
$\delta(C_i, C_j) = \min\{\|x - y\| \,|x \in C_i, y \in C_j\}$
$\delta(C_i, C_j) = \max\{\|x - y\| \,|x \in C_i, y \in C_j\}$
(c)
$\delta(C_i, C_j) = \|\mu_i - \mu_j\|$

Figure 2.6: Different notions of similarity between clusters: (a) single linkage, (b) complete linkage, and (c) centroid linkage, μ_i, μ_j represent respectively the centroids of clusters C_i, C_j.

Additionally, there is flexibility in the choice of distance metrics between data points. For numerical attributes, the most commonly used measures are Euclidean and Manhattan distances.

Hierarchical clustering takes $\mathcal{O}(n^2 \log n)$, where n is the number of data points to cluster, making it one of the most computationally intensive clustering algorithms. Nevertheless, when the dataset size is small, hierarchical clustering is often chosen as a first approach due to its interpretability and effectiveness.

2.6.2. DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is another clustering algorithm that employs the local density of data points to identify the clusters. DBSCAN does not require to specify in advance the number of clusters to be determined, and it automatically provides the outliers. The pseudocode from [65] is sketched in Alg. 2.5.

Algorithm 2.5 DBSCAN Clustering

```
1: **Input:** Dataset $\mathcal{D}$, radius $\epsilon$, minimum points $minpts$
2: **Output:** Clusters, Core points, Border points, Noise
3: Initialize $Core \leftarrow \emptyset$
4: for all $\mathbf{x}_i \in \mathcal{D}$ do
5:    Compute neighborhood $N_\epsilon(\mathbf{x}_i)$
6:    $id(\mathbf{x}_i) \leftarrow \emptyset$
7:    if $|N_\epsilon(\mathbf{x}_i)| \geq minpts$ then
8:       $Core \leftarrow Core \cup \{\mathbf{x}_i\}$
9:    end if
10: end for
11: $k \leftarrow 0$
12: for all $\mathbf{x}_i \in Core$ such that $id(\mathbf{x}_i) = \emptyset$ do
13:    $k \leftarrow k + 1$
14:    $id(\mathbf{x}_i) \leftarrow k$
15:    DensityConnected($\mathbf{x}_i$, $k$)
16: end for
17: $C_i \leftarrow \{\mathbf{x} \in \mathcal{D} \mid id(\mathbf{x}) = i\}$, for $i = 1, \ldots, k$
18: $Noise \leftarrow \{\mathbf{x} \in \mathcal{D} \mid id(\mathbf{x}) = \emptyset\}$
19: $Border \leftarrow \mathcal{D} \setminus (Core \cup Noise)$
20: return  Clusters $\{C_i\}_{i=1}^k$, Core, Border, Noise
21: function DensityConnected($\mathbf{x}, k$)
22: for all $\mathbf{y} \in N_\epsilon(\mathbf{x})$ do
23:    if $id(\mathbf{y}) = \emptyset$ then
24:       $id(\mathbf{y}) \leftarrow k$
25:       if $\mathbf{y} \in Core$ then
26:          DensityConnected($\mathbf{y}, k$)
27:       end if
28:    end if
29: end for
```

The goal of DBSCAN is to identify groups of densely packed and connected data points. This allows the algorithm to discover clusters of arbitrary shape, while also detecting outliers—points that do not belong to any cluster. The two input parameters are:

- ϵ : the maximum distance within which two points are considered directly connected.
- $minpts$: the minimum number of points that must be directly connected to a given

point for it to be considered a core point, capable of initiating a cluster. Clusters that are directly connected through core points are then merged into a single, larger cluster.

2.6.3. Clustering Validation

Two main types of measures are commonly used to evaluate clustering quality: internal or external validation measures. Internal measures assess the structure quality of the resulting clusters using criteria such as intra-cluster cohesion and inter-cluster separation. External measures, on the other hand, require ground-truth labels and evaluate how well the clustering results align with known class assignments provided by expert knowledge. This thesis focuses on external validation measures. Among the various existing approaches, the two most widely used categories are: entropy-based measures, such as Normalized Mutual Information, and pairwise measures, such as the Fowlkes-Mallows index, [65].

Entropy-based Measures & Normalized Mutual Information.

Entropy-based measures try to evaluate how much information the predicted clustering provides about the ground truth labels, and vice versa. Entropy-based measures are all based on the notion of Entropy and conditional Entropy. Defining the clustering C and the ground truth $\mathcal{T}$, as a label assignments to the data points, one uses:

- n as the total number of points,
- n_i as the number of points assigned to cluster C_i,
- m_j as the number of points assigned to ground truth partition $\mathcal{T}_j$,
- n_{ij} as the number of points simultaneously assigned to cluster C_i and to ground truth partition $\mathcal{T}_j$,
- $Pr(C_i) = \frac{n_i}{n}$,
- $Pr(\mathcal{T}_j) = \frac{m_j}{n}$.

The entropies $H(C)$ and $H(\mathcal{T})$ are defined as:

$$\begin{aligned} H(C) &= -\sum_{i=1} Pr(C_i) \cdot \log_2 \left(Pr(C_i)\right), \\ H(\mathcal{T}) &= -\sum_{i=1} Pr(\mathcal{T}_i) \cdot \log_2 \left(Pr(\mathcal{T}_i)\right). \end{aligned} \tag{2.50}$$

The conditional entropy $H(\mathcal{T}|C)$ is defined as:

$$H(\mathcal{T}|C) = \sum_{i=1} Pr(C_i) \cdot H(\mathcal{T}|C_i)\,, \tag{2.51}$$

where:

$$H(\mathcal{T}|C_i) = -\sum_{j=1} \left(\frac{n_{ij}}{n_i}\right) \cdot \log_2 \left(\frac{n_{ij}}{n_i}\right). \tag{2.52}$$

Finally, the Normalized Mutual Information score between a clustering C and a ground truth of labels $\mathcal{T}$, is defined as:

$$NMI(C,\mathcal{T}) = \frac{I(C,\mathcal{T})}{\sqrt{H(C)\cdot H(\mathcal{T})}} = \frac{H(\mathcal{T}) - H(\mathcal{T}|C)}{\sqrt{H(C)\cdot H(\mathcal{T})}} = \frac{H(\mathcal{C}) - H(C|\mathcal{T})}{\sqrt{H(C)\cdot H(\mathcal{T})}}\,. \tag{2.53}$$

The latter is a measure in the range [0,1] where 1 means that the clustering perfectly matches the ground truth.

Pairwise Measures & Fowlkes-Mallows score

Given a data point x_i, then y_i denotes the true partition label while $\hat{y}_i$ denotes the predicted clustering label for point x_i. If two points x_i and x_j belong to the same cluster, then $\hat{y}_i = \hat{y}_j$. The basic elements of pairwise measures are defined as:

- $TP = \left|\{(x_i, x_j) : y_i = y_j \wedge \hat{y}_i = \hat{y}_j\}\right|$
- $FN = \left|\{(x_i, x_j) : y_i = y_j \wedge \hat{y}_i \neq \hat{y}_j\}\right|$
- $FP = \left|\{(x_i, x_j) : y_i \neq y_j \wedge \hat{y}_i = \hat{y}_j\}\right|$
- $TN = \left|\{(x_i, x_j) : y_i \neq y_j \wedge \hat{y}_i \neq \hat{y}_j\}\right|.$

The Fowlkes-Mallows score is defined as:

$$FM = \sqrt{\left(\frac{TP}{TP+FP}\right)\cdot\left(\frac{TP}{TP+FN}\right)} \tag{2.54}$$

and it is desirable to have a value close to 1.

3 | Problem Description

As mentioned in Chapter 1, the goal of this thesis is to develop a scalable and distributed reinforcement learning architecture for power grid management, with a focus on topology-based control actions. The standard framework to test and develop such architectures is Grid2Op. The Grid2Op framework realistically simulates power grid operations and supports many types of interventions that can be performed on a power grid (the specific details will be given in Section 3.1). However, topology-based actions are undoubtedly the most interesting and extensively studied, as they have no operational cost and are highly complex due to their non-local effects and combinatorial nature, making them particularly challenging for humans to execute effectively. Notably, they have received significant attention from RTE, the organizers of the challenges, who have prioritized topology reconfigurations in their recent works [3, 33, 34].
This chapter provides a formal and detailed description of the problem, including an overview of the Grid2Op simulator and an outline of the main challenges posed by power grid control via topology-based actions, which motivate and shape the research direction explored in this thesis.

3.1. Problem Formulation

Grid2op framework models a power network as a graph of connected elements, an example is provided in Fig. 3.1.

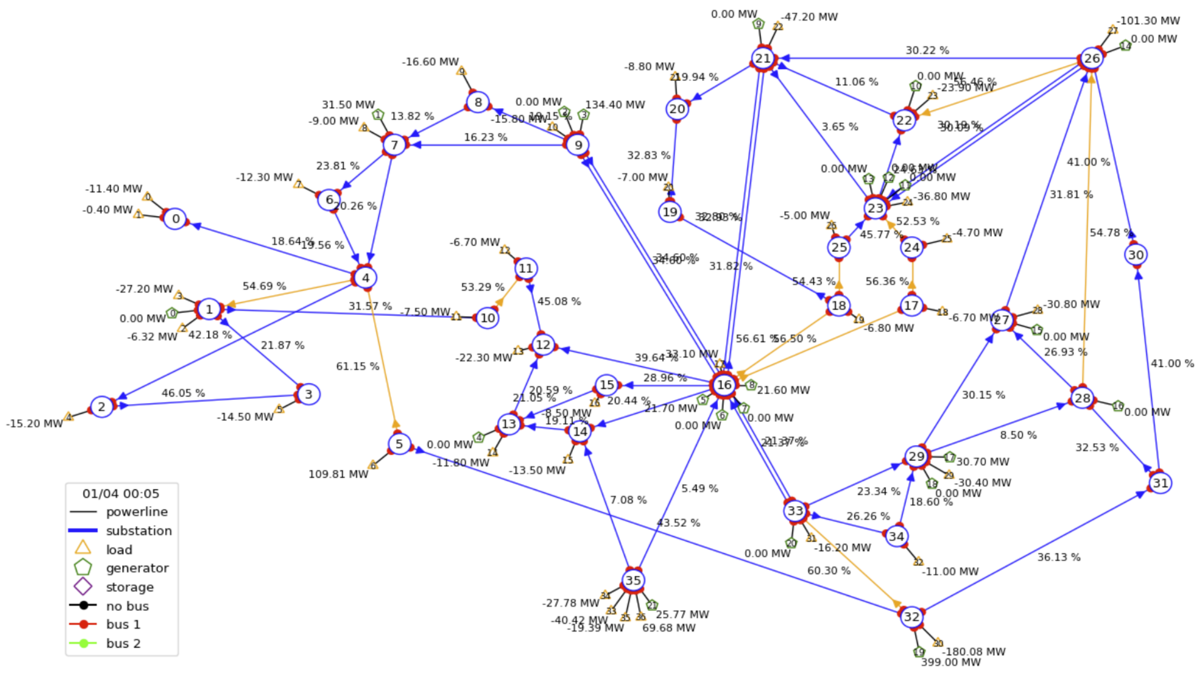


Figure 3.1: Example grid: "l2rpn_icaps_2021_small".

The nodes represent substations and the edges represent powerlines. Each substation has two buses to which its elements can be connected, meaning that the substation manages two circuits. In other words, substations can be seen as routers in the power network. Other elements of the power grid are: generators (renewable and non renewable) and loads (storages are not considered in this work). Generators produce electricity while loads consume it. A single substation manages its topology by controlling to which bus its elements are connected.

The power grid is simulated for a given period, several days at 5-minutes intervals. At each timestep, the agent is called to take an action on the simulator, leading to the next simulator's state. The simulation runs for at most T timesteps, but it ends prematurely in case of a blackout. A blackout can happen if the energy demand (loads request) is not satisfied by the current configuration. The main cause of a blackout is the so called "line overload": when the current flow of a specific powerline crosses a given threshold, the environment automatically disconnects the powerline, potentially leading to a grid configuration that violates loads' requirements. Even a single disconnection is a dangerous condition for a power grid that must be avoided to keep the system safe.

Due to system stability requirements from real-world power grids, control actions are typically restricted to a single substation at each time step. Simultaneous interventions on multiple substations can introduce unpredictable interactions and lead to highly unstable grid conditions. Therefore, the simulator allows to act on a single substation at a time.

The power grid management problem can be formalized as a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mu, \gamma)$.

State Space

The state space $\mathcal{S}$ is the set of possible configurations of the power grid. The state's features provided by Grid2op simulator includes :

- Loads' features:

Feature	**Description**
`load_p`	Load active power (MW)
`load_q`	Load reactive power (MVar)
`load_v`	Voltage magnitude of the bus to which each load is connected (kV)
`load_to_subid`	Id of the substation to which the load is connected

- Renewable generators' features:

Feature	**Description**
`gen_p`	Active power production (MW)
`gen_q`	Reactive power production (MVar)
`gen_v`	Voltage magnitude of the bus the generator is connected to (kV)
`gen_pmin`	Minimum active power for stable generator operation (MW)
`gen_pmax`	Maximum active power output of the generator (MW)
`gen_p_before_curtail`	Active Power value before curtailment (MW)
`curtailment_limit`	Agent-imposed curtailment limit (as a ratio of `gen_pmax`)
`curtailment_limit_effective`	Effective curtailment ratio applied by the agent
`curtailment_mw`	Amount of power curtailed (MW)
`gen_to_subid`	Id of the substation to which the generator is connected

- Non-Renewable generators' features:

Feature	Description
gen_p	Active production value of (MW)
gen_q	Reactive production value
gen_v	Voltage magnitude of the bus to which each generator is connected (kV)
gen_pmin	Minimum active power production needed for a generator to work properly
gen_pmax	Maximum active power production needed for a generator to work properly
gen_max_ramp_up	Maximum active power variation possible between two consecutive timestep
gen_max_ramp_down	Minimum active power variation possible between two consecutive timestep
gen_min_uptime	Minimum time (number of timesteps) a generator needs to be turned on
gen_min_downtime	Minimum time (number of timesteps) a generator needs to be turned off
gen_cost_per_MW	"Operating cost", e.g., the cost in terms of "used currency" for the production
gen_startup_cost	Cost to start a generator
gen_shutdown_cost	Cost to shut down a generator
target_dispatch	Target redispatching imposed by the agent
actual_dispatch	Actual redispatching that is currently being implemented by the environment
gen_margin_up	How much generators production can be increased within the next timestep
gen_margin_down	How much generators production can be decreased within the next timestep
gen_to_subid	Id of the substation to which the generator is connected

- Lines' features:

Feature	Description
rho	Capacity of the powerline: current flow divided by its thermal limit
line_status	Status of the powerline (connected/disconnected)
p_or and p_ex	Active power flow at the origin or extremity side of the powerline
q_or and q_ex	Reactive power flow at the origin or extremity side of the powerline
v_or and v_ex	Voltage magnitude at the bus to which the origin or extremity side of the powerline is connected
a_or and a_ex	Current flow at the origin or extremity side of the powerline (A)
sub_id_or and sub_id_ex	Id of the substation to which the origin or extremity side of the powerline is connected

- Bus' features:

Feature	Description
p	Total active production
q	Total reactive production
v	Total voltage

- Topology feature: `topo_vect`, which, for each object (load, generator, ends of a powerline), gives on which bus this object is connected to, within its substation.

Action Space

The types of interventions supported by the simulator include redispatching, curtailment, and topology-based actions. Redispatching and curtailment apply to generators and allow for direct control over energy production. Specifically, redispatching enables adjusting the setpoint of non-renewable generators, either increasing or decreasing their output. Curtailment, on the other hand, applies to renewable generators and can only reduce their production, as increasing output is not feasible due to their dependence on uncontrollable environmental conditions.

Topology-based actions, instead, allow for modifying the internal connectivity within substations. To clarify this concept, a simple example is provided. Consider a substation at time $t = 0$, with its configuration shown in Fig. 3.2.

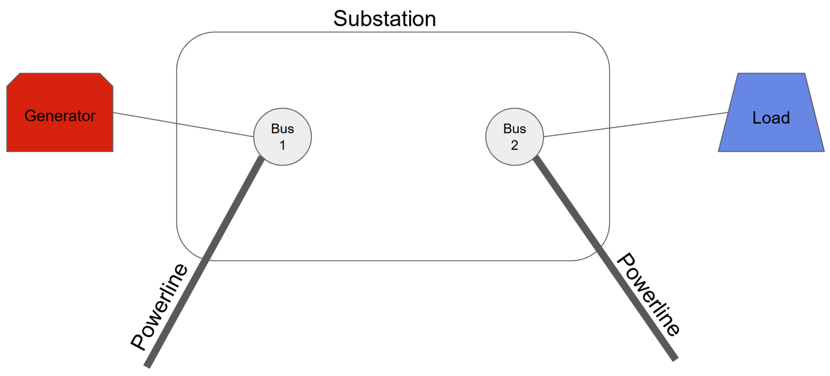


Figure 3.2: Example substation $t = 0$.

Suppose that a topological action is applied that switches the powerline on the left to the other bus. The resulting configuration at time $t = 1$ is shown in Fig. 3.3.

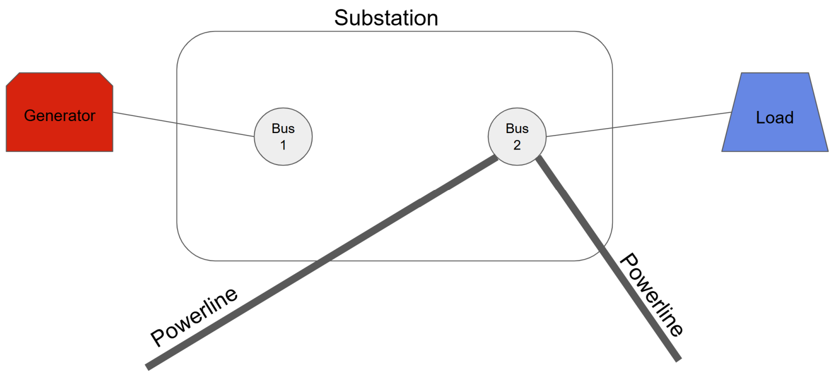


Figure 3.3: Example substation $t = 1$.

As anticipated in Chapters 1 and 3, the focus of the work will be on topology-based actions. When focusing solely on topology actions, the simulator defines two types of topological action spaces: “set-like” and “change-like.” A set-like action assigns a specific configuration to a substation regardless of its current state. In contrast, a change-like action modifies the current configuration by switching the bus assignment of the target component. For example, if a generator is connected to bus 1, applying a change-like action will switch it to bus 2; applying the same action again will revert it back to bus 1. "Set-like" actions are preferred from an RL perspective since the behavior of "change-like" actions strongly depends on the current configuration making the dynamics of the MDP more complex.
In principle, a substation i with N_i elements counts 2^{N_i} possible topology actions (2 buses), thus a grid with S substations counts $\sum_{i \in S} 2^{N_i}$ possible actions. In practice, considering a substation with N_i elements, many of the 2^{N_i} configurations are not "valid" (e.g., a configuration with a load not connected to a powerline), so the number of available actions is a bit smaller than 2^{N_i}.

Reward

The Grid2Op simulator supports many different rewards depending on the specific task of interest. The one used in L2RPN challenges is defined as the sum of squared margin of the powerlines [1], namely:

$$r_t = \sum_{l \in \text{Lines}} \Big(\max\left(1 - rho_l^t\,,\, 0\right) \Big)^2 . \tag{3.1}$$

Therefore, the cumulative reward over an episode lasting T timesteps is defined as:

$$R_T = \sum_{t=0}^{T} r_t\,. \tag{3.2}$$

Nevertheless, the main performance metric used to assess the model’s performance is the *survive time*, namely, the number of time steps the agent managed to control the grid without blackouts.

[1]The quantity rho represents the percentage of capacity utilization. While it can exceed 1, it typically remains below this threshold under normal operating conditions. If rho exceeds 1 for a given power line, that line will be disconnected in the subsequent time step.

3.2. Challenges and research direction

Topology-based grid control presents several key challenges: **the combinatorial nature of the action space**; **the increasing complexity of the state representation**; and **the computational requirements of simulation-based methods** as the size of the power grid increases.

The topological action space, which includes configuration changes, grows with the number of controllable components.

Similarly, as the size of the grid increases, the state representation becomes hard to be learned effectively due to the grid's graph structure: the complexity of the network's topology, and the number of interconnections and relations among elements grows with the size of the grid. Therefore, given a very large power grid, traditional learning models struggles to learn interesting structural patterns in such environment. Moreover, a learning algorithm that effectively captures the interconnection relationships among grid components can learn more efficiently by focusing only on relevant parts of the system. For example, when making a decision about a specific element in a large power grid, the state of distant and possibly unconnected components may have little to no impact. By understanding the grid's topology, the algorithm can ignore irrelevant information and focus only on the local context, leading to faster and more accurate learning.

Furthermore, modern state-of-the-art systems for grid management are simulation-based [10, 33]. However, simulating a power grid involves solving a large and complex system of equations that grows with the size of the power grid itself. Typically, that system of equations is solved through iterative numerical methods that yield only approximate solutions. In very large power grids, convergence to a solution could not be feasible. To cope with this problem, the current research trend is to decompose a large grid into independent subgrids, and to simulate only the subgrid of interest. A possible way to identify the independent subgrids is by means of pseudo-block diagonalization of an adjacency matrix obtained through the computation of the pairwise mutual information between states and actions, as studied in [29]. However, this method is static, as it defines fixed subgrids, even though the grid's topology, and therefore the actual independent subgrids, may change over time.

These challenges point toward a research direction focused on developing topology-based controllers designed for very large power grids. Such controllers should: operate over distributed action and observation spaces, exploit the grid's graph structure for richer state representations, and avoid the dependence on computationally intensive long-term simulations. Additionally, a complementary research direction involves exploring methods for dynamically decomposing large power grids based on their evolving topology, thereby enabling more flexible factorizations.

The ultimate goal of this work is to assess whether a fully distributed approach—both in the action and observation spaces—is feasible and competitive. We also aim to investigate whether graph-learning modules can construct sufficiently informative local representations to mitigate the partial observability caused by decomposing the observation space. Ideally, this approach should lead to a reduction in inference time compared to expert simulation-based methods while maintaining strong performance. Furthermore, this thesis wants to propose effective solutions to standard challenges encountered in distributed approaches within power grids, and to pave the way for more scalable extensions of the proposed framework.

4 | Related Work

This chapter offers a comprehensive overview of the principal research efforts conducted within the domain of power grid management using the Grid2Op simulation environment [9].

A brief introduction to the L2RPN challenges will be provided for completeness, followed by a discussion of the commonly adopted strategies that have gained wide acceptance within the research community and are frequently employed in competitive solutions.

Subsequently, the focus will shift to research directions that align with the objectives of this thesis, with particular emphasis on the application of Graph Neural Networks and distributed multi-layer architectures to the problem of power grid control. In addition, a brief analysis of factorization methods for power grids will be included.

4.1. L2RPN Challenges & Common Strategies

The L2RPN challenge series, organized by RTE (Réseau de Transport d'Électricité), was established to promote the development of innovative solutions for power grid management. Held annually from 2019 to 2023, all the editions, except the first one, were built upon the Grid2Op simulation environment, which provides a realistic framework for modeling and controlling power transmission networks.

Within the scope of the challenges, certain strategies emerged as standard practices, since they have been adopted by all or most of the winning solutions and widely recognized as effective approaches to power grid control. In particular, the use of a rule-based top-level decision logic – responsible for determining when the RL agent should intervene – and action space reduction techniques to manage the combinatorial complexity of the action space have proven to be essential components in successful approaches to power grid management.

The rule-based top-level logic comprises two steps: make the agent act only when it is necessary and reconnect powerlines as soon as possible in case of a disconnection.

The first fundamental step, adopted by all winning solutions, is to have the agent intervene only in emergency situations. Given the complexity and instability of power grids, taking no action is often the most effective strategy under normal operating conditions. Consequently, all competitive solutions are designed so that the agent takes actions on the power grid only when a line is almost overloaded, that is, when its capacity (`rho`) exceeds a predefined threshold. This threshold, which typically lies within the range $[0.9, 1]$, determines how frequently the agent intervenes, ensuring that actions are taken only in critical situations.

The second fundamental step, firstly introduced by [64] and then adopted by all the following winning solutions, consists in reconnecting a powerline as soon as possible in case of a disconnection.

In summary, the rule-based top-level logic for managing the agent intervention is defined in Alg. 4.1.

Algorithm 4.1 Top-Level rule-based logic

```
1:  if No line disconnected and no line overloaded then
2:      {Safe situation}
3:      Do nothing
4:  else
5:      if There is a disconnected line then
6:          {Critical situation}
7:          Reconnect the line
8:      else
9:          {Danger situation}
10:         Let the agent plays
11:     end if
12: end if
```

With regard to action space reduction, most winning solutions, along with several methods proposed beyond the challenges, employ a range of strategies that share the common goal of simplifying the action space to effectively manage its combinatorial complexity. A comprehensive survey of action space reduction techniques for power grid topology control is provided in [53]. A fundamental step, common to all solutions, is the elimination of actions that lead to invalid grid configurations (e.g., isolating a load). On top of that, many other procedures – both domain-specific and domain-agnostic – have been proposed to further reduce the action space. In general, several works have highlighted that training a competitive algorithm without incorporating action reduction techniques is notably difficult and requires substantial computational time [25, 53].

4.2. Graph Neural Networks & Power Grids

Given the inherent graph structure of power grids, Graph Neural Networks (GNNs) represent a natural and reasonable choice for learning meaningful representations from such data. Therefore, Graph Reinforcement Learning (GRL), which combines Graph Neural Networks with Reinforcement Learning, is a promising framework for addressing the power grid management problem. However, there is not a well established and standard framework to use GNNs within the context of power grid management via topology changes. Many approaches have been proposed over the years, differing in many aspects of the procedures, even in the representation of the power grid as a graph. A comprehensive survey about the use of GNNs within power grids is provided in [16].
The first work to introduce GNNs for power grid topology control was [61], where the GNN was used as a feature extractor for a standard DQN agent. This solution represented the power grid as an heterogeneous graph in which the nodes were: generators, loads and end of powerlines, with two elements linked if connected to the same bus.
The same graph representation was adopted in several subsequent works [50, 62].
The first solution introducing a new graph representation is the winning solution of the L2RPN 2020 challenge [64]. It represented the power grid as a graph in which nodes are substations, thus using a homogeneous graph representation.
Notably, a heterogeneous graph representation of the power grid rises two important problems: the so called "bus-bar information asymmetry" and the need for heterogeneous GNN models. The former highlights the fact that, within heterogeneous graph representations, elements linked to different buses on the same substation are not connected together. Therefore, information about potential connection is lost. The latter, instead, underlines a common problem of using standard GNN with heterogeneous graphs: they fail to generalize. This topic has been analyzed by [7] that solved both problems by slightly improving the heterogeneous graph representation and by using an heterogeneous GNN model. More precisely, it used edge weights to identify two classes of connections: elements connected to the same bus are linked by edge type 1 and elements connected to different buses in the same substation are linked by edge type 2. This representation is handled by a heterogeneous GNN, thus increasing the model's complexity.

4.3. Multi-Agent Solutions for Power Grids

A rational way to handle the huge combinatorial action space is by means of Multi-Agent Reinforcement Learning (MARL). In this way, the action space could be split in many subsets, each managed by a different agent. Moreover, given the natural graph structure of the power grids, the "logistic" placement of these agents is relatively straightforward: either place them on substations or on lines.
The first approach that introduced Multi-Agent (MA) solutions within power grid topology management problem was presented in [31]. The solution is a two-layers Distributed RL model: a single RL manager and multiple low-level agents (one for substation). When it is time to play, the manager selects the low-level agent that consequently plays on its controlled substation.
Another MA approach for power grid topology control was implemented by [52]. In this work, each substation is managed by a different agent trained with PPO [45]. The high-level logic deciding which agent comes to play upon critical situation is a rule-based logic called CAPA[1], that was firstly introduced by [64] but for a different scope. When a line is in overload, the CAPA logic selects as candidate agents the two substations extremities to the overloaded powerline, and later the substation with the highest average line utilization is selected among the two candidates. The CAPA logic presents an important limitation in the performance of such Multi-Agent architecture: it becomes difficult for the agents to develop long-term cooperative strategies.
The two previous approaches are unified in a recent work [8]. Here, the authors compare several two-layers RL solutions. In particular the effect of CAPA high-level logic against a RL manager. The results show that the RL manager outperforms the system with a CAPA high-level manager in large power grids.

4.3.1. Discussion

All the proposed MA solutions use the same observation as input to all the agents. Both the manager and the low-level agents receive a vector representation of the entire power grid, consisting of the concatenation of flow information, line capacities, and a topology vector. Note that, although the action space has been correctly split into independent subsets, the dimension of the input observation linearly grows with the size of the power grid, possibly becoming a limit when dealing with very large power grids.
Moreover, all the proposed MA solutions design the placement of agents in the same way:

[1]The acronym CAPA was not explicitly defined in the original paper that introduced it.

each low-level agent controls a different substation. This choice is probably motivated by the fact that the simulator allows to act on a single substation at a given timestep. However, the actual task of the power grid management problem is: maintain the grid's stability by avoiding lines' overloads. Therefore, a deployment of agents over powerlines better reflects the task of the control problem.

4.4. Power Grid Decomposition

As discussed above, Multi-Agent solutions effectively partition the action space into independent subsets, but they do not address the increasing complexity of the observation space. While low-level agents can be designed to operate based on local observations, the RL manager requires access to the full environment state in order to make informed decisions, which poses significant challenges to scalability.
Promising research directions are addressing this problem by designing algorithms to decompose the power grid in smaller, possibly independent, subgrids and manage them separately, as studied in [29]. However, the latter method, which computes a static decomposition of a power grid into separate subgrids, could suffer from the fact that the topology of the power grid changes over time, possibly resulting in dynamic independent subgrids, thus not fixed a priori.

5 | Proposed Solution

The core idea of the proposed solution is to design a two-layers distributed model that decomposes the large combinatorial action space into subsets, allowing each to be managed almost independently. Rather than relying on a full global view of the system, the approach uses local observations. To make these observations more informative, the model leverages graph-structured data, allowing agents to extract meaningful features from their local neighborhood and better cope with the partial observability inherent in such distributed settings.

Finally, the thesis explores and evaluates practical techniques – such as imitation learning and reward shaping – to accelerate training and improve performance in this multi-agent, two-layers setup.

Lastly, a minor contribution was made toward implementing a learning-based approach for decomposing power grids into independent sub-grids. This approach aims to enable future research into region-based controllers, which could improve the scalability of the architecture. This topic will be discussed in detail in Chapter 7.

The proposed algorithm is a two-layers Multi-Agent (MA) system: the low-level agents are independent Deep Q-Learners and share a Graph Neural Network (GNN) as feature extractor; the top-level agent is again a Deep Q-Learner and its role is to coordinate the low-level agents. Each low-level agent controls a single powerline of the grid, thus playing over a limited action space, a subset of the total grid topology actions. The GNN processes the grid observation, enriching the information available to the single agent. The top-level agent selects which low-level agent should act in a given situation.
Since convergence in such two-layers MA setting is hard to achieve, both low-level and top-level agents are pre-trained to learn expert demonstrations deriving from an Expert simulation-based algorithm [33], by means of Deep-Q-Learning from demonstrations (DQfD) [18]. Moreover, in such MA collaborative setting, the use of a global reward may lead to instabilities or difficulties during learning. Therefore, a potential-based reward-shaping technique [1] is implemented, yielding interesting results. The complete imple-

mentation is available at: https://github.com/Carlo000ml/RL4PG.

5.1. High-Level Model Architecture

The proposed solution adopts a two-layers MA architecture composed of three main learning components:

1. Low-Level agents: Deep Dueling Double Q-learning (DDDQN) agents [37, 55, 58] with Prioritized Experience Replay Buffer (PERB) [44] in charge of managing the powerlines. Each agent controls a distinct powerline, performing only the topology actions associated with that specific line. Moreover, each agent perceives only its powerline, receiving a local observation composed exclusively by the subset of features related to that powerline. Therefore, from a single agent perspective, all the other agents belong to the environment, which becomes complex and partially observable. Although common MARL approaches for power grid management assign agents to substations [8, 52, 64], we choose to assign agents to powerlines instead, which more naturally aligns with the objective of controlling the grid stability avoiding lines' overloads.

2. Shared Graph Neural Network (GNN): all low-level agents share a common GNN, which processes a graph-based representation of the power grid to enrich the information available to the single agents. The GNN acts as a feature extractor that enhances the informativeness of the single line observation by incorporating neighborhood information. The GNN aims to reduce the degree of partial observability of the environment from each agent's perspective. Moreover, since the GNN is shared among agents, it implicitly promotes cooperation and possibly enhances generalization capabilities. This procedure potentially allows graph-neural transfer learning, where a model trained on a smaller grid can be used to initialize the model on a larger one, possibly improving convergence.

3. High-Level Controller: DDDQN with PERB agent that manages the low-level agents deciding which one has to act on a specific situation. It receives as input the powerlines' thermal limits and current flow information, and a binary topology vector. Upon danger situation, the controller selects which agent has to act. Differently from many of the distributed solutions, which involved a rule-based selection procedure called CAPA [52, 64], this work enables a more flexible solution by means of an RL top-level agent, following a new promising research direction as [31] and [8].

To better illustrate the computational steps the model performs to produce an action, Alg.

5.1 outlines the flow executed by the architecture when called to act on the environment.

Algorithm 5.1 Action computation flow

```
1: $i \leftarrow$ Manager.select_agent$(s_t|\mu^t)$
2: $g_t \leftarrow$ convert_graph$(s_t)$
3: $o_t \leftarrow$ GNN$(g_t|\phi^t)[i]$
4: $a_t \leftarrow$ Agent$_i$.policy_play$(o_t|\theta_i^t, \alpha_i^t, \beta_i^t)$
5: return $a_t$
```

An abstract high-level representation of the model is provided in Fig. 5.1.

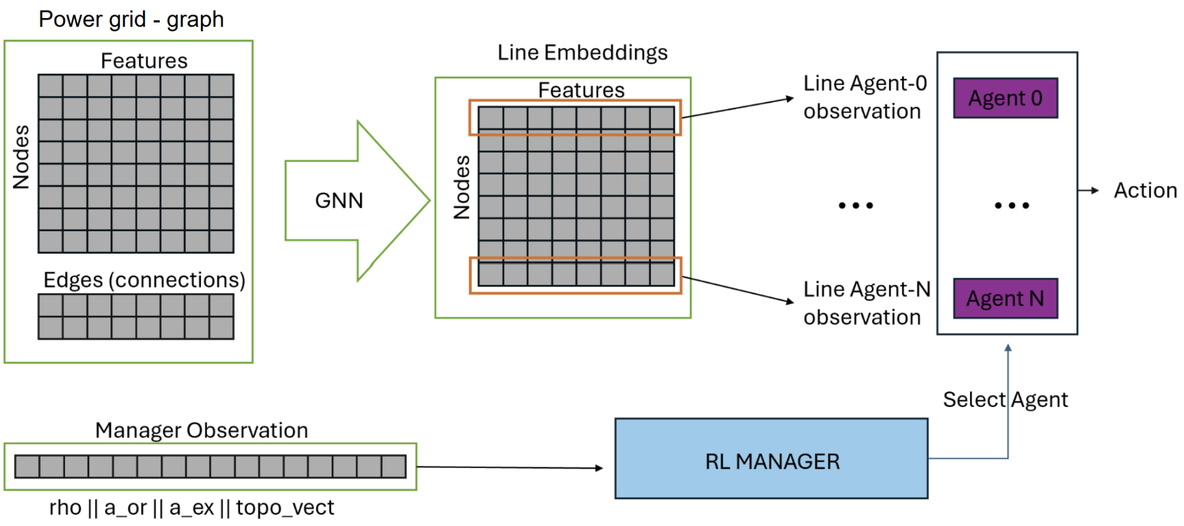


Figure 5.1: Abstract high-level representation of the model.

5.2. Graph-Based Observation Construction

The raw simulator observation has been pre-processed to obtain graph-like data before passing it to the GNN. Both the Grid2Op simulator and the research community have proposed methods for representing the power grid as a graph; however, each approach presents certain limitations. Most of existing methods work by considering each element as a node (load, generator, powerline, bus ...) rising the problem of graph heterogeneity: nodes present dissimilar features or belongs to different classes. In general, the problem of graph heterogeneity/heterophily strongly affects the performance of GNNs [66], which usually fails to generalize in such setting. Therefore, the common approach to handle heterogeneous graphs with GNNs is to employ models specifically designed for such structures, which inevitably increases model complexity [26, 32, 66]. Moreover, most of

existing methods represent buses in the graph but omit substations, meaning that the two buses belonging to the same substation are not connected. This leads to what is known as "bus-bar information asymmetry", where each bus lacks access to information about potential connections originating from its counterpart within the same substation. The pre-processing procedure of this work is novel, returns an homogeneous graph representation and naturally handles the problem of bus-bar information asymmetry without using heterogeneous graphs as in [7].
Initially every element of the grid, except for substations, is directly represented as a vector. Next, the vectors of the elements connected to the same bus are summed to create the bus embedding. To represent a substation, the embeddings of its buses, including the bus of disconnected elements, are concatenated to form the substation embeddings. Finally, the powerlines' embeddings are obtained by concatenating the embeddings of their two terminal substations. To clarify the procedure, a more detailed explanation with illustrations is provided below.
Each element (powerline, load, generator or bus) is represented as a feature vector using a one-hot-like encoding, where the vector is zero in all positions except for the subset of dimensions reserved for its specific type, as illustrated in Fig. 5.2.

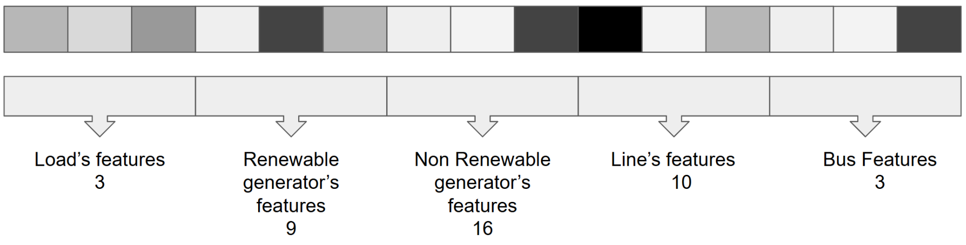


Figure 5.2: Vector representation of a grid element. The vector depicted in the illustration serves as an explanatory example; its dimensionality is simplified and does not correspond to the actual feature vector used in the model.

An example is provided in Fig. 5.3, only the dimensions associated with loads' features are non-zero. Note that the features for each type are the ones provided by the simulator and illustrated in Section 3.1.

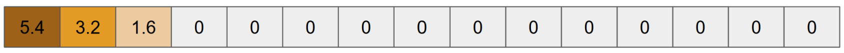


Figure 5.3: Explanatory example of the vector representing a load. The vector in the illustration is for explanatory purpose only and does not represent a real load's vector from Grid2Op.

In this way it is possible to represent a substation by considering the vectors of the elements connected to it, an example is provided in Fig 5.4.

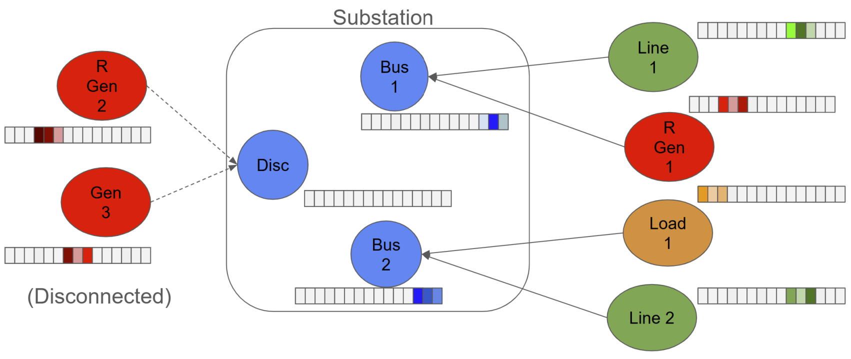


Figure 5.4: Explanatory example, representation of a substation.

By summing the vectors bus-wise, we obtain the bus embeddings, which are then concatenated to construct the substation embeddings, as shown in Fig. 5.5.

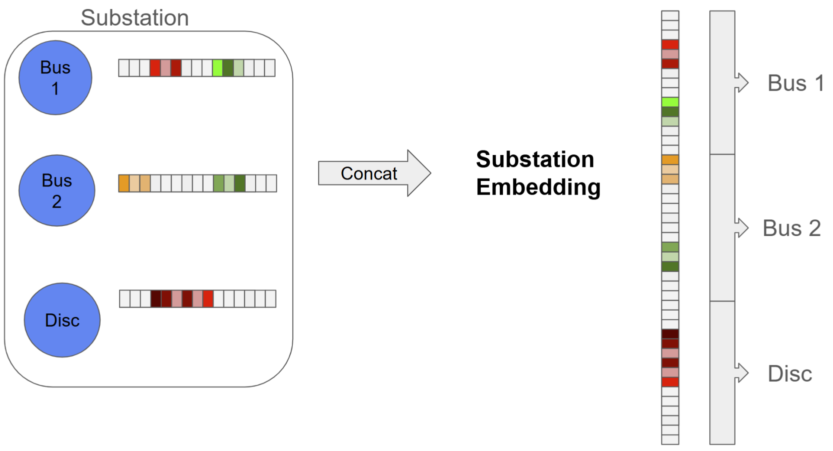


Figure 5.5: Explanatory example, substation's embedding.

Finally, the line embedding is created as the concatenation of the embeddings of its origin and extremity substations, as illustrated in Fig. 5.6.

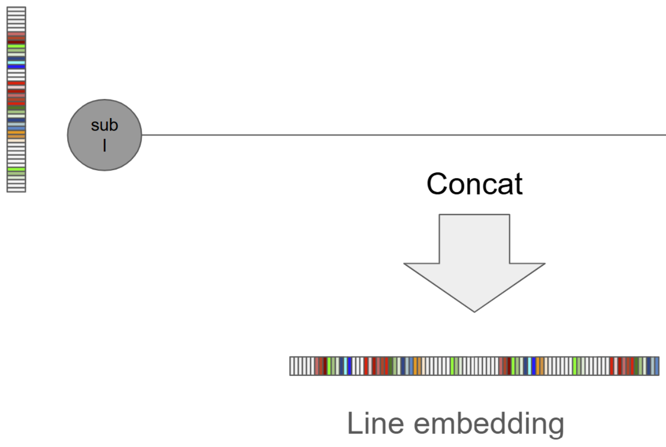


Figure 5.6: Explanatory example, line's embedding.

At this point, each line is represented by a vector that captures the information from both substations it connects.
To build the graph, each powerline is treated as a node, and two nodes are connected if their corresponding powerlines share a common substation. This transformation converts the original power grid into a line graph, where powerlines (originally edges) become nodes, and substations (originally nodes) become edges.
With this procedure, a sequence of power grid observations is turned into a sequence of undirected homogeneous graphs.

5.3. Graph Processing

Once the power grid is represented as a line graph, it is fed into a GNN with learnable parameters ϕ, which enriches the node features and computes the agents' observations. Formally, the observation for a single agent i is computed from the original graph-state $g_t = \text{convert\_graph}(s_t)$ as:

$$o_t^i = \Big[GNN\big(g_t|\phi^t\big)\Big]_i, \tag{5.1}$$

where $[\dots]_i$ is the i-th element of the input vector. Essentially, before applying the GNN, each node contains only its own line-specific features. After processing with the GNN, each node's representation is enriched with aggregated information from its neighboring nodes, capturing the local topology. An abstract example is provided in Fig. 5.7.

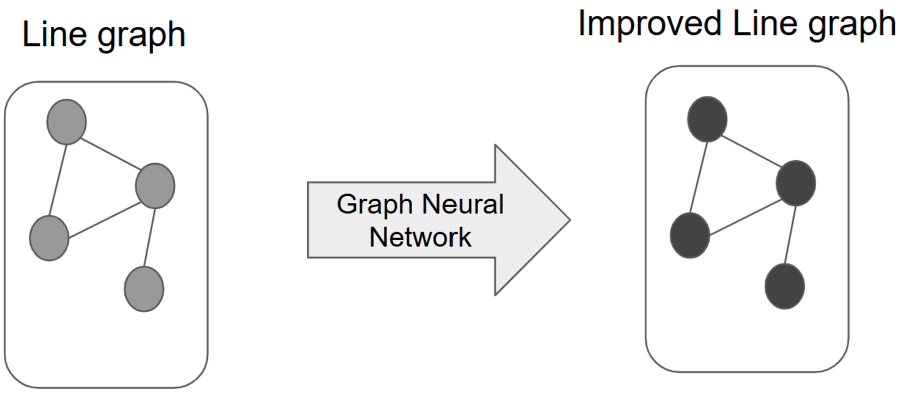


Figure 5.7: Explanatory example illustrating how a Graph Neural Network (GNN) enriches node features. The lighter color indicates that the initial node features are limited and sparse, while the darker gray signifies that, after applying the GNN, the nodes contain more informative and enriched representations.

While the initial model employed a Graph Convolutional Network (GCN), empirical results demonstrated superior performance with Graph Attention Networks (GAT), specifically the GATv2 extension.

5.4. Low-Level Line Agents

The Low-Level Line agents are Deep Dueling Double Q-learning (DDDQN) agents [37, 55, 58] with Prioritized Experience Replay Buffer (PERB) [44]. Each agent has its own Observation Space Ω_i and its own Action Space $\mathcal{A}_i$, both related to the specific line under control. The agents are independent learners, meaning that, from a single agent perspective, other agents belong to the environment, which becomes Partially Observable. However, the reward function is shared among the agents, which observe a global reward for their actions. Mathematically speaking, agent i plays over the POMDP: $< \mathcal{S}_i, \mathcal{A}_i, \mathcal{R}, \Omega_i, \mathcal{T}_i, \mathcal{O}_i, \mu_i, \gamma_i >$. Diving into the details of the model, the target computation (without bootstrapped reward shaping) for a single agent receiving in input the line graph $g_t = \text{convert\_graph}(s_t)$ is:

$$y_t^i = r_t + \gamma \cdot Q\Big(g_{t+1}, \arg\max_a Q\big(g_{t+1}, a|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\big)|\phi^{t-}, \theta_i^{t-}, \alpha_i^{t-}, \beta_i^{t-}\Big), \tag{5.2}$$

where ϕ represents the learnable parameters of the GNN and are shared among all the agents (no subscript i), θ_i, α_i and β_i are the Q-network parameters, respectively: the

shared ones, the ones of the advantage stream and the ones of the value stream. Remember that the input to the i-th line agent is the i-th component of the output of the GNN.
The parameters are learned via mini-batch gradient descent, to minimize the standard deep Q-learning loss (see Eq. 2.21). Being the GNN shared among all the agents, its parameters are jointly optimized together, forcing a sort of information sharing among the agents. Following a common strategy adopted by several competitive and state-of-the-art algorithms for power grid management [20, 53], this work employs action space reduction to accelerate the learning process. Specifically, a single-step simulation is performed at runtime to eliminate unsafe actions, i.e., those that result in an increased maximum `rho`, indicating greater grid instability. Before a low-level agent executes an action, the actions at its disposal are simulated for a single time step, and only those that do not worsen the grid's stability are considered.

5.5. High-Level Controller

The high-level RL manager is again a DDDQN agent [37, 55, 58] with PERB [44], driven by parameters μ^t. It receives as input the powerlines' thermal limits (`rho`) and current flow information (`a_or` and `a_ex`), and a binary topology vector (`topo_vect`). Therefore the dimensionality of its input is: $3|L|+|N|$ where $|L|$ is the number of powerlines and $|N|$ is the number of elements of the power grid (generators, loads and ends of powerlines). When a critical situation occurs, the high-level controller selects the appropriate line agent to intervene. Therefore, the action space of the top-level agent has size $|L|$.
Note that the High-Level manager is the only non-scalable component of the proposed solution as its observation and action space grow linearly with the size of the power grid. Therefore, factorization methods and regional-based controllers are promising research directions to deal with very large power grids.

5.6. Deep Q-Learning from Demonstrations

After collecting a dataset of expert experiences, the DQfD algorithm was implemented to accelerate convergence. Regarding the losses detailed in Section 2.2.5, the *n-step double Q-learning loss* was adopted with $n = 10$, matching the value used in the original paper's experiments.

As for the hyperparameters $\lambda_1, \lambda_2, \lambda_3$, a comprehensive tuning was not feasible due to the substantial computational cost of each experiment (as discussed in Section 6.4). Instead, the individual contribution of each loss component to the overall performance was analyzed separately, and the final values of the λ hyperparameters were adjusted based on these results.

5.7. Potential-Based Reward Decomposition

Potential-based reward shaping was used to refine the reward signal in order to give more accurate rewards to the low-level agents. In particular, Bootstrapped Reward Shaping (BSRS) was implemented by each low level agent. More precisely, the single low-level agent reward is computed decomposing the global reward as:

$$\tilde{r}_t^i = r_t + \gamma \cdot \eta \cdot V\left(g_{t+1}|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right) - \eta \cdot V\left(g_t|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right), \tag{5.3}$$

where the value function is computed with double learning as:

$$V\left(g_t|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right) = Q\Big(g_t, \arg\max_a Q\left(g_t, a|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right)|\phi^{-t}, \theta_i^{-t}, \alpha_i^{-t}, \beta_i^{-t}\Big). \tag{5.4}$$

The final target computation becomes:

$$\begin{aligned} y_t^i &= r_t + \gamma \cdot \eta \cdot V\left(g_{t+1}|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right) - \eta \cdot V\left(g_t|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right) \\ &\qquad + \gamma \cdot Q\Big(g_{t+1}, \arg\max_a Q\left(g_{t+1}, a \mid \phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right) \mid \phi^{t-}, \theta_i^{t-}, \alpha_i^{t-}, \beta_i^{t-}\Big) \\ &= \Big[r_t - \eta \cdot V\left(g_t|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right)\Big] + \gamma \cdot (1+\eta) \cdot V\left(g_{t+1}|\phi^t, \theta_i^t, \alpha_i^t, \beta_i^t\right). \end{aligned} \tag{5.5}$$

5.8. Complete Algorithm

A high-level overview of the implemented Graph-based Distributed Double Dueling DQN (G4DQN) algorithm is presented in Alg. 5.2. It consists of two main phases: the *Pre-Training* phase and the *Online-Training* phase. The Pre-Training phase simply performs Deep Q-Learning from Demonstrations to learn expert experiences while the Online-Training phase is a standard RL Deep Q-learning interaction loop. The latter encompasses several parts:

- The *Exploitation* part runs the agents in pure exploitation mode to monitor performance throughout training and to validate the model.

- The *Learning* part allows agents to learn from the experiences stored in their buffers. Notably, it takes the graph processor as input, meaning that graph processor and the agents are optimized together to minimize the same loss.

- The *Target Synchronization* acts as a global synchronization step for the low-level agents, during which all low-level agents, graph processor included, simultaneously synchronize their target networks and their respective main networks in a coordinated manner. The RL manager synchronizes its networks independently of the low-level agents.

The `Train-Episode` routine, that runs an episode in train mode, is detailed in Alg. 5.3. The `Exploit-Episode` routine, which runs an episode in exploit mode, follows the same structure as `Train-Episode` except that it plays agents in pure exploitation (greedy-policy) and does not collect experiences.

Algorithm 5.2 High-level G4DQN Training Procedure

```
1: Input: Training and validation environments, line agents, high-level controller, graph
   processor, number of episodes $N$, validation period $V$ learning period $L$, low-level
   agents sync period $S_1$, high-level agent sync period $S_2$
2: Pre-training:
3: for each line agent do
4:     Learns from demonstrations jointly with the graph processor
5: end for
6: High-level controller learns from demonstrations
7: Online Training:
8: for episode = 1 to $N$ do
9:     if episode is an exploitation episode then
10:        Run training episode in exploit mode
11:    else
12:        Run training episode in training mode
13:    end if
14:    if episode = 0 mod $L$ then
15:        for each line agent do
16:            Learns from experience replay buffer jointly with the graph processor
17:        end for
18:        high-level controller learns experiences from its buffer
19:    end if
20:    if episode = 0 mod $S_1$ then
21:        Synchronize target networks for line agents and graph processor
22:    end if
23:    if episode = 0 mod $S_2$ then
24:        Synchronize target networks for high-level controller
25:    end if
26:    if episode = 0 mod $V$ then
27:        for each validation episode do
28:            Run validation episode in exploit mode
29:        end for
30:    end if
31: end for
```

Algorithm 5.3 G4DQN Train Episode

```
Require: $Env$ (Environment), $GP(\phi^t)$ (Graph Processor), $\{Agent_i(\theta_i^t, \alpha_i^t, \beta_i^t)\}_{i=1}^n$ (Line Agents), $MA(\mu^t)$ (High-Level Controller), $\bar{\rho}$ (Safety Threshold)
1: Initialize environment state $s_0 \leftarrow Env.reset()$, $done \leftarrow$ False, $t \leftarrow 0$
2: while not $done$ do
3:   if $is\_safe(s_t, \bar{\rho})$ then
4:     {Safe situation}
5:     Take no action: $a_t \leftarrow DoNothing()$
6:   else if $disconnected\_Line(s_t)$ then
7:     {critical situation}
8:     Reconnect lines: $a_t \leftarrow Reconnect\_Line(s_t)$
9:   else
10:    {Danger situation}
11:    Select line agent $i \leftarrow MA.select\_agent(s_t|\mu^t)$
12:    Construct graph $g_t \leftarrow convert\_graph(s_t)$
13:    Build observation $o_t \leftarrow GP(g_t|\phi^t)[i]$
14:    Agent acts $a_t \leftarrow Agent_i.policy\_play(o_t|\theta_i^t, \alpha_i^t, \beta_i^t)$
15:  end if
16:  Execute $a_t$ in environment: $(s_{t+1}, r_t, done) \leftarrow Env.step(a_t)$
17:  if an agent acted then
18:    Update graph $g_{t+1} \leftarrow Build\_graph(s_{t+1})$
19:    Store agent experience $\rightarrow Agent_i.collect\_experience(g_t, a_t, r_t, g_{t+1})$
20:    if line agent exploited then
21:      Store top-level experience $\rightarrow MA.collect\_experience(s_t, i, r_t, s_{t+1})$
22:    end if
23:  end if
24:  $t \leftarrow t + 1$
25: end while
```

6 | Experimental Results

The goal of the analysis is to assess whether a scalable, distributed model—based on split action and observation spaces—can achieve competitive performance in the power grid control task. This chapter presents the experimental results obtained during the study. It begins with a detailed description of the experimental setup and the evaluation metrics used. Subsequently, it introduces the baseline model and compares its performance with that of the proposed approach. This is followed by a qualitative analysis and an ablation study aimed at further understanding the model's behavior. The chapter concludes with a discussion of the computational requirements of the method.

6.1. Experimental Setup

6.1.1. Environment Description

The analysis was conducted on the Grid2Op [9] simulator, in particular on the *"l2rpn_case14_sandbox"* environment (Fig. 6.1). The latter represents a power grid counting 14 substations, 20 lines, 6 generators and 11 loads, and it encompasses 1004 different episodes. Each episode is represented as a set of time series (referred to as a chronics), which models the behavior of generators and loads over time. The agent must keep the power grid stable despite the unknown evolution of energy production (time series on generators) and consumption (time series on loads). The simulation runs for a maximum of 8064 timesteps. If the agent survives until the end, the simulation terminates regardless of whether a blackout has occurred.

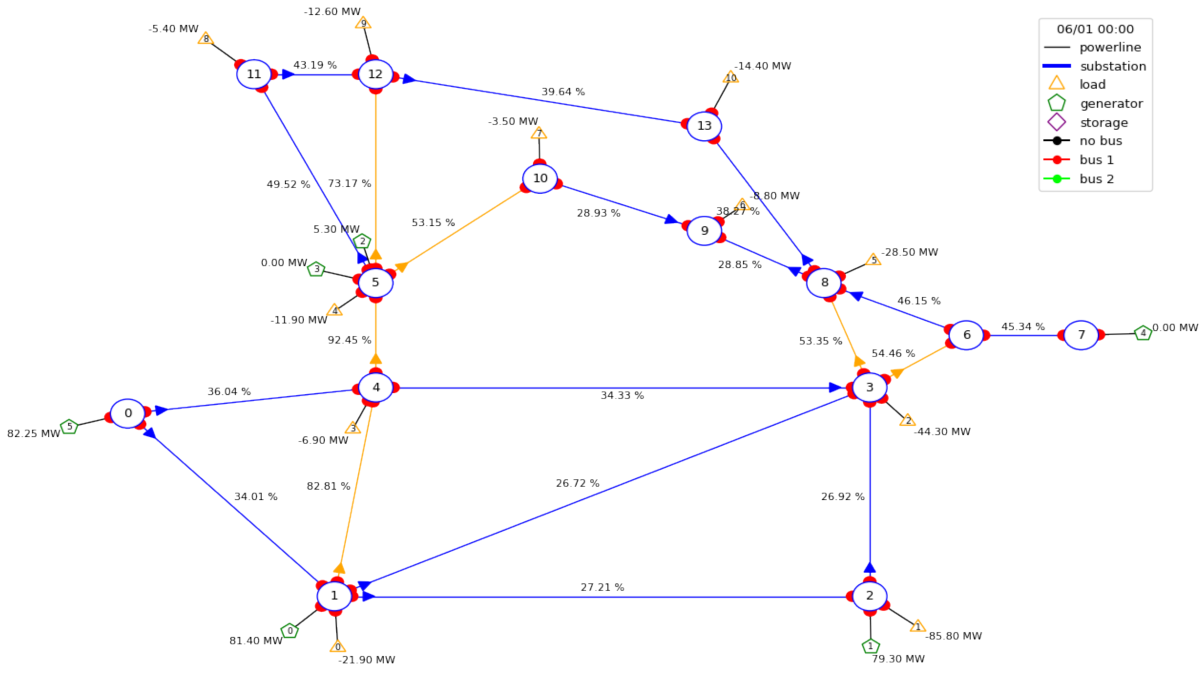


Figure 6.1: "l2rpn_case14_sandbox" power grid.

6.1.2. Training Configuration

Each powerline is controlled by a dedicated low-level agent, resulting in 20 such agents operating in parallel. Including the RL manager, the system consists of 21 agents in total. Each agent maintains a pair of neural networks—a main network and a target network. In addition, a graph neural network is employed as feature extractor, with both main and target versions. In total, 44 neural networks are simultaneously loaded on the computing device, with 22 actively trained and the remaining 22 kept frozen for target computation. The complexity of the system led to significant training instabilities: basic implementation often resulted in unstable/increasing loss of individual agents. Thus, carefully chosen strategies were necessary to ensure stable and effective learning. To this end, gradient clipping and soft main-target synchronization were adopted to achieve stable learning and an effective training. Each main neural network was optimized using Adam optimizer [23].

Inspired by the original DQN paper [37], gradients were clipped in the range $[-3, 3]$. In addition, soft updates between the main and target networks were employed, as this is a widely adopted practice for enhancing training stability in Deep Q-learning methods [27]. Soft updates provide a smooth synchronization mechanism between the main and target networks by incrementally updating the target network's weights θ' according to the rule: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, where θ denotes the weights of the main network and

$\tau \ll 1$ is a small update coefficient that governs the rate of change. However, the most significant contribution to training stability came from increasing the synchronization interval. Synchronizing the main and target networks too frequently was observed to cause an exponentially growing loss.
The actual hyperparameters used for the final solution are shown in Appendix B.

6.1.3. Exploration Strategy

All agents followed an $\epsilon-$greedy exploration policy, with exponential ϵ decay controlled by half-life. The choice of this strategy was guided by the goal of maintaining a transparent and interpretable exploration behavior. Several exploration methods were initially considered, including Boltzmann exploration, standard exponential decay, and exponential decay with half-life control. However, due to the high computational cost associated with each experiment, extensive hyperparameter tuning was not feasible. As a result, the most straightforward and easily configurable option—exponential ϵ decay based on half-life—was selected for all experiments. In this approach, the value of ϵ decays according to an exponential schedule defined by a half-life parameter, which represents the number of steps required for ϵ to be reduced by half. The implementation follows [41], formally, the ϵ updates are computed as:

$$\epsilon_t = \epsilon_{min} + (\epsilon_0 - \epsilon_{min}) \cdot e^{-\frac{t \cdot ln(2)}{h}} \tag{6.1}$$

so that $\epsilon_t = \epsilon_0$ when $t = 0$, while $\epsilon_t = \frac{1}{2}(\epsilon_{min} + \epsilon_0)$ when $t = h$

6.1.4. Evaluation Protocol

The main performance metric was the survive time, measured as the number of timesteps the agent was able to keep the power grid operating safely without causing a blackout, for a given chronic. This value ranges from 0 to 8064, with a score of 8064 indicating that the agent successfully completed the entire chronic without failure. For a correct evaluation of the model, the 1004 chronics were split into 98% training, 1% validation and 1% test, as suggested by the official Grid2Op tutorials [4]. The overall number of chronics for each set is:

- 984 training chornics
- 10 validation chornics
- 10 test chornics.

The agent's performance was tracked throughout training on both the training and validation sets, by evaluating its behavior in exploit mode, across a sufficiently large number of chronics. The test set was used the least possible to avoid making informed decisions based on its outcomes.

6.1.5. Training Reward

The reward collected by each agent was slightly different from the reward provided by the simulator and presented in Section 3.1. The agent's reward at timestep t, results of action a_t, is denoted as r_t and it is defined as follows:

$$r_t = \begin{cases} \dfrac{1}{20} \sum\limits_{l \in \text{Lines}} \left(\max(1 - rho_l^{t+1},\ 0)\right)^2, & \text{if no blackout occurs,} \\ -1, & \text{otherwise,} \end{cases} \tag{6.2}$$

where rho_l^{t+1} represents the capacity of line l at time $t+1$. This reward formulation encourages the agent to maintain line loads below their thermal limits (i.e., $rho_l^{t+1} < 1$). The closer the line capacity is to zero, the higher the reward contribution. A blackout triggers a fixed negative reward of -1.

6.2. Results

6.2.1. Do Nothing Baseline

To evaluate the effectiveness of the proposed approach, we compare it against a straightforward yet surprisingly competitive baseline: the *Do-Nothing* policy, the same baseline used in L2RPN challenges [35]. This baseline represents a passive control strategy where the agent takes no action throughout the episode, regardless of the system's state.

The strength of this policy lies in the design of the environment itself. By design, the initial configuration, where all components in the grid are connected to bus 1 of their respective substations, is a stable configuration. This setup allows the system to maintain safe operation for a considerable duration without intervention, especially under moderate or favorable chronics. As a result, the Do-Nothing policy often achieves non-trivial survival times and serves as a meaningful reference for evaluating learned strategies.

Moreover, prior work has highlighted the importance of the initial configuration. For example, in a qualitative analysis of winning solutions to the L2RPN competition, Yoon et al. [64] showed that the best-performing models tended to deviate the least from the initial

topology, suggesting that minimal intervention is often the most robust strategy. Similarly, the solution proposed by [20] explicitly included in the action space an action "return to initial configuration" highlighting the desirable stability of the initial configuration.
The performance of the *Do-Nothing* baseline over the training chronics is provided in Fig. 6.2. The average performance is 1628.59, with bootstrap-based 95% confidence bounds of [1541.42, 1720.62].

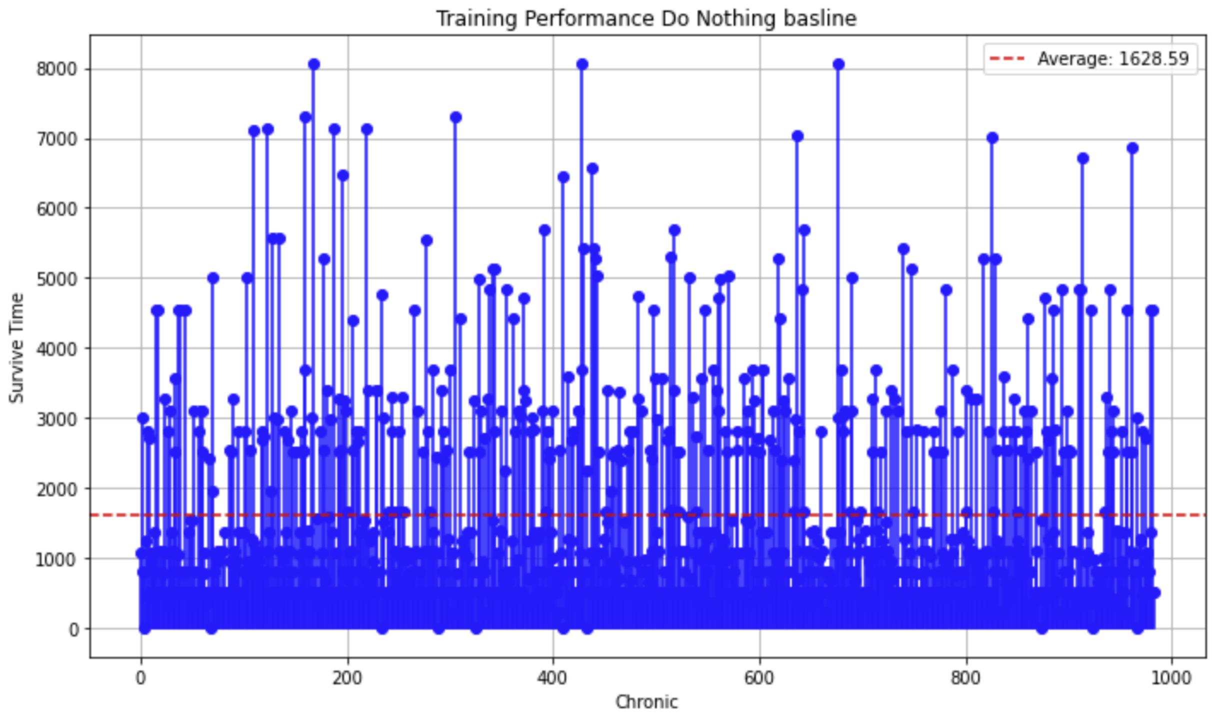


Figure 6.2: Do-Nothing training performance.

6.2.2. Performance of the proposed architecture

The proposed architecture is evaluated and tested against the *Do Nothing* baseline.

Validation Performance

As shown in Fig. 6.3, the proposed architecture outperforms the *Do-Nothing* baseline on the validation set after training. The x-axis denotes the id of the specific validation chronic (10 in total), while the y-axis reports the survive time.

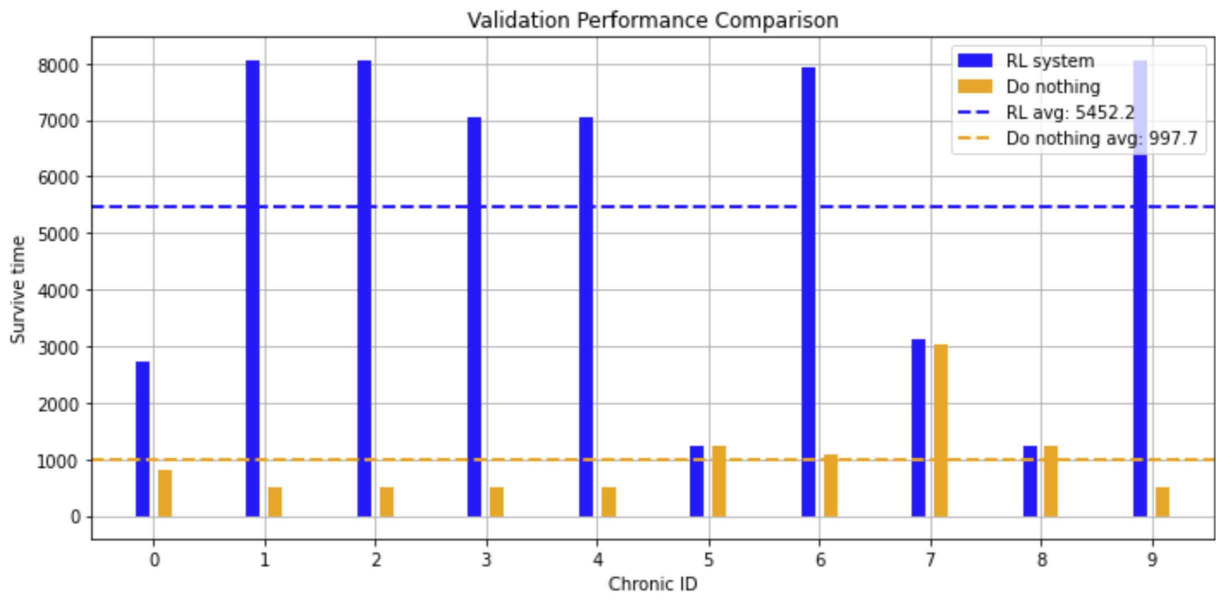


Figure 6.3: Comparison on the validation set: Proposed model against *Do Nothing* baseline.

To give a better understanding of the validation performance, the actual number of performed actions for each chronic is reported in Fig. 6.4.

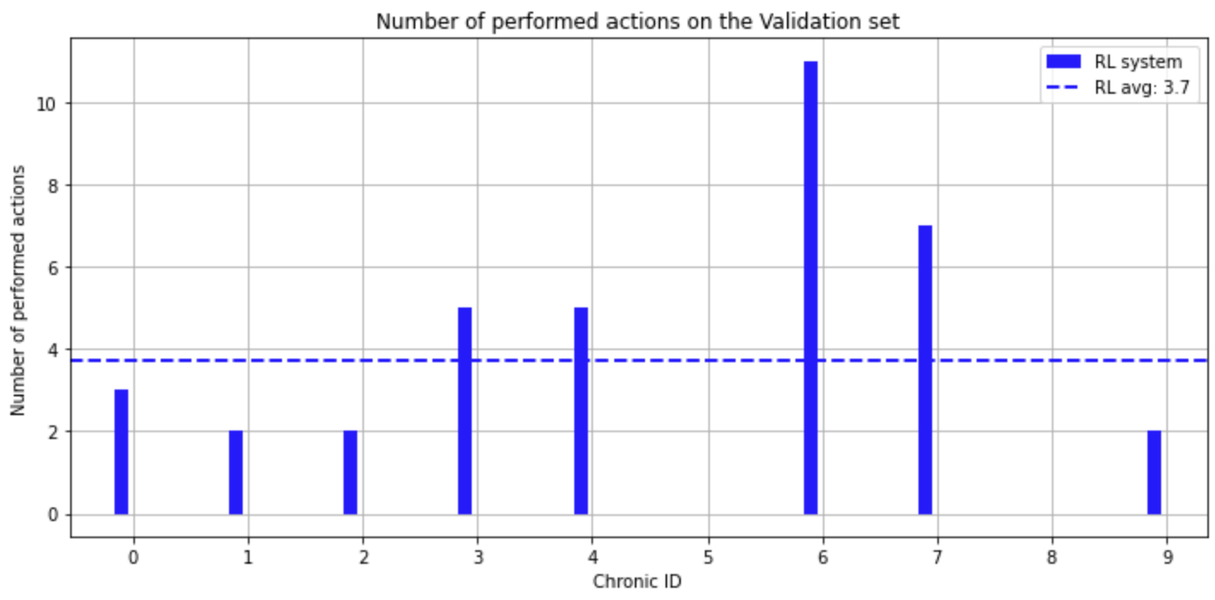


Figure 6.4: Number of actions taken on the validation set, excluding "do nothing" actions.

The validation performance has been monitored throughout the training process, the results are provided in Fig. 6.5.

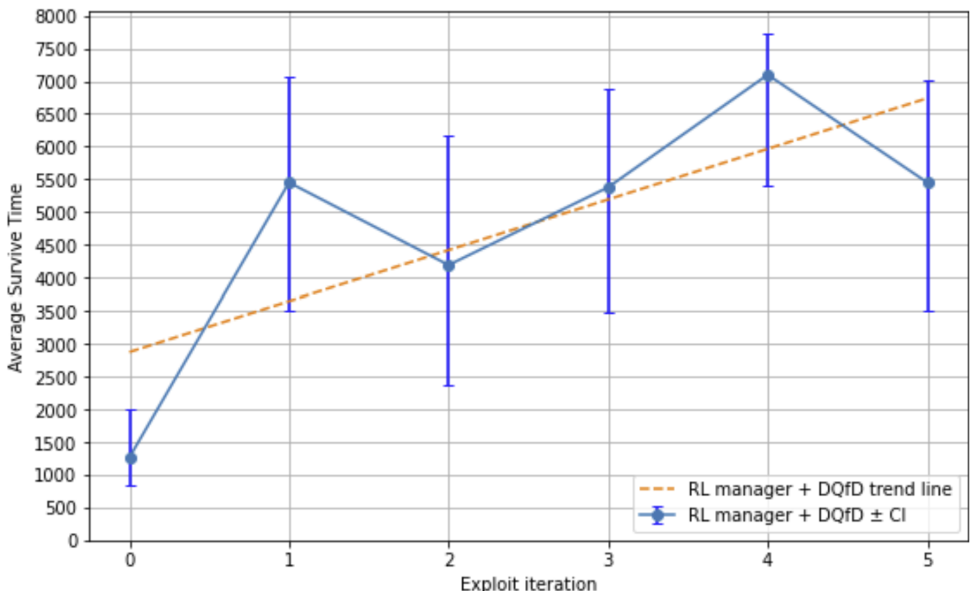


Figure 6.5: Validation performance monitored throughout training. The x-axis indicates the validation iteration, each corresponding to a full evaluation over the 10 validation chronics. The y-axis shows the average survival time across these chronics, along with associated confidence intervals.

Test Performance

Similarly, the proposed model and the *Do Nothing* baseline are compared on the test set and the results are provided in Fig. 6.6.

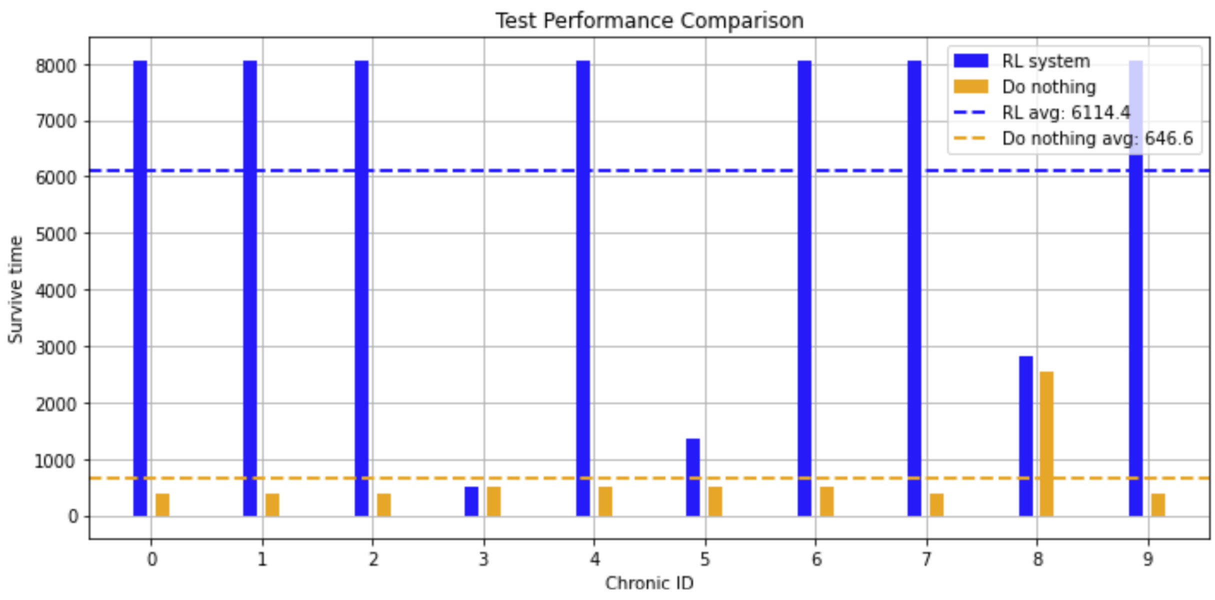


Figure 6.6: Comparison on the test set: Proposed model vs *Do Nothing* baseline.

The number of performed actions per episode are reported in Fig. 6.7.

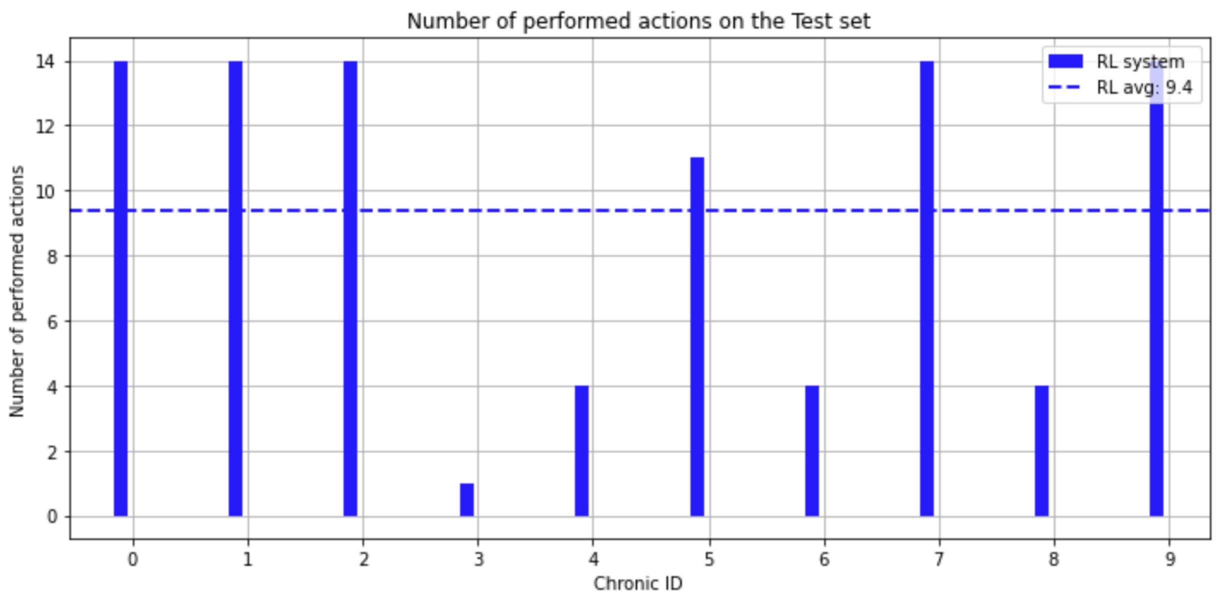


Figure 6.7: Number of actions taken on the test set, excluding "do nothing" actions.

Training Performance

The performance of the model was monitored during the training phase. Every 500 training episodes the model was tested in exploit-mode for 50 training chronics and the results were statistically processed to report the averages and confidence intervals over those 50 training chronics. These results are provided in Fig. 6.8

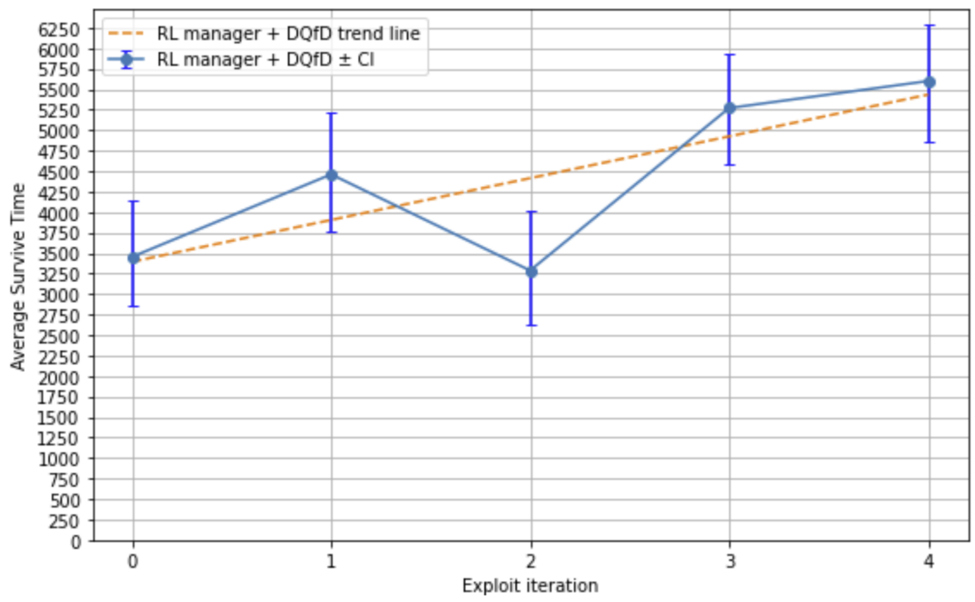


Figure 6.8: Survive Time in exploit mode monitored during the training. The x- axis represent the exploit iteration, while the y-axis reports the average survive time and confidence interval: each point is the average over 50 episodes. The confidence intervals are computed with bootstrapping method.

The number of actions performed during the training exploit monitoring is reported in Fig. 6.9

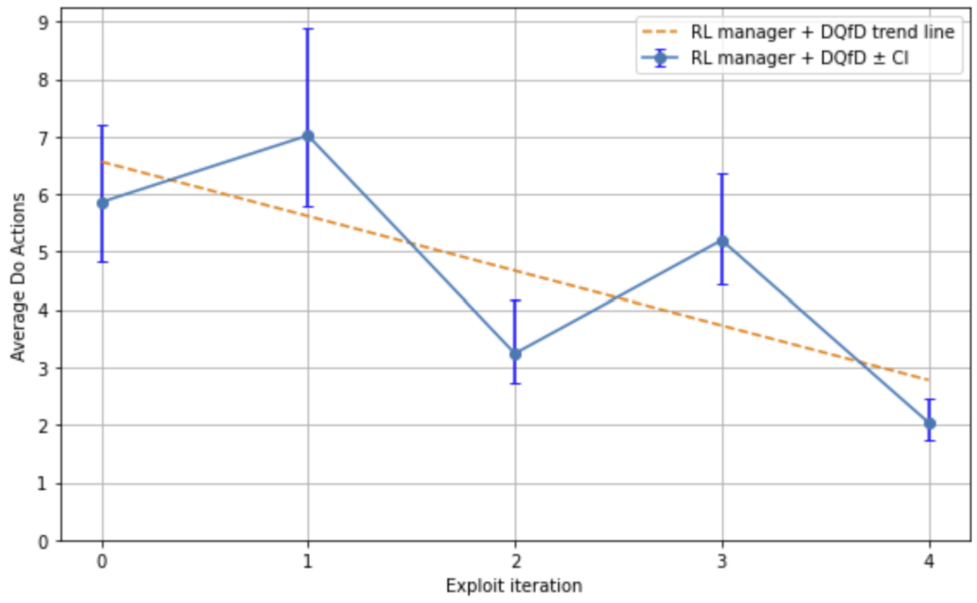


Figure 6.9: Number of performed actions during the training exploit monitoring.

6.2.3. Approximation Loss

To analyze the approximation accuracy of the deep neural network models, their estimation loss has been tracked using tensorboard [51] throughout the pre-training and training phase. The loss from the pre-training phase is reported in Fig. 6.10. While the loss obtained during the training phase is provided in Fig. 6.11.

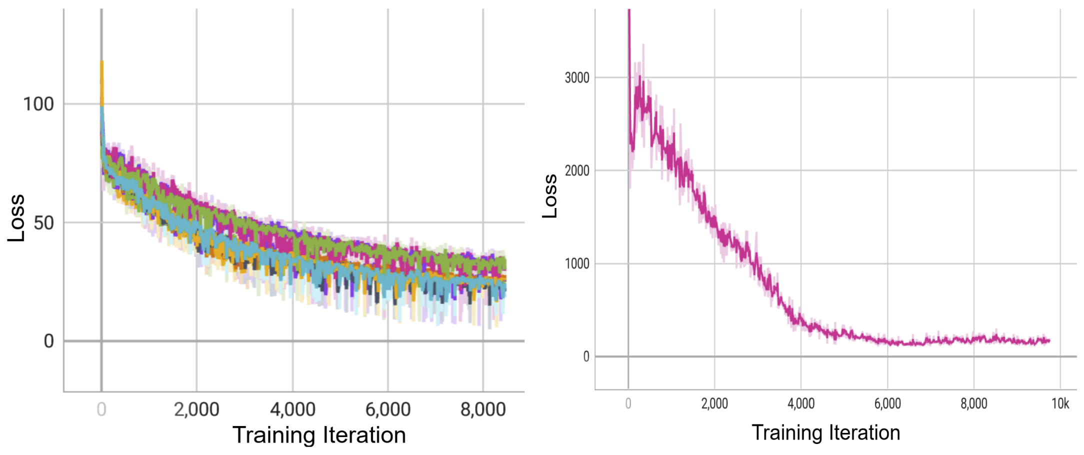


Figure 6.10: Loss from the pre-training phase. On the left the loss of the line agents, on the right the loss of the manager.

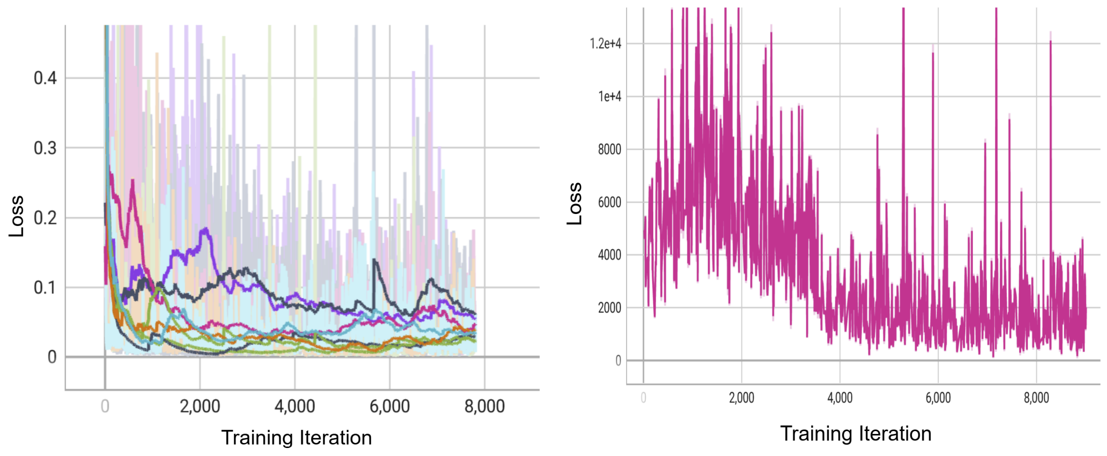


Figure 6.11: Loss from the training phase. On the left the loss of the line agents, on the right the loss of the manager. The loss of the line agents is provided as a moving average of their values, the actual loss is the shadow area around the lines.

6.2.4. Qualitative Analysis of GNN Weights

An analysis of the learned weights from the first GNN layer is conducted to better understand the network's internal behavior. Fig. 6.12 displays a heatmap of these weights, providing a qualitative perspective on how the model processes structured power grid information. Recall that the input vector processed by the GNN is formed by concatenating the feature vectors of the origin and extremity substations of each powerline. This structural symmetry is reflected in the heatmap, which displays a clearly symmetrical pattern with respect to the column indices corresponding to these two segments. This symmetry suggests that the GNN has effectively learned the structure of its input representation.

Furthermore, two distinct vertical regions exhibit near-zero weights: the first around column index 100 and the second toward the far right of the matrix. These regions correspond to the features associated with the buses of disconnected elements. It seems that the GNN has not focused its learning on these features, which is reasonable given that the action space does not include actions that disconnect elements.

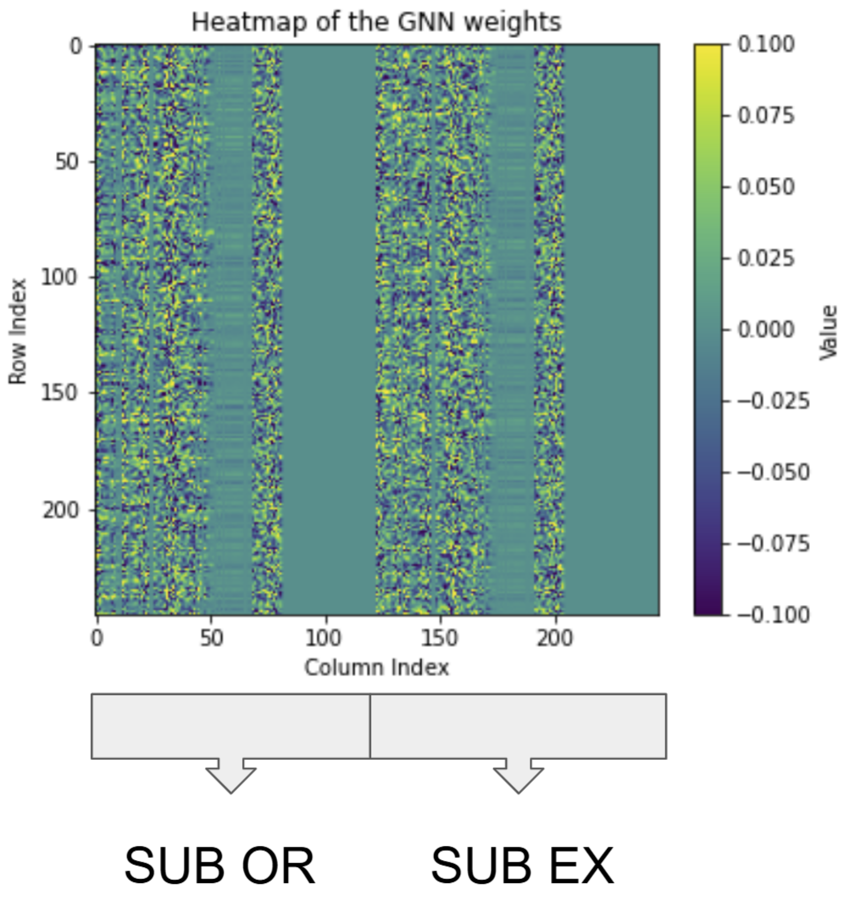


Figure 6.12: Heatmap of the weights in the first GNN layer.

6.3. Ablation Study

This section analyzes the contribution of the components of the proposed model by removing them one at a time and observing the impact on performance. The proposed model's main components tested in this ablation study are: DQfD, GNN, the RL manager, reward shaping, and the greedy action space reduction. Each component is removed in isolation to evaluate its individual impact on the model's performance. A summary of the ablation study is provided in the Tab. 6.1, while each component has a dedicated subsection in which the relative results are presented and commented. The computational time for each experiment is also reported in the table, while a more detailed version is provided in Section 6.4.4.

Table 6.1: Ablation study results: average survival time with 95% bootstrapped confidence intervals ([*lower bound,upper bound*]) on validation and test sets. The best result per column is in bold, the second-best is underlined.

Configuration	**Validation Performance**	**Test Performance**	**Computational Time**
No DQfD	1985 [1278,3153.2]	1877.9 [1141.8, 2725]	15.48 days
No GNN	1206.4 [780, 1982.1]	785.2 [464, 1889.9]	2.889 days
No RL Manager (CAPA)	2562.5 [1745.4, 3632.9]	—	11.57 days
No Reward Shaping	**5667.1** [3724.1, 7073]	<u>5324.3</u> [3723.1, 6291.1]	3.631 days
No Action Filtering	1157.6 [796.7,1764.2]	1634.2 [1057.4,2554.12]	15.49 days
Complete System	<u>5452.2</u> [3509.2, 7022]	**6114.4** [3791.7, 7538.5]	5.642 days

6.3.1. Deep Q-Learning from Demonstrations

The impact of DQfD on overall performance was analyzed by running a version of the model without a pre-training phase and without access to expert demonstrations. Since the pre-training phase significantly accelerates convergence, the version without DQfD was trained for five times as many episodes as the complete algorithm. The final performance on the validation and test sets is reported in Fig. 6.13. It is important to note that this experiment was conducted on a different machine using a different train/validation/test split, compared to other experiments. Consequently, the chronics differ from those used in the other experiments, which explains the slight performance discrepancy of the complete algorithm in this setting compared to the main results.

To further investigate the effect of removing DQfD, the training performance in exploit mode was monitored throughout the run. The results, shown in Fig. 6.14, demonstrates the slow rate of convergence of a model without access to expert demonstrations.

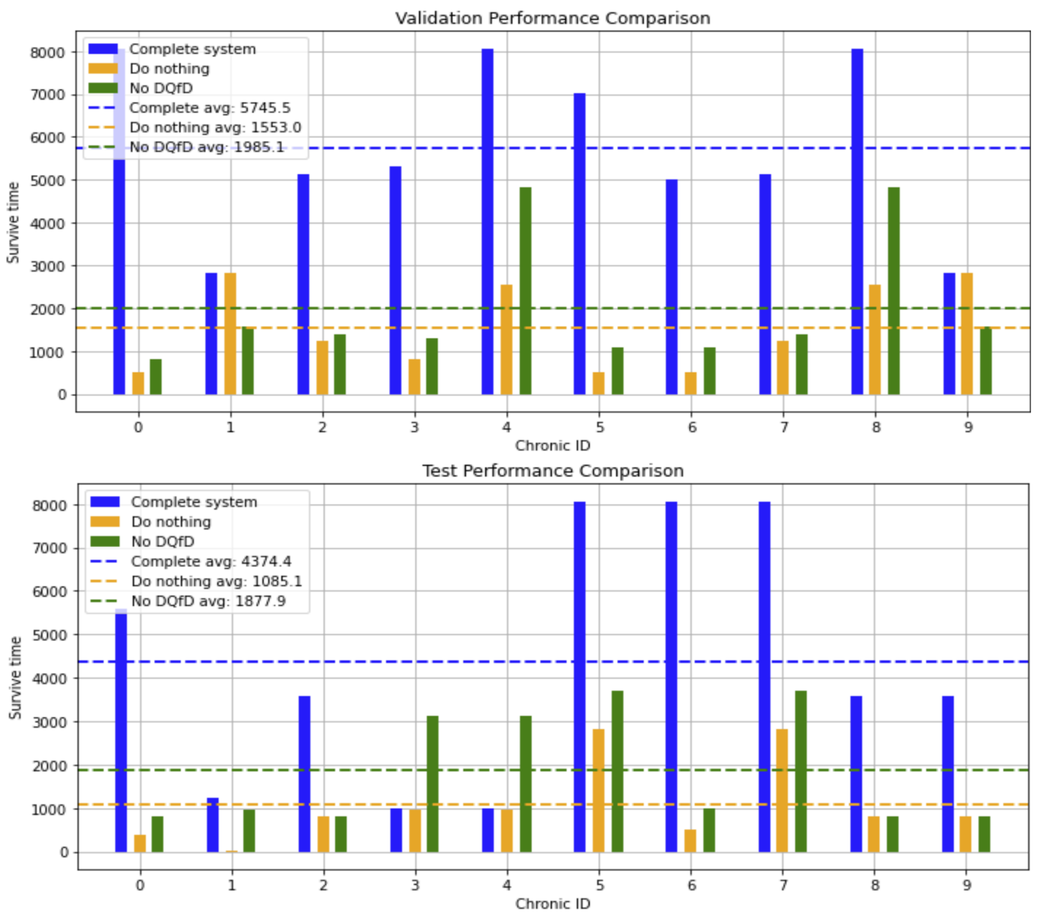


Figure 6.13: Validation performance comparison (top), Test performance comparison (bottom). The x-axis represents the chronic id, while the y axis reports the survive time.

6.3.2. Graph Neural Networks

To assess the impact of the GNN component, we evaluated a variant of the model in which the GNN was removed. In this ablated version, the agent received only the raw line observations, with no message passing. As a result, the agent lacked a structured representation of the grid topology.

Figs. 6.16 and 6.15 shows that removing the GNN leads to a substantial degradation in performance on both the test and validation sets. This indicates that the GNN plays a critical role in enabling the agent to generalize across chronics by learning relational information between lines.

In addition, we examined the approximation loss magnitude during both the demonstration pre-training phase and the subsequent online learning phase, the results are provided in Figs. 6.17 and 6.18.

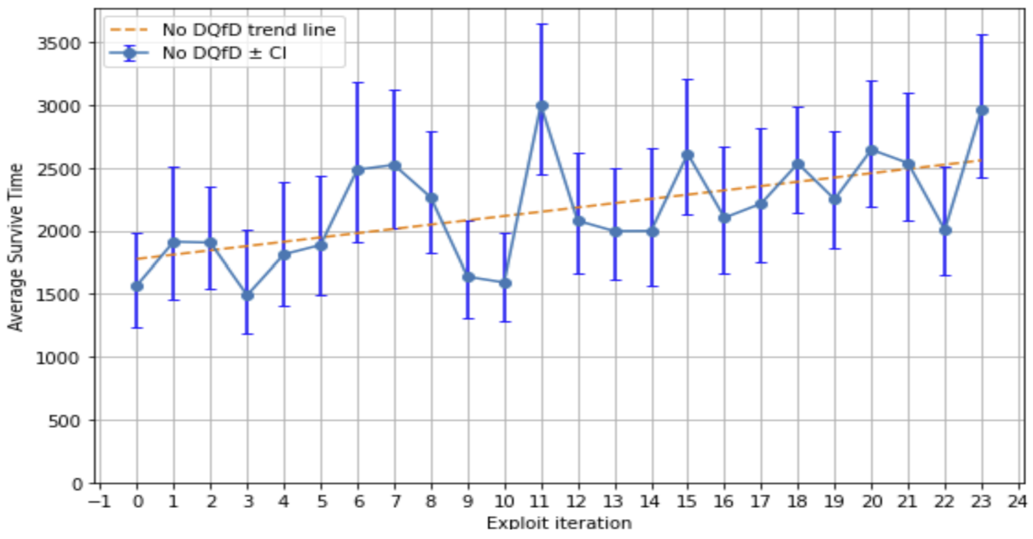


Figure 6.14: Survive Time in exploit mode monitored during the training. The x- axis represent the exploit iteration, while the y-axis reports the average survive time and confidence interval: each point is the average over 50 episodes. The confidence intervals are computed with bootstrapping method.

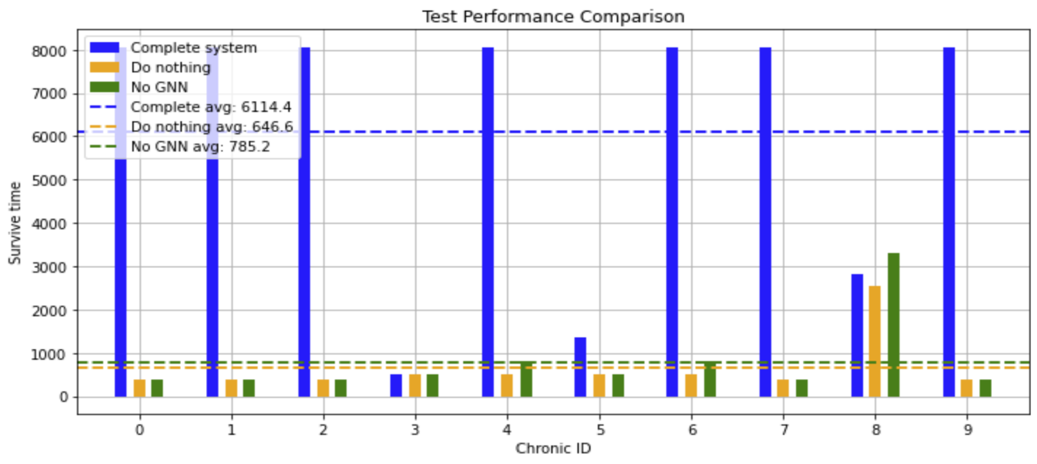


Figure 6.15: Test performance comparison between the complete model and the variant without GNN.

Both plots are shown in logarithmic scale. As observed, the loss magnitude of the model without the GNN is consistently several orders of magnitude higher than that of the complete model (see Section 6.2.3). This reflects the model's inability to approximate the Q-function effectively when the input lacks structured relational features, further emphasizing the importance of the GNN for stable learning and accurate value estimation.

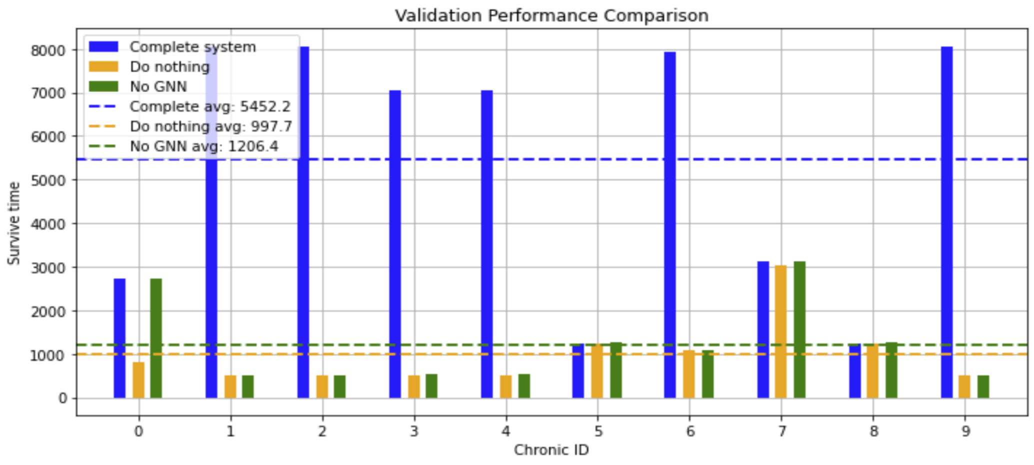


Figure 6.16: Validation performance comparison between the complete model and the variant without GNN.

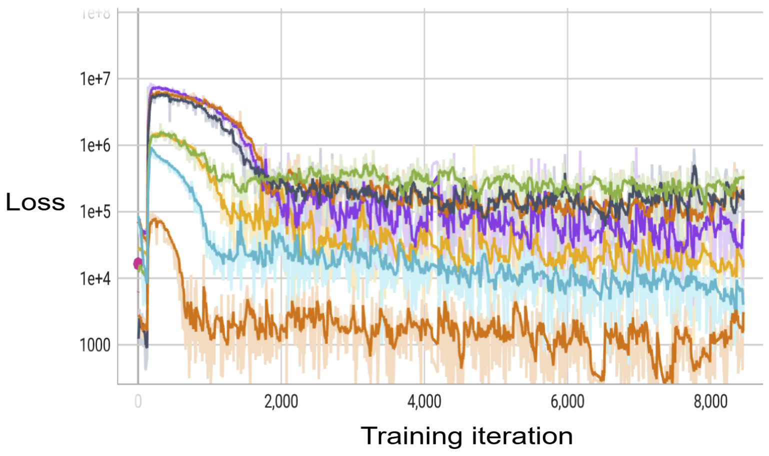


Figure 6.17: Loss during the demonstration pre-training phase (log scale).

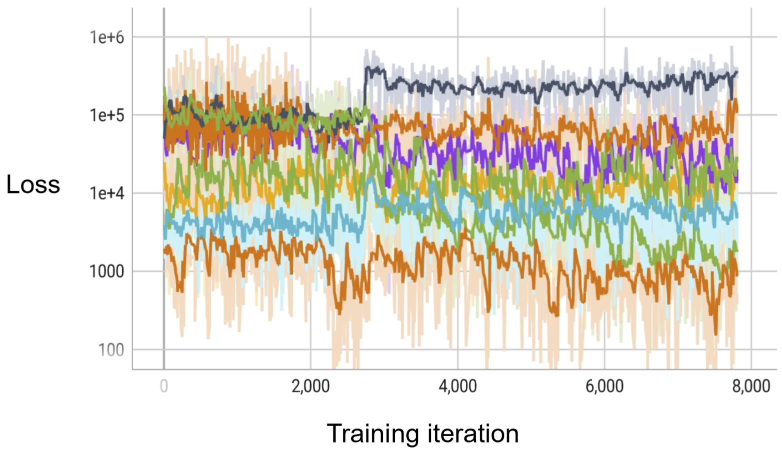


Figure 6.18: Training loss during reinforcement learning (log scale).

6.3.3. RL Manager

To assess the advantages of using a learnable RL manager, we compared it with CAPA, a rule-based manager adopted in several distributed control approaches [52, 64]. The CAPA manager selects, at each risky time step, the agent controlling the most overloaded power line. All other components of the model are kept identical.

Since CAPA relies on a rule-based coordination policy, the expert demonstrations provided to the line agents are filtered to keep only those experiences that are consistent with the rule-based logic, i.e., experiences in which the acting line agent is the one of the overloaded line.

The CAPA-based model was evaluated on the validation set and during the training. It was not tested on the final test set, as its performance did not indicate strong potential. The same considerations about the different computational environment done in Section 6.3.1 applies also here. This explains the slight discrepancy in the performance of the complete model.

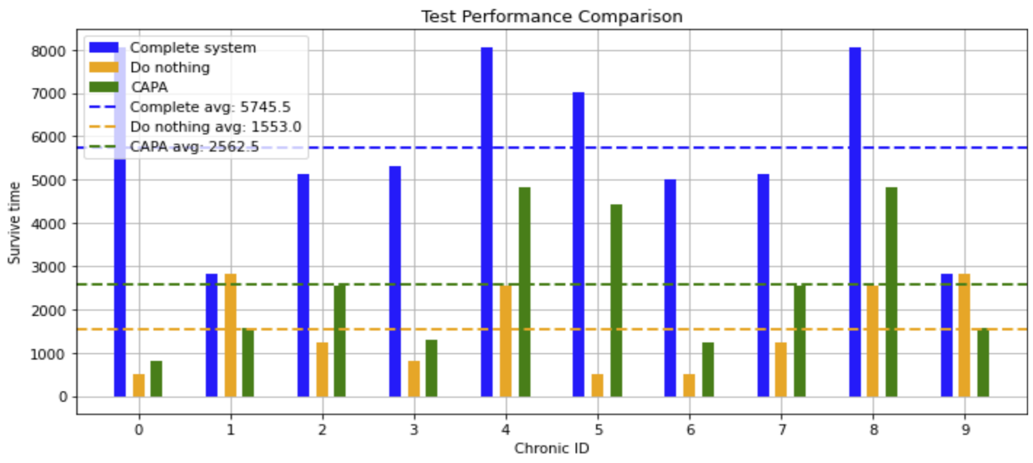


Figure 6.19: Validation performance comparison between the complete system (with RL manager), the CAPA manager, and the do-nothing baseline.

Figs. 6.19 and 6.20 show that while CAPA outperforms the do-nothing baseline, its performance remains significantly below that of the complete system equipped with the learned RL manager. These results suggest that the rule-based CAPA manager limits the overall system's performance.

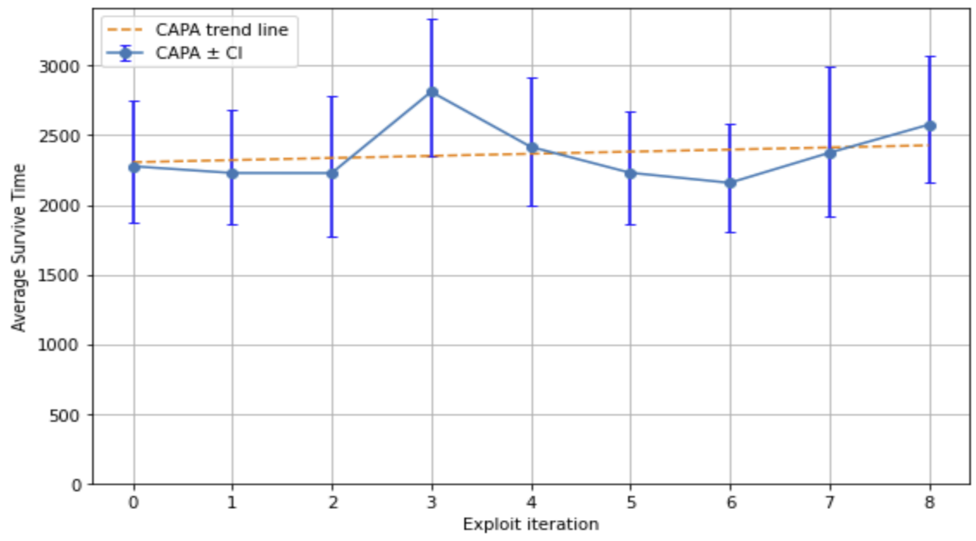


Figure 6.20: Exploit performance comparison over training iterations, evaluated on 50 training chronics at each exploit iteration. Confidence intervals are shown for both models.

6.3.4. Effect of Reward Shaping

To evaluate the impact of reward shaping, we compare the full model with a variant in which the shaped components of the reward function are removed. The rest of the architecture and training procedure remains unchanged.

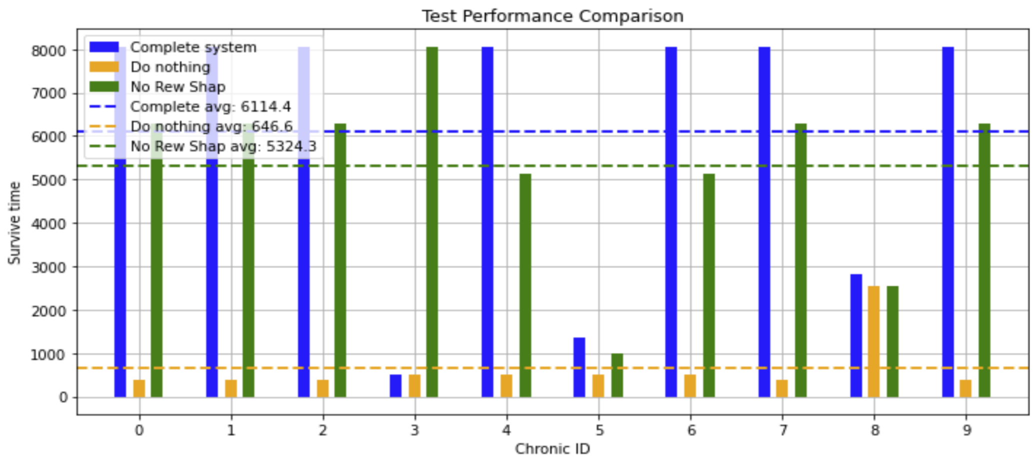


Figure 6.21: Test performance comparison between the complete system and the version without reward shaping.

As shown in Figs. 6.21 and 6.22, the absence of reward shaping has a minimal effect on the model's performance across both the test and validation sets. This behavior aligns with expectations, given the relatively small scale of the power grid used in the experiments.

However, in larger and more complex grid topologies, where diluted rewards could be

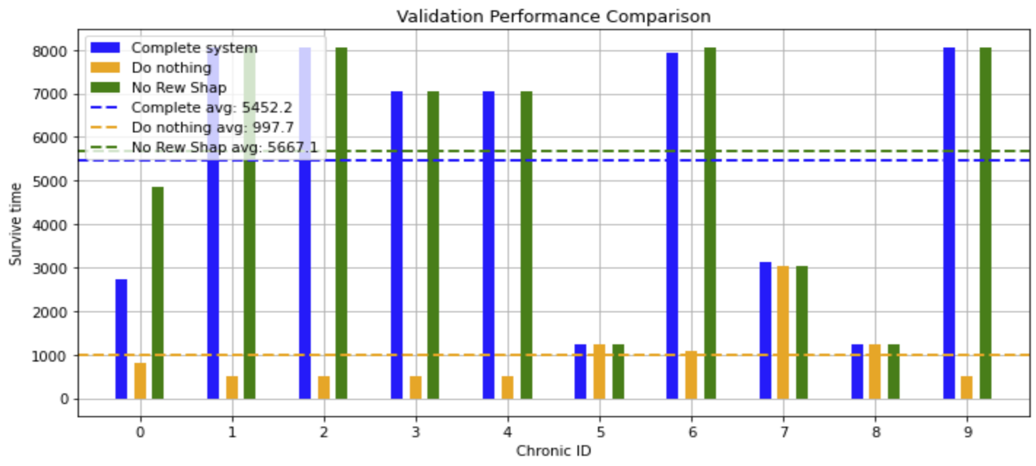


Figure 6.22: Validation performance comparison between the complete system and the version without reward shaping.

problematic, reward shaping may play a more critical role. Investigating its impact in such settings remains an important direction for future work.

6.3.5. Effect of Action Filtering

In order to assess the importance of action filtering, we conducted an ablation study by training the system without any form of action reduction technique. The performance of the resulting model is summarized in Figs. 6.23, 6.24, and 6.25. These show the validation, test, and exploit-mode performances, respectively.

As already observed in prior literature, the absence of action reduction techniques significantly hampers the learning process [25, 53]. RL algorithms operating over large discrete action spaces—such as those encountered in power grid control tasks—are notoriously difficult to train effectively without any form of action pruning or guidance. In this experiment, the policy trained without action filtering fails to converge, and even degrades over time, despite being trained for a number of episodes five times larger than that used for the complete system. This phenomenon, where performance initially deteriorates before (possibly) improving, is not unusual in RL—particularly in high-dimensional environments. Moreover, a version of the model not employing action reduction techniques likely requires more careful hyperparameter tuning, as the optimal settings may differ significantly from those used for the complete system. Nonetheless, the observed degradation highlights the critical role of action filtering in making training both feasible and sample-efficient.

The same considerations about the different computational environment done in Section

6.3.1 applies also here. This explains the slight discrepancy in the performance of the complete model.

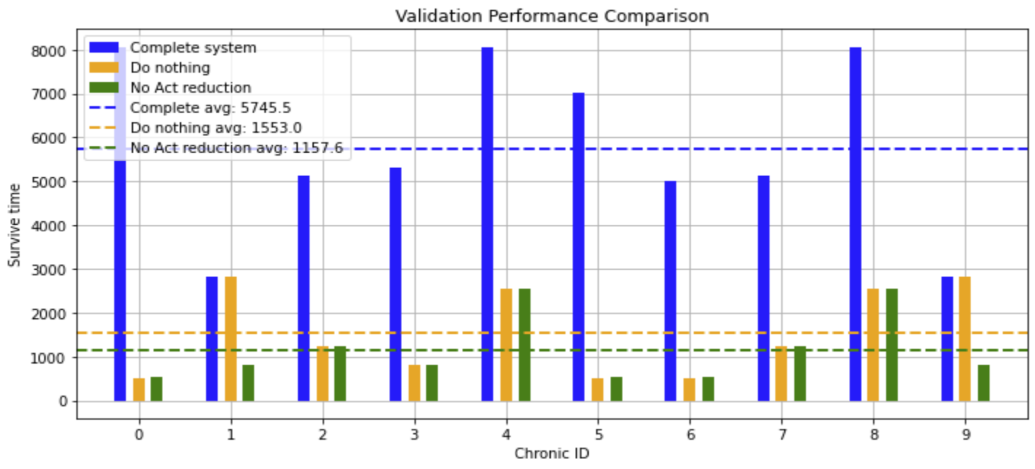


Figure 6.23: Validation performance comparison between the complete system and the version without action filtering. The x-axis represents the chronic ID; the y-axis reports the survival time.

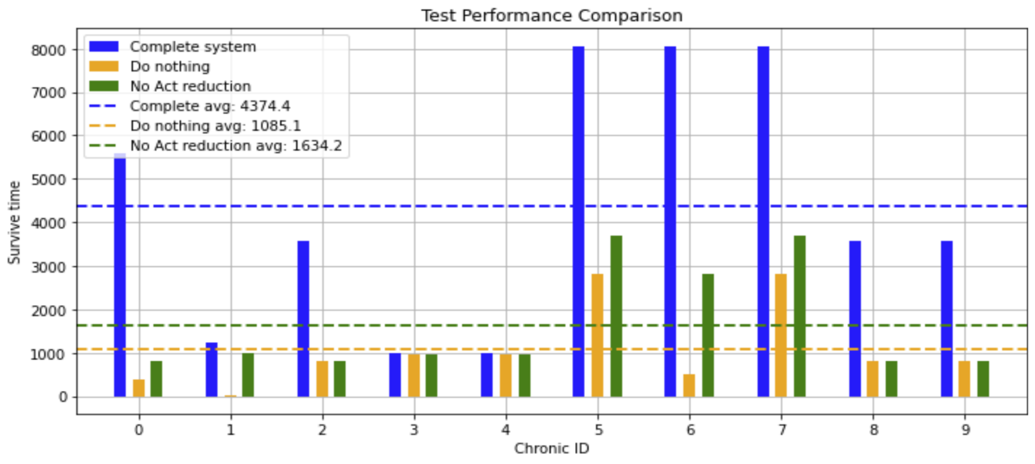


Figure 6.24: Test performance comparison between the complete system and the version without action filtering.

6.4. Computational Requirements

Simulating a power grid is, by nature, a computationally intensive task. In this section, we analyze the computational requirements of the proposed system from multiple perspectives. First, we provide an overview of the hardware specifications used for the experiments. Then, we report the average computational time required to simulate a single environment step, excluding any overhead introduced by the agent's decision-making

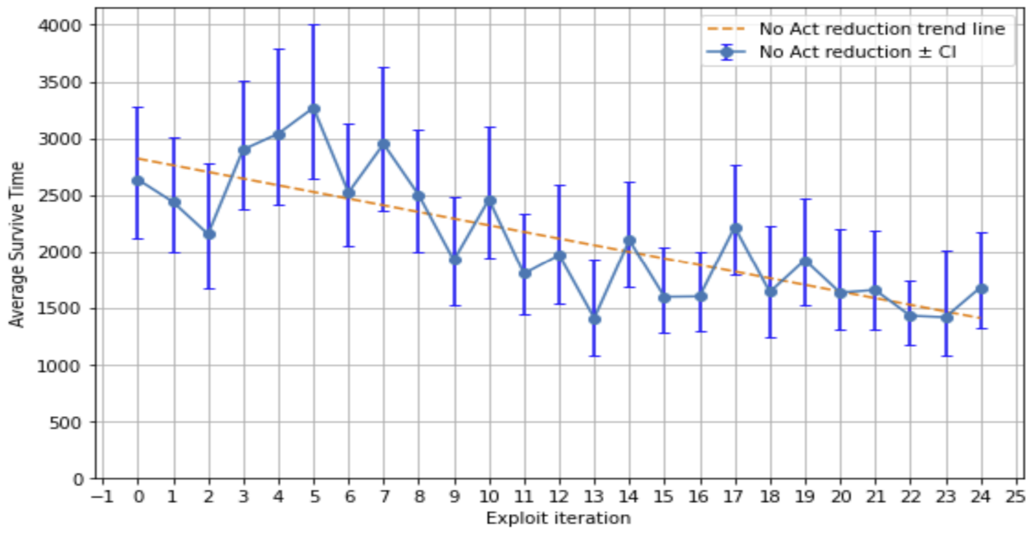


Figure 6.25: Average survival time in exploit mode for the model trained without action filtering. Confidence intervals are computed via bootstrapping over 50 episodes.

logic. Next, we evaluate the inference efficiency of the trained model by measuring the time needed to select a single action during exploitation, and compare it to the inference time of the expert agent. Finally, we present a summary of the total runtime for the experiments described in Section 6.2.

6.4.1. Hardware Specifications

The experiments presented in this thesis were carried out on two distinct hardware configurations: a local machine and a remote shared server. The specifications of both computational devices are detailed in Tab. 6.2.

Component	**Local Machine (Dell XPS 15 9500)**	**Remote Server (Linux)**
CPU	Intel(R) Core(TM) i7-10750H @ 2.60GHz (6 cores / 12 threads)	Dual Intel(R) Xeon(R) E7-8880 v4 @ 2.20GHz (44 physical cores / 88 threads)
RAM	32 GB DDR4	94 GiB
Operating System	Windows 11 Pro, 64-bit	Linux, 64-bit

Table 6.2: Hardware configurations used for training and experimentation.

Each reported computational time will be accompanied by a specification of the hardware setup used for the corresponding experiment.

6.4.2. Plain Grid Simulation

To establish a reference for the computational cost of pure simulation, the average execution time per timestep when playing the simple *Do-Nothing* policy was measured. This setup reflects the cost of simulating the power grid without any decision-making overhead from an agent.

The experiment consisted of 10 complete episodes. For each episode, the total simulation time was divided by the number of timesteps survived to compute the average time per step. The resulting per-episode averages were then aggregated to compute the overall mean and standard deviation across episodes.

The results show that simulating a single timestep under the *Do-Nothing* policy requires, on average, **0.1097 seconds**, with a standard deviation of **0.0058 seconds**. Therefore, simulating an entire episode of 8064 timesteps requires on average: **14.74 minutes**.

6.4.3. Inference time

The inference time of the proposed approach includes both the time required by the manager to select the appropriate line agent and the time taken by the selected line agent to determine the action to execute. The inference time is highly affected by the action space reduction procedure that involves a single simulation step. When the technique is applied, the average inference time is **0.187 seconds** with a standard deviation of **0.145 seconds**. In contrast, without using the technique, the inference time drops to an average of **0.0097 seconds** with a standard deviation of **0.0026 seconds**. These inference times are compared against those of the expert agent that was used to collect demonstration data, whose average inference time is **2.56 seconds** with a standard deviation of **0.223**. These results are summarized in the Tab. 6.3.

Table 6.3: Inference Time Comparison Across Different Agents.

Model	**Inference Time (avg $\pm$ std) [s]**
Proposed Model (w/ Action Space Reduction)	0.187 ± 0.145
Proposed Model (w/o Action Space Reduction)	0.0097 ± 0.0026
Expert Agent	2.56 ± 0.223

6.4.4. Experiments Runtime Summary

The actual computational time was monitored using tensorboard [51] for all experiments discussed in this chapter, including both the Results and Ablation Study sections. The results are provided in Tab. 6.4.

Table 6.4: Runtime summary of the experiments.

Configuration	**Computational time**	**Device**
No DQfD	15.48 days	Remote Server
No GNN	2.889 days	Local Machine
No RL Manager (CAPA)	11.57 days	Remote Server
No Reward Shaping	3.631 days	Local Machine
No Action Filtering	15.49 days	Remote Server
Complete System	5.642 days	Local Machine

7 | Dynamic Power Grid Clustering

A minor contribution was given towards the implementation of a technique for dynamic clustering of substations within a power grid. The goal was to develop an adaptive clustering of substations, i.e., a clustering that evolves over time as the topology of the power grid changes. The proposed solution employs a Transformer Autoencoder to generate embeddings from the time series associated with each substation. These embeddings are then used as input for the clustering process. This chapter illustrates the key operations and components of the proposed solution. It begins by detailing how the evolution of the power grid is encoded as time series associated with the substations. Additionally, the specific data collection procedure is provided. Next, it presents the Transformer Autoencoder architecture along with the training procedure. Finally, the clustering procedure and the experimental results are presented.

7.1. Substations' Time Series

The proposed representation allows to model substations as evolving components within a power grid. Referring to section 3.1, the features considered here are the three bus features: Active power (`p`), Reactive power (`q`) and Voltage (`v`), and in particular their evolution over time. A single substation at time t can be represented as the concatenation of its two buses vectors, as illustrated in Fig. 7.1.

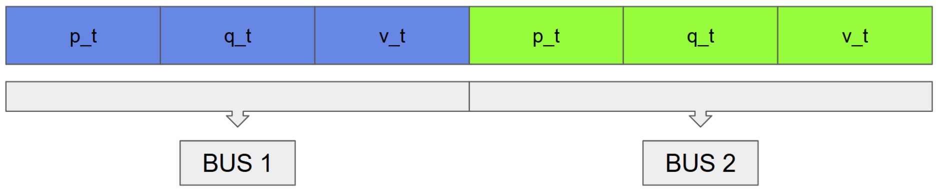


Figure 7.1: Representation of a substation at time t. It is the concatenation of its bus_1 and bus_2 vectors.

Thus, a single substation in an episode of T timesteps can be represented by stacking its vectors for each time step, as in Fig. 7.2.

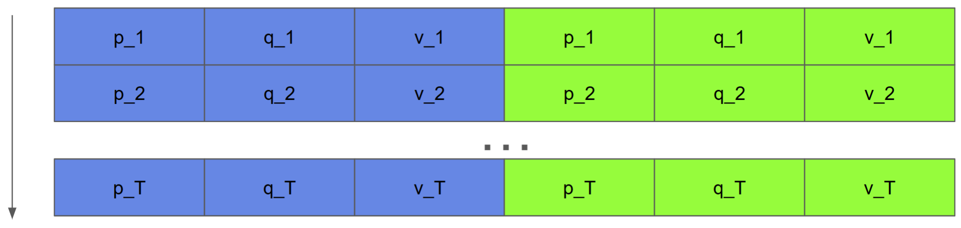


Figure 7.2: Representation of a substation in an episode of T timesteps.

The evolution of a power grid ,with N substations, over an episode of T timesteps, is defined by the representation of all its substations, as reported in Fig. 7.3.

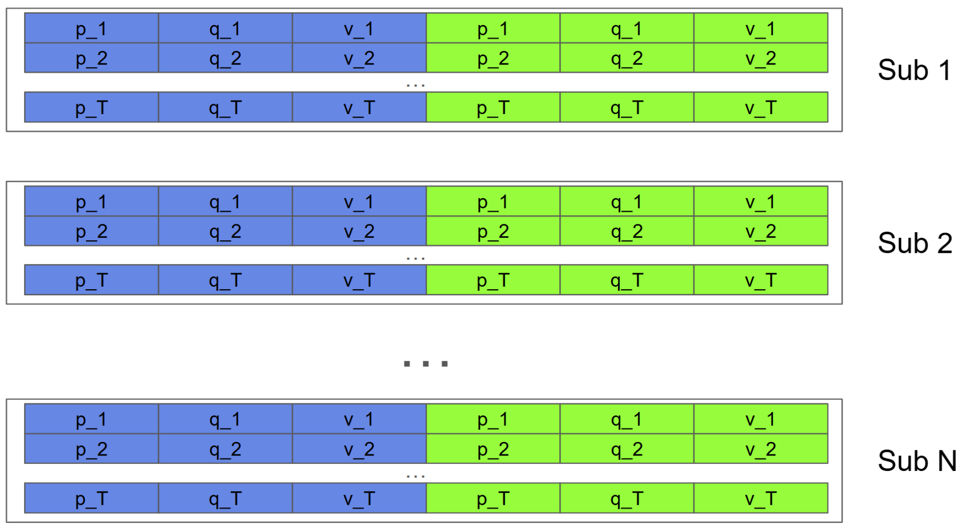


Figure 7.3: Representation of a N substations in an episode of T timesteps.

In conclusion, the N substation of a power grid are defined as timeseries over their bus information.

7.2. Data Collection

The previous section defines how to represent substations within an episode of T rounds. However, it remains to be defined how to represent power grids across multiple episodes of varying durations and, more importantly, how these episodes were generated—that is, what policy was used to produce the corresponding time series. As in [29], a random policy was used to collect the data: each episode was generated by executing random actions for up to 30 timesteps. After the 30th step, the episode was terminated regardless of the state of the power grid. Zero-padding was applied at the beginning of each time series to ensure that all the sequences were extended to a uniform length of 30 timesteps. To explain the zero-padding, let's consider substation k over two different episodes, one lasting T_i timesteps and the other lasting T_j timesteps. The time series corresponding to the former will be T_i elements long, while the one corresponding to the latter will be T_j elements long. Thus, a number of zero elements required to reach a total length of 30 timesteps is stacked at the beginning of each time series, as illustrated in Fig. 7.4.

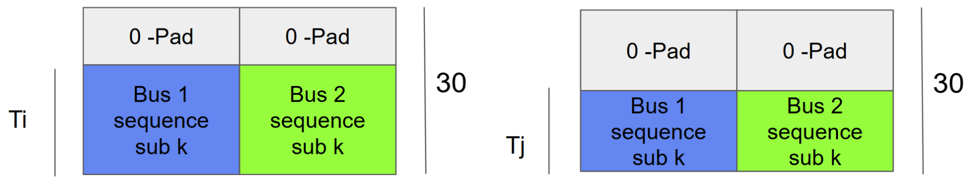


Figure 7.4: Explanatory example: zero-padding application.

For a correct evaluation, 984 Training chronics were used to produce Training Data and 10 Test chronics to assess the correctness of the model, as shown in Fig. 7.5.

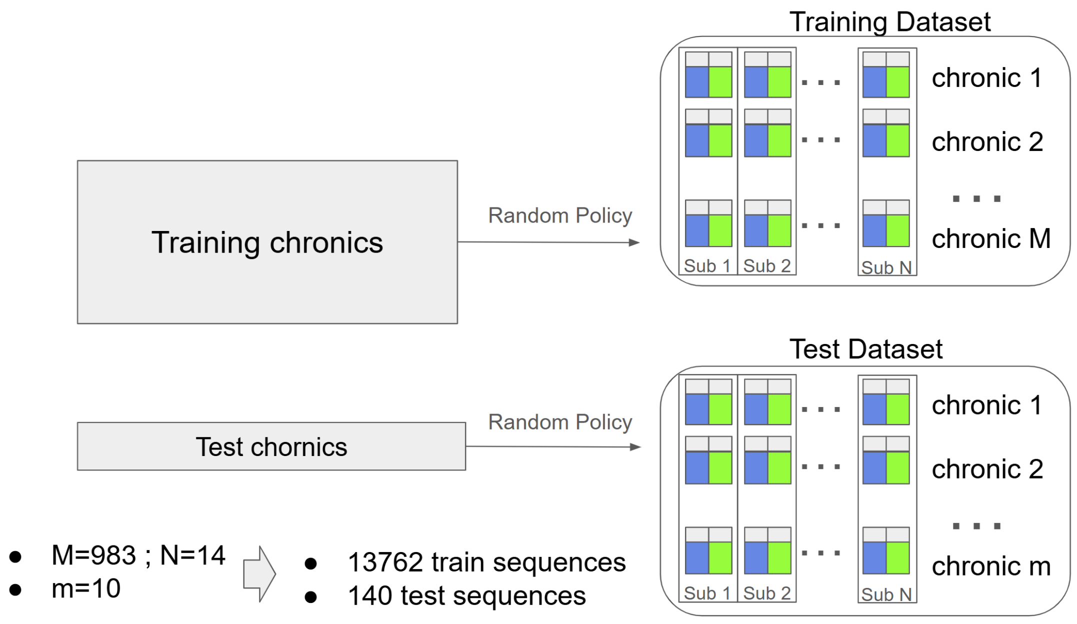
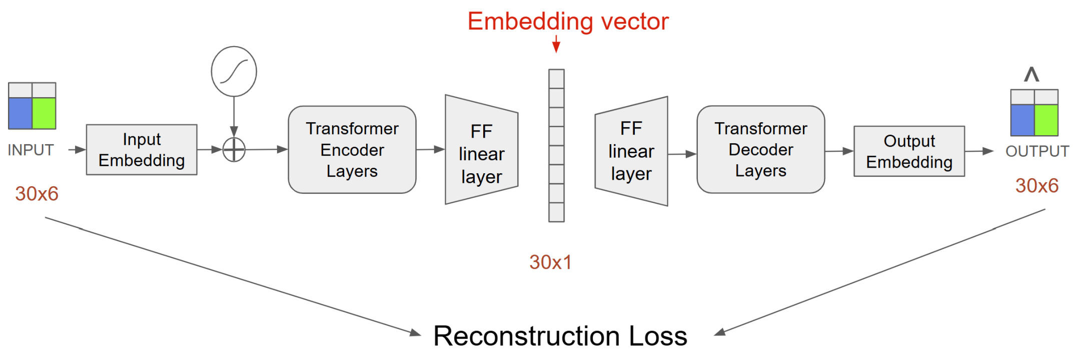


Figure 7.5: Data collection procedure: random policy played on training chronics to produce training data and random policy played on test chronics to produce test data. A single time series is obtained for each substation in a single episode. Thus, having 14 substation in the power grid, we obtain 984×14 training time series and 10×14 test time series.

7.3. Transformer Autoencoder

The Transformer Autoencoder is trained to reconstruct input sequences by learning a process of compression and decompression. Once the model achieves sufficient reconstruction accuracy, the decoder is discarded, and only the encoder is retained. The result is a robust, learned mechanism, for compressing time series data. The overall training procedure is summarized in Fig.7.6.



$$\frac{1}{B}\sum_{i=1}^{B} MSE(sequence_i, pred\_sequence_i)$$

Figure 7.6: Transformer Autoencoder structure overview.

More specifically, the Transformer Autoencoder is trained to minimize the Mean Squared Error (MSE) between the input and reconstructed sequences across each batch. The training performance was monitored by tracking both training and validation losses. Specifically, the training sequences were split into 80% for training and 20% for validation. The validation set was used to evaluate the model's generalization ability throughout the training process, the results are provided in section 7.5. After enough training, the decompression component is removed and only the compression one is kept, as shown in Fig. 7.7.

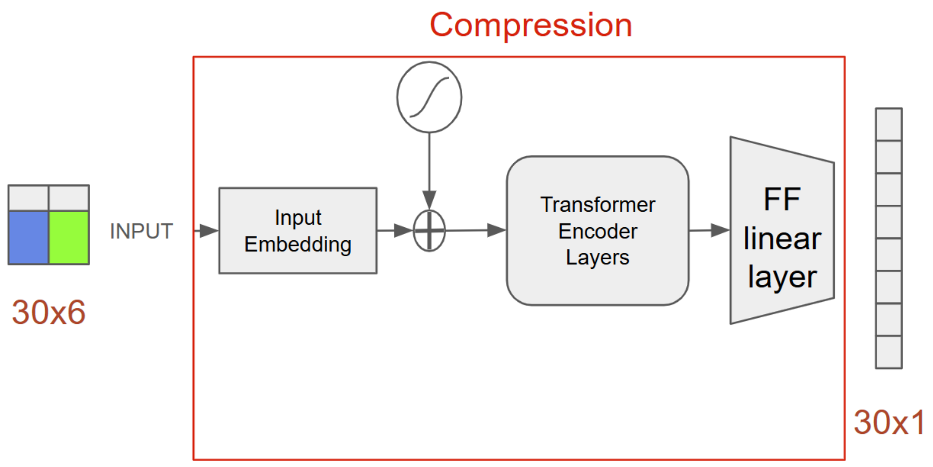


Figure 7.7: Transformer Autoencoder compression.

7.4. Clustering of Substations

The evolution of the N substations within a single chronic can be represented as N distinct time series, as explained in section 7.1. These N time series are individually compressed using the Transformer Autoencoder from section 7.3, producing N corresponding data points in a 30-dimensional latent space. To identify groups of substations with similar temporal behavior, hierarchical clustering and DBSCAN are then applied to these compressed representations. The overall procedure is illustrated in Fig. 7.8.

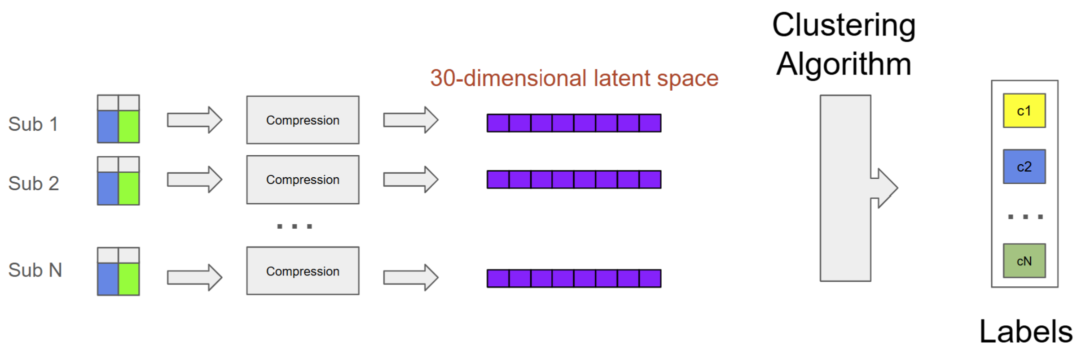


Figure 7.8: Clustering procedure overview.

Both hierarchical clustering and DBSCAN are applied to simultaneously ensure consistency of the results and to test different methodologies.

7.5. Experimental Results

The procedure defined above was applied to the power grid *"l2rpn_ case14_ sandbox"*, counting 14 substations. In particular the method was tested over time series produced on the 10 test chronics, and the resulting clusterings were compared to the ones provided by the static factorization from [29], assumed to be the ground truth. More precisely, the clustering and the ground truth were compared by means of the Normalized Mutual Information and Fowlkes-Mallows score. For completeness, the ground truth provided by [29], is shown in Fig. 7.9.

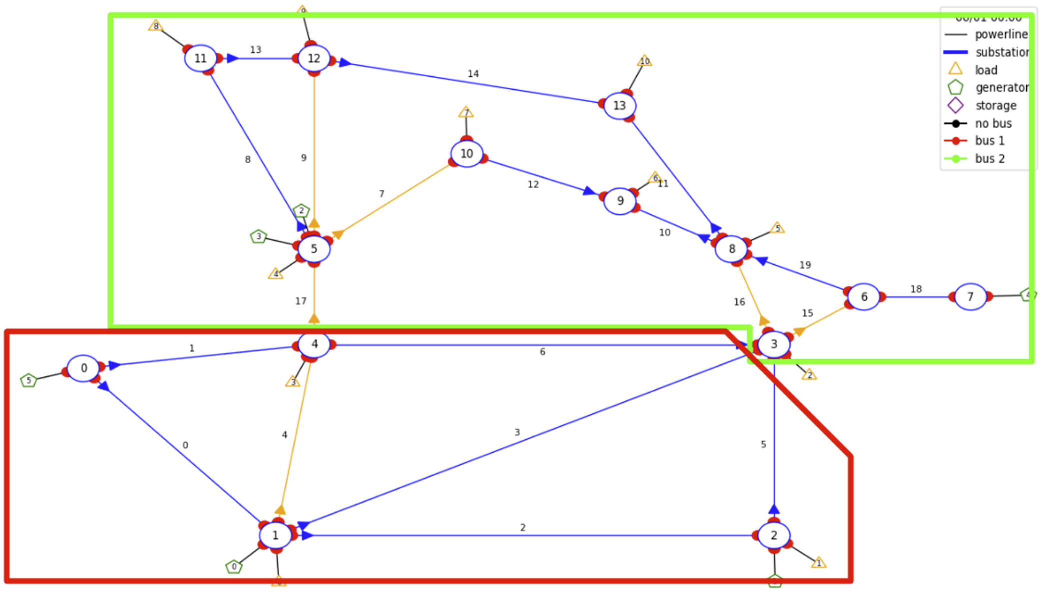


Figure 7.9: Ground truth clustering labels, provided by [29].

The train and validation loss of the Transformer Autoencoder are shown in Fig. 7.10.

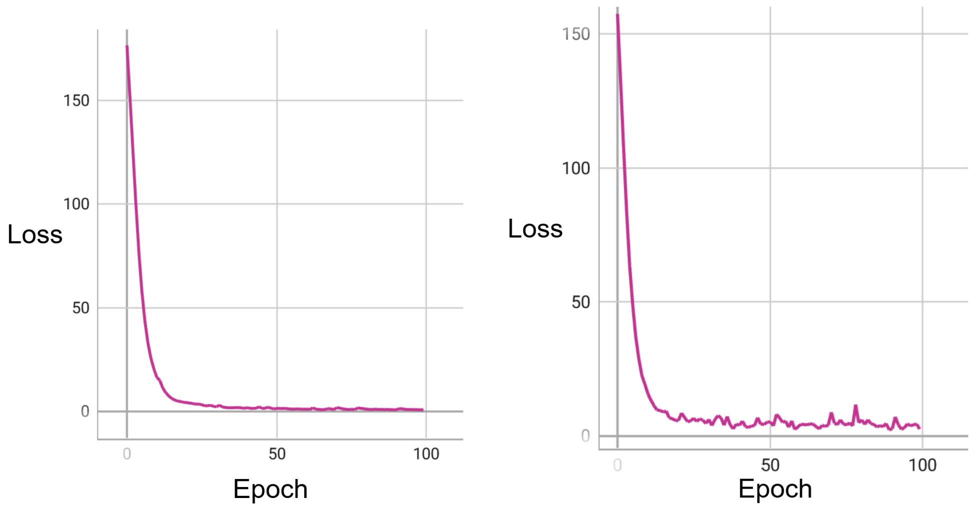


Figure 7.10: On the left the training loss, on the right the validation loss of the Transformer Autoencoder.

7.5.1. Hierarhcical clustering

The result of hierarchical clustering for a single test chronic (test chronic 0) is provided in Fig. 7.11.

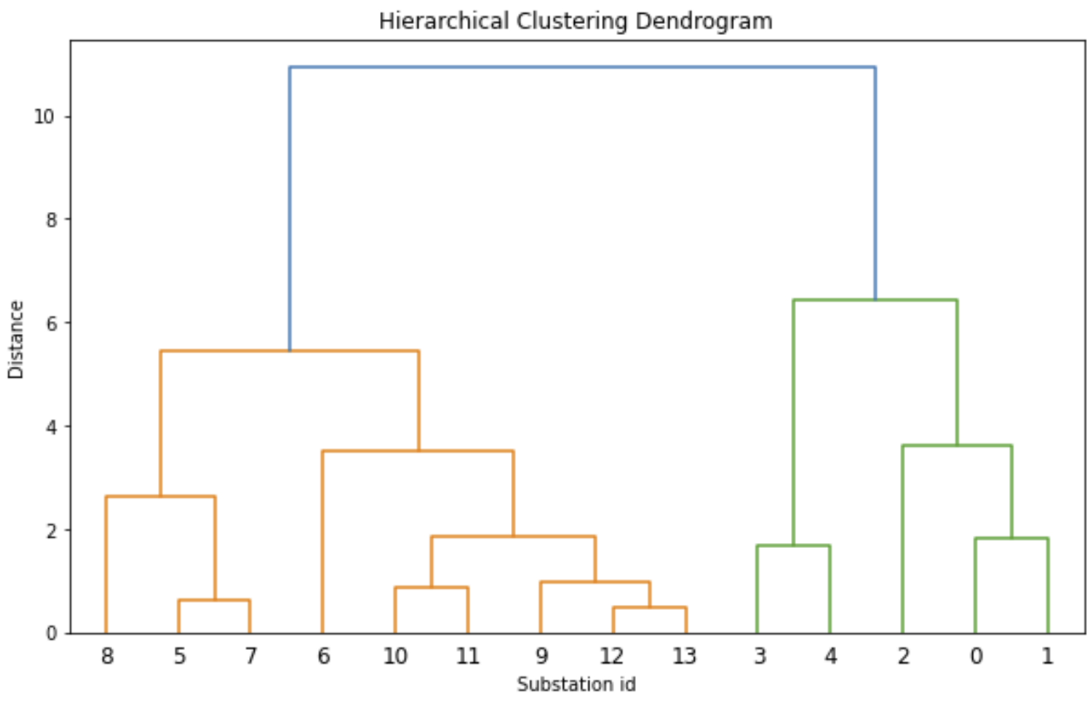


Figure 7.11: Dendrogram for test chornic 0.

The dendrogram shows the largest increase in distance when the number of clusters is reduced to one, suggesting that two is the most appropriate number of clusters—consistent with the ground truth. The dendrogram is then cut to obtain two clusters, and the resulting assignments are compared to the ground truth, as reported in Table 7.1. The results show that, the obtained clusterings agree quite well with the ground truth.

Chronic id	**Normalized Mutual Information**	**Fowlkes-Mallows Score**
0	0.671273	0.867132
1	0.671273	0.867132
2	0.671273	0.867132
3	0.671273	0.867132
4	0.671273	0.867132
5	0.671273	0.867132
6	0.671273	0.867132
7	0.671273	0.867132
8	0.671273	0.867132
9	0.671273	0.867132

Table 7.1: Evaluation of hierarchical clustering performance using Normalized Mutual Information and Fowlkes-Mallows Score across test chronics.

The resulting clustering, which was identical across all test chronics, is shown in Fig. 7.12. It differs from the ground truth in only one assignment: substation 3.

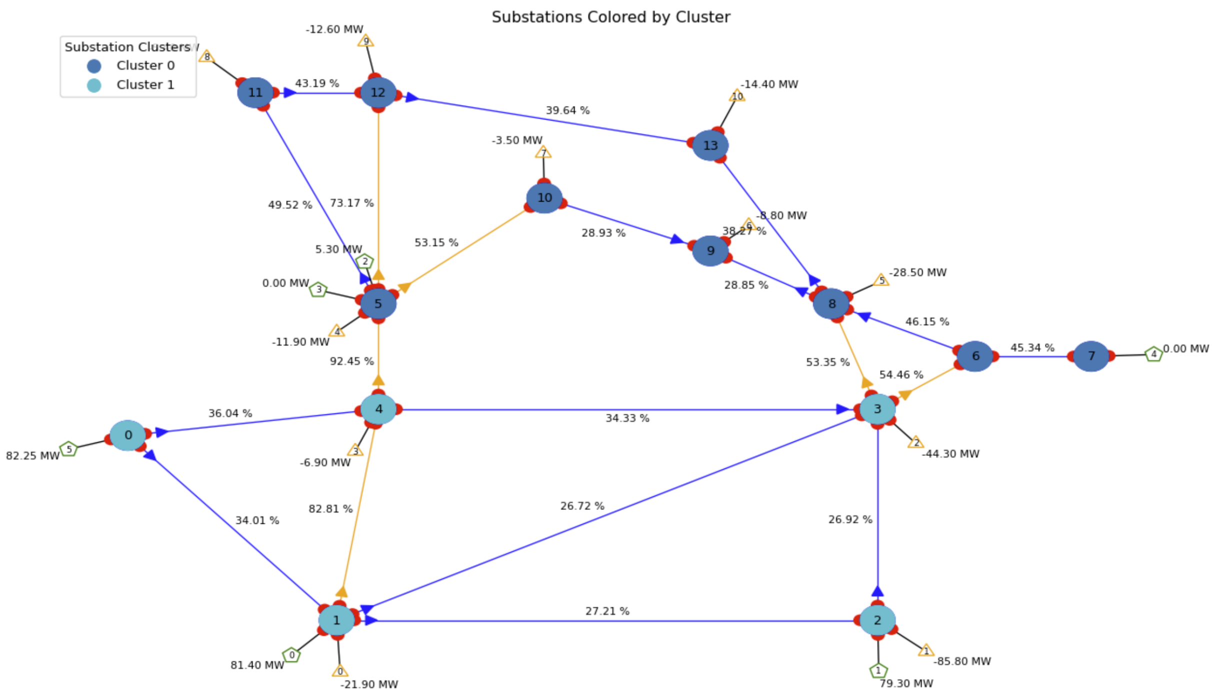


Figure 7.12: Clustering computed for test chronic 0.

7.5.2. DBSCAN

The results obtained using DBSCAN are presented in Table 7.2, and once again, they show good alignment with the ground truth.

Chronic id	Normalized Mutual Information	Fowlkes-Mallows Score
0	0.669958	0.863475
1	0.671273	0.867132
2	0.569167	0.772328
3	0.671273	0.867132
4	0.671273	0.867132
5	0.671273	0.867132
6	0.671273	0.867132
7	0.669958	0.863475
8	0.569167	0.772328
9	0.671273	0.867132

Table 7.2: Evaluation of DBSCAN clustering performance using Normalized Mutual Information and Fowlkes-Mallows Score across test chronics.

The cluster assignments produced by DBSCAN match those from hierarchical clustering in all cases except for test chronics 0 and 7—which share the same labels—and 2 and 8, which also share identical labels. The resulting clusterings for these four chronics are shown in Fig. 7.13, where DBSCAN identified three distinct clusters in one case and an outlier in the other.

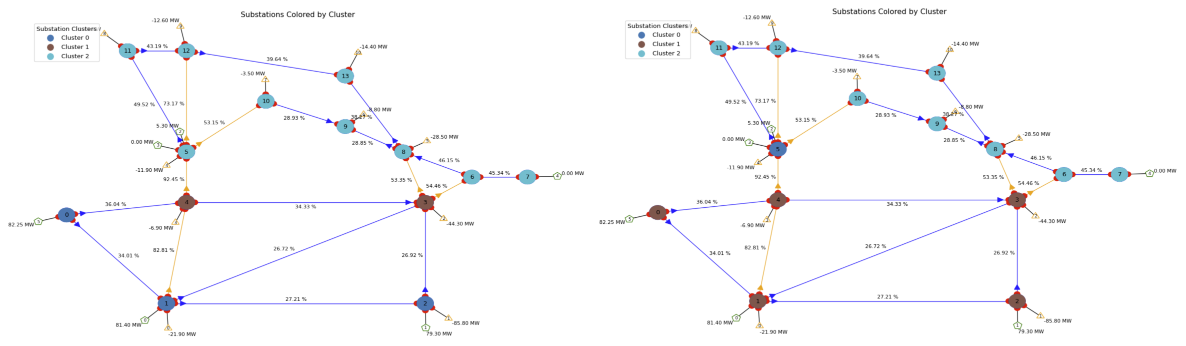


Figure 7.13: On the left the clustering obtained for test chronics 0 and 7, which yield three distinct clusters. On the right the clustering obtained for text chronics 2 and 8. Here, substation 5 represents an outlier.

8 | Conclusions and Future Developments

8.1. Summary of the Work

This thesis deals with some of the main challenges faced by today's power grids. The proliferation of renewable energy sources, which are highly variable and unpredictable, along with the expanding size of power networks, has made power grid management significantly more complex. The action and observation spaces become vast, complex and often unmanageable by traditional control systems. To address these challenges, a novel Graph-based Multi-Agent Reinforcement Learning algorithm for real-time scalable power grid control is proposed. The method consists of a network of distributed low-level agents, each controlling a different powerline, coordinated by a high-level manager. Each low-level agent operates solely on its associated powerline and receives a local observation preprocessed by a shared Graph Neural Network (GNN). This procedure makes it possible to decompose the action space and the observation space simultaneously, enabling a fully scalable solution. To further improve learning efficiency and stability, the framework incorporates Deep Q-Learning from Demonstrations and potential-based reward shaping techniques.

8.2. Main Findings

The results obtained in the Grid2Op simulation environment demonstrate the effectiveness of the proposed Graph-based MARL framework. The method significantly outperforms the standard Do-Nothing baseline in terms of survival time, while also showing much lower inference time compared to the Expert System. These outcomes confirm the scalability and efficiency of the architecture, particularly in scenarios requiring rapid decision-making across large-scale networks. The main contribution of this work lies in the novel application of Graph Neural Networks to decompose the observation space while constructing informative local observations for the decentralized agents. Both the conversion of the environment's observation into a homogeneous graph structure and the use of a shared Graph Neural Network at the base of all distributed agents are carefully motivated design choices. Together, they contribute significantly to the strength and coherence of the proposed approach, enabling structured local observations and efficient information sharing. The combination of Deep Q-Learning from Demonstrations and potential-based reward shaping also plays an important role in the proposed framework. Deep Q-Learning from Demonstrations proves to be essential for reaching competitive performance within a limited amount of time, while potential-based reward shaping appears to be a promising research direction for addressing the problem of diluted rewards in large-scale power networks.

8.3. Limitations

While the proposed approach has its strength, it still faces some limitations. Firstly, although the low-level agents receive local observations, the manager needs access to a complete global view of the environment to make decisions. This global view limits the scalability of the overall architecture. To overcome this limitation, multi-layer architectures emerge as a promising research direction. In these architectures, multiple high-level managers control different, independent portions of the power grid, identified through power grid decomposition methods.
Secondly, applying imitation learning assumes the availability of a dataset of expert demonstrations, which is not always the case. This is particularly problematic when dealing with large, unmanageable power grids for which expert strategies are unknown.
Lastly, while the model effectively decomposes the action space, it still relies on a single-step simulation at runtime to perform a greedy action-space reduction. This step likely helps to achieve good performance in a shorter amount of time, but it may not be strictly necessary. Alternative action space reduction techniques should be explored. A promising direction is the N-1 criterion proposed by [8, 53]. This criterion statically reduces the action space by nearly half by retaining only those actions that preserve N-1 stability, ensuring that the power grid remains operational even if a random powerline is disconnected.

8.4. Future Developments

Future developments should focus on addressing the limitations discussed in the previous section.

To eliminate the need for runtime simulation and significantly reduce inference time, alternative action space reduction techniques should be explored. One promising technique is the previously mentioned N-1 criterion. This criterion retains only configurations that ensure redundancy in line connections, meaning at least two lines connect a pair of buses. This redundancy ensures that if one line is disconnected, the other can still maintain the connection.

Multi-layer architectures should be implemented and evaluated on large-scale power grids, alongside the exploration of different grid decomposition methods. This involves identifying independent subgrids, each managed by a different high-level manager. A rule-based top-level logic could then determine which subgrid requires intervention during critical situations.

Lastly, an interesting research direction introduced by the use of a single shared GNN is GNN-based transfer learning. This involves using a GNN trained on a smaller power grid as initialisation for a model operating on a larger grid. This approach could significantly accelerate convergence in large-scale scenarios.

Bibliography

[1] J. Adamczyk, V. Makarenko, S. Tiomkin, and R. V. Kulkarni. Bootstrapped reward shaping, 2025.

[2] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications, 2021.

[3] R. Bacher and H. Glavitsch. Network topology optimization with security constraints. *IEEE Transactions on Power Systems*, 1(4):103–111, 1986.

[4] R. Baudin and the Grid2Op Contributors. Grid2op: Training an agent - getting started notebook, 2024. URL `https://github.com/Grid2op/grid2op/blob/master/getting_started/04_TrainingAnAgent.ipynb`.

[5] D. P. Bertsekas. *Neuro-dynamic programming*, pages 2555–2560. Springer US, 2009.

[6] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks?, 2022.

[7] M. de Jong, J. Viebahn, and Y. Shapovalova. Generalizable graph neural networks for robust power grid topology control, 2025.

[8] B. de Mol, D. Barbieri, J. Viebahn, and D. Grossi. Centrally coordinated multi-agent reinforcement learning for power grid topology control, 2025.

[9] B. Donnot. Grid2op: A testbed platform to model sequential decision making in power systems, 2020. URL `https://github.com/Grid2Op/grid2op`.

[10] M. Dorfer, A. R. Fuxjäger, K. Kozak, P. M. Blies, and M. Wasserer. Power grid congestion management via topology optimization with alphazero, 2022.

[11] J. Fan, Z. Wang, Y. Xie, and Z. Yang. A theoretical analysis of deep q-learning, 2020.

[12] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning, 2016.

[13] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry, 2017.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[15] T. Guardian. What caused the blackout in spain and portugal—and did renewable energy play a part?, 2025.

[16] M. Hassouna, C. Holzhüter, P. Lytaev, J. Thomas, B. Sick, and C. Scholz. Graph reinforcement learning for power grids: A comprehensive survey, 2024.

[17] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps, 2017.

[18] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep q-learning from demonstrations, 2017.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[20] I. Hrgović, I. Pavic, M. Puljiz, and M. Brcic. Deep reinforcement learning-based approach for autonomous power flow control using only topology changes. *Energies*, 15:6920, 09 2022.

[21] S. Iqbal and F. Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970. PMLR, 09–15 Jun 2019.

[22] S. Kar, J. M. F. Moura, and H. V. Poor. Qd-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations. *IEEE Transactions on Signal Processing*, 61(7):1848–1862, 2013.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[24] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017.

[25] T. Lan, J. Duan, B. Zhang, D. Shi, Z. Wang, R. Diao, and X. Zhang. Ai-based autonomous line flow control via topology adjustment for maximizing time-series atcs, 2019.

[26] J. Li, Z. Wei, J. Dan, J. Zhou, Y. Zhu, R. Wu, B. Wang, Z. Zhen, C. Meng, H. Jin, Z. Zheng, and L. Chen. Hetero^{2}net: Heterophily-aware representation learning on heterogenerous graphs, 2023.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019.

[28] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018.

[29] G. Losapio, D. Beretta, M. Mussi, A. M. Metelli, and M. Restelli. State and action factorization in power grids, 2024.

[30] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.

[31] B. Manczak, J. Viebahn, and H. van Hoof. Hierarchical reinforcement learning for power network topology control, 2023.

[32] Q. Mao, Z. Liu, C. Liu, and J. Sun. Hinormer: Representation learning on heterogeneous information networks with graph transformer, 2023.

[33] A. Marot, B. Donnot, S. Tazi, and P. Panciatici. Expert system for topological remedial action discovery in smart grids. In *MedPower*, 2018.

[34] A. Marot, B. Donnot, C. Romero, L. Veyrin-Forrer, M. Lerousseau, B. Donon, and I. Guyon. Learning to run a power network challenge for training topology controllers, 2019.

[35] A. Marot, B. Donnot, G. Dulac-Arnold, A. Kelly, A. O'Sullivan, J. Viebahn, M. Awad, I. Guyon, P. Panciatici, and C. Romero. Learning to run a power network challenge: a retrospective analysis, 2021.

[36] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Hysteretic q-learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69, 2007.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

[38] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287, 1999.

[39] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability, 2017.

[40] A. Oroojlooy and D. Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023.

[41] A. Paszke. Reinforcement learning (dqn) tutorial - pytorch, 2024. URL `https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`.

[42] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2016.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[46] G. Serré, E. Boguslawski, B. Donnot, A. Pavão, I. Guyon, and A. Marot. Reinforcement learning for energies of the future and carbon neutrality: a challenge design, 2022.

[47] M. T. Spaan. Reinforcement learning and markov decision processes. In *Reinforcement Learning: State of the Art*. Springer, 2012.

[48] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[49] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[50] S. Taha, J. Poland, K. Knezovic, and D. Shchetinin. Learning to run a power network under varying grid topology. In *2022 IEEE 7th International Energy Conference (ENERGYCON)*, pages 1–6, 2022.

[51] TensorFlow Developers. TensorBoard, 2023. URL `https://www.tensorflow.org/tensorboard`. Software available from tensorflow.org.

[52] E. van der Sar, A. Zocca, and S. Bhulai. Multi-agent reinforcement learning for power grid topology optimization, 2023.

[53] E. van der Sar, A. Zocca, and S. Bhulai. Optimizing power grid topologies with reinforcement learning: A survey of methods and challenges, 2025.

[54] H. Van Hasselt. Double q-learning., 01 2010.

[55] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015.

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[57] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018.

[58] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning, 2016.

[59] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[60] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

[61] P. Xu, Y. Pei, X. Zheng, and J. 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, 2020.

[62] P. Xu, J. Duan, J. Zhang, Y. Pei, D. Shi, Z. Wang, X. Dong, and Y. Sun. Active power correction strategies based on deep reinforcement learning—part i: A simulation-driven solution for robustness. *CSEE Journal of Power and Energy Systems*, 8(4): 1122–1133, 2022.

[63] S. Yang. Hierarchical graph multi-agent reinforcement learning for traffic signal control. *Information Sciences*, 634:55–72, 2023.

[64] D. Yoon, S. Hong, B.-J. Lee, and K.-E. Kim. Winning the l2rpn challenge: Power grid management via semi-markov afterstate actor-critic. In *International Conference on Learning Representations*, 2021.

[65] M. J. Zaki and W. M. Jr. *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge University Press, 2nd edition, 2020.

[66] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs, 2020.

A | Appendix A

The implementation of an efficient prioritized-based data structure is not trivial. A very basic implementation could be storing an object o a number of times proportional to its priority and then, when sampling a batch, uniformly sample objects from the data structure. Clearly, the latter is a very inefficient implementation, but states the idea of a prioritized-based data structure: an object o should ideally occupy a space in the data structure, proportional to its priority. To efficiently implement the latter concept a *Sum-tree* data structure is used: leaf nodes store the objects and their priorities, while internal nodes stores intermediate sums of the children priorities. The root node contains the sum of all priorities p_{total}, an example is provided in Fig. A.1.

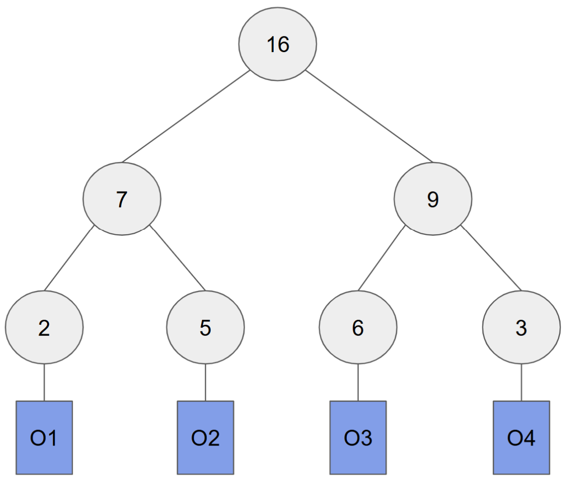


Figure A.1: Sum-Tree storing 4 objects: O1 priority-2 , O2 priority-5, O3 priority-6 and O4 priority-3.

This data structure allows the searching of an object by cumulative priority and the storing of an object in $O(\log N)$, where N is the size of the data structure. Searching an object with a specific cumulative priority starts from the root and the rationale is: compare the priority value to the one of the leftmost child, if it is lower continue the search on the leftmost child, if it is greater, continue the search on the rightmost child subtracting from the value the priority of the leftmost child. An example is provided in Fig. A.2.

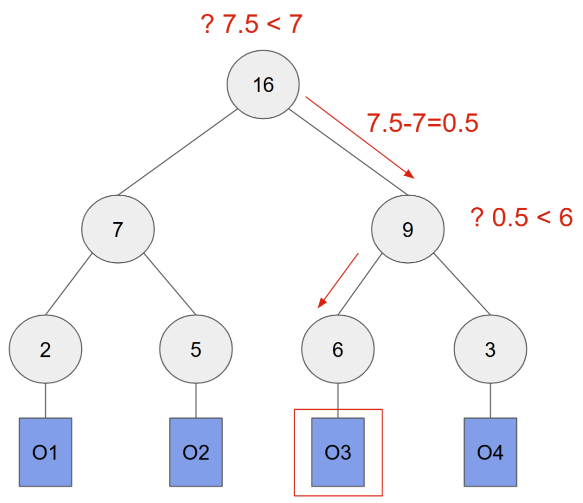


Figure A.2: Search of an object in a Tree-Sum, with a cumulative value of 7.5. The search correctly returns O3, which occupies the priority range [7,13].

To sample a batch of k objects from the Sum-Tree, the range $[0, p_{total}]$ is divided into k equal ranges, from each range a value is sampled uniformly and finally the k objects, corresponding to the k sampled values, are sampled from the Sum-Tree.

B | Appendix B

The complete set of hyperparameters used in the final solution is shown below:

Table B.1: Policy Hyperparameters.

Hyperparameter	**Value**
policy_kargs.epsilon	0.5
policy_kargs.min_epsilon	0.1
policy_kargs.decay_mode	half-life
policy_kargs.half_life	75

Table B.2: Line Agents Hyperparameters.

Hyperparameter	**Value**
line_agents.lr	0.0001
line_agents.weight_decay	5e-7
line_agents.use_double	true
line_agents.gamma	0.999
line_agents.Q_estimator_net_type	Dueling
line_agents.Q_estimator_net_shared_stream	[128]
line_agents.Q_estimator_net_value_stream	[128]
line_agents.Q_estimator_net_advantage_stream	[128,32]
line_agents.gradient_clipping	3
line_agents.tau_soft_updates	0.2
line_agents.policy_type	epsilon-greedy
line_agents.start_training_capacity	250
line_agents.target_update_freq	300
line_agents.buffer_type	prioritized
line_agents.num_training_iters	130

Table B.3: Manager Hyperparameters.

Hyperparameter	**Value**
MA_manager.lr	1e-5
MA_manager.weight_decay	5e-6
MA_manager.gamma	0.99
MA_manager.Q_estimator_net_shared_stream	[128]
MA_manager.Q_estimator_net_value_stream	[128]
MA_manager.Q_estimator_net_advantage_stream	[128,32]
MA_manager.policy_type	epsilon-greedy
MA_manager.start_training_capacity	1000
MA_manager.num_training_iters	150
MA_manager.target_update_freq	1000
MA_manager.lambda1	0.3
MA_manager.lambda2	0.3
MA_manager.lambda3	0.2

Table B.4: GNN hyperparameters.

Hyperparameter	**Value**
GNN.input_dim	246
GNN.n_layers	3
GNN.type	GAT
GNN.dropout	0.1
GNN.batch_size	32
GNN.lr	5e-5

Table B.5: Configuration Hyperparameters.

Hyperparameter	**Value**
config.environment.name	l2rpn_case14_sandbox
config.environment.seed	42
config.number of episodes	850

List of Figures

List of Tables

Acknowledgements

Un grazie di cuore alla mia famiglia, che mi ha sempre sostenuto con amore e pazienza.