



**POLITECNICO
DI TORINO**

POLITECNICO DI TORINO

Master Degree course in Energy and Nuclear Engineering

Master Degree Thesis

**Multi-Agent Reinforcement Learning
Building Energy Optimization:
Taxonomy and Experimental Evaluation
in CityLearn**

Supervisors

Prof. Lorenzo BOTTACCIOLI

PhD. Pietro RANDO-MAZZARINO

Candidate

Costantin Cosmin COSTEA

ACADEMIC YEAR 2025-2026

*To Mum and Dad,
Thank you for your unwavering support.*

Abstract

This thesis is structured in two main parts.

In the first part, a taxonomy of reinforcement learning control strategies for building energy management is proposed based on previous literature. The objective is to provide a clear classification framework for the analyzed algorithms and to better understand their suitability for different applications.

In the second part, a comparative analysis is conducted between a single-agent Proximal Policy Optimization (PPO) algorithm and three multi-agent PPO-based approaches within the CityLearn environment. The goal is to evaluate their performance in terms of energy cost and carbon emissions at the single-building level, to analyze how load profiles change compared to a baseline and a Rule-Based Control (RBC), and to assess district-level Key Performance Indicators (KPIs) from a grid perspective.

For the single-agent implementation, the Stable-Baselines3 library is used, and optimal hyperparameters are identified through Optuna. For the multi-agent implementations, RLlib is employed, with hyperparameter optimization also performed using Optuna. Simulations are conducted separately for the summer and winter periods. In both cases, the data are divided into two months for training and one month for deployment (testing).

All multi-agent frameworks are deployed in a decentralized manner. The first two algorithms are Independent PPO (IPPO) variants based on Schroeder et al. (2021). The first configuration follows the original paper and uses shared policy parameters across agents, while the second generalizes the approach by assigning independent neural networks to each agent. Finally, a Multi-Agent PPO (MAPPO) implementation based on Yu et al. (2022) is considered, featuring a centralized shared critic. All experiments are conducted on four randomly selected buildings from a dataset of seventeen.

The results indicate that the considered reinforcement learning approaches achieve overall comparable performance across the evaluated metrics. At both the building and district levels, differences among the algorithms remain limited, with each method demonstrating strengths depending on the specific key performance indicator and seasonal conditions.

Overall, all approaches outperform the baseline scenario across the main indicators. Moreover, each model contributes to improving the district’s average daily load profile in both winter and summer, highlighting their effectiveness in enhancing demand flexibility and overall energy management performance.

Contents

1	Introduction	5
2	State of the Art	7
2.1	Reinforcement Learning Fundamentals	8
2.1.1	Reinforcement learning problem	8
2.1.2	Markov Decision Process	11
2.1.3	Bellman Equations	13
2.1.4	Models, Policies and Learning set-up	14
2.1.5	Dynamic Programming	15
2.1.6	Model-Free	16
2.1.7	Model-based	20
2.2	Deep Reinforcement Learning	20
2.2.1	Fundamentals of Deep Learning	21
2.2.2	Value-based Methods	26
2.2.3	Policy gradient methods	28
2.2.4	PPO - Proximal Policy Optimization	30
2.3	Single Agent Taxonomy	32
2.4	MAS - Multi-Agent Systems	33
2.4.1	Benefits of MARL	34
2.4.2	Challenges in MARL	34
2.4.3	Hierarchy of Multi-Agent System Games	35
2.4.4	Traning and Execution Paradigms	37
2.4.5	IPPO - Independent Proximal Policy Optimization	38
2.4.6	MAPPO - Multi Agent Proximal Policy Optimization	41
2.4.7	Taxonomy	44
2.5	Related work	46
3	Technological Background	49
3.1	CityLearn	49
3.1.1	Control Variables	50
3.1.2	Observations	51
3.1.3	Datasets	52
3.2	StableBaselines3 - SB3	54
3.3	Optuna	54

3.4	Ray	54
4	Methodology	57
4.1	Outline of the system	57
4.2	KPIs monitored	62
4.3	Simulation Workflow	63
5	Results	67
5.1	Summer Results	68
5.1.1	Training phase	68
5.1.2	Testing Phase	70
5.2	Winter Results	76
5.2.1	Training Phase	76
5.2.2	Testing phase	78
6	Conclusion	85
6.1	Taxonomy	85
6.2	Experimental Evaluation	85
6.3	Future Work	86
	Bibliography	87

Chapter 1

Introduction

Buildings account for approximately 39% of global final energy consumption and 36% of greenhouse gas emissions [1]. The rapid integration of renewable energy sources, distributed photovoltaic systems, and battery storage technologies are transforming buildings into active participants in modern energy systems. Building Energy Management Systems (BEMS), in the near future, must therefore optimize energy consumption, minimize operational costs, maintain occupant comfort, and support grid stability [2].

Demand Response (DR) becomes crucial as solar and wind energy expands. DR involves adjusting power usage in response to market prices or grid signals. This adds the challenge to integrate the renewable energy production while maximizing renewable energy on site [3]. While buildings can be managed individually, coordinated control at the district level enables advanced demand response strategies and improved grid resilience.

Three primary modeling approaches are commonly adopted:

White-box models are physics-based and highly accurate but complex and difficult to scale.

Gray-box models, typically based on RC state-space representations, balance interpretability, and computational efficiency. They are widely used in Model Predictive Control (MPC).

Black-box models are purely data-driven, they are scalable but often difficult to integrate into optimization frameworks.

In this thesis the latter is used.

Between 2010 and 2019, adaptive control strategies such as MPC dominated building energy management research, achieving better results compared to Rule-Based Control (RBC) under optimal simulated conditions. However, MPC requires accurate system models and frequent recalibration [4].

Reinforcement Learning (RL) provides a model-free alternative capable of learning optimal policies directly from interaction data. This approach has widely developed from 2020 till now with both a single agent and multi-agent approach. RL naturally handles sequential decision-making and adapts to dynamic environments. It allows the control of multiple buildings or neighborhoods at once.

The first goal of this thesis is to provide a taxonomy, based on [5], [6], [7], [8] for part of the multi-agent algorithms available today.

The second goal of this thesis is to compare the evolution of a single agent algorithm to a multi agent with both an independent and cooperative approach in CityLearn.

The CityLearn environment [9] has become a benchmark for both multi agent and single agent energy coordination and demand response. CityLearn allows the comparison of the algorithms based on different scenarios given the multiple dataset and configurations available.

Recent works have explored centralized training and decentralized execution (CTDE) actor-critic algorithms such as MAPPO [10], MATRPO [11], HAPPO and HATRPO [12]. Wu et al. [13] demonstrated integration of these algorithms within CityLearn using MARLlib. Wang et al. [14] applied MAPPO [15] to hybrid residential systems. A similar work was completed by Yang et al. [16] where CityLearn was used to compare PPO, IPPO and MAPPO, adding a forecasting method and using a different reward function.

In this thesis PPO, IPPO, IPPO-SP and MAPPO are compared to better understand how the algorithms behave by considering a single agent (PPO), an independent multi-agent (IPPO) with both centralized and decentralized training, an independent PPO with shared parameters (IPPO-SP) and a cooperative Multi-Agent (MAPPO). CityLearn environment is used as a benchmark, evaluating performance against RBC and baseline through key performance indicators such as cost reduction, peak demand mitigation, renewable self-consumption.

Chapter 2

State of the Art

Reinforcement Learning (RL) is a learning paradigm in which an agent interacts with an environment by observing states, taking actions, and receiving rewards. The objective of the interaction is to learn a policy that maximizes long-term cumulative reward. Classical RL approaches typically rely on tabular methods or simple function approximators, which makes them suitable for problems with small, discrete state and action spaces but limits their scalability. Reinforcement learning alongside with supervised and unsupervised learning represent the three paradigms of machine learning.

Deep Reinforcement Learning (DRL) extends RL by integrating deep neural networks as function approximators for value functions and/or policies, enabling the handling of high-dimensional and continuous state spaces and the automatic extraction of relevant features from raw observations. Deep Reinforcement Learning significantly improves scalability and applicability to complex real-world systems, such as robotics and energy management, but it comes at the cost of higher computational requirements and increased training complexity compared to traditional RL methods.

Understanding the current state of the art is essential to properly interpret and contextualize the results presented in this thesis. For this reason, the exploration of the Reinforcement Learning (RL) field constitutes the primary objective of this chapter. The chapter begins by introducing the fundamental definitions, notation, and theoretical foundations of RL, establishing the conceptual framework required for the subsequent analysis. It then examines how Deep Reinforcement Learning (DRL) extends traditional RL by leveraging deep neural networks, with particular attention given to the evolution of the main algorithms and the motivations behind their development. Afterward, the discussion moves from single-agent formulations to multi-agent reinforcement learning, providing a comprehensive overview of the field and defining a base taxonomy following the [5], [6], [7], [8]. Finally, the methodological baseline for this thesis is presented, whose ultimate goal is the coordinated management of electrical energy across multiple households within the same neighborhood. The paragraphs completing this section are inspired by [5], [6], [17], [18], .

2.1 Reinforcement Learning Fundamentals

This section aims to present notions, definitions, and ideas behind this research with the goal to help the reader developing a solid base for the understanding of the DRL section.

2.1.1 Reinforcement learning problem

The reinforcement learning problem involves an *agent* interacting with a complex *environment* to solve a problem by trial and error with the goal of maximizing a *cumulative reward*.

Let's go step by step and analyze one by one the key elements of the previous statement. In the following figure the main characters, the agent and the environment are shown as their interaction loop.

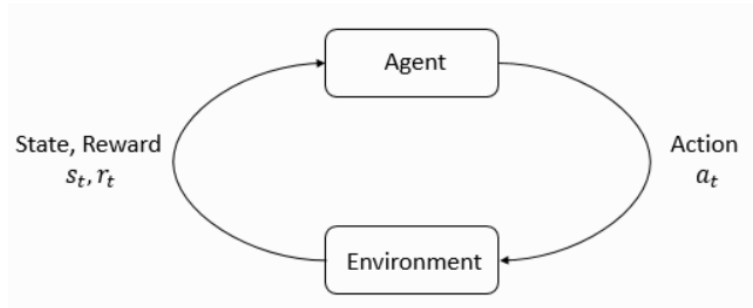


Figure 2.1: Agent - Environment interaction [17]

Agent - Actions and action space

The *agent* is the entity interacting with the *environment*. It may take one or multiple actions (a_t or $[a_{t1}, a_{t2}, \dots]$) from a set allowed by the environment that is called *action space*. The choice is made based on the observations it receives from the environment processed within an algorithm. The agent has no control over the environment, it just interacts with it by choosing actions a_t at each timestep. Actions are the only means that can influence and modify the environment.

The actions compose the action space which can be described as a vector if multiple actions are allowed. The action value can be discrete or continuous. An example of a discrete action space may be the choice between left and right, while continuous action space may imply the choice of how much to charge or discharge a battery.

For the following paragraphs we will consider an agent that takes only one action for simplicity. The goal of the agent is to maximize its cumulative reward, called *return*, given as feedback by the environment at each timestep.

Environment - Observations, reward and state space

The *environment* is the entity with which the *agent* interacts with. When the agent

executes an action, the environment transitions to a new state and returns an *observation* and a *reward*.

The environment can be either *deterministic* or *stochastic*. In a deterministic environment, the next state is uniquely determined by the current state and the action taken: executing the same action in the same state will always lead to the same outcome. In a stochastic environment, the next state is not uniquely determined by the current state and action, as randomness is incorporated into the state transition process. In this case, the same state-action pair may lead to different next states according to a probability distribution. Agents operating in stochastic environments must learn policies that are robust to uncertainty.

The *observation* represent the information available to the agent about the environment and are typically expressed as a vector or a matrix containing relevant state variables.

The *reward* is a scalar signal that defines the objective of the reinforcement learning problem. It represents feedback provided by the environment, indicating whether the agent's action had a positive, negative, or neutral effect, according to how the reward was defined by the user. The reward signal reflects only the immediate outcome of the most recent action; however, the consequences of an action can extend over longer time horizons, with rewards that can be delayed.

As in human decision-making, receiving a large reward at a given time step does not preclude receiving smaller rewards afterward. In many situations, it may be beneficial to sacrifice an immediate gain in order to maximize the cumulative reward in the future. The impact on future states stress the importance of time. The actions are somehow sequentially linked by the agent or agents, resulting in data being no more independent.

The *state* s_t is a complete description of the world at timestep t . All the world information is included in the state.

An *observation* o_t partially describes the state. It is important to note that the observation might not include all the variables in the state. The observation can be real-valued vector, matrix or tensors used by the agent to choose the next action.

Trajectories

The sequence of actions and states is called *trajectory* (τ), and it can be defined:

$$\tau = (s_0, a_0, s_1, a_1, \dots) \quad (2.1)$$

At the beginning of an episode the initial state (s_0) is randomly set from the start state distribution, usually denoted ρ_0 . This distribution is formally expressed as:

$$s_0 \sim \rho_0(\cdot) \quad (2.2)$$

As the interactions unfold, the environment evolves over discrete time steps. The environment moves to state s_{t+1} as consequence of agent taking action a_t .

The transition can be **deterministic**, where the next state is uniquely determined by a transition function:

$$s_{t+1} = f(s_t, a_t) \quad (2.3)$$

or **stochastic**, where the next state is drawn from a probability distribution conditioned on the current state and action:

$$s_{t+1} \sim P(\cdot | s_t, a_t) \quad (2.4)$$

At each timestep the agent choose the actions according to its policy.

Definition of the cumulative return

In the previous paragraphs, an intuitive explanation of the basic concepts were presented; now it is time to formalize them.

The main purpose of the agent is to maximize the cumulative return in the long run. The cumulative return can be defined as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \quad (2.5)$$

Where:

- r_{t+k+1} is the reward value output by the environment at timestep $t + k + 1$;
- $\gamma \in [0,1)$ is the discount factor.

The need of γ is to ensure convergence over an infinite horizon. It is also to mitigate the weight of the future rewards as the more you go in the future, the more uncertain the reward is. If γ is set to 0, the cumulative reward is the last reward. The function is a geometric series that converges to a finite value $1/(\gamma - 1)$. Given this, the value $\gamma = 1$ makes sense only when the horizon is finite.

Definitions of policy and value functions

A *policy* is a rule the agent follows to decide what action to take. It can be deterministic, usually denoted by μ :

$$a_t = \mu(s_t) \quad (2.6)$$

Or it can be stochastic, usually denoted by π :

$$a_t = \pi(s_t) \quad (2.7)$$

The policy can be interpreted as the decision-making mechanism of the agent. In Deep Reinforcement Learning (DRL), policies are parameterized, meaning that their outputs are computed by functions that depend on a set of parameters and can be adjusted through optimization algorithms to modify the agent's behavior. From this perspective, the central objective of reinforcement learning is to learn the optimal policy π^* , which maximizes the expected cumulative return and defines the best action to take in a given state. To achieve this objective, reinforcement learning exploits value functions, which estimate the long-term expected return associated with states or state-action pairs and provide a principled foundation for policy evaluation and improvement.

With *value functions* the expected return is intended. If agent starts in a state or action-state pair and then, acts according to a particular policy. There are four fundamental value functions commonly used in Reinforcement Learning [17]:

The **on-policy state-value function** $V^\pi(s)$ represents the expected return obtained by starting from state s and thereafter following policy π :

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_0 = s]. \quad (2.8)$$

The **on-policy action-value function** $Q^\pi(s, a)$ gives the expected return obtained by starting from state s , taking an arbitrary initial action a , and then following policy π :

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_0 = s, a_0 = a]. \quad (2.9)$$

The **optimal state-value function** $V^*(s)$ represents the maximum expected return achievable from state s under any policy:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_0 = s]. \quad (2.10)$$

The **optimal action-value function** $Q^*(s, a)$ represents the maximum expected return achievable by starting from state s , taking action a , and thereafter following the optimal policy:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_0 = s, a_0 = a]. \quad (2.11)$$

2.1.2 Markov Decision Process

The RL problem can be framed through a Markov Decision Process (MDP). The formal framework of finite Markov Decision Process involves evaluative feedback and introduce an associative component, where the agent must learn to choose different actions in different situations. MDP provide a classical and rigorous model for sequential decision-

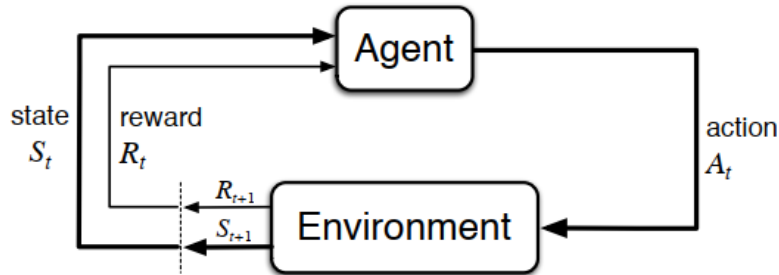


Figure 2.2: Representation of the agent-environment interaction in a MDP [18]

making, in which each action affects not only the immediate reward, but also the next state and consequently future rewards. This introduces the concept of delayed rewards and the challenge of balancing immediate versus long-term gains.

In MDP, we estimate the state-action value $q(s, a)$, which can be translated as the expected return of taking action a in state s , or the state value $v(s)$ that is the expected return from state s in optimal behavior. These state-dependent measures are crucial to properly assign credit to actions that influence outcomes over time.

Finite MDP represent a mathematically idealized version of the reinforcement learning problem, allowing for precise theoretical analysis.

Based on the previous definitions, it can be said that:

- *agent*: the learner and the decision maker;
- *environment*: the "thing" agent interacts with and returns feedback after interacting with the agent.

Each time the agent interacts with the environment, this will react by presenting new situations to the agent and giving a reward as feedback fro the actions. The following figure presents the Markov Decision Process and the agent-environment interaction. MDP is defined by tuple:

$$\langle (S, A, P, R, \gamma) \rangle$$

where:

- S is a finite set of states;
- A is a finite set of actions;
- $P : S \times A \rightarrow \delta(S)$ is the state transition probability;
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function. It is a scalar value received by the agent for a transition from state-action pair (s_t, a_t) to state s_{t+1} ;
- γ is the discount factor as previously presented.

The state signal $s_t \in S$ describes the environment at each time step t . The agent can alter the state by taking the action $a_t \in A$. This will result in an environment change according to the state transition probability function P :

$$s_t \rightarrow s_{t+1}$$

The probability of ending in s_t executing a_t is given by:

$$P(s_t, u_t, s_{t+1})$$

Once in the new state, the agent receives a scalar according to the reward function:

$$R : r_{t+1} = R(s_t, a_t, s_{t+1}) \in \mathbb{R}$$

This reward immediately assesses the impact of the action a_t , but does not speak about the long-term effect of this action.

The agent is expected to maximize, at each time step, the expected cumulative return defined in (2.5) and repurposed here:

$$R_t = \mathbb{E} \left[\sum_{j=0}^{\infty} \gamma^j r_{k+j+1} \right] \quad (2.12)$$

The reward is not sufficient to determine optimal policy as it does not account for future rewards that a specific action may lead to. The agent's goal is to maximize its long-term performance, while only having a-step feedback and then use it to choose the best action.

2.1.3 Bellman Equations

Both equations 2.8 and 2.9 can be expressed as the relation of two consecutive states as they obey special self-consistency equations called Bellman equations [19]. The idea behind it is that the expected value is the sum of the value expected in the beginning, plus the expected discounted reward.

The Bellman equations for the on-policy value functions are:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi, s' \sim P} [r(s, a) + \gamma V^\pi(s')] \quad (2.13)$$

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim P} [r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} [Q^\pi(s', a')]] \quad (2.14)$$

Where $s' \sim P$ means that the next state is sampled from the environment according to the distribution P and $a \sim \pi$ shows that the policy π determines the next action. As discussed when the policy was introduced, the goal is to find the best policy π^* , and it can be done through *Bellman optimality equations*:

$$\begin{aligned} V^*(s_t) &= \max_a \mathbb{E}_{s' \sim P} [r(s_t, a) + \gamma V^*(s_{t+1})] \\ &= \max_a \sum_{s' \in S, r \in R} P(s', r | s, a) [r + \gamma V^*(s')] , \end{aligned} \quad (2.15)$$

$$\begin{aligned} Q^*(s_t, a_t) &= \mathbb{E}_{s' \sim E} \left[r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a') \right] \\ &= \sum_{s' \in S, r \in R} P(s', r | s, a) \left[r + \gamma \max_{a'} Q^*(s', a') \right] . \end{aligned} \quad (2.16)$$

Value functions induce a partial ordering over policies. Specifically, given two policies π and π' , policy π is said to be at least as good as π' if

$$V^\pi(s) \geq V^{\pi'}(s), \quad \forall s \in \mathcal{S},$$

which is denoted as $\pi \geq \pi'$. This ordering provides a formal basis for stating the *sanity theorem*. The theorem asserts that, for any Markov Decision Process (MDP), there exists at least one optimal policy π^* that is better than or equal to all other policies, that is,

$$\pi^* \geq \pi, \quad \forall \pi.$$

Moreover, all optimal policies achieve the same optimal state-value function V^* and the same optimal action-value function Q^* , thereby characterizing optimal behavior uniquely in terms of value functions.

2.1.4 Models, Policies and Learning set-up

Now that we have an idea of the components and how they interact, it is possible to go more in detail on each one of them.

Starting from the *algorithm model*, it can be:

- **model based:** the agent interacts with a virtual physical model of the environment. This set up is useful and quite powerful in deterministic environments. It allows the agent to plan ahead and solve the optimization problem in a more accurate way. Learning a model for the agent can be complex, as it may fall in a bias on how the model is built and have disastrous results when tested in a similar one.
- **model free:** the environment is a black box. The agent does not know exactly what is happening or how things are happening inside. It just relies on the feedback the environment returns. The agent takes the observations, computes them, chooses the action and checks if the reward improved or not. The agent is somehow trying to build his impression of the environment and its logic to be able to make the best decision at each time step.

Reinforcement learning algorithms can be categorized into *value-based* and *policy-based* methods.

- **value based:** aim to solve the optimization problem by estimating the value of each possible action. At each time step, the agent assigns a numerical value to available actions, typically representing the expected cumulative return. Then, selects the action with the highest estimated value, which is assumed to maximize long-term reward. Value-based methods can be further classified into:
 - **on-policy methods:** which learn exclusively from data generated by the current policy;
 - **off-policy:** which are able to learn from data generated by different or past policies.
- **policy based:** in contrast, do not explicitly estimate action values. Instead, the agent directly learns a policy represented as a probability distribution over actions. At each time step, the algorithm outputs this distribution conditioned on the current state, and the agent selects an action according to it. Agents typically favor actions with higher probabilities. The policy is optimized to maximize the expected cumulative return on the long run.

The learning setting can be *online* or *offline*:

- **on-line** setting: the agent learns while continuously interacting with the environment and collecting new experiences. The policy is updated using freshly gathered data, allowing the agent to adapt dynamically to changes in the environment;
- **off-line** setting: the agent learns from a fixed and limited dataset collected beforehand, without further interaction with the environment. In this case, generalization becomes a critical challenge, as the agent cannot explore new state-action pairs or correct suboptimal behavior through additional experience.

Another important distinction among reinforcement learning algorithms concerns how

the policy is used during training. In on-policy algorithms, learning strictly depends on data generated by the current policy, as these methods are designed to update the policy using only the last data available.

In contrast, off-policy algorithms can leverage data generated by different policies or collected in the past. This flexibility allows the use of large experience replay buffers or previously recorded episodes, improving sample efficiency and enabling learning from historical data.

2.1.5 Dynamic Programming

Dynamic Programming (DP) comprises a class of algorithms that compute optimal policies assuming full knowledge of the environment model, and therefore primarily applies to model-based reinforcement learning problems, where the environment is formalized as a Markov Decision Process (MDP).

Although DP methods can theoretically guarantee optimal solutions, they are rarely used in practice due to their reliance on a perfect and complete model of the environment, as well as their typically high computational cost, which scales poorly with the size of the state and action spaces.

Policy evaluation and Policy improvement

Policy Iteration addresses the control problem in reinforcement learning by iteratively improving a policy. Starting from an initial policy, the algorithm alternates between *policy evaluation*, where the value function of the current policy is computed, and *policy improvement*, where the policy is updated by selecting more rewarding actions.

During the policy improvement step, a better policy is obtained by exploiting the *action-value function* $Q_\pi(s, a)$, which represents the expected return obtained by taking action a in state s and then following the current policy π . If, for a given state s , there exists an action such that $Q_\pi(s, a) > V_\pi(s)$, then this action yields a higher expected return than the one currently prescribed by the policy. Replacing the original action with this improved one results in a new policy that is guaranteed to be no worse than the previous policy, and typically strictly better. This process is repeated until the policy converges to the optimal policy.

Value Iteration Value Iteration solves the control problem by directly approximating the optimal value function. The algorithm iteratively applies the Bellman optimality update to the state-value function, updating each state by selecting the maximum expected return over all possible actions. Once convergence is achieved, the optimal policy is derived by acting greedily with respect to the resulting value function.

GPI - Generalised Policy Iteration Generalized Policy Iteration (GPI) underlies the interaction between policy evaluation and policy improvement in both value iteration and policy iteration algorithms. Rather than treating these two steps as strictly separate processes, GPI describes their continuous and interleaved application.

In the policy evaluation step, the current policy is exploited to construct an approximation of the corresponding value function. In the policy improvement step, the value

function is then used to derive a better policy by favoring actions that yield higher expected returns. Through repeated alternation of these two processes, the policy and value function progressively improve, eventually leading to convergence aiming for an optimal solution.

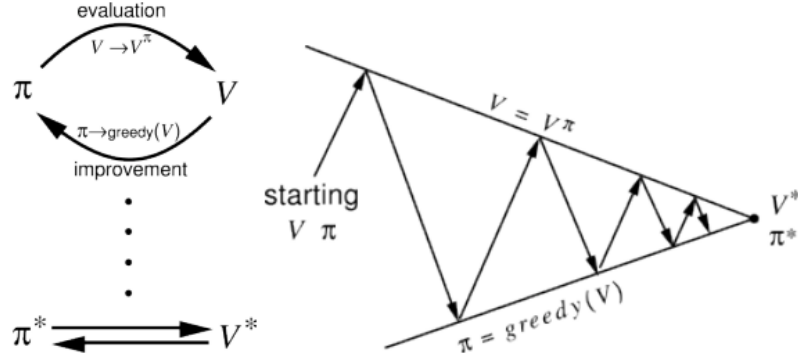


Figure 2.3: GPI schema. [18]. Value and Policy interact to compute the optimal value function and the optimal policy

2.1.6 Model-Free

Model-free reinforcement learning allows an agent to learn optimal policies or value functions directly from experience, without relying on an explicit model of the environment. The agent interacts with the environment by selecting actions and receives observations and rewards in return. As a result, no detailed knowledge of the environment dynamics is required.

A critical aspect of model-free reinforcement learning is the exploration–exploitation trade–off. During exploration, the agent deliberately selects different actions, not necessarily those that currently appear optimal, in order to better understand how the environment responds. This phase is essential, particularly in the early stages of learning. As more information is gathered and better actions are identified, the agent increasingly shifts toward exploitation, selecting actions that maximize the expected cumulative return in the long run.

Despite their flexibility, model-free methods often suffer from sample inefficiency, as they typically require a large number of interactions with the environment before converging to a satisfactory policy. Moreover, they are affected by the curse of dimensionality: as the size of the state or action space grows, learning becomes increasingly inefficient. Consequently, scaling model-free methods to very large or continuous state–action spaces can be challenging and may lead to slow convergence. *Monte Carlo* (MC) methods and *Temporal-Difference* (TD) are two model free approaches.

Monte Carlo Methods

Monte Carlo methods learn directly from complete episodes of experience. This implies

that each episode must terminate, regardless of the actions selected by the agent. During training, the agent interacts with the environment, collects sequences of states, actions, and rewards, and estimates value functions by averaging the returns observed over multiple episodes.

Monte Carlo methods are therefore applicable only to episodic Markov Decision Processes (MDPs), since the episode must be completed before value updates can be performed. The accumulated reward obtained at the end of an episode is used to estimate the state-value or action-value function.

As with other model-free approaches, Monte Carlo methods must address the exploration–exploitation dilemma. The agent needs to balance exploring new actions and exploiting ones, while ensuring sufficient coverage of the state–action space. A common strategy is the ϵ – *greedy* policy, in which the agent selects a random action with probability ϵ and follows the current learned policy with probability $(1 - \epsilon)$. Compared to Dynamic Programming methods, Monte Carlo approaches offer several advantages:

- they learn optimal behavior directly through interaction with the environment, without requiring a model;
- they can be applied using simulated or sampled environments, making them suitable for a wide range of practical applications;
- they naturally focus computation on visited subsets of the state space, rather than requiring exhaustive sweeps over all states.

More in depth information on these methods can be found in [18], Chapter 5.

Temporal Difference Learning

Temporal Difference (TD) learning combines key ideas from both Monte Carlo methods and Dynamic Programming. Like Monte Carlo methods, TD learning can learn directly from raw experience through interaction with the environment. Like Dynamic Programming, it updates value estimates by relying on other value estimates rather than waiting for fully observed returns. TD learning is therefore a model-free approach that, unlike Monte Carlo methods, updates value functions using a temporal difference error instead of the total accumulated reward at the end of an episode.

The temporal difference error is defined as the difference between the current value estimate and a bootstrapped target constructed from the immediate reward and the estimated value of the subsequent state. This allows value functions to be updated incrementally at each time step, without requiring the episode to terminate.

The TD(0) update rule for the state-value function is given by the following equation:

$$V(s_t) \leftarrow V(s_t) + \alpha \underbrace{\left(\overbrace{r_{t+1} + \gamma V(s_{t+1})}^{\text{TD target}} - V(s_t) \right)}_{\text{TD error } (\delta_t)} \quad (2.17)$$

where α is the learning rate and γ is the discount factor.

As a result, TD methods can be applied to both episodic and continuing (non-terminating) environments. Compared to Monte Carlo methods, TD learning generally exhibits lower

variance, but introduces additional bias due to bootstrapping. *SARSA* (*State-Action-Reward-State-Action*) and *Q-Learning* are the algorithms worth of mentioning due to their extensive use in solving RL problems.

SARSA is an on-policy Temporal Difference (TD) algorithm. Unlike TD methods that estimate state-value functions, SARSA learns an action-value function, focusing on the expected return of state-action pairs and the value of transitions between them.

The convergence guarantees that apply to TD(0) for state-value functions also extend to the corresponding action-value formulation. The SARSA update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2.18)$$

This update is performed after every transition from a non-terminal state. The update rule explicitly uses the full sequence of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, from which the name *SARSA* is derived.

Algorithm 1: SARSA (on-policy TD control) for estimating $Q \approx q^*$

Input: step size $\alpha \in (0,1]$, exploration parameter $\varepsilon > 0$
Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, except $Q(\text{terminal}, \cdot) = 0$;
for each episode **do**
 Initialize s_t ;
 Choose a_t from s_t using a policy derived from Q (e.g., ε -greedy);
 repeat
 Take action a_t , observe r_{t+1} and s_{t+1} ;
 Choose a_{t+1} from s_{t+1} using a policy derived from Q (e.g., ε -greedy);
 $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$;
 $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$;
 until s_t is terminal;
end

Q-Learning [20] Q-learning is an off-policy Temporal Difference (TD) control algorithm and one of the earliest major advances in reinforcement learning. It learns an approximation of the optimal action-value function $Q \approx q^*$ while following a potentially different behavior policy. In practice, actions are often selected according to an ε -greedy policy to ensure exploration, whereas the update uses a greedy target obtained by taking the maximum action value in the next state. This separation between the behavior policy and the target policy is what makes Q-learning off-policy.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \quad (2.19)$$

Algorithm 2: Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Input: step size $\alpha \in (0,1]$, exploration parameter $\varepsilon > 0$
 Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, except $Q(\text{terminal}, \cdot) = 0$;
for each episode **do**
 Initialize s_t ;
 Choose a_t from s_t using policy derived from Q (e.g., ε -greedy);
 repeat
 Take action a_t , observe r_{t+1} and s_{t+1} ;
 Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g., ε -greedy);
 $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$;
 $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$;
 until s_t is terminal;
end

Temporal Difference Lambda Learning Temporal Difference λ (TD(λ)) learning extends standard TD methods by combining ideas from both Monte Carlo and Dynamic Programming through a weighted average of multi-step returns. Instead of updating value estimates using a single one-step look forward, TD(λ) considers returns that look n steps into the future, assigning exponentially decaying weights to longer horizons.

This leads to the definition of the λ -return, denoted by G_t^λ , which interpolates between the one-step TD target and the full Monte Carlo return. The λ -return is defined as

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \quad (2.20)$$

where the n -step return is given by

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V(s_{t+n}). \quad (2.21)$$

By adjusting the parameter $\lambda \in [0,1]$, TD(λ) smoothly transitions from TD(0), which is based purely on one-step bootstrapping, to Monte Carlo methods, which use complete returns.

TD(λ) also introduces the concept of *eligibility traces*, which quantify how strongly previously visited states should be updated at each time step. Eligibility traces reflect both the recency and frequency of state visitation and decay exponentially over time.

For state-value methods, the eligibility trace for state s is updated as

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) + 1, & \text{if } s = s_t, \\ \gamma \lambda e_{t-1}(s), & \text{otherwise.} \end{cases} \quad (2.22)$$

The value function is then updated according to

$$V(s) \leftarrow V(s) + \alpha \delta_t e_t(s), \quad (2.23)$$

where the temporal difference error is defined as

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \quad (2.24)$$

Eligibility traces allow TD(λ) to efficiently assign credit to states that contributed to future rewards, providing a unifying framework that bridges Monte Carlo and Dynamic Programming methods while offering a flexible bias–variance trade-off.

2.1.7 Model-based

Model-based reinforcement learning explicitly incorporates a model of the environment dynamics. This model enables the agent to perform *planning*, which can significantly improve *sample efficiency* compared to model-free methods.

A model-based approach typically relies on either *prior knowledge* of the environment or on a learned model. When prior knowledge is available, it can accelerate learning by providing an initial approximation of the environment dynamics. Alternatively, the agent can learn the model directly through interaction with the environment and use it to simulate future trajectories.

One of the main drawbacks of model-based methods is that the assumed or learned model may be inaccurate, leading to biased predictions and potentially sub-optimal policies. Moreover, model-based algorithms must learn both the environment model and the policy, which increases computational complexity. Despite this, model-based approaches often require fewer real environment interactions to learn an effective policy, making them particularly attractive in data-scarce or costly environments.

2.2 Deep Reinforcement Learning

The primary motivation for Deep Reinforcement Learning (DRL) lies in the integration of neural networks as function approximators within reinforcement learning algorithms. Classical reinforcement learning methods that explicitly compute state–value or action–value functions become impractical when dealing with large-scale or continuous state and action spaces, as the computational cost grows rapidly and the curse of dimensionality becomes prohibitive.

To address these limitations, function approximation is employed to represent value functions and policies compactly. Instead of maintaining tabular representations, value functions are parameterized by a vector of parameters

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_n)^\top. \quad (2.25)$$

These parameters are used to approximate the state–value and action–value functions, allowing the agent to generalize from previously visited states to unseen ones. The parameters are updated using Monte Carlo or Temporal Difference learning methods.

Formally, the approximated value functions are expressed as

$$V(s; \boldsymbol{\theta}) \approx V_\pi(s), \quad Q(s, a; \boldsymbol{\theta}) \approx Q_\pi(s, a), \quad (2.26)$$

where the mapping from the parameter vector θ to the value function is learned through interaction with the environment.

This parameterization leads to a significant reduction in the number of parameters that must be learned, reducing memory requirements and improving sample efficiency. When neural networks are used as function approximators, they allow efficient learning in high-dimensional and continuous domains, bridging the gap between reinforcement learning and deep learning. This integration marks the transition from classical reinforcement learning to Deep Reinforcement Learning.

Pioneering work by DeepMind [21] and [22] demonstrated the effectiveness of this approach in complex decision-making and control problems, representing a major step toward general artificial intelligence.

Given the scope of this work, the remainder of this section focuses on model-free deep reinforcement learning algorithms. In particular, it presents the theoretical foundations and state-of-the-art developments leading to the main algorithm used in this thesis: the **Proximal Policy Optimization (PPO) algorithm** [23]. PPO is an actor-critic algorithm and the core algorithm used to develop the experimental results presented in this work.

2.2.1 Fundamentals of Deep Learning

Artificial Neural Networks

Deep learning is based on the approximation of a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, parameterized by a vector of weights $\mathbf{w} \in \mathbb{R}^{n_w}$, with $n_w \in \mathbb{N}$, such that

$$y = f(\mathbf{x}; \mathbf{w}). \quad (2.27)$$

The fundamental building block of deep learning models is the artificial neuron, which is inspired by the biological neurons found in human and animal nervous systems. A biological neuron receives signals through multiple inputs (others neurons), known as *dendrites*. These signals are processed within the cell body, and if the accumulated potential exceeds a certain threshold, the neuron emits a signal through its single output, called the *axon*.

In the same way, an artificial neuron processes an input vector by computing a weighted sum of its inputs, adding a bias term, and applying a nonlinear activation function. This operation can be described by the following equation:

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right), \quad (2.28)$$

where x_i are the input features, w_i are the corresponding weights, b is the bias term, and $f(\cdot)$ is the activation function.

The learning process consists of adjusting the weights $\mathbf{w}$ and the bias b in order to find a parameter configuration that best approximates the target function. Figure 2.7 illustrates the analogy between the biological neuron and its artificial counterpart.

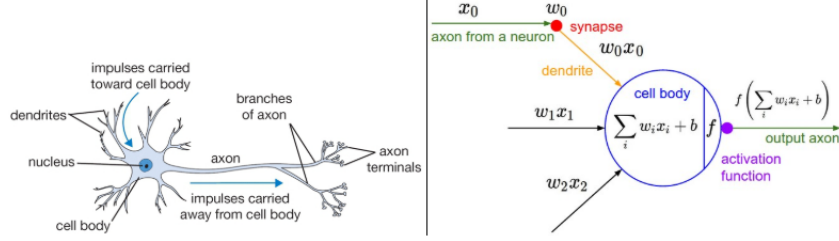


Figure 2.4: Biological (left) and artificial(right) neuron [24].

Deep Neural Networks (DNNs) are composed of multiple layers of artificial neurons, each performing a nonlinear transformation of its input. The successive application of these transformations enables the network to learn hierarchical representations, where higher layers capture increasingly abstract features of the data. This progressive abstraction plays a key role in the learning capability of deep models and has been extensively studied in the literature [25].

To better understand the behavior of deep neural networks, it is convenient to represent them as a sequence of fully connected layers, where each neuron in a layer is connected to all neurons in the subsequent layer. Figure 2.5 illustrates the structure of a fully connected deep neural network with one hidden layer.

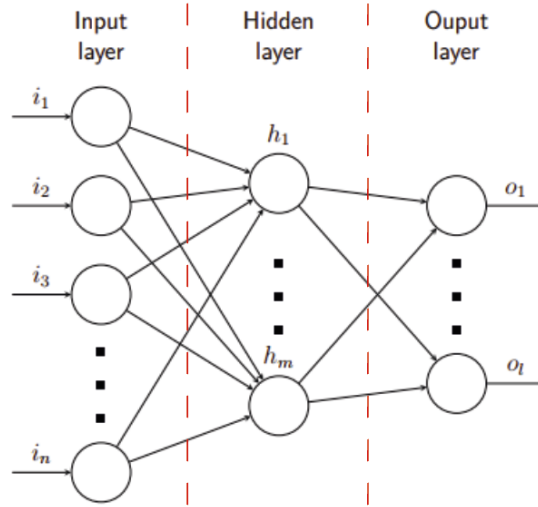


Figure 2.5: Example of Deep Neural Network with one fully connected hidden layer.

According to the figure, the input layer receives a column vector of inputs $\mathbf{i} \in \mathbb{R}^n$, where $n \in \mathbb{N}$ denotes the number of input features. The hidden layer is composed of $m \in \mathbb{N}$ neurons and produces a hidden representation $\mathbf{h} \in \mathbb{R}^m$, obtained through a nonlinear transformation of the input.

The forward propagation through the network can be described by the following

equations:

$$\mathbf{h} = g(\mathbf{W}_1 \mathbf{i} + \mathbf{b}_1), \quad (2.29)$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2, \quad (2.30)$$

where $g(\cdot)$ denotes an element-wise activation function.

The elements appearing in the 2.29 and 2.30 are defined as follows:

- $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ is the weight matrix connecting the input layer to the hidden layer;
- $\mathbf{W}_2 \in \mathbb{R}^{l \times m}$ is the weight matrix connecting the hidden layer to the output layer;
- $\mathbf{b}_1 \in \mathbb{R}^m$ is the bias vector associated with the hidden layer;
- $\mathbf{b}_2 \in \mathbb{R}^l$ is the bias vector associated with the output layer;
- $g(\cdot)$ denotes an element-wise activation function;
- $\mathbf{o} \in \mathbb{R}^l$ represents the output vector of the network.

Learning Process

Learning in a neural network consists of finding a set of parameters $\boldsymbol{\theta}$ that provides the best possible approximation of a target function. This learning process is performed iteratively and can be decomposed into four main steps.

- **Forward pass:** the input data $\mathbf{X}$ is propagated through the network layer by layer. Using the current parameters $\boldsymbol{\theta}$, the network produces a predicted output:

$$\hat{\mathbf{Y}} = f(\mathbf{X}; \boldsymbol{\theta}). \quad (2.31)$$

- **Loss computation:** the predicted output is compared with the true target value, and a loss function $L(\boldsymbol{\theta})$ is evaluated. The choice of the loss function depends on the specific learning task. One of the most widely used loss functions in Reinforcement Learning is the Mean Squared Error (MSE), defined as:

$$L_{\text{MSE}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \|y_i - \hat{y}_i\|_2^2, \quad (2.32)$$

where N is the number of samples, y_i denotes the true value, and $\hat{y}_i$ is the predicted value.

- **Backpropagation:** backpropagation [26] is the core mechanism that enables neural networks to learn. Each neuron can be interpreted as a mathematical function, and the network itself is a composition of such functions. Applying the chain rule of calculus, the gradient of the loss function with respect to each network parameter is computed. These gradients are propagated backward through the network, in the opposite direction of the forward pass.
- **Parameter update:** the network parameters are updated using gradient-based optimization methods. The standard gradient descent update rule is given by:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k), \quad (2.33)$$

where α is the learning rate controlling the step size of the update.

Backpropagation does not require explicit prior knowledge of the internal structure of the network beyond its differentiable formulation, but its performance may be affected by imperfect or noisy training data.

Several optimization algorithms extend basic gradient descent. Stochastic Gradient Descent (SGD) performs updates using mini-batches of data [27], while adaptive methods such as AdaGrad [28] and Adam [29] further improve convergence by dynamically adjusting the learning rate.

Regularization

Regularization refers to a set of machine learning techniques aimed at preventing overfitting. Overfitting is a phenomenon in which models learn noise or random details from the training data instead of the underlying patterns. Regularization works by adding a penalty term to the loss function, discouraging overly complex models and large parameter values. [30]

Given a loss function $L(\boldsymbol{\theta})$, regularization modifies the optimization objective as:

$$L_{\text{reg}}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}), \quad (2.34)$$

where $\Omega(\boldsymbol{\theta})$ is the regularization term and $\lambda > 0$ controls the strength of the regularization and they are better described by 2.35 and 2.36.

Two widely used regularization methods are L1 and L2 regularization.

- **L1 regularization:** penalty is defined as the ℓ_1 norm of the parameter vector:

$$\Omega_{\text{L1}}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i|. \quad (2.35)$$

L1 regularization leads to a sparser solution by driving some parameters exactly to zero, performing implicit feature selection.

- **L2 regularization:** penalty is defined as the squared ℓ_2 norm of the parameter vector:

$$\Omega_{\text{L2}}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_i \theta_i^2. \quad (2.36)$$

L2 regularization shrinks parameter values toward zero but rarely forces them to be exactly zero, leading to smoother and more stable models.

L1 and L2 can be combined to obtain another regularization method called Elastic Net Regularization [31]. Regularization plays a fundamental role in modern deep learning, where models often contain millions of parameters. Its importance has been extensively discussed as a key mechanism enabling deep neural networks to generalize effectively [32].

Activation Functions

An activation function performs two tasks: determines whether a neuron is activated and controls the output it produces. After computing the weighted sum of the inputs and the bias, the activation function introduces nonlinearity into the model, enabling neural networks to approximate complex non-linear functions.

Among the most commonly used activation functions are the following:

- **Sigmoid:** the sigmoid function maps the input to the interval $(0,1)$ and is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (2.37)$$

It is often used when the output is interpreted as a probability.

- **Hyperbolic tangent:** the hyperbolic tangent function maps the input to the interval $(-1,1)$ and is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (2.38)$$

Compared to the sigmoid function, it is zero-centered and often leads to faster convergence.

- **Rectified Linear Unit (ReLU):** the ReLU function introduces a simple piecewise linear nonlinearity and is defined as

$$\text{ReLU}(x) = \max(0, x). \quad (2.39)$$

ReLU is widely used in deep neural networks due to its computational efficiency and its ability to mitigate the vanishing gradient problem.

All these variants can be set as f in 2.28.

Batch Normalization Mini-batch learning allows a learning algorithm to update its parameters using a subset of the training data, called a mini-batch, rather than a single sample or the entire dataset. This approach improves computational efficiency and provides a balance between stochastic and full-batch gradient descent [18].

In reinforcement learning, mini-batches are often constructed from previously collected transitions consisting of observations, actions, rewards, and next states. During training, this experience may be replayed one or multiple times to update the model parameters. Training with mini-batches is generally faster and more stable than processing the entire dataset at once.

Batch Normalization [33] is a technique designed to stabilize and accelerate the training of deep neural networks. It normalizes the inputs of a layer within a mini-batch so that they have approximately zero mean and unit variance.

Given a mini-batch $\mathcal{B} = \{x_1, \dots, x_m\}$, batch normalization computes the batch mean and variance as

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2. \quad (2.40)$$

The normalized activations are then obtained as

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}, \quad (2.41)$$

where $\varepsilon > 0$ is a small constant introduced to avoid division by zero.

Finally, the batch normalization layer applies a learned affine transformation

$$y_i = \gamma \hat{x}_i + \beta, \quad (2.42)$$

where γ and β are learnable parameters that are optimized together with the original network parameters θ .

By introducing these additional parameters the optimization dynamics are improved in the neural network. Applying batch normalization to the input data or to the outputs of hidden layers typically results in faster convergence, improved regularization, and reduced overfitting.

2.2.2 Value-based Methods

Value-based reinforcement learning algorithms aim to learn an approximation of the optimal action–value function $Q^*(s, a)$ by parameterizing it as $Q_\theta(s, a)$ and optimizing an objective derived from the Bellman optimality equation, see 2.14. The learned action–value function is then used to infer the optimal policy 2.16.

Most value-based algorithms are off-policy, meaning that they can exploit all collected experience during training, including samples generated by past experiences, rather than relying only on the most recent interactions. This allows for efficient reuse of data, often through experience replay mechanisms. This is the reason that makes value-based methods, generally, more sample-efficient than policy-based approaches. However, they are also known to be less stable, particularly when combined with function approximation and bootstrapping.

Action selection in value-based methods is typically performed by choosing the action that maximizes the estimated action–value function:

$$a(s) = \arg \max_a Q_\theta(s, a). \quad (2.43)$$

Once the action–value function has been learned, the corresponding policy can be obtained directly by acting greedily with respect to Q_θ .

One of the most widely used and influential value-based algorithms is the Deep Q-Network (DQN) [21], which extends classical Q-learning [34] by approximating the action–value function with a deep neural network. DQN introduced key stabilizing techniques, such as experience replay and target networks, enabling value-based methods to scale to high-dimensional state spaces.

Deep Q-Network

The Deep Q-Network (DQN) algorithm builds upon the classical Q-learning framework by using a deep neural network as a function approximator for the action–value function. DQN was introduced by DeepMind in 2014, where it successfully learned to play a wide range of Atari 2600 games directly from raw pixel inputs [21].

In DQN, the action–value function $Q(s, a)$ is approximated by a neural network parameterized by θ . For visual input domains, this network is implemented as a convolutional neural network (CNN) [35], enabling the extraction of meaningful features from high-dimensional image data.

The network outputs a vector of Q-values, one for each available action. Action selection is typically performed using an ε -greedy policy, choosing the action that maximizes the estimated Q-value with high probability. The action with the highest Q-value is assumed to give the best expected cumulative return.

The optimization of the DQN loss function is performed using a stochastic gradient-based optimizer (SGD); in the original implementation, a variant of RMSProp was employed [36]. To address the instabilities arising in the value-based methods, DQN introduces two key mechanisms: a target network, which provides stable targets for the Q-value updates, and an experience replay buffer, which stores past transitions and allows efficient reuse of data while reducing temporal correlations.

Target Network The presence of a *target network* allows the loss function to be computed using a separate and more stable set of parameters, instead of those of the online (local) network that is updated at every iteration. This mechanism is essential in value-based DRL to reduce instabilities that arise during the training phase.

To better understand it, let θ denote the parameters of the online network and θ^- those of the target network. While θ is updated at each gradient step, the target network parameters are updated only every $I \in \mathbb{N}$ iterations. During the I iterations between updates, the target network remains fixed, preventing rapid oscillations and divergence caused by simultaneously learning the value estimates and their targets.

The target network is periodically synchronized with the online network according to

$$\theta^- \leftarrow \theta \quad \text{every } I \text{ iterations.} \quad (2.44)$$

Using the target network, the TD target for Q-learning is defined as

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-), \quad (2.45)$$

where r_t is the immediate reward and $\gamma \in [0,1]$ is the discount factor.

The loss function minimized by the online network is given by

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[(y_t - Q(s_t, a_t; \theta))^2 \right], \quad (2.46)$$

where $\mathcal{D}$ denotes the experience replay buffer.

By keeping the target network fixed for I iterations, the learning process resembles supervised learning with slowly varying targets, leading to improved training stability and more reliable convergence.

Experience Replay Experience replay memory is another key component introduced with DQN to improve training stability and data efficiency [37]. In reinforcement learning, consecutive transitions collected during interaction with the environment are temporally correlated, which can lead to inefficient learning and overfitting when used directly to train a neural network.

To mitigate this issue, DQN employs an experience replay buffer that stores a finite set of past transitions. Each transition is typically represented as a tuple $(s_t, a_t, r_{t+1}, s_{t+1})$. The buffer has a fixed capacity $N_{\text{replay}} \in \mathbb{N}$; when the buffer is full, older experiences are discarded as new ones are added.

During training, the algorithm samples mini-batches uniformly at random from the replay buffer rather than using consecutive transitions. This random sampling breaks temporal correlations between samples, provides a more diverse set of state-action pairs for each update, and leads to more stable gradient estimates. As a result, experience replay reduces variance in the learning updates and improves the generalization capability of the neural network.

The following algorithm comes from [21]:

Algorithm 3: Deep Q-learning with Experience Replay

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ ;
Initialize action-value function  $Q(\cdot, \cdot; \boldsymbol{\theta})$  with random weights  $\boldsymbol{\theta}$ ;
for  $episode = 1$  to  $M$  do
    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed state  $\phi_1 = \phi(s_1)$ ;
    for  $t = 1$  to  $T$  do
        With probability  $\varepsilon$ , select a random action  $a_t$ ;
        Otherwise select
            
$$a_t = \arg \max_a Q(\phi(s_t), a; \boldsymbol{\theta})$$

        Execute action  $a_t$  in the environment and observe reward  $r_t$  and next
        observation  $x_{t+1}$ ;
        Set  $s_{t+1} = (s_t, a_t, x_{t+1})$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ ;
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in replay memory  $\mathcal{D}$ ;
        Sample a random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ ;
        Set target
            
$$y_j = \begin{cases} r_j, & \text{if } \phi_{j+1} \text{ is terminal,} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \boldsymbol{\theta}), & \text{otherwise} \end{cases}$$

        Perform a gradient descent step on
            
$$(y_j - Q(\phi_j, a_j; \boldsymbol{\theta}))^2$$

        according to the Q-learning update rule;
    end
end

```

2.2.3 Policy gradient methods

Policy gradient methods directly optimize a parameterized policy $\pi_\theta(a | s)$ by maximizing a performance objective defined over some trajectories. The objective function is the expected cumulative reward, defined as:

$$J(\theta) = \mathbb{E} \left[\sum_{t=0}^N r(s_t, a_t) ; \pi_\theta \right] \quad (2.47)$$

The 2.47 can also be written as a sum over trajectories:

$$J(\theta) = \sum_{\tau} P(\tau; \theta) r(\tau), \quad (2.48)$$

where:

- $\tau = (s_0, a_0, \dots, s_N, a_N)$ denotes a trajectory generated by following the policy π_{θ} ;
- $P(\tau; \theta)$ is the probability of observing trajectory τ under the policy, and $r(\tau)$ is the cumulative reward associated with the trajectory.

The objective of policy gradient methods is to find the optimal policy parameters by solving:

$$\theta^* = \arg \max_{\theta} J(\theta). \quad (2.49)$$

Since the goal is to maximize $J(\theta)$, gradient ascent is employed rather than gradient descent. Once the policy gradient $\nabla_{\theta} J(\theta)$ has been computed, the policy parameters are updated according to:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta_t), \quad (2.50)$$

where $\alpha > 0$ is the learning rate, it controls the step size of the update. A large α allows faster learning with the risk of overshooting or causing divergence. A small α implies a more stable and smooth convergence but it brings over a slow learning, so more iterations could be needed for convergence.

Policy gradient methods offer several advantages. The policy is updated directly rather than being derived from an estimated value function therefore they offer a more stable convergence than value-based methods. They naturally handle continuous action spaces, since actions are sampled from a parameterized distribution rather than selected via maximization. These methods are also well suited to partially observable environments, as they can learn stochastic policies.

However, policy gradient methods also suffer from important limitations. In particular, the optimization objective is generally non-convex, which makes these methods prone to convergence toward local optima. As a consequence, convergence to a globally optimal policy cannot be guaranteed.

Actor-Critic

Actor–Critic methods are a class of reinforcement learning algorithms that combine the strengths of policy-based and value-based approaches in order to improve learning stability and sample efficiency [38].

The **actor** is responsible for selecting actions by directly parameterizing and updating a policy $\pi_{\theta}(a | s)$. Given a state, the actor outputs a probability distribution over the available actions and samples an action accordingly.

The **critic** evaluates the actions taken by the actor by estimating a value function that can be either a state-value function $V(s)$ or an action–value function $Q(s, a)$. The value function is a feedback to the actor given by estimating the quality of the selected action in the given state.

The interaction between the actor and the critic is driven by the Temporal Difference (TD) error, which measures the difference between the observed reward and the critic’s

current value estimate. The TD error is used to update both components: the critic updates its value estimates, while the actor adjusts its policy parameters to favor actions that yield higher than expected returns.

By combining direct policy optimization with value-based evaluation, actor-critic methods reduce the high variance typical of pure policy gradient methods while avoiding the instability often observed in value-based methods with function approximation.

2.2.4 PPO - Proximal Policy Optimization

Proximal Policy Optimization (PPO) [23] is a policy gradient algorithm designed to improve training stability while maintaining simplicity and sample efficiency. Unlike classical policy gradient methods, PPO allows multiple epochs of mini-batch updates using the same collected data, significantly improving sample efficiency.

PPO is closely related to Trust Region Policy Optimization (TRPO) [39] and was proposed to address the same fundamental question: *what is the largest policy improvement that can be made without destabilizing learning or collapsing the policy?* TRPO addresses this problem by enforcing a hard constraint on the Kullback–Leibler (KL) divergence between the old and new policies [40]. It measures how a probability distribution $q(x)$ differs from a reference distribution $p(x)$, representing the information lost when $q(x)$ approximates $p(x)$. In TRPO, the KL-Divergence measures how far is the new policy from the old one. This results in a second-order optimization problem that is computationally expensive and complex to implement. PPO achieves a similar objective using first-order optimization. It makes the solution simpler while it retains most of the benefits of TRPO.

Two main variants of PPO have been proposed.

- **PPO with KL Penalty (PPO-Penalty)**: this variant constrains the policy update by adding a KL-divergence penalty term to the objective function. Unlike TRPO, the constraint is not enforced strictly; instead, the penalty coefficient is adjusted adaptively during training to keep policy updates within a desired trust region.
- **PPO with Clipping (PPO-Clip)**: this variant removes the explicit KL-divergence term and relies on a clipped surrogate objective. The clipping mechanism limits the change in the probability ratio between the new and old policies, preventing excessively large policy updates. This approach is simpler, more robust in practice, and does not require tuning or monitoring a KL constraint.

Due to its simplicity, stability, and strong empirical performance, the PPO-Clip variant is the focus of this thesis. The following equations follow the process in [41].

In PPO-Clip, policy updates are performed by maximizing a surrogate objective as in equation 2.52. The surrogate objective is used instead of the true objective as it is more manageable and easier to optimize. It is a local approximation of how much better or worse is the new policy compared to the older one. At iteration k , the policy parameters are updated as:

$$\theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{s,a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \quad (2.51)$$

In practice, this maximization is approximated using multiple epochs of mini-batch

stochastic gradient ascent (SGD). The PPO clipped surrogate objective is defined as:

$$L(s, a, \theta_k, \theta) = \min \left(\underbrace{r(\theta) A^{\pi_{\theta_k}}(s, a)}_{\text{Policy Gradient Objective}}, \underbrace{\text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) A^{\pi_{\theta_k}}(s, a)}_{\text{Clipped Poly Gradient Objective}} \right) \quad (2.52)$$

where:

•

$$r(\theta) = \frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)} \quad (2.53)$$

is called *probability ratio* and it measures how much the new policy has changed with respect to the older one. Given the action a in state s $r(\theta)$ indicates how much an action is likely to happen:

- $r(\theta) > 1$ means action a is more likely to happen given state s ;
- $r(\theta) < 1$ means action a is less likely to happen given state s .
- $A^{\pi_{\theta_k}}(s, a)$ is the advantage function. A positive value mean the action is good, therefore increase its probability to happen. A negative value of A means that action is bad, therefore the probability of it happening has to be decreased.
- $\epsilon > 0$ is a hyperparameter controlling the maximum allowed policy deviation.

An equivalent and simplified form of the objective is given by

$$L(s, a, \theta_k, \theta) = \min(r(\theta) A, g(\epsilon, A)), \quad (2.54)$$

with

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A, & A \geq 0, \\ (1 - \epsilon)A, & A < 0. \end{cases} \quad (2.55)$$

The clipping mechanism acts as a regularizer by removing incentives for large policy updates. Once the probability ratio exceeds the interval $[1 - \epsilon, 1 + \epsilon]$, further deviations no longer improve the objective. ϵ measures how much a new policy can detach from the older one with the same goal. As a result, PPO achieves stable learning while retaining the simplicity of first-order optimization methods. The pseudo code for the algorithm presented is from [41]:

Algorithm 4: PPO-Clip

Initialize policy parameters θ_0 and value function parameters ϕ_0 ;
for *iteration* $k = 0, 1, 2, \dots$ **do**
 Collect a set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running the current policy π_{θ_k} in the environment;
 Compute rewards-to-go $\hat{R}_t$ for all time steps t ;
 Compute advantage estimates $\hat{A}_t$ using the current value function V_{ϕ_k} ;
 for *epoch* $= 1$ **to** K **do**
 Sample a minibatch of state-action pairs (s_t, a_t) from $\mathcal{D}_k$;
 Update the policy parameters by performing stochastic gradient ascent on

$$L^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(s_t, a_t) \in \mathcal{B}} \min\left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t\right)$$

 using Adam or another first-order optimizer;
 end
 for *epoch* $= 1$ **to** K_v **do**
 Sample a minibatch of states s_t from $\mathcal{D}_k$;
 Update the value function parameters by minimizing the mean squared error

$$L^V(\phi) = \frac{1}{|\mathcal{B}|} \sum_{s_t \in \mathcal{B}} \left(V_{\phi}(s_t) - \hat{R}_t\right)^2$$

 using gradient descent;
 end
end

2.3 Single Agent Taxonomy

Having introduced the main characteristics of the principal reinforcement learning algorithms, it is now possible to organize them within a unified framework. Based on the taxonomy proposed in [42], reinforcement learning methods can be systematically classified according to their learning strategy, policy representation, and use of environment models.

This taxonomy provides a structured overview of the relationships between value-based, policy-based, and actor-critic methods, as well as the distinction between model-free and model-based approaches. Figure 2.6 illustrates this classification and highlights how modern deep reinforcement learning algorithms fit within this framework.

As shown, this taxonomy provides a structured map to navigate the extensive amount of research produced in recent years in the field of reinforcement learning. In addition to organizing algorithms into coherent categories, it also highlights relationships and shared principles across different algorithmic families.

Within the scope of this thesis, *Proximal Policy Optimization (PPO)* is the primary algorithm of interest the core focus of the proposed analysis and experimental evaluation.

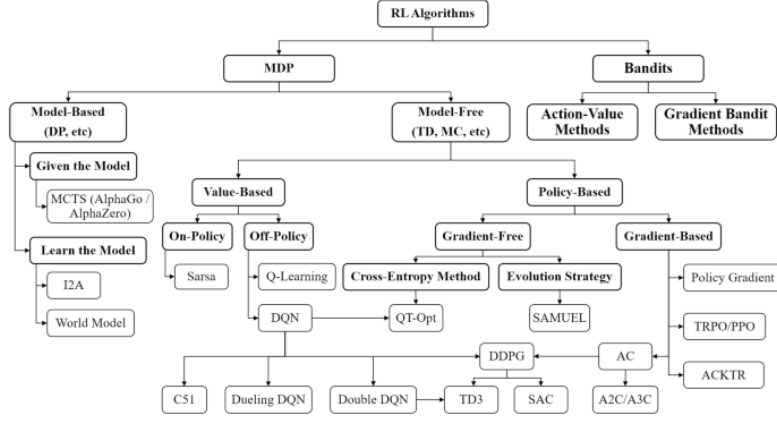


Figure 2.6: Taxonomy Classification according to [42].

2.4 MAS - Multi-Agent Systems

In this section, the fundamentals of multi-agent systems is presented. Markov Decision Process (MDP) introduced in section 2.1.2 for the single-agent case is extended to a multi-agent formulation. The objective of a Multi-Agent Reinforcement Learning (MARL) system is to learn optimal decision policies for two or more artificial agents interacting within a shared environment. All the following elements in this section are inspired by [43] and [44].

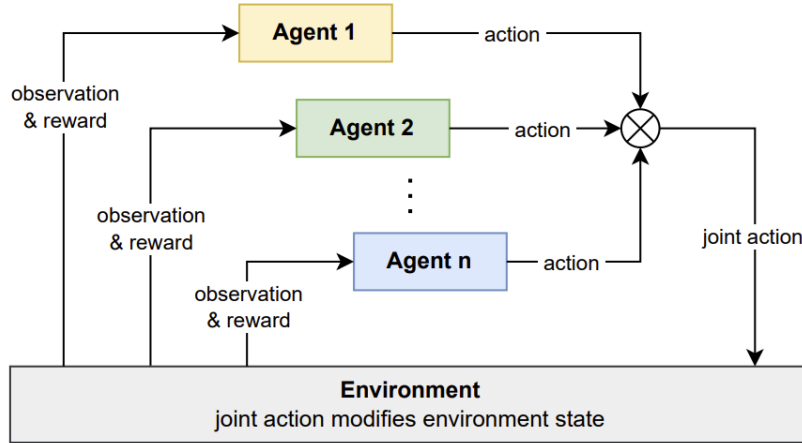


Figure 2.7: Multi-Agent schematic. Each agent receives reward and contribute to choose the best joint action. Source of image [43].

As in the single-agent setting, the fundamental building blocks of a MARL system are the environment and the agents. However, in this case, multiple agents simultaneously observe, act, and learn, often influencing both the environment and other agents. The

overall goal is to learn optimal policies for a set of agents operating within a Multi-Agent System (MAS).

The use of MARL becomes particularly valuable when it is necessary to decompose large-scale problems or to implement decentralized decision-making strategies, as commonly required in real-world scenarios such as energy systems, traffic control, and distributed robotics.

2.4.1 Benefits of MARL

The main advantages of MARL include:

- **Parallelization and training speed-up:** Learning can be distributed among multiple agents, allowing parallel experience collection and policy updates. This often results in faster convergence or improved responsiveness compared to fully centralized control approaches.
- **Experience sharing:** When agents face similar or related tasks, sharing experiences or learned representations can significantly improve learning efficiency and generalization.
- **Robustness and fault tolerance:** The distributed nature of MARL systems enhances robustness. If one agent fails or behaves suboptimally, other agents can compensate, maintaining overall system performance.
- **Scalability:** MARL systems are scalable by design. Decentralized decision-making enables control over complex and large-scale environments that would be impractical for a single centralized controller.

2.4.2 Challenges in MARL

Despite its advantages, MARL introduces several fundamental challenges:

- **Non-stationarity:** The presence of multiple learning agents causes the environment to become non-stationary from the perspective of any individual agent. As agents update their policies, the transition dynamics and reward distributions change, making previously learned policies quickly outdated and complicating convergence.
- **Scalability and curse of dimensionality:** The joint state and action spaces grow exponentially with the number of agents, leading to a combinatorial explosion that significantly increases computational complexity.
- **Policy optimality:** In single-agent reinforcement learning, optimality is defined as maximizing the expected cumulative return. In MARL, however, multiple equilibrium points may exist, and defining or identifying a globally optimal solution becomes significantly more complex.
- **Credit assignment:** Determining how individual agents contribute to a shared reward is a major challenge. Properly assigning credit to agents' actions is crucial for stable and efficient learning, especially in cooperative or mixed cooperative-competitive settings.

2.4.3 Hierarchy of Multi-Agent System Games

Multi-Agent Systems (MAS) can be formally modeled as *games*, depending on the interaction structure among agents and between agents and the environment. These games formulations can be organized in a hierarchy of increasing complexity, ranging from normal-form games to stochastic games and eventually partially observable stochastic games.

Normal-form games represent the simplest class of multi-agent interactions. They describe a single, simultaneous interaction among two or more agents. Each agent selects an action without knowledge of the actions chosen by the others. The outcome of the interaction is determined by a payoff function defined over the joint action space.

Normal-form games can be categorized as:

- **Zero-sum games**, where one agent's gain corresponds exactly to another agent's loss (e.g., Rock–Paper–Scissors);
- **Common-reward (cooperative) games**, where all agents share the same reward function;
- **General-sum games**, where agents may have partially aligned or conflicting objectives.

These games are static and do not model temporal dynamics or state transitions.

Stochastic Games

Stochastic games, first described by Shapley in 1953 [45] generalize both normal-form games and Markov Decision Processes (MDPs), introduced in 2.1.2. This can be done by introducing state transitions and sequential decision-making in a multi-agent setting. At each time step, all agents select actions, the environment transitions to a new state according to a joint transition function, and each agent receives an individual reward.

In their standard formulation, stochastic games assume full observability of the environment. That is, all agents have access to the true global state, allowing them to construct complete state–action–reward histories and define trajectories for learning and control. The following mathematical definitions and deductions come from [43]:

Definition 1 (Stochastic Game) *A stochastic game is defined by the tuple:*

$$\langle I, S, \bar{S}, \{A_i\}_{i \in I}, \{R_i\}_{i \in I}, T, \mu \rangle,$$

where:

- $I = \{1, \dots, n\}$ is a finite set of agents;
- S is a finite set of states, with $\bar{S} \subset S$ denoting the set of terminal states;
- for each agent $i \in I$:
 - A_i is a finite set of actions;
 - $R_i : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, where $A = A_1 \times \dots \times A_n$ is the joint action space;
- $T : S \times A \times S \rightarrow [0,1]$ is the state transition probability function, such that:

$$\forall s \in S, \forall a \in A : \sum_{s' \in S} T(s, a, s') = 1;$$

- $\mu : S \rightarrow [0,1]$ is the initial state distribution, satisfying:

$$\sum_{s \in S} \mu(s) = 1 \quad \text{and} \quad \forall s \in \bar{S} : \mu(s) = 0.$$

A stochastic game develops as follows:

- The game starts at time $t = 0$ with an initial state $s_0 \in S$ sampled from μ .
- At each time step t , each agent $i \in I$ observes the current state s_t and selects an action $a_t^i \in A_i$ according to its policy $\pi_i(a_t^i | h_t)$, where:

$$h_t = (s_0, a_0, s_1, a_1, \dots, s_t)$$

denotes the state-action history.

- Each action is part of the joint action is $a_t = (a_t^1, \dots, a_t^n)$. Assuming full observability, all agents observe the same history h_t .
- Given s_t and a_t , the environment transitions to a next state s_{t+1} with probability:

$$T(s_{t+1} | s_t, a_t),$$

and each agent i receives a reward:

$$r_t^i = R_i(s_t, a_t, s_{t+1}).$$

The interaction continues until a terminal state $s_t \in \bar{S}$ is reached or a maximum time horizon is exceeded. In non-terminating games, the process may continue indefinitely.

Stochastic games satisfy the Markov property:

$$\Pr(s_{t+1}, r_t | s_t, a_t, \dots, s_0, a_0) = \Pr(s_{t+1}, r_t | s_t, a_t),$$

where $r_t = (r_t^1, \dots, r_t^n)$ is the joint reward vector. For this reason, stochastic games are also referred to as *Markov games*.

Partially Observable Stochastic Games

Partially Observable Stochastic Games (POSGs) extend stochastic games by relaxing the full observability assumption. Instead of observing the true global state, each agent receives a partial and possibly noisy observation of the environment.

This limited observability substantially increases the complexity of the learning problem, as agents must operate under uncertainty and may need to rely on internal memory or belief states. POSGs provide a more realistic framework for modeling real-world multi-agent systems, such as decentralized control and cooperative robotics. The following mathematical definitions and deductions come from [43]:

Definition 2 (Partially Observable Stochastic Game) *A partially observable stochastic game (POSG) extends a stochastic game 1 by introducing partial observability. In addition to the elements of a stochastic game, a POSG defines for each agent $i \in I$:*

- a finite set of observations O_i ;

- an observation function:

$$O_i : A \times S \times O_i \rightarrow [0,1],$$

such that

$$\forall a \in A, \forall s \in S : \sum_{o_i \in O_i} O_i(a, s, o_i) = 1.$$

A POSG starts in an initial state $s_0 \in S$ sampled from the initial stated distribution μ . At each time step t , the environment is in state $s_t \in S$ and the previous joint action is $a_{t-1} \in A$ (with $a_{-1} = \emptyset$ for $t = 0$). Each agent $i \in I$ receives an observation $o_t^i \in O_i$ with probability

$$O_i(o_t^i | a_{t-1}, s_t).$$

Each agent selects an action $a_t^i \in A_i$ according to its policy $\pi_i(a_t^i | h_t^i)$, where

$$h_t^i = (o_0^i, o_1^i, \dots, o_t^i)$$

denotes the agent's observation history. The joint action is $a_t = (a_t^1, \dots, a_t^n)$.

Given the current state s_t and joint action a_t , the environment transitions to the next state $s_{t+1} \in S$ according to

$$T(s_{t+1} | s_t, a_t),$$

and each agent i receives a reward

$$r_t^i = R_i(s_t, a_t, s_{t+1}).$$

The interaction continues until a terminal state $s_t \in \bar{S}$ is reached or a fixed time horizon T is exceeded; otherwise, the game may be non-terminating.

POSGs can be classified as zero-sum, common-reward, or general-sum games. Common-reward POSGs are also known as *Decentralized Partially Observable Markov Decision Processes* (Dec-POMDPs) [46].

POSGs include stochastic games as the special case in which $o_t^i = (s_t, a_{t-1})$ for all agents, and POMDPs as the special case in which $|I| = 1$. Observation functions in POSGs can model diverse observability conditions, including discrete, continuous, or mixed observation spaces.

2.4.4 Training and Execution Paradigms

In multi-agent reinforcement learning (MARL), the learning process can be conceptually divided into two distinct phases: the *training phase* and the *execution phase*. Each phase can be either centralized or decentralized, leading to different learning paradigms.

Training Phase During training, agents update their policies or value functions based on observed data.

- **Centralized Training (CT):** A centralized training setting assumes access to global information during learning. This may include the full environment state, as well as the observations and actions of all agents. A central controller or centralized critic can exploit this global information to stabilize learning and mitigate the non-stationarity induced by simultaneously learning agents.

- **Decentralized Training (DT):** In decentralized training, each agent learns independently using only its local observations, actions, and rewards. No information about other agents' observations, actions, or policies is available. This setting corresponds to independent reinforcement learning and typically suffers from non-stationarity, since other agents are perceived as part of a changing environment.

Execution Phase During execution (deployment), agents interact with the environment using the learned policies.

- **Centralized Execution (CE):** In centralized execution, a central controller selects the joint action for all agents at each time step. This requires access to global information at runtime and may not be feasible in distributed or large-scale systems.
- **Decentralized Execution (DE):** In decentralized execution, each agent selects its own action based solely on its local observation:

$$a_i = \pi_i(o_i).$$

No centralized coordinator or global information is required at runtime. This setting is generally preferred in practical applications due to scalability and robustness considerations.

Combined Paradigms Combining these choices leads to different MARL paradigms:

- **Centralized Training and Decentralized Execution (CTDE):** Agents are trained using global information, but execute independently using local observations only. This paradigm stabilizes learning while preserving decentralized deployment and is currently the dominant framework in MARL [47].
- **Decentralized Training and Decentralized Execution (DTDE):** Agents learn and act independently, without access to global information at any stage. While highly scalable and simple to implement, this paradigm may suffer from instability due to non-stationarity [48]

2.4.5 IPPO - Independent Proximal Policy Optimization

The algorithm presented in this section is based on PPO presented in section 2.2.4. IPPO was firstly introduced in [49] and it operates in a Dec-POMDP. IPPO is a multi agent algorithm where each agents acts as an independent entity with its own actor network and critic network. Agents maintain individual observations histories and choose actions via decentralized policies. The only available information for each agent are the own observations and actions, it does not have access to others actions or observations. There is no central critic here. It is defined by the tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, \mathcal{O}, \rho, \gamma \rangle$, where:

- $\mathcal{N} = \{1, \dots, N\}$ is the set of N cooperative agents.
- $s \in \mathcal{S}$ is the global environment state; $s_0 \sim \rho$.
- $\mathcal{U}$ is the shared action space; each agent a selects $u_t^a \in \mathcal{U}$.
- $P(s_{t+1} | s_t, \mathbf{u}_t)$ is the transition kernel over joint actions $\mathbf{u}_t = \{u_t^a\}_{a=1}^N$.
- $r_t = r(s_t, \mathbf{u}_t)$ is a *shared* scalar team reward.
- Each agent a draws a local observation $z_t^a \in \mathcal{Z}$ via $\mathcal{O}(s_t, a)$.

- $\gamma \in [0,1)$ is the discount factor.

Agents maintain individual observation histories $\tau_t^a \in \mathcal{T}_t$ and choose actions via decentralized policies $u_t^a \sim \pi^a(\cdot \mid \tau_t^a)$. The team maximizes the expected discounted return:

$$J(\pi) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \quad (2.56)$$

Independently of the CTDE/DTDE distinction, IPPO can be implemented in two ways depending on whether agents share network weights:

- **IPPO with parameter sharing** [49] A single actor network π_θ and a single critic network V_ϕ are maintained. At each training update the experiences collected by all N agents are pooled into one batch and used to update this one shared network. Execution remains fully decentralized: every agent queries the same network within its own local observation. This variant is appropriate for *homogeneous* agents — agents whose observation and action spaces are semantically identical — and delivers N times more training signal per update step. Check 5 for the code.
- **Fully independent IPPO.** Each agent maintains its own actor π_{θ^a} and critic V_{ϕ^a} , trained exclusively on that agent’s own rollout data. No weights, gradients, or experiences are ever shared between agents. This variant is more appropriate for *heterogeneous* agents — agents whose observations carry different semantics across agents — since each network is free to specialize. The cost is that each network sees only $1/N$ of the total experience, making learning slower and more data-hungry. Check 6 for the code.

Both variants use the same PPO clipping objective. Some of the key characteristics from [49] are the following:

- **Full decentralisation.** Both actor and critic use only local observations in both IPPO variants, making the algorithm executable without any communication at test time. This distinguishes IPPO from MAPPO, where the critic requires global state during training.
- **Policy clipping is essential.** Ablation studies confirm that removing the clipped surrogate objective (reducing IPPO to Independent Actor-Critic) causes significant performance degradation, particularly on hard cooperative tasks [49]. The clipping mechanism stabilizes training under the non-stationarity induced by simultaneously updating all agents’ policies.
- **Value clipping is beneficial but task-dependent.** Value clipping provides additional regularization on tasks where the critic suffers from high variance. Both policy and value clipping use the same $\epsilon = 0.2$ in the original paper.
- **The sharing of parameters improves the efficiency of the sample for homogeneous agents.** In the parameter-sharing variant, all N agents’ experiences are pooled into a single batch, effectively multiplying the amount of training data by N per update. This is only meaningful when agents are homogeneous, i.e. when observations carry the same semantics across all agents.
- **Fully independent networks are preferable for heterogeneous agents.** When agents differ in their observation characteristics — as is the case for buildings in the CityLearn environment — separate networks allow each agent to specialize.

The trade-off is reduced sample efficiency, since each network is updated only from its own agent’s experience.

- **Competitive with centralized methods.** Despite using no global state information, IPPO matches or surpasses value-factorization algorithms such as QMIX on several hard SMAC maps, challenging the assumption that centralized critics are necessary for strong cooperative performance.

Algorithm 5: IPPO with Parameter Sharing (Schroeder et al., 2021)

```

for iteration  $k = 0, 1, 2, \dots$  do
  Set  $\theta_{\text{old}} \leftarrow \theta_k, \phi_{\text{old}} \leftarrow \phi_k$ ;
  (Decentralized rollout collection with shared policy);
  foreach agent  $a \in \mathcal{N}$  in parallel do
    Collect trajectories  $\mathcal{D}_k^a = \{\tau_i^a\}$  by running the shared policy  $\pi_{\theta_{\text{old}}}$  in the
    environment;
    Compute rewards-to-go  $\hat{R}_t^a$ ;
    Compute TD-errors  $\delta_t^a$ ;
    Compute GAE advantages  $\hat{A}_t^a$  using the shared value function  $V_{\phi_{\text{old}}}$ ;
    Normalize  $\hat{A}_t^a$  over the rollout;
  end
  Aggregate dataset:  $\mathcal{D}_k = \bigcup_{a \in \mathcal{N}} \mathcal{D}_k^a$ ;
  (Shared policy update);
  for epoch = 1 to  $K$  do
    Sample a minibatch  $\mathcal{B}$  from  $\mathcal{D}_k$ ;
    Update shared policy parameters by stochastic gradient ascent on

    
$$L^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(z_t^a, u_t^a) \in \mathcal{B}} \min\left(r_t^a(\theta) \hat{A}_t^a, \text{clip}(r_t^a(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^a\right) + \lambda_{\text{entropy}} \mathcal{H}[\pi_\theta]$$


    Clip gradient norm to  $g_{\text{max}}$  and update  $\theta$  using Adam;
  end
  (Shared value function update);
  for epoch = 1 to  $K_v$  do
    Sample a minibatch of states  $z_t^a$  from  $\mathcal{D}_k$ ;
    Update shared value parameters by minimizing

    
$$L^V(\phi) = \frac{1}{|\mathcal{B}|} \sum_{z_t^a \in \mathcal{B}} \left(V_\phi(z_t^a) - \hat{R}_t^a\right)^2$$


    Clip gradient norm to  $g_{\text{max}}$  and update  $\phi$  using Adam;
  end
end

```

Algorithm 6: Fully Independent IPPO

```

for iteration  $k = 0, 1, 2, \dots$  do
  (Decentralized rollout collection and independent learning);
  foreach agent  $a \in \mathcal{N}$  in parallel do
    Set  $\theta_{\text{old}}^a \leftarrow \theta_k^a, \phi_{\text{old}}^a \leftarrow \phi_k^a$ ;
    Collect trajectories  $\mathcal{D}_k^a = \{\tau_i^a\}$  by running the agent-specific policy  $\pi_{\theta_{\text{old}}^a}^a$ 
      in the environment;
    Compute rewards-to-go  $\hat{R}_t^a$ ;
    Compute TD-errors  $\delta_t^a$ ;
    Compute GAE advantages  $\hat{A}_t^a$  using the agent-specific value function
       $V_{\phi_{\text{old}}^a}$ ;
    Normalize  $\hat{A}_t^a$  over the rollout;
    (Agent-specific policy update);
    for epoch = 1 to  $K$  do
      Sample a minibatch  $\mathcal{B}$  from  $\mathcal{D}_k^a$ ;
      Update agent-specific policy parameters by stochastic gradient ascent
        on
        
$$L^{\text{CLIP}}(\theta^a) = \frac{1}{|\mathcal{B}|} \sum_{(z_t^a, u_t^a) \in \mathcal{B}} \min\left(r_t^a(\theta^a) \hat{A}_t^a, \text{clip}(r_t^a(\theta^a), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^a\right) + \lambda_{\text{entropy}} \mathcal{H}[\pi_{\theta^a}]$$

      Clip gradient norm to  $g_{\text{max}}$  and update  $\theta^a$  using Adam;
    end
    (Agent-specific value function update);
    for epoch = 1 to  $K_v$  do
      Sample a minibatch of states  $z_t^a$  from  $\mathcal{D}_k^a$ ;
      Update agent-specific value parameters by minimizing
        
$$L^V(\phi^a) = \frac{1}{|\mathcal{B}|} \sum_{z_t^a \in \mathcal{B}} \left(V_{\phi^a}(z_t^a) - \hat{R}_t^a\right)^2$$

      Clip gradient norm to  $g_{\text{max}}$  and update  $\phi^a$  using Adam;
    end
  end
end

```

2.4.6 MAPPO - Multi Agent Proximal Policy Optimization

MAPPO (Multi-Agent Proximal Policy Optimization) is an on-policy multi agent reinforcement learning algorithm proposed by Yu et al. [15]. It extends PPO to cooperative agent settings using CTDE paradigm. How it works. Each agent has its own local observation as input and the policy network outputs an actions that will be executed. Each agent sees only its own observations and actions. The critic network take in all the agents'

observations. This signal gives the critic much better signal for computing the value estimation, which is transformed in better policy gradient updates. The key characteristics can be resumed as:

- **Centralized critic, decentralized actors.** The critic sees the global state during training but agents act independently on local observations at execution time.
- **Parameter sharing.** In environments with homogeneous agents, both policy and value function parameters are shared across the agent. In this implementation all buildings are homogeneous.
- **Difference with the paper:** Here the domain is energy management in CityLearn, where each building controls the charge/discharge of its electrical storage to minimize electricity cost, carbon emissions, peak demand and ramping at the district level. There is no death masking since buildings do not become inactive mid-episode, and the cooperative reward is shaped around real-world energy KPIs rather than game scores.

Algorithm 7: MAPPO with Centralized Critic and Parameter Sharing

```

for iteration  $k = 0, 1, 2, \dots$  do
  Set  $\theta_{\text{old}} \leftarrow \theta_k, \phi_{\text{old}} \leftarrow \phi_k$ ;
  (Decentralized rollout collection; centralized critic input);
  Reset environment; observe  $\{o_0^i\}_{i=1}^n$ ;
  Build global state  $\mathbf{s}_0 = [o_0^1 \parallel \dots \parallel o_0^n]$ ;
  for  $t = 0, 1, \dots, T - 1$  do
    foreach agent  $i \in \{1, \dots, n\}$  do
      | Sample action  $a_t^i \sim \pi_{\theta_{\text{old}}}(\cdot \mid o_t^i)$ ;
    end
    Execute joint action  $\mathbf{A}_t = (a_t^1, \dots, a_t^n)$ ;
    Receive shared reward  $r_t$  and next observations  $\{o_{t+1}^i\}_{i=1}^n$ ;
    Build next global state  $\mathbf{s}_{t+1} = [o_{t+1}^1 \parallel \dots \parallel o_{t+1}^n]$ ;
    Store  $(\{o_t^i, a_t^i\}_{i=1}^n, r_t, \mathbf{s}_t, \mathbf{s}_{t+1})$ ;
  end
  (Advantage estimation with centralized critic);
  for  $t = T - 1, \dots, 0$  do
    |  $\delta_t \leftarrow r_t + \gamma V_{\phi_{\text{old}}}(\mathbf{s}_{t+1}) - V_{\phi_{\text{old}}}(\mathbf{s}_t)$ ;
    |  $\hat{A}_t \leftarrow \delta_t + (\gamma\lambda)\hat{A}_{t+1}$ ;
  end
  Normalize  $\hat{A}_t$  over the rollout buffer;
  Compute returns  $R_t \leftarrow \hat{A}_t + V_{\phi_{\text{old}}}(\mathbf{s}_t)$ ;
  (Shared policy update);
  for epoch = 1 to  $K$  do
    Sample a minibatch  $\mathcal{B}$  of size  $M$  from the rollout buffer;
    Update shared actor parameters by stochastic gradient ascent on

    
$$L^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \sum_{i=1}^n \min\left(r_t^i(\theta)\hat{A}_t, \text{clip}(r_t^i(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t\right),$$


    
$$r_t^i(\theta) = \frac{\pi_{\theta}(a_t^i \mid o_t^i)}{\pi_{\theta_{\text{old}}}(a_t^i \mid o_t^i)}$$


    Clip gradient norm to  $g_{\text{max}}$  and update  $\theta$  using Adam;
  end
  (Centralized value function update);
  for epoch = 1 to  $K_v$  do
    Sample a minibatch  $\mathcal{B}$  of size  $M$  from the rollout buffer;
    Update critic parameters by minimizing

    
$$L^V(\phi) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} (V_{\phi}(\mathbf{s}_t) - R_t)^2$$


    Clip gradient norm to  $g_{\text{max}}$  and update  $\phi$  using Adam;
  end
end

```

2.4.7 Taxonomy

Now that the main challenges and the main characteristics of MAS were presented it is possible to define a taxonomy to identify and to categorize them. It is not possible to have a tree structure like the single-agent approach therefore an orthogonal axis definition better suits the need.

Busoniu et al. [5] approached the taxonomy of MARLs focusing primarily on how agents learn in relation to each other. Their taxonomy is focused on the the interaction model structuring them as:

- **Independent learners.** Each agent treats the others as part of the environment;
- **Join-action Learners.** Agents are aware of others in the environment;
- **Centralized vs decentralized approach.** It classifies the algorithm comparing the learning paradigm, whether they use only local information or fully distributed approach;
- **Comparative vs Competitive Settings.** The paper later adds the interaction section by distinguishing cooperative, competitive or mixed settings;

This taxonomy focuses more on how agents relate to each other during learning.

Canese et al. [6] the focus shifts on the task structure whether the problem is fully cooperative, fully competitive one or a mixed/general sum. A major part of their taxonomy is structured around the core technical challenges of MARL, including: Non-stationarity, Credit Assignment, scalability, Partial observability, communication learning, and coordination mechanism. They highlight as well the Centralized Training with Decentralized Execution, a Fully decentralized learning and a communication-based learning. However, in this case, it is not a main organizing principle. The shift focus on the MARL architectures categorizing approaches as:

- Value Decomposition Methods;
- Policy-gradient-based multi-agent methods;
- Communication architecture.

Zhang et al. [42] the main focus is on the mathematical model organizing the algorithms on the game structure they solve. They also clearly separate the algorithmic Learning Paradigms in:

- Value-based methods;
- Policy-based methods;
- Actor-Critic methods;
- Fictitious play;
- Gradient-based dynamics.

Another distinctive trait is on the convergence and sample complexity, as they care more about stability, and mathematical tractability. Also in here the information structure accounted for.

Hu et al. [8] the focus is more on algorithmic evolution and practical application, many of the algorithms categorized in this section were found in this paper.

Existing MARL surveys organize algorithms around single dominant organizing principles (interaction structure, task structure, or theoretical framework). However, none provide a fully orthogonal multi-axis taxonomy where independent dimensions of MARL are explicitly separated.

Orthogonal Axis An orthogonal classification of the Multi-Agent algorithms is set up. Each axis answer represents a specific dimension aligned with a specific characteristic. The six axis are presented here:

1. **Policy type** - What kind of policy optimization paradigm does the algorithm use?
This axis tells *how* the agents updates their policy:
 - **Value-based**: It derives a policy implicitly from value estimation.
 - **Policy-Gradient**: Direct optimization of a parametrized policy, no value functions and no critics involved.
 - **Actor-Critic**: Two network are used: a policy (actor) optimized by policy gradients and a value function (critic) learned separately, both are required and are jointly trained.
 - **Population based**: Optimizes policies using evolutionary, genetic, or sampling-based population search instead of gradient descent. No value functions or policy-gradient updates are used.
2. **Learning Regime**: How are the policy parameters updated during the learning?
This axis represents the sampling strategy and is controlled by the training set up. It tells *from where* the learning data comes:
 - **On-Policy**: Agents learn only from trajectories generated by the current policy.
 - **Off-Policy**: Agents learn from trajectories generated by other policies or older experience (replay buffer).
 - **Evolutionary**: Policies are updated via evolutionary operators.
3. **Task type** - What is the relationship among agents in terms of goals and rewards?
This axis defines *reward relation* among agents:
 - **Fully cooperative**: maximize a common return.
 - **Fully competitive**: Agents compete with others for the same resources. The main goal is to maximize their own reward and minimize others.
 - **Mixed**: Combination of cooperation and competition. The rewards are inter-dependent (not identical, nor opposite).
4. **Awareness Degree** - To what extent does each agent model, observe, or account for the presence, actions, or policies of other agents? This axis tells the *perception* of others.
 - **Independent**: The other agents present are treated as part of the environments by the others. No explicit modeling of other learners.
 - **Aware**: There is an implicit Coordination. Agents sense or model others' action/rewards and make use of them in the updates.
 - **Tracking**: There is an explicit coordination. Agents explicitly learn models, others' policies and continuously adapt.
5. **Observability** - What information of others in terms of state and/or action spaces

are available to each agent? This axis tells *what information* are available to each other.

- **Global** Global state and others' actions known.
- **Actions Only** Actions from the other agents are known. No other states.
- **Local** Agents are fully blind to what others do. No observations or actions available from other agents.

6. **Training** - How is the training happening? This axis tells *how the agents are trained*.

- **Centralized**: The training set up is centralized, agents are sharing states or actions.
- **Decentralized**: The agents are training themselves only with local observations and local actions.

Table 2.1: Taxonomy legend

POLICY TYPE		
VB	Value-Based	Derives policy implicitly from value estimates
PG	Policy Gradient	Learns a differentiable policy directly; no separate critic network
AC	Actor-Critic	Two networks: actor (policy) + separately learned critic (value)
POP	Population	Maintains a population of policies; optimised via selection / oracle training
ON/OFF-POLICY		
ON	On-Policy	Learns only from trajectories of the current policy
OFF	Off-Policy	Learns from a replay buffer of past experience
EVO	Evolutionary	Iteratively grows a strategy population; no gradient required
TASK TYPE		
COOP	Fully Cooperative	Agents share a single team reward
COMP	Fully Competitive	Zero-sum: one agent's gain is another's loss
MIXED	Mixed	Agents have partially aligned / partially conflicting rewards
AWARENESS		
IND	Independent	Other agents treated as part of the environment; no explicit modelling
AWA	Aware	Agent uses others' actions / obs in training (e.g. centralised critic)
TRK	Tracking	Agent explicitly models and adapts to others' changing policies
OBSERVABILITY		
GLOB	Global	Critic / agent sees full joint state: others' observations AND actions
ACT	Actions only	Agent observes others' actions but NOT their private observations
LOC	Local	Agent sees only its own observation and action history
TRAINING		
CTR	Centralised	Training uses a central controller with global information
DEC	Decentralised	Training uses only local information; no central controller

In the following table some of the main algorithms are presented, categorized with the 6 axis taxonomy.

2.5 Related work

This work is based on the CityLearn 2022 dataset, which includes all 17 buildings. This section focuses on studies that use this dataset with battery control actions.

In [50], the objective was to minimize electricity consumption, electricity cost, and carbon emissions, while improving grid-level KPIs and preserving building load profiles. The authors implemented an independent Soft Actor-Critic (SAC) controller within a

Table 2.2: Main MARL algorithms classified according to the taxonomy defined

First Author	Year	Algorithm / Paper	Policy Type	On/Off-Policy	Task Type	Awareness	Observability	Training
Tan	1993	Ind. vs Coop. Agents	VB	OFF	COOP	IND	LOC	DEC
Schroeder	2020	IPPO	AC	ON	COOP	IND	LOC	DEC
Yu et al.	2022	MAPPPO	AC	ON	COOP	AWA	GLOB	CTR
Chu et al.	2020	IA2C / IAC	AC	ON	COOP	AWA	GLOB	CTR
Foerster	2018	COMA	AC	ON	COOP	AWA	GLOB	CTR
Lowe et al.	2017	MADDPG	AC	OFF	MIXED	AWA	GLOB	CTR
Kuba et al.	2021	HATRPO / HAPPO	PG	ON	COOP	AWA	GLOB	CTR
Sunehag	2018	VDN	VB	OFF	COOP	AWA	GLOB	CTR
Rashid	2018	QMIX	VB	OFF	COOP	AWA	GLOB	CTR
Son et al.	2019	QTRAN	VB	OFF	COOP	AWA	GLOB	CTR
Wang et al.	2021	QPLEX	VB	OFF	COOP	AWA	GLOB	CTR
Yang et al.	2020	QATTEN	VB	OFF	COOP	AWA	GLOB	CTR
Hu & Wellman	2003	Nash Q-Learning	VB	OFF	MIXED	AWA	ACT	DEC
Heinrich	2016	NFSP	VB	OFF	COMP	IND	ACT	DEC
Lanctot	2017	PSRO	POP	EVO	MIXED	AWA	ACT	CTR
Foerster	2018	LOLA	PG	ON	MIXED	TRK	ACT	DEC
Wen et al.	2019	PR2	PG	ON	MIXED	TRK	ACT	DEC
Jiang et al.	2020	DGN	VB	OFF	COOP	AWA	GLOB	CTR
Ackermann	2019	MATD3	AC	OFF	COOP	AWA	GLOB	CTR
Zhang et al.	2020	MASAC	AC	OFF	COOP	AWA	GLOB	CTR
Iqbal & Sha	2019	MAAC	AC	OFF	MIXED	AWA	GLOB	CTR
Greenwald	2003	Correlated Q-Learning	VB	OFF	MIXED	AWA	ACT	CTR
Bowling	2002	WoLF-PHC	PG	ON	MIXED	IND	ACT	DEC
Bowling	2002	WoLF-IGA	PG	ON	MIXED	IND	ACT	DEC
Conitzer	2007	AWESOME	VB	OFF	MIXED	TRK	ACT	DEC
Tian et al.	2019	ROMMEO	AC	OFF	MIXED	TRK	ACT	DEC
Li et al.	2023	ACE	AC	OFF	COOP	AWA	GLOB	CTR
Ma et al.	2022	R2G	AC	OFF	MIXED	TRK	GLOB	CTR
Kim et al.	2022	VM3-AC	AC	ON	COOP	AWA	GLOB	CTR
Nasiri	2023	HAMDPO	PG	ON	COOP	AWA	GLOB	CTR
Mei et al.	2023	MAC-PO	AC	OFF	COOP	AWA	GLOB	CTR
Du et al.	2023	SCIC	AC	OFF	COOP	AWA	GLOB	CTR
Kar et al.	2013	QD-Learning	VB	OFF	COOP	AWA	ACT	DEC
Palmer	2018	Lenient DQN	VB	OFF	COOP	IND	LOC	DEC
Foerster	2017	Stabilizing MA-DQN	VB	OFF	MIXED	AWA	ACT	DEC
Hausknecht	2015	DRQN	VB	OFF	COOP	IND	LOC	DEC
Omidshafiei	2017	Dec-HDRQN	VB	OFF	COOP	IND	LOC	DEC
Gupta et al.	2017	PS-TRPO	PG	ON	COOP	IND	LOC	CTR
Wang & Sand.	2002	OAL	VB	OFF	COOP	AWA	ACT	DEC
Yang et al.	2018	Mean Field MARL	AC	OFF	MIXED	AWA	ACT	DEC
Macua et al.	2018	DAC	AC	ON	COOP	AWA	ACT	DEC
Macua et al.	2018	DiffDAC	AC	ON	COOP	AWA	ACT	DEC
Lee et al.	2018	Distributed GTD	VB	OFF	COOP	AWA	ACT	DEC
Littman	1994	Minimax Q-Learning	VB	OFF	COMP	AWA	ACT	DEC
Perolat	2018	AC Fictitious Play	AC	ON	MIXED	TRK	ACT	DEC
Best et al.	2019	Dec-MCTS	POP	EVO	COOP	AWA	LOC	DEC
Liang et al.	2021	MAHPPPO	AC	ON	COOP	AWA	GLOB	CTR
Sukhbaatar	2016	CommNet	PG	ON	COOP	AWA	GLOB	CTR
Shen et al.	2020	Comm. MADDPG	AC	OFF	MIXED	TRK	GLOB	CTR
Mahajan	2019	MAVEN	VB	OFF	COOP	AWA	GLOB	CTR
Jaques	2019	SIMOA	AC	ON	MIXED	AWA	ACT	DEC
Wang et al.	2020	DOP	AC	OFF	COOP	AWA	GLOB	CTR
Zhou et al.	2020	LICA	AC	ON	COOP	AWA	GLOB	CTR
Su et al.	2021	VDAC	AC	ON	COOP	AWA	GLOB	CTR
Li & He	2021	MATRPO	PG	ON	COOP	IND	ACT	DEC
Ma & Luo	2022	VDPPPO	AC	ON	COOP	AWA	GLOB	CTR
Dhamankar et al.	2020	VFAC	AC	OFF	COOP	AWA	GLOB	CTR
Fang et al.	2021	MADQN+OES	VB	OFF	MIXED	AWA	GLOB	CTR
Vasquez-Canteli et al.	2020	MARLISA	AC	OFF	COOP	AWA	ACT	DEC
Han et al.	2025	SDMAPD	AC	OFF	COOP	AWA	GLOB	CTR
Tungom et al.	2024	SWOAM	POP	EVO	COOP	IND	GLOB	CTR
Tungom et al.	2025	SWAPP	POP	EVO	COOP	IND	GLOB	CTR

decentralized control architecture, where each building is controlled by a separate agent. The controllers are trained in the CityLearn environment, and learned policies can be transferred across buildings. Three deployment strategies were developed:

- In the first strategy, each building trains its own SAC policy over the entire year, with no transfer learning applied;
- In the second strategy, one building is used as a source, trained on the full dataset,

and its policy is transferred to the other buildings;

- In the third strategy, only five months of data are used for training; the learned policy is then transferred and further updated.

The results show improvements in both building-level and district-level KPIs compared to baseline controllers. Transfer learning caused only limited performance degradation compared to building-specific training, while maintaining superior performance relative to the baselines and reducing training cost.

In [51], the authors proposed a novel gradient-free algorithm called Swarm Optimized Agents for Sequential Decision Making (SWOAM). Unlike reinforcement learning approaches, this method is based on Particle Swarm Optimization (PSO), which optimizes policy parameters globally through metaheuristic search. The approach introduces state classes obtained through K-means clustering, providing a simplified representation of the environment. Instead of using raw observations, the agents operate on cluster identifiers, thereby reducing the complexity of the decision space. Each particle represents an agent, and agents train on different time windows such that, collectively, the swarm covers the full dataset. The authors report significant reductions in electricity cost and carbon emissions, as well as improvements in grid ramping, outperforming single-agent and multi-agent RL approaches in CityLearn.

In [52], the SWOAM framework is extended by introducing adaptive action-bound refinement. The algorithm dynamically shrinks or expands the allowable action range for each state class during training, improving convergence speed and policy stability.

In [14], the authors proposed a MAPPO framework combined with imitation learning to optimize a residential energy system including space heating and battery storage. Unlike the district-level setting in CityLearn, this study focuses on a single building composed of multiple thermal zones. The agents correspond to the heating controller and the battery controller. The work compares single-agent PPO with MAPPO and shows that it converges faster and improves both self-sufficiency and PV self-consumption.

In [16], authors combined forecasting models with Multi-Agent Reinforcement Learning to improve microgrid scheduling decisions. Instead of relying solely on RL, future load and solar generation are first predicted and then incorporated into the reinforcement learning controller. This creates a two-stage AI pipeline: the first stage forecasts future conditions, and the second stage optimizes battery scheduling using these forecasts. Two forecasting methods are trained: Gradient Boosted Decision Trees (GBDT) and Linear Regression (LR). The authors compare centralized PPO, Independent PPO (IPPO), and MAPPO, all augmented with forecast information. MAPPO achieves the best performance, highlighting the importance of cooperation in multi-agent scheduling problems.

Chapter 3

Technological Background

3.1 CityLearn

CityLearn [9] is a Gymnasium-based environment that includes energy models of buildings and distributed energy resources (DERs). For each building, it is possible to control cooling, heating, and domestic hot water (DHW) systems. The controllable assets may include heating storage, cooling storage, and electrical storage, but this depends on the chosen dataset. Not all storage systems are necessarily available in every building. CityLearn guarantees occupant thermal comfort at all times.

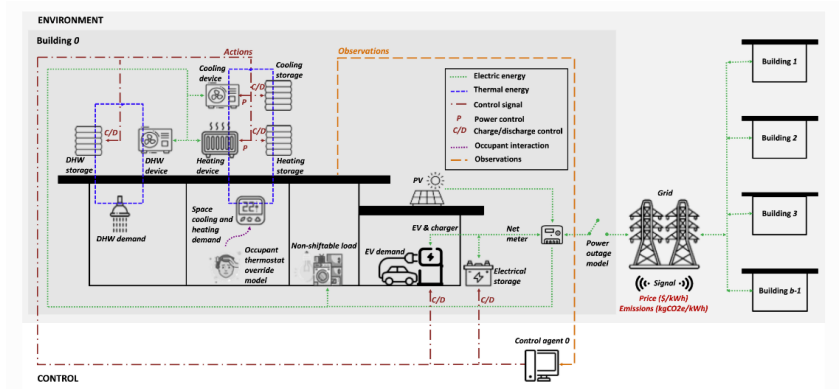


Figure 3.1: Structure of the CityLearn environment [9].

A building may include up to five types of loads: cooling, heating, domestic hot water (DHW), electrical appliances, and electric vehicle (EV) charging. Cooling and heating loads refer to the energy required to maintain the indoor dry-bulb temperature at its setpoint. In CityLearn, each building is modeled as a single thermal zone, where heat loads influence the indoor temperature. A long short-term memory (LSTM) regression model is used to approximate the thermal dynamics of the building, quantifying the effect of heat loads on temperature evolution. The occupant model may override temperature setpoints if necessary to ensure comfort.

The DHW heating load represents the total thermal energy required to satisfy hot

water demand. Electrical appliances are defined as non-shiftable electrical loads such as lighting, entertainment devices, and kitchen appliances. EV charging demand represents the energy required to charge an electric vehicle to a specified state of charge (SoC) before its scheduled departure. All loads are either derived from real-world measurements or generated a priori using building energy performance simulation (BEPS). It should be noted that not all loads are present in every building. For example, in very hot climates, heating storage may not be required. To satisfy these loads, different devices are modeled in the environment: cooling devices (heat pumps), heating devices, and domestic hot water devices (either heat pumps or electric heaters). All these devices are electrically powered and may be used to charge a thermal energy storage (TES) system. The available control variables correspond to the storage systems, that can be charged or discharge. Each building may have up to five storage units, which can be categorized into thermal and electrical storage systems. The thermal storage systems include:

- Cooling storage;
- Heating storage;
- Domestic hot water storage.

The electrical storage systems include:

- Electrical storage: a battery energy storage system (BESS) that can supply power to HVAC devices;
- Electric vehicle storage: available in some datasets, operating according to predefined arrival and departure schedules.

Buildings may also be equipped with photovoltaic (PV) systems that generate electrical energy. In all cases, buildings are connected to the electrical grid and can import energy without constraints, except during power outages. Excess on-site generation is exported to the grid as net electricity export.

3.1.1 Control Variables

Storage and devices can be controlled. The control action space is continuous in CityLearn and five control variables are dedicated to the Energy Storage systems and other two are dedicated to the cooling and heating device, respectively. The table 3.1 resume the control actions.

Table 3.1: CityLearn Control Variables and Action Ranges

Category	Name	Range	Description
Energy Storage	cooling storage	$[-1, 1]$	Proportion of cooling storage capacity to be charged ($a > 0$) or discharged ($a < 0$).
	heating storage	$[-1, 1]$	Proportion of heating storage capacity to be charged ($a > 0$) or discharged ($a < 0$).
	DHW storage	$[-1, 1]$	Proportion of DHW storage capacity to be charged ($a > 0$) or discharged ($a < 0$).
	electrical storage	$[-1, 1]$	Proportion of electrical storage capacity to be charged ($a > 0$) or discharged ($a < 0$).
	EV storage	$[-1, 1]$	Proportion of EV storage capacity to be charged ($a > 0$) or discharged ($a < 0$).
Electric Device	cooling device	$[0, 1]$	Proportion of nominal cooling device power supplied.
	heating device	$[0, 1]$	Proportion of nominal heating device power supplied.

Farama Foundation Gymnasium interface for standard RLC environment design are used in CityLearn environment. At each time step, the agent observes the system, takes an action, receives a reward, and learns a policy that maximizes long-term performance over the whole simulation. Citylearn allows three configurations:

- Single-Agent: one agent controls all the buildings, taking in all the observations, giving as output an action for each building and receiving a single reward value.
- Independent multi-agent: it has one agent per building, this means there is policy per building, therefore a reward per each building is given.
- Coordinate multi-agent: with this configuration sharing information among agents is allowed for the achievement of common objectives.

For this work, only the electrical storage will be controlled.

3.1.2 Observations

Different observations are available within CityLearn. They can be categorized as Calendar, Weather and Building observations. The calendar observations are presented in table 3.2, the weather observations are presented in table 3.3 and finally the building observations are shown in table 3.4.

Table 3.2: Calendar observations.

Category	Name	Source	Unit	Description
Calendar	month	building_id.csv		Month of year ranging from 1 (January) through 12 (December).
Calendar	day_type	building_id.csv		Day of week ranging from 1 (Monday) through 7 (Sunday).
Calendar	hour	building_id.csv		Hour of day ranging from 1 to 24.
Calendar	daylight_savings_status	building_id.csv		Boolean that indicates if the current day is daylight savings period.

Table 3.3: Weather observations

Category	Name	Source	Unit	Description
Weather	outdoor_dry_bulb_temperature	weather.csv	C	Outdoor dry bulb temperature.
Weather	outdoor_dry_bulb_temperature_predicted_6h	weather.csv	C	Outdoor dry bulb temperature predicted 6 hours ahead.
Weather	outdoor_dry_bulb_temperature_predicted_12h	weather.csv	C	Outdoor dry bulb temperature predicted 12 hours ahead.
Weather	outdoor_dry_bulb_temperature_predicted_24h	weather.csv	C	Outdoor dry bulb temperature predicted 24 hours ahead.
Weather	outdoor_relative_humidity	weather.csv	%	Outdoor relative humidity.
Weather	outdoor_relative_humidity_predicted_6h	weather.csv	%	Outdoor relative humidity predicted 6 hours ahead.
Weather	outdoor_relative_humidity_predicted_12h	weather.csv	%	Outdoor dry bulb temperature predicted 12 hours ahead.
Weather	outdoor_relative_humidity_predicted_24h	weather.csv	%	Outdoor dry bulb temperature predicted 24 hours ahead.
Weather	diffuse_solar_irradiance	weather.csv	W/m2	Diffuse solar irradiance.
Weather	diffuse_solar_irradiance_predicted_6h	weather.csv	W/m2	Diffuse solar irradiance predicted 6 hours ahead.
Weather	diffuse_solar_irradiance_predicted_12h	weather.csv	W/m2	Diffuse solar irradiance predicted 12 hours ahead.
Weather	diffuse_solar_irradiance_predicted_24h	weather.csv	W/m2	Diffuse solar irradiance predicted 24 hours ahead.
Weather	direct_solar_irradiance	weather.csv	W/m2	Direct solar irradiance.
Weather	direct_solar_irradiance_predicted_6h	weather.csv	W/m2	Direct solar irradiance predicted 6 hours ahead.
Weather	direct_solar_irradiance_predicted_12h	weather.csv	W/m2	Direct solar irradiance predicted 12 hours ahead.
Weather	direct_solar_irradiance_predicted_24h	weather.csv	W/m2	Direct solar irradiance predicted 24 hours ahead.

Table 3.4: Building Observations

Category	Name	Source	Unit	Category
Building	indoor_dry_bulb_temperature	building_id.csv/Runtime calculation	C	Building
Building	average_unmet_cooling_setpoint_difference	building_id.csv	\	Building
Building	indoor_relative_humidity	building_id.csv	%	Building
Building	non_shiftable_load	building_id.csv	kWh	Building
Building	solar_generation	building_id.csv	kWh	Building
Building	cooling_storage_soc	Runtime calculation	kWh/kWhcapacity	Building
Building	heating_storage_soc	Runtime calculation	kWh/kWhcapacity	Building
Building	dhw_storage_soc	Runtime calculation	kWh/kWhcapacity	Building
Building	electrical_storage_soc	Runtime calculation	kWh/kWhcapacity	Building
Building	net_electricity_consumption	Runtime calculation	kWh	Building
Building	electricity_pricing	pricing.csv	\$/kWh	Building
Building	electricity_pricing_predicted_6h	pricing.csv	\$/kWh	Building
Building	electricity_pricing_predicted_12h	pricing.csv	\$/kWh	Building
Building	electricity_pricing_predicted_24h	pricing.csv	\$/kWh	Building
Building	cooling_device_efficiency	Runtime calculation	1	Building
Building	heating_device_efficiency	Runtime calculation	1	Building
Building	dhw_device_efficiency	Runtime calculation	1	Building
Building	cooling_demand	building_id.csv/Runtime calculation	kWh	Building
Building	heating_demand	building_id.csv/Runtime calculation	kWh	Building
Building	dhw_demand	building_id.csv/Runtime calculation	kWh	Building
Building	cooling_electricity_consumption	Runtime calculation	kWh	Building
Building	heating_electricity_consumption	Runtime calculation	kWh	Building
Building	dhw_electricity_consumption	Runtime calculation	kWh	Building
Building	indoor_dry_bulb_temperature_set_point	building_id.csv	C	Building
Building	indoor_dry_bulb_temperature_delta	building_id.csv/Runtime calculation	C	Building
Building	power_outage	building_id.csv/Runtime calculation		Building

The observations used in this work are:

- hour;
- day_type;
- solar_generation;
- net_electricity_consumption;
- electircal_storage_soc;

3.1.3 Datasets

There are different datasets available. Each dataset may differ per number of buildings available, and per actions allowed to be performed. Also the simulation period is different, it can vary from thirty days to one year. The timestep is 1h, this means each step in the environment shifts forward of one hour. Different datasets were presented also for the yearly challenges. In the table ?? a comprehensive presentation of the available datasets with their respective number of buildings, control variables and simulation periods are presented.

Table 3.5: CityLearn datasets with number of buildings, control variables and simulation length

CityLearn Datasets	No. Bldgs	Action Space	Sim. Period [1h]
ca_alameda_county_neighbourhood	73	Charge/Discharge of: Domestic Hot Water storage; Battery; Cooling device; Heating device; Cooling or heating device.	1y

Continued on next page

CityLearn Datasets	No. Bldgs	Action Space	Sim. Period [1h]
Challenge_2020_climate_zone_1	9	Charge/Discharge of: Cooling storage; Heating storage; Domestic Hot Water storage; Battery.	1y
Challenge_2020_climate_zone_2	9	Charge/Discharge of: Cooling storage; Heating storage; Domestic Hot Water storage; Battery.	1y
Challenge_2020_climate_zone_3	9	Charge/Discharge of: Cooling storage; Heating storage; Domestic Hot Water storage; Battery.	1y
Challenge_2020_climate_zone_4	9	Charge/Discharge of: Cooling storage; Heating storage; Domestic Hot Water storage; Battery.	1y
Challenge_2021	9	Charge/Discharge of: Cooling storage; Heating storage; Domestic Hot Water storage; Battery.	4y
Challenge_2022_phase_1	17	Charge/Discharge of a battery.	1y
Challenge_2022_phase_2	17	Charge/Discharge of a battery.	1y
Challenge_2022_phase_3	17	Charge/Discharge of a battery.	1y
Challenge_2022_phase_all	17	Charge/Discharge of a battery.	1y
Challenge_2022_phase_all_plus_evs	17	Charge/Discharge of a battery.	1y
Challenge_2023_phase_1	3	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	30d
Challenge_2023_phase_2_local eval	3	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	30d
Challenge_2023_phase_2_online eval_1	3	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	30d
Challenge_2023_phase_2_online eval_2	3	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	30d
Challenge_2023_phase_2_online eval_3	3	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	30d
Challenge_2023_phase_3_1	6	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	92d
Challenge_2023_phase_3_2	6	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	92d
Challenge_2023_phase_3_3	6	Charge/Discharge of: Cooling storage; Domestic Hot Water storage; Battery.	92d
Quebec_neighbourhood_with demand_response_set_points	20	Heating device.	270d
Quebec_neighbourhood_without demand_response_set_points	20	Heating device.	270d
tx_travis_county_neighborhood	100	Charge/Discharge of: Domestic Hot Water; Electrical Storage; Cooling or heating device.	1y
vt_chittenden_county_neighborhood	47	Charge/Discharge of: Domestic Hot Water; Electrical Storage; Cooling or heating device.	1y

For this work the dataset "Challenge_2022_phase_all" is used. This dataset presents a 17 zero net energy (ZNE) buildings. They are family home situated in Sierra Crest Zero Net Energy community in Fontana, California. This dataset is the same used in the CityLearn Challenge 2022. The data spans from August 1, 2016 till July 31, 2017. The buildings date from the mid to late 2010s. The gross floor area spans between 177 up to 269 square meters. High performance materials were used for the envelope to improve efficiency along with high performance appliances, electric heating and water heating systems. Each building has a home energy management system that provides one-minute power time series data. The CityLearn dataset differs a bit from the as-built

community, 8/15 buildings have 6,5 kWh batteries with 5kW power rating, 90% round trip efficiency and 75% depth of discharge. PV capacity of 4kW or 5kW are installed for buildings without batteries. To make the dataset open source it was transformed and the following modifications were applied:

- The minute data was converted into hourly energy data measured in kWh.
- All buildings now have batteries with 6.4kWh capacity, 5kW power rating, 90% round-trip efficiency with a depth of discharge up to 100%.
- HVAC demands, DHW and plug loads have been included as non-shiftable loads.
- Data quality issues, including outliers and missing values, were handled through interquartile range (IQR)-based outlier detection, interpolation techniques, and supervised learning models for predictive imputation.

3.2 StableBaselines3 - SB3

StableBaselines3 (SB3) [53] is an open-source library of implementation of deep reinforcement learning algorithms with Pytorch. It provides a well-tested and reproducible implementations of state-of-the-art algorithms such as PPO, A2C, SAC. SB3 is used for the single-agent implementation of PPO as it can be easily integrated with CityLearn since it follows the OpenAI Gym interface. A CityLearn environment can be wrapped as a standard gym environment, making the agent-environment interaction smoother.

3.3 Optuna

Optuna [54] is an open-source framework for automatic hyperparameter optimization. It is designed to efficiently find good hyperparameter settings for machine learning models. Optuna's goal is to accelerate and automate the tuning process of hyperparameter as finding them manually can be difficult and very time consuming. Once the grid search is defined, a *study* can be created for the full optimization. In the study different *trials* are executed, each one with a specific set of hyperparameters chosen by optuna. The optimization goal is defined by an *objective function* defined by the user. At each trial, once the model is trained, the objective function is computed. Optuna chooses from the grid search the combinations of parameters that maximize or minimize the objective function.

3.4 Ray

Ray [55] is an open-source computing framework. It was developed at UC Berkeley and designed to scale Python applications from a single machine to large clusters with minimal code changes. RLlib [56] is Ray reinforcement library. It was released as part of Ray ecosystems and provides a unified API for training RL agents at scale. RLlib's primary aim is to abstract away the complexity of distributed rollout collection, gradient computation, and policy optimization, enabling researchers and practitioners to focus on algorithm design rather than infrastructure. It supports single-agent, multi-agent, model-based and offline RL. It natively integrates Ray Tune for scalable hyperparameter

search. In this work RLlib was used to implement IPPO and MAPPO in the CityLearn environment. A custom wrapper was created to integrate the API. Each building was treated as an independent agent identified by its name. For the MAPPO a custom module was implemented for a centralized critic enabling it to receive the concatenated global states. Hyperparameter optimization was carried out by integrating Ray Tune with Optuna combined with ASHAScheduler [57] for early stopping of unpromising trials. This allows an efficient exploration of the training configuration space.

Chapter 4

Methodology

4.1 Outline of the system

As presented in the subsection 3.1.3 the "challenge_2022_phase_all" dataset is used. The dataset spans over an entire year. From it a summer period and a winter period were analyzed. Each period is then divided into a train dataset and a winter dataset. The train and test datasets were structured as follow:

- Summer train: months of June and July, for a total of 60 days;
- Summer test: months of August for a total of 30 days;
- Winter train: months of November and December for a total of 60 days;
- Winter test: month of December for a total of 30 days.

The 60 days and 30 days period was chosen as it is easier to set up the batch sizes for the training set up.

The electricity pricing available in the dataset are in $\$/kWh$:

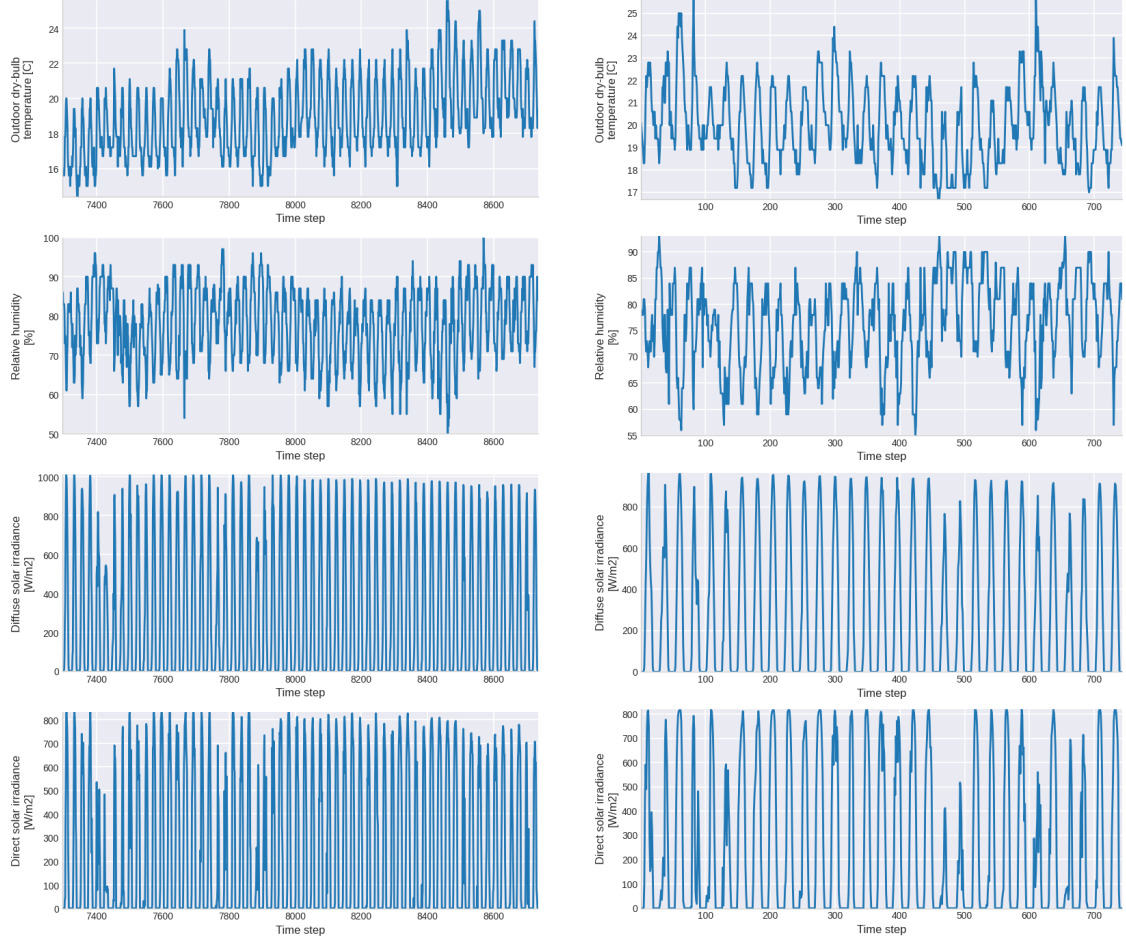
Table 4.1: Electricity Pricing

	June-September		October-May	
Time	Weekday	Weekend	Weekday	Weekend
8 AM-4 PM	0.21	0.21	0.20	0.20
4 PM-9 PM	0.54	0.40	0.50	0.50
9 PM-8 AM	0.21	0.21	0.20	0.20

In the figures 4.1, 4.2, summer and winter conditions in terms of Outdoor dry-bulb temperature, relative humidity, diffuse solar irradiance, and direct solar irradiance are presented.

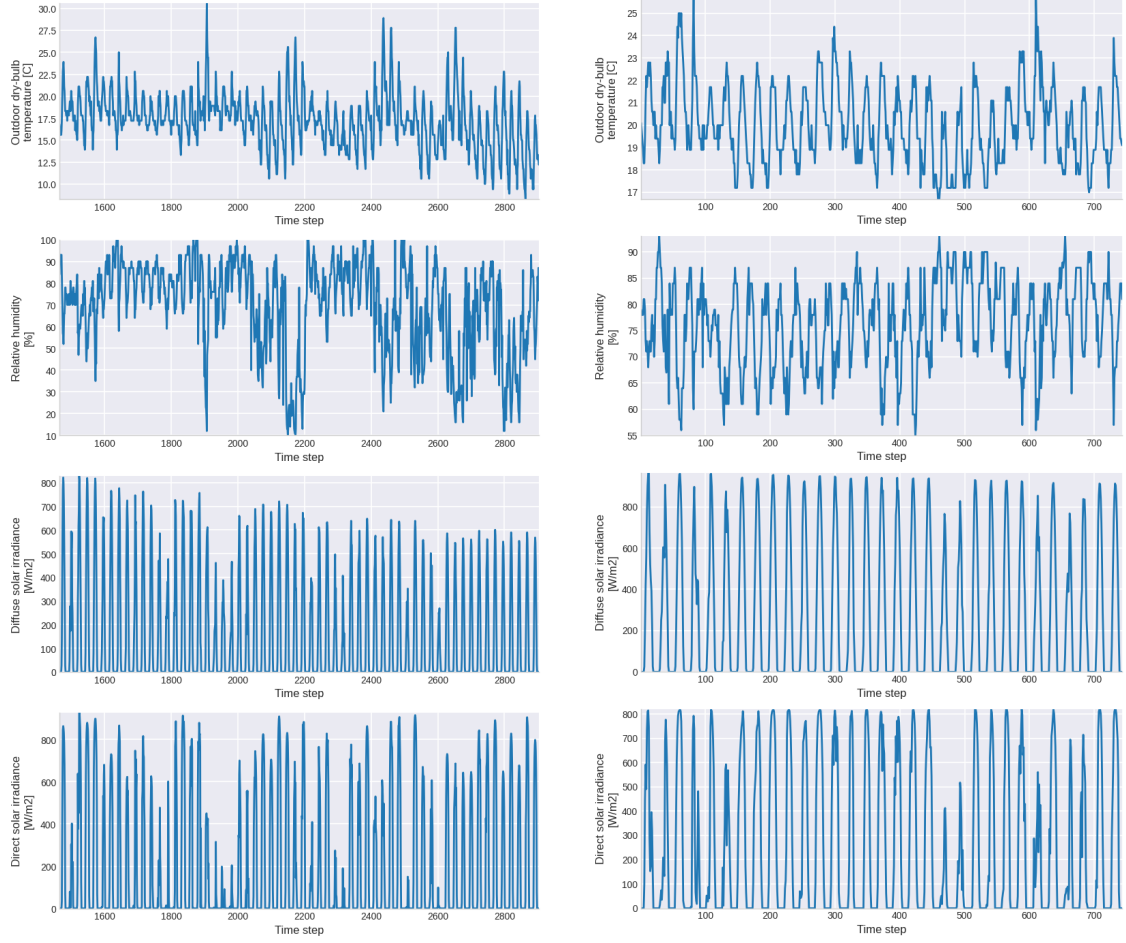
As it can be seen in the figure 4.3 and 4.4 the net load profiles are different from each buildings in absolute terms but on a qualitative basis they are similar. The objective of the study is to compensate the negative values, when the consumption is more than the production. It is also important to notice that the selected buildings came from a random choice, and are Buildings 1, 10, 13 and 16 as it can be seen in the figures.

Then the RBC map for both summer and winter is built as shown in table 4.2 [58].



(a) Summer - Train dataset weather conditions (b) Summer - Test dataset weather conditions

Figure 4.1: Summer weather conditions for train and test time span.



(a) Winter - Train dataset weather conditions (b) Winter - Test dataset weather conditions

Figure 4.2: Winter weather conditions for train and test time span.

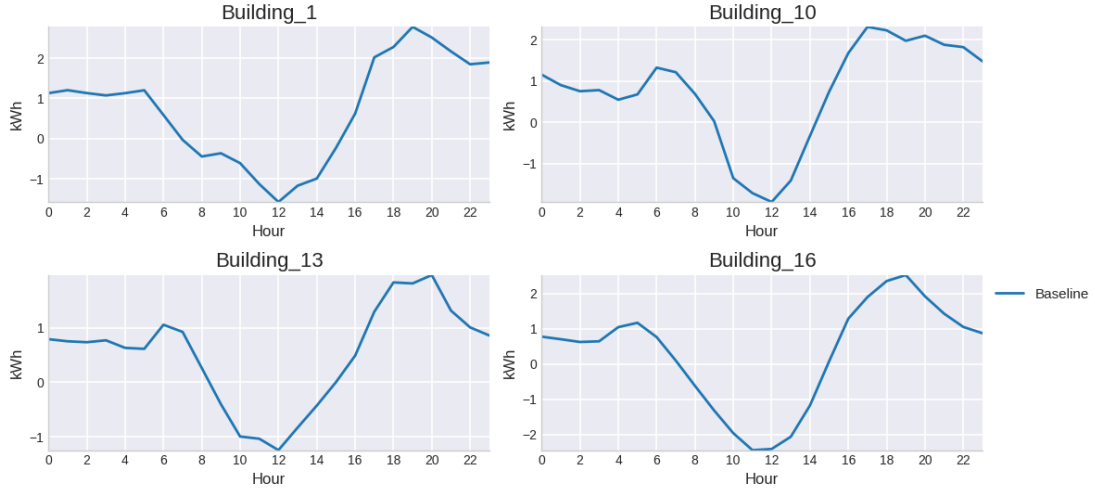


Figure 4.3: Summer net consumptions profile for the analyzed buildings

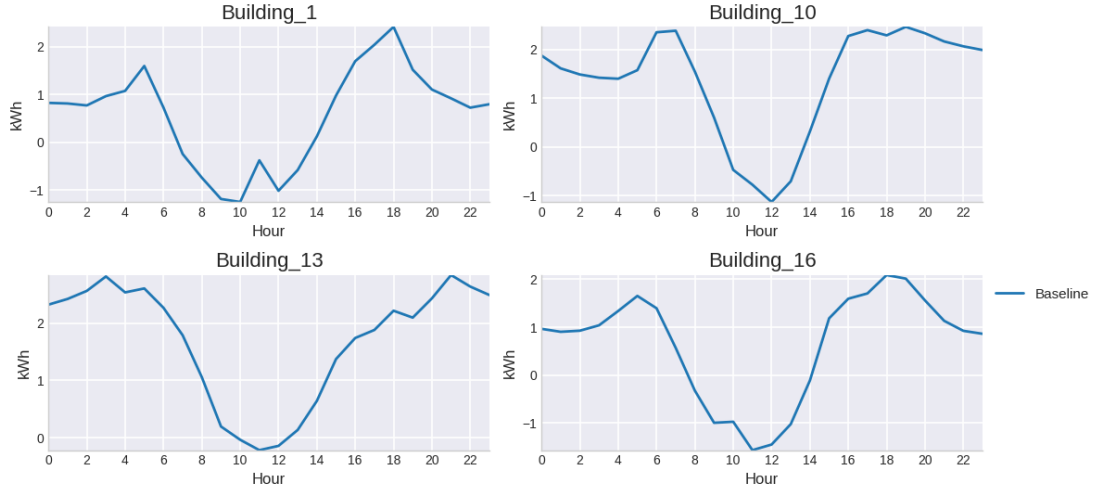


Figure 4.4: Winter net consumption profile for the analyzed buildings

Hour	Battery Action	Hour	Battery Action
1	0.00	13	0.15
2	0.00	14	0.15
3	0.00	15	0.15
4	0.00	16	0.05
5	0.00	17	0.00
6	0.00	18	-0.10
7	0.00	19	-0.20
8	0.10	20	-0.20
9	0.10	21	-0.10
10	0.10	22	-0.10
11	0.10	23	-0.10
12	0.10	24	-0.10

Table 4.2: Rule-based control (RBC) battery action schedule. Positive values indicate charging, while negative values indicate discharging.

Episodes Each hour in CityLearn is a timestep, therefore, a day is composed by 24 timesteps. To make the agent aware of the entire period, the episode will be considered completed at the end of the 60 days of training. This makes each training episode of 1440 timesteps. For the testing part, each episode will be half of it, 720 timesteps.

Actions There is only one control variable: the state of charge (SoC) of the battery of each building. The value can range from $[-1,1]$ where a positive sign means charge until you reach the integer value (e.g. $[0.6]$ charge the battery until the state of charge of 60%). A negative sign means discharge until you reach the integer value (e.g. $[-0.3]$ discharge the battery until the state of charge reaches 30%).

Observations The agents, whether if single-agent or multi agents, have the following active observations available:

- **hour**: it can be a value from 0 to 23.
- **day_type**: it can be a value from 1 to 7 where 1 states for Monday, 2 for Tuesday etc.
- **solar_generation**: it is measured in kWh and stands for the PV electricity generation.
- **net_electricity_consumption**: it is measured in kWh and stands for total building consumption.
- **electrical_storage_soc**: it is measured $kWh/kWhCapacity$ and stands for the SoC of the battery, it spans from 0, empty, to 1, full capacity.

Reward Function The primary objective of the control strategy is to minimize both electricity costs and carbon emissions. To achieve this, the agent must learn to charge the battery during periods when electricity prices are low. In addition, photovoltaic (PV) generation provides an opportunity to store locally produced renewable energy at no cost. In particular, the time window between 9:00 AM and 4:00 PM is advantageous for charging, as it typically coincides with high renewable energy sources (RES) production, mainly PV, and lower grid emission intensity. By storing energy during these hours, the agent can later discharge the battery during morning and evening peak periods, when electricity demand and carbon intensity are higher. This strategy contributes to reducing net grid consumption and flattening the overall load profile. The agent should learn to discharge whenever there is a positive net grid load and sufficient stored energy is available, mitigating peak demand and improving grid-level performance. The reward equation is the one defined in [58]:

$$r = \sum_{i=0}^n (p_i \times |C_i|) \quad (4.1)$$

$$p_i = - \left(1 + \text{sign}(C_i) \times \text{SoC}_i^{\text{battery}} \right) \quad (4.2)$$

Where:

- C_i is the energy cost and can be positive or negative;
- p_i is the penalty term for the building i ;

- $SoC_i^{battery}$ is the state of charge of the battery for the building i

The total reward of the action is a sum of the single buildings reward. It is important to mention that this happens for the single case agent as for the MAPPO implementation and for the IPPO with the parameter sharing. During training the overall reward is used. For the Independent IPPO each building receives its own reward and only this value is used for the actor and critic network training.

4.2 KPIs monitored

CityLearn monitors different KPIs within the environment. This work will monitor the following: electricity cost, carbon emission, average daily peak, ramping and load factor. The first two can be measured both at building level as a district level, while all five can be measured at district levels. All the KPIs are normalized with respect to the baseline to understand how much the agent improved the performances of the buildings. It is important to note that the baseline means that there are no battery and no photovoltaic generation.

Electricity Cost Electricity cost is computed as the cumulative cost of net positive electricity consumption at the building level. Let E_h^{building} denote the net electricity consumption at hour h , and p_h the electricity price at hour h . The total cost over n hours is defined as:

$$\text{cost} = \sum_{h=0}^{n-1} \max(0, E_h^{\text{building}} \cdot p_h) \quad (4.3)$$

Carbon Emissions Carbon emissions are calculated as the total emissions associated with positive grid electricity consumption. Let O_h denote the carbon intensity (kgCO_{2e}/kWh) at hour h :

$$\text{carbon emissions} = \sum_{h=0}^{n-1} \max(0, E_h^{\text{building}} \cdot O_h) \quad (4.4)$$

Average Daily Peak The average daily peak is defined as the mean of the daily maximum district-level net electricity consumption:

$$\text{average daily peak} = \frac{1}{n} \sum_{d=0}^{n-1} \max_{h \in \{0, \dots, 23\}} (E_{24d+h}^{\text{district}}) \quad (4.5)$$

Ramping Ramping measures the smoothness of the district load profile:

$$\text{ramping} = \sum_{h=1}^{n-1} |E_h^{\text{district}} - E_{h-1}^{\text{district}}| \quad (4.6)$$

Load Factor The load factor evaluates consumption efficiency. Since optimization is formulated as minimization, we minimize $(1 - \text{load factor})$:

$$1 - \text{load factor} = \frac{1}{n} \sum_{m=0}^{n-1} \left(1 - \frac{\frac{1}{d_m} \sum_{h=0}^{d_m-1} E_{h+H_m}^{\text{district}}}{\max_{h \in \{0, \dots, d_m-1\}} (E_{h+H_m}^{\text{district}})} \right) \quad (4.7)$$

Normalized KPI All KPIs are normalized with respect to the baseline scenario without battery storage and without PV:

$$\text{KPI} = \frac{\text{KPI}_{\text{control}}}{\text{KPI}_{\text{baseline (no battery)}}} \quad (4.8)$$

4.3 Simulation Workflow

Period and building Selection In this section the simulation work flow is presented. The first step was the training and testing simulation period as presented the section 4.1. The choices of the buildings are random. For the simulation the seed: 42 was set and the buildings 1, 10, 13 and 16 were chosen. These choices allow for a reduction of the cost of the training by: reducing the training period and the number of buildings.

Network configuration For the neural network configuration two layers of 64 neurons have been chosen for both critic and actor.

Hyper parameter selection Once the period and the buildings were selected, the next step is to build the algorithm. For the single agent the PPO from StableBaselines 3 [53] is chosen, the code can be seen in 4. The available PPO parameters are the following:

Table 4.3: PPO (SB3) Parameters and Default Values

Name	Standard Value	Description
policy	–	Policy model (e.g., MlpPolicy, CnnPolicy, MultiInputPolicy).
env	–	Environment instance or registered Gym environment ID.
learning_rate	3×10^{-4}	Learning rate (can be constant or schedule depending on training progress).
n_steps	2048	Number of steps per environment per update (rollout buffer size is $n_{\text{steps}} \times n_{\text{envs}}$).
batch_size	64	Minibatch size used during optimization.
n_epochs	10	Number of epochs used to optimize the surrogate loss per update.
gamma	0.99	Discount factor for future rewards.
gae_lambda	0.95	Bias-variance trade-off parameter for Generalized Advantage Estimation (GAE).
clip_range	0.2	Clipping parameter for the policy objective (can be constant or scheduled).

clip_range_vf	None	Clipping parameter for value function updates (OpenAI-specific implementation).
normalize_advantage	True	Whether to normalize the advantage estimates.
ent_coef	0.0	Entropy coefficient encouraging exploration.
vf_coef	0.5	Weight of the value function loss in the total loss.
max_grad_norm	0.5	Maximum gradient norm for gradient clipping.
use_sde	False	Whether to use generalized State Dependent Exploration (gSDE).
sde_sample_freq	-1	Frequency (in steps) for sampling new gSDE noise; -1 means once per rollout.
rollout_buffer_class	None	Custom rollout buffer class (automatically selected if None).
rollout_buffer_kwargs	None	Additional arguments for the rollout buffer.
target_kl	None	Optional limit on KL divergence between updates.
stats_window_size	100	Window size for logging average episode statistics.
tensorboard_log	None	Directory for TensorBoard logging (disabled if None).
policy_kwargs	None	Additional keyword arguments passed to the policy constructor.
verbose	0	Verbosity level (0: none, 1: info, 2: debug).
seed	None	Random seed for reproducibility.
device	auto	Device used for training (cpu, cuda, etc.; auto selects GPU if available).
_init_setup_model	True	Whether to build the model at initialization.

To find the best parameters a grid search was implemented, the grid search was implemented on the parameters in the table 4.4, while the others where set up at their default value. This selection remained the same for both single agent and multi agent.

Table 4.4: Sampled PPO Hyperparameters

Hyperparameter	Candidate Values
n_{steps}	{240, 480, 720, 1440}
Batch fraction	{0.25, 0.5, 1.0}
Learning rate (α)	$\{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}, 5 \cdot 10^{-3}, 7 \cdot 10^{-3}, 10^{-2}\}$
Number of epochs (n_{epochs})	{5, 10, 15, 20}
Discount factor (γ)	{0.80, 0.85, 0.875, 0.90, 0.92, 0.94, 0.96, 0.98, 0.99, 0.995, 0.999}
Entropy coefficient (c_{ent})	{0.001, 0.003, 0.01, 0.03, 0.1, 0.2}

The batch size is not sampled directly but derived from n_{steps} and a sampled batch fraction. It is computed as:

$$\text{batch_size} = n_{\text{steps}} \times \text{batch_fraction} \quad (4.9)$$

where $\text{batch_fraction} \in \{0.25, 0.5, 1.0\}$. Additionally, the batch size is adjusted when necessary to ensure that it exactly divides n_{steps} , i.e.,

$$n_{\text{steps}} \bmod \text{batch_size} = 0. \quad (4.10)$$

The batch size was derived from the batch section due to SB3 training constraints where:

$$\text{batch_size} \leq n_{\text{steps}} \quad (4.11)$$

Optuna Search and best model training The selection and the grid search is the same for both single-agent and multi-agent implementation.

Once the hyperparameter grid was defined a study was defined with Optuna [54] to find the best combination among the fixed ones and the combination presented in table 4.4. Each trial is set up to be launched and run for 100 episodes. After the Optuna screening for the best parameters, the best model is built with that configuration and is then trained for 400 episodes. This happens for both single-agent and multi-agent.

Chapter 5

Results

This chapter presents the experimental results obtained from the comparative evaluation of the four control strategies – PPO, IPPO, IPPO-SP, and MAPPO – tested against a no-storage baseline and a Rule-Based Controller (RBC). The evaluation is structured across two seasonal scenarios, summer and winter, each covering a training phase of 400 episodes and a testing phase of 30 days. For each scenario, results are analyzed along three dimensions: training convergence, building-level KPIs (energy cost, carbon emissions, and load flattening), and district-level KPIs (net load profile, peak demand, load factor, and grid ramping). Across both seasons, all control strategies consistently outperform the uncontrolled baseline, confirming the effectiveness of battery-based energy management. The single-agent PPO emerges as the most robust approach, achieving the strongest load flattening performance at both building and district level. Among the multi-agent methods, IPPO-SP demonstrates competitive cost reduction, particularly in winter, while MAPPO shows the weakest overall performance and occasional training instability. A recurring observation throughout the results is that building-specific characteristics significantly influence the achievable improvements, suggesting that the effectiveness of a given control architecture depends not only on its algorithmic design but also on the heterogeneity of the building stock it manages. For a more clear understanding, the different models characteristics are resumed here:

- **PPO**: Single-Agent PPO implementation.
- **IPPO-SP**: Multi-Agent PPO implementation with shared actor-critic networks. The input is each agent observation but the weights are shared.
- **IPPO**: Multi-Agent PPO implementation with independent agent and an independent critic for each building, no weights shared across agents.
- **MAPPO**: Multi-Agent PPO implementation each actor receives only its own observation while the critic gets a concatenated observation from all the agents.

5.1 Summer Results

5.1.1 Training phase

During the training phase each model was trained for 400 episodes, each episode is composed of 1440 timesteps, which represent 60 days of data, each model reaching a total timesteps of 576'000.

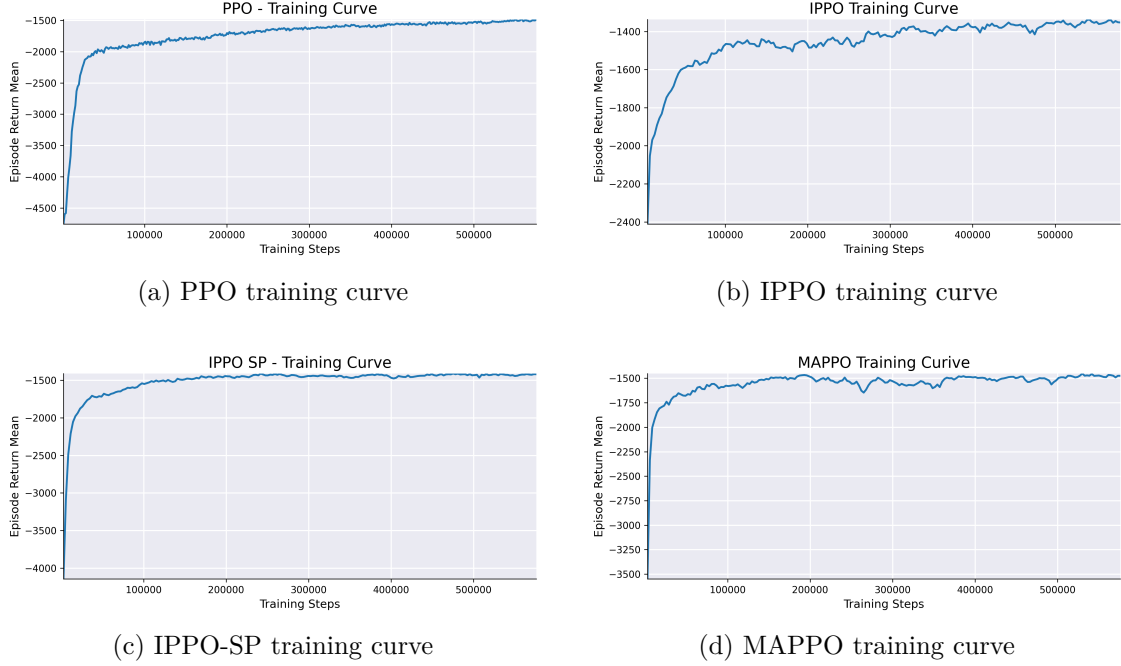
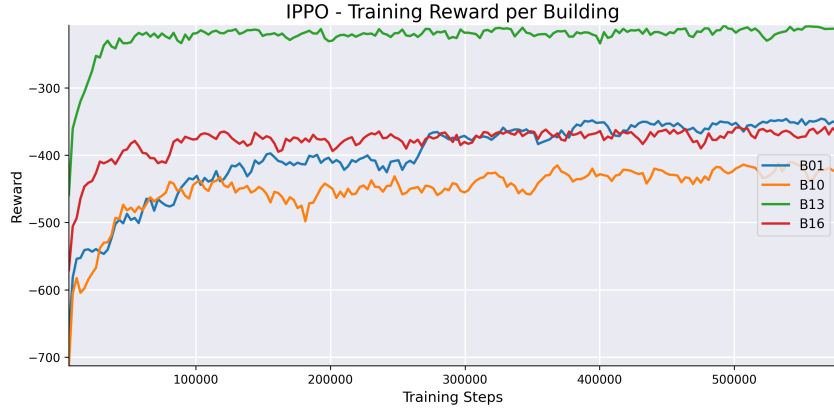
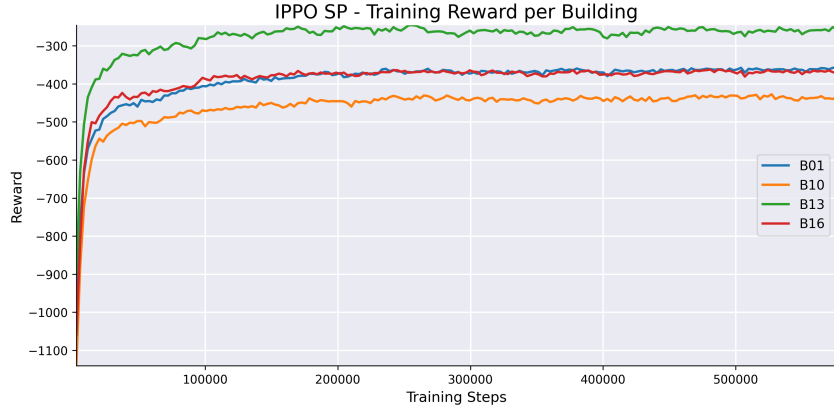


Figure 5.1: Training reward evolution for different agent configuration.

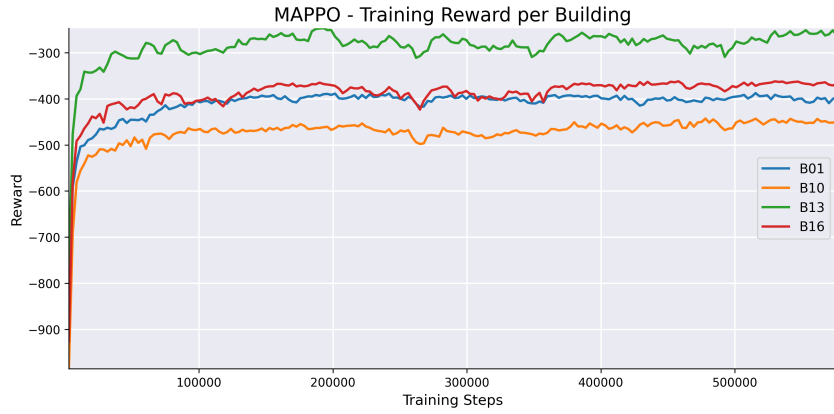
As it can be observed from figure 5.1 all the models reach a plateau. The MAPPO and IPPO-SP are the fastest to reach convergence, followed by PPO, and the IPPO-SP. In the figure 5.2 the single reward per building is presented. The single value differs due to a different load profile. It is important to remember that all the loads are considered not-shiftable loads.



(a) IPPO



(b) IPPO-SP



(c) MAPPO

Figure 5.2: Agent reward for each MA set-up

From figure 5.2 we can see that in each case the one with the major reward return are the B13 while the lowest value is given by B10. It is interesting how B01 and B16

have similar path in the IPPO-SP and MAPPO case while it starts to coincide only after 300 000 timesteps in case of the IPPO. This shows how shared parameters influence the training by fastening the convergence of buildings with similar load profile.

5.1.2 Testing Phase

KPIs The simulation results for the summer for the testing period are presented in this section. The testing period for summer starts at 00:00 AM of 1st of August and ends at 11:00 PM of the 30th of August. This allows to have 30 complete days for testing. The current results are obtained by testing the full trained best model of each set up.

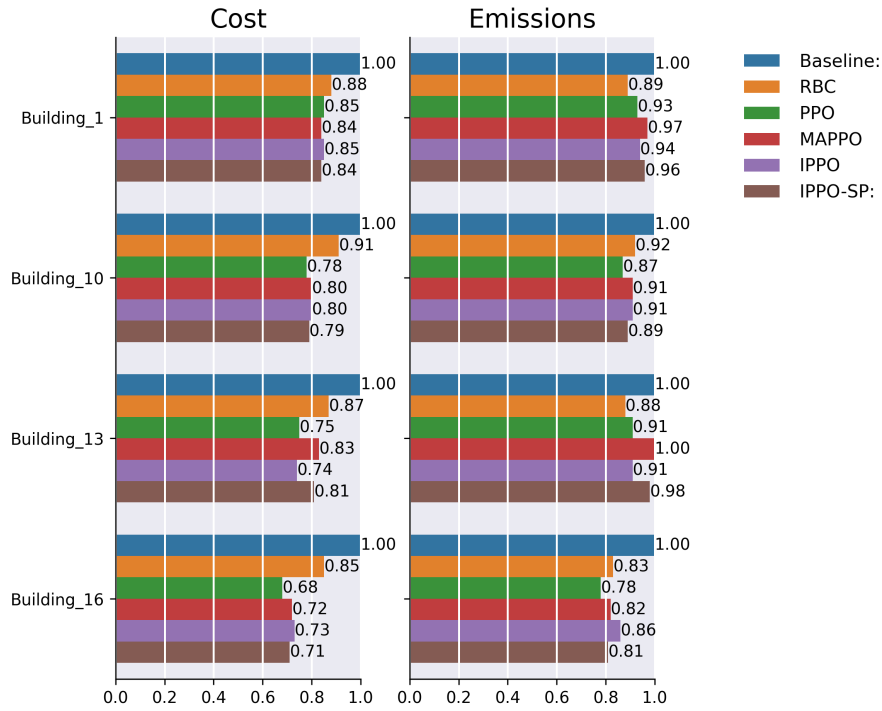


Figure 5.3: Summer - Building KPIs

In the figure 5.3 the KPIs at building level are presented for all the configuration available. Every agent is normalized with respect to the baseline. Lower value in the results implies an improvement in the model.

From a district point of view, the figure 5.4 the results are different and some of them are worsening the situation rather than improving it, in particular for the ramping. With the multi-agent approach the ramping grows lowering the smoothness of the the district net load profile.

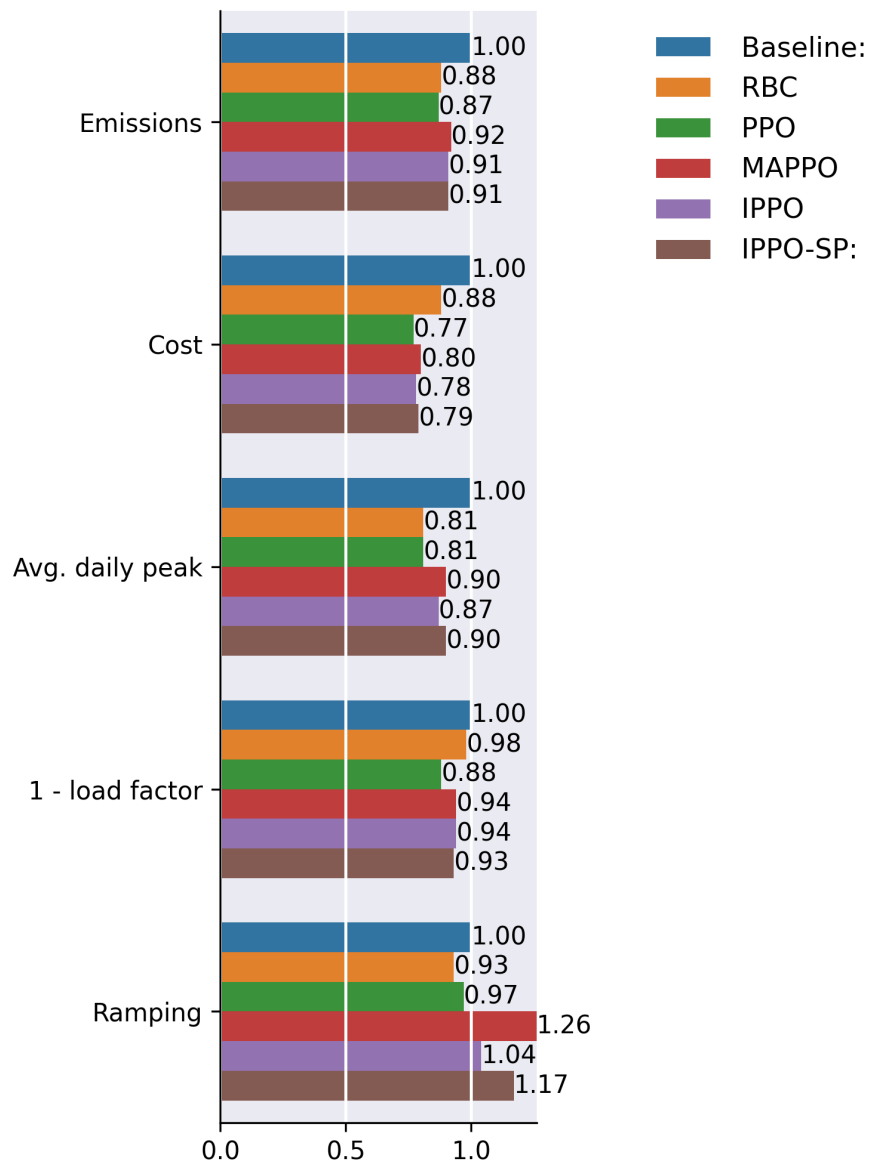


Figure 5.4: Summer - District KPIs

Building Net Load Profile In the following figures the net load profiles are presented. To introduce here the measure

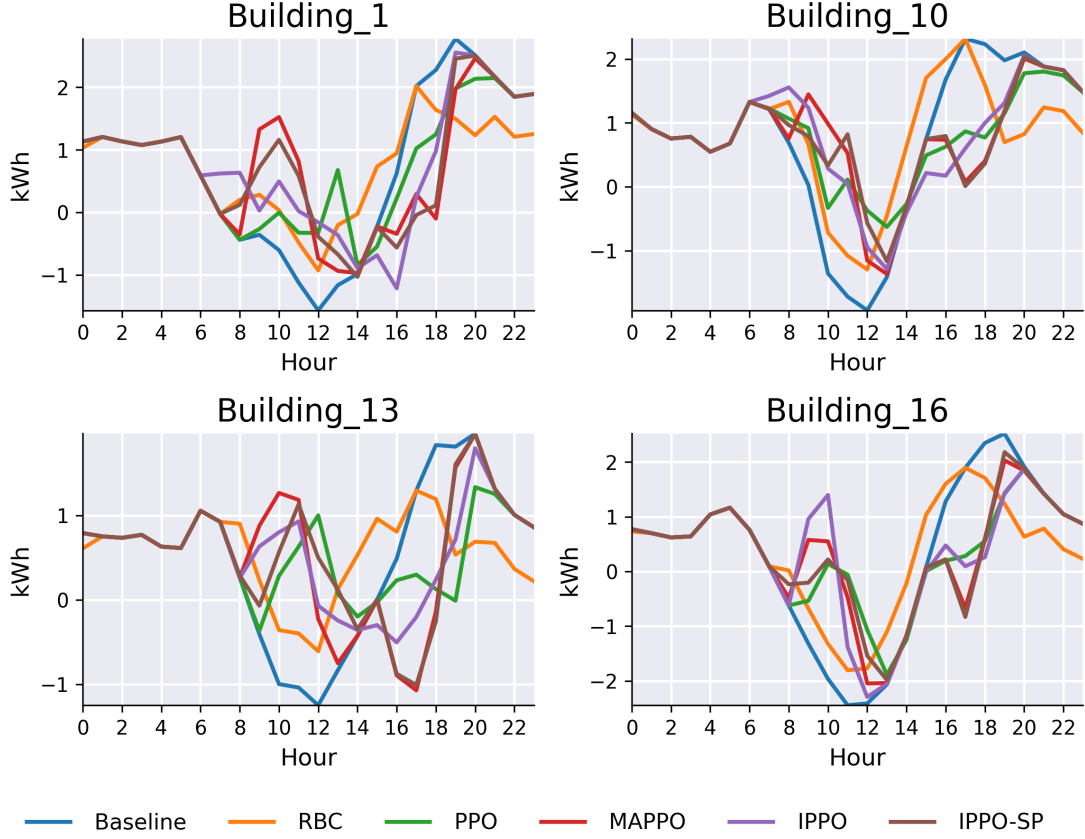


Figure 5.5: Summer - Average Daily Profile per building

To quantify the degree of load profile flattening with respect to the baseline, the standard deviation of the daily net electricity consumption is used as a flatness indicator. Given a daily load profile x_h , where $h = 0, \dots, 23$ denotes the hourly time step, the mean daily load is defined as

$$\mu_x = \frac{1}{24} \sum_{h=0}^{23} x_h. \quad (5.1)$$

The corresponding standard deviation is computed as

$$\sigma_x = \sqrt{\frac{1}{24} \sum_{h=0}^{23} (x_h - \mu_x)^2}. \quad (5.2)$$

A lower value of σ_x indicates a flatter load profile, as it reflects smaller deviations from the daily average consumption. To evaluate the improvement with respect to the baseline profile b_h , the percentage reduction in standard deviation is calculated as

$$\Delta_{\text{flat}}(\%) = 100 \left(1 - \frac{\sigma_x}{\sigma_b} \right), \quad (5.3)$$

where σ_b is the standard deviation of the baseline load profile. Positive values of Δ_{flat} indicate an improvement in load flattening compared to the baseline, while negative values denote increased variability.

Table 5.1: Flatness comparison

Building	Agent	Std_Dev	Improvement_%
Building_1	Baseline	1,26	0,00
Building_1	RBC	0,73	42,39
Building_1	PPO	0,90	28,31
Building_1	MAPPO	1,00	21,13
Building_1	IPPO	1,00	20,79
Building_1	IPPO - SP	0,98	22,57

Building_10	Baseline	1,24	0,00
Building_10	RBC	0,87	29,38
Building_10	PPO	0,67	46,23
Building_10	MAPPO	0,82	34,15
Building_10	IPPO	0,83	32,64
Building_10	IPPO - SP	0,74	40,39

Building_13	Baseline	0,90	0,00
Building_13	RBC	0,47	47,57
Building_13	PPO	0,46	49,55
Building_13	MAPPO	0,77	14,38
Building_13	IPPO	0,57	37,35
Building_13	IPPO - SP	0,70	22,42

Building_16	Baseline	1,46	0,00
Building_16	RBC	1,01	30,30
Building_16	PPO	0,89	39,00
Building_16	MAPPO	1,02	29,57
Building_16	IPPO	1,08	25,84
Building_16	IPPO - SP	0,98	32,42

In table 5.1 a confrontation on a building level based on the previous equations is done. Some remarks can be done:

- **Building 1** The best result is achieved by the RBC, followed by the single-agent PPO. The multi-agent configurations exhibit very similar performance among themselves, but remain clearly below the RBC.
- **Building 10** The best result is achieved by the single-agent PPO, followed by IPPO-SP, MAPPO, and IPPO, while the RBC ranks last.
- **Building 13** A different trend emerges in this case. The single-agent PPO achieves the best result, followed closely by the RBC, with a difference of approximately two percentage points.
- **Building 16** The single-agent PPO again achieves the best result, followed by IPPO-SP. The RBC performs marginally better than MAPPO, while IPPO ranks

last.

From a building perspective, PPO shows the most consistent and robust flattening performance. RBC can achieve competitive results but it lacks general consistency. Multi-agent approaches demonstrate stable but lower improvements. This suggests that their coordination may not translate into clear advantages when evaluated solely on local load variability metrics. The results indicate that building characteristics play a significant role to determine the achievable degree of load flattening.

District Net Load Profiles In this paragraph the attention is moved to the district load profile, shown in figure 5.6

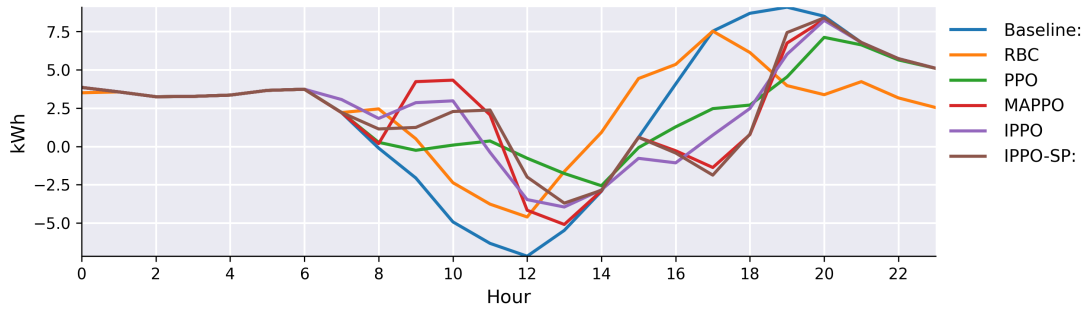


Figure 5.6: Summer - District Load Profile

In table 5.2 a comparison in the standard deviation like the one done in the previous paragraph is shown. From the results the best results with respect to the baseline is achieved by single agent PPO, the RBC approach follows. The last three places are IPPO-SP, IPPO and MAPPO respectively.

Table 5.2: District Flatness

Agent	Std_Dev	Improvement_%
Baseline	4,73	0,00
RBC	2,89	38,94
PPO	2,52	46,77
MAPPO	3,34	29,56
IPPO	3,10	34,61
IPPO-SP	3,08	35,00

Building Daily SoC Profile In this paragraph the 24h SoC profile is shown. The y-axis represent the percentage charge of the battery, a value of 1 means the battery is fully charged. All the agents had similar profiles with respect to the RBC, except for PPO in in the chart of building 1. The state of charge, on average, never goes above 80%. For the other buildings, on average the PPO shows the lowest value during mid day when the production is maximum. All of them show a steep curve for charging and for discharging.

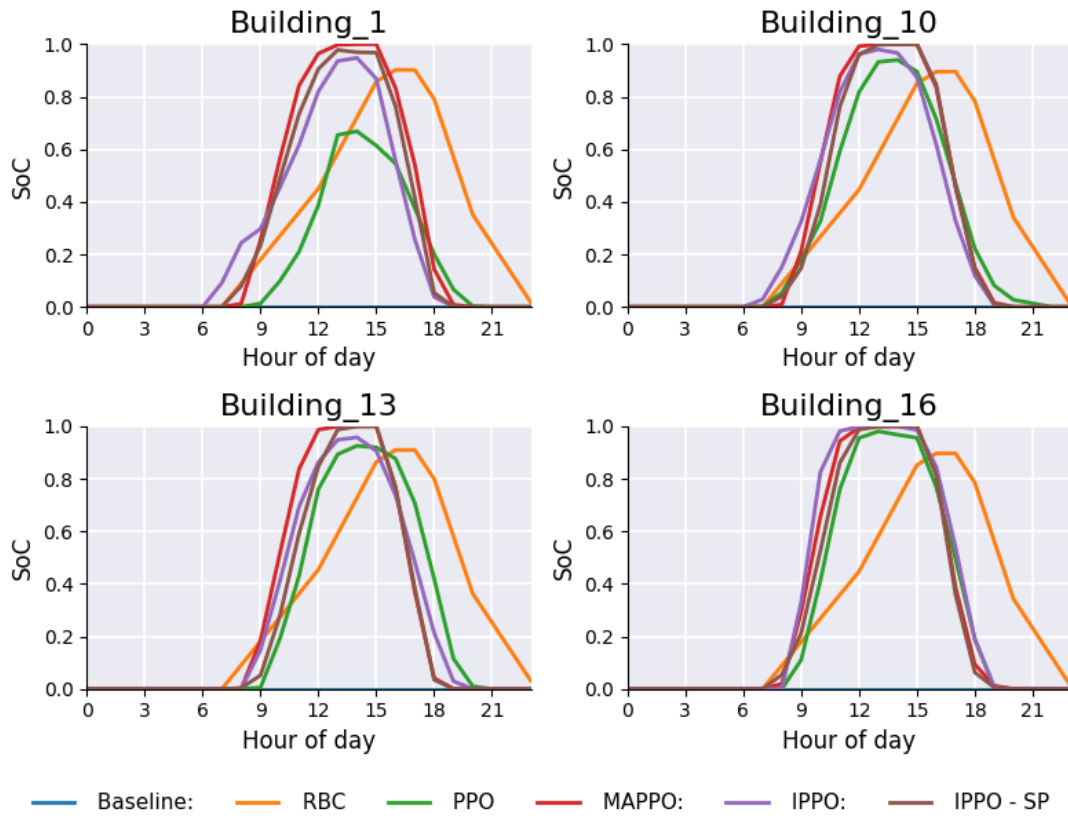
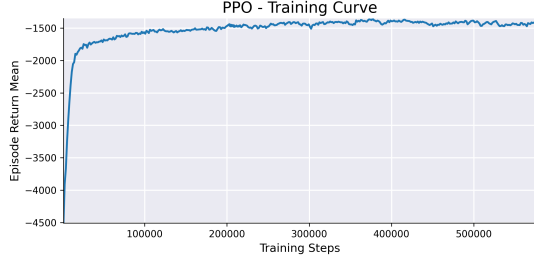


Figure 5.7: Summer - 24h Average Daily SoC Profile

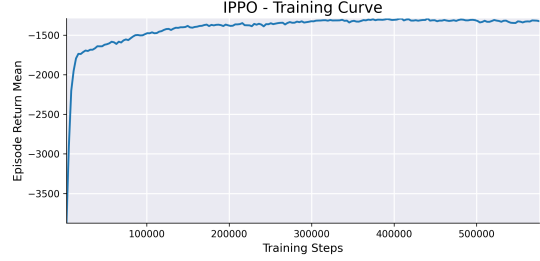
5.2 Winter Results

5.2.1 Training Phase

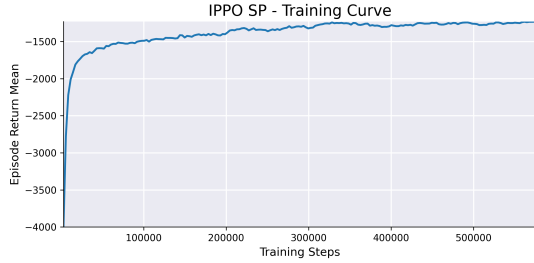
In the following images, as done for the summer phase, the convergence curves are shown.



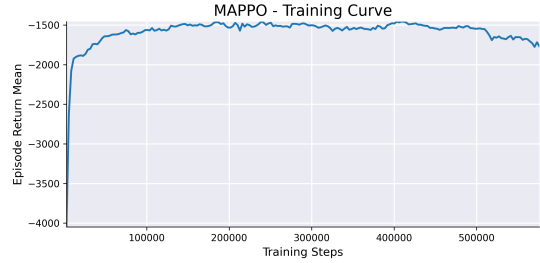
(a) PPO training curve



(b) IPPO training curve



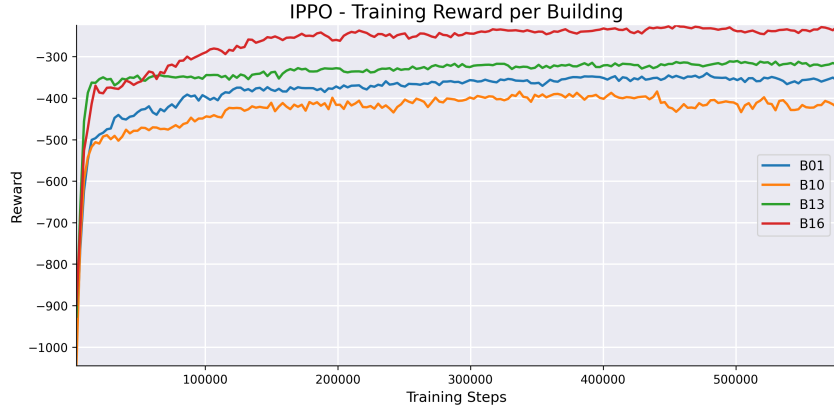
(c) IPPO-SP training curve



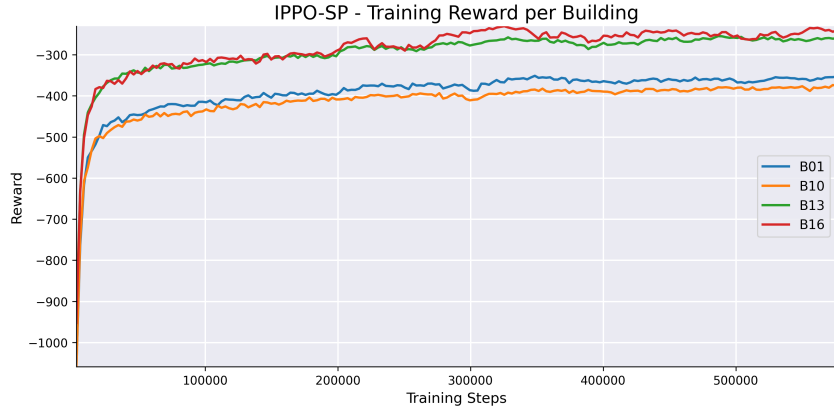
(d) MAPPO training curve

Figure 5.8: Training reward evolution for different agent configuration.

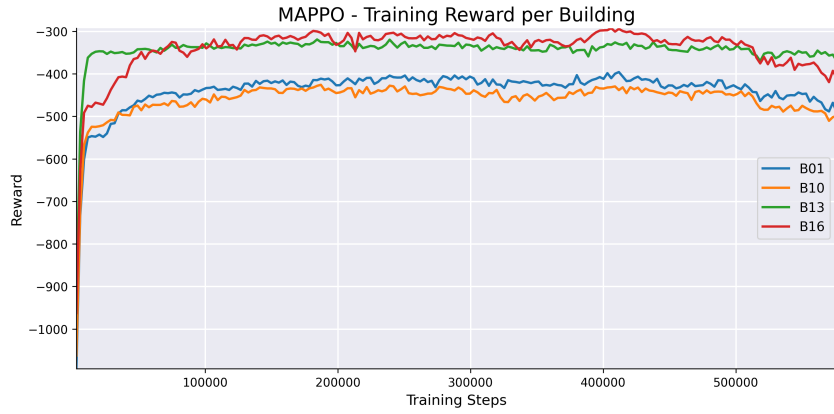
As before, all the models reach a plateau. In this case, a decrease in episode reward is observed in the final training phase for MAPPO. This may indicate training instability or policy oscillation. Such behavior is common in on-policy reinforcement learning methods and does not necessarily imply overfitting. In the following images the reward per building during training is plotted.



(a) IPPO



(b) IPPO-SP



(c) MAPPO

Figure 5.9: Agent reward for each MA set-up

The distance among the rewards value is different than in the IPPO configuration.

The IPPO-SP and the MAPPO show a coupled reward. This can be explained by the concatenated observations received by the MAPPO critic and the shared parameters of the IPPO-SP. For the IPPO configuration the Building 16 shows a grow in the reward until 150 000 timesteps while the other buildings stabilizes quicker and fluctuate around a best value. In all three set-ups the minor contributor in absolute terms to the reward function is the building 16, followed by building 14, building 01 and eventually building 10.

5.2.2 Testing phase

Keeping the same structure presented in subsection 5.1.2, the key results are presented.

KPIs In figure 5.10 the buildings KPIs are presented. In this case we have a different situation with respect to summer. For the cost IPPO-SP showed the best achievements for all buildings followed by the single agent PPO. The IPPO has the worst performances among reinforcement learning agents for all buildings except 16 where the worst was MAPPO.

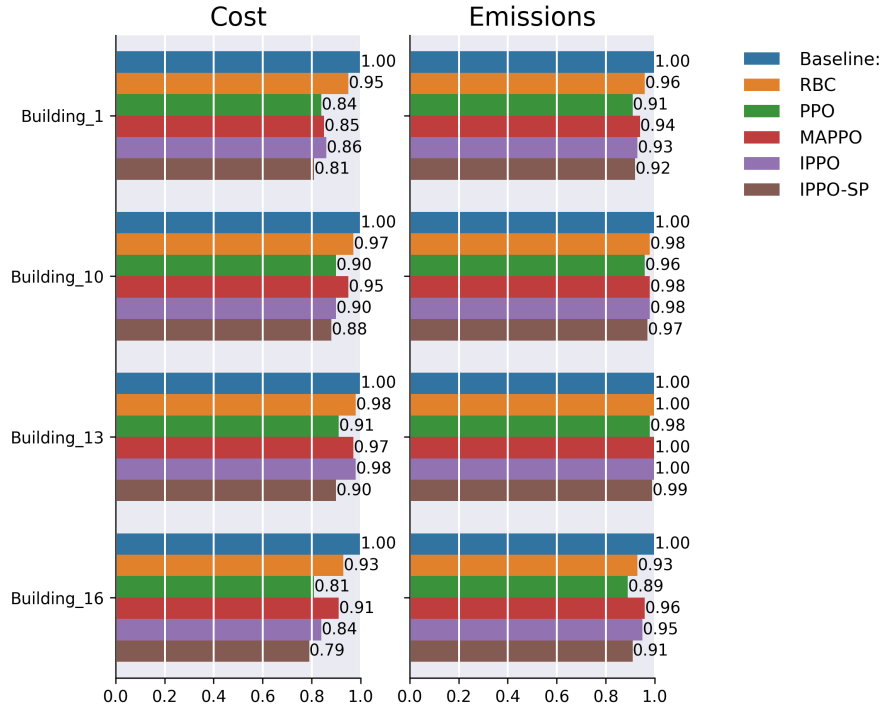


Figure 5.10: Winter - Building KPIs

The 5.11 shows the results from a district point of view. In terms of emissions the best result is achieved by PPO, followed by IPPO-SP, IPPO and MAPPO that has the same score as RBC. Looking at Cost the best value is achieved by IPPO-SP as expected from the building KPIs. The PPO takes the second place followed by IPPO and MAPPO.

In terms of average daily peak the PPO, IPPO and IPPO-SP achieve the same value while MAPPO still has the worst performance among the reinforcement learning agents. Moving on to the load factor all the agents achieved better performances with respect to the baseline improving, even a bit. For the ramping, PPO is the only agent that improves the situation with all the others worsening it.

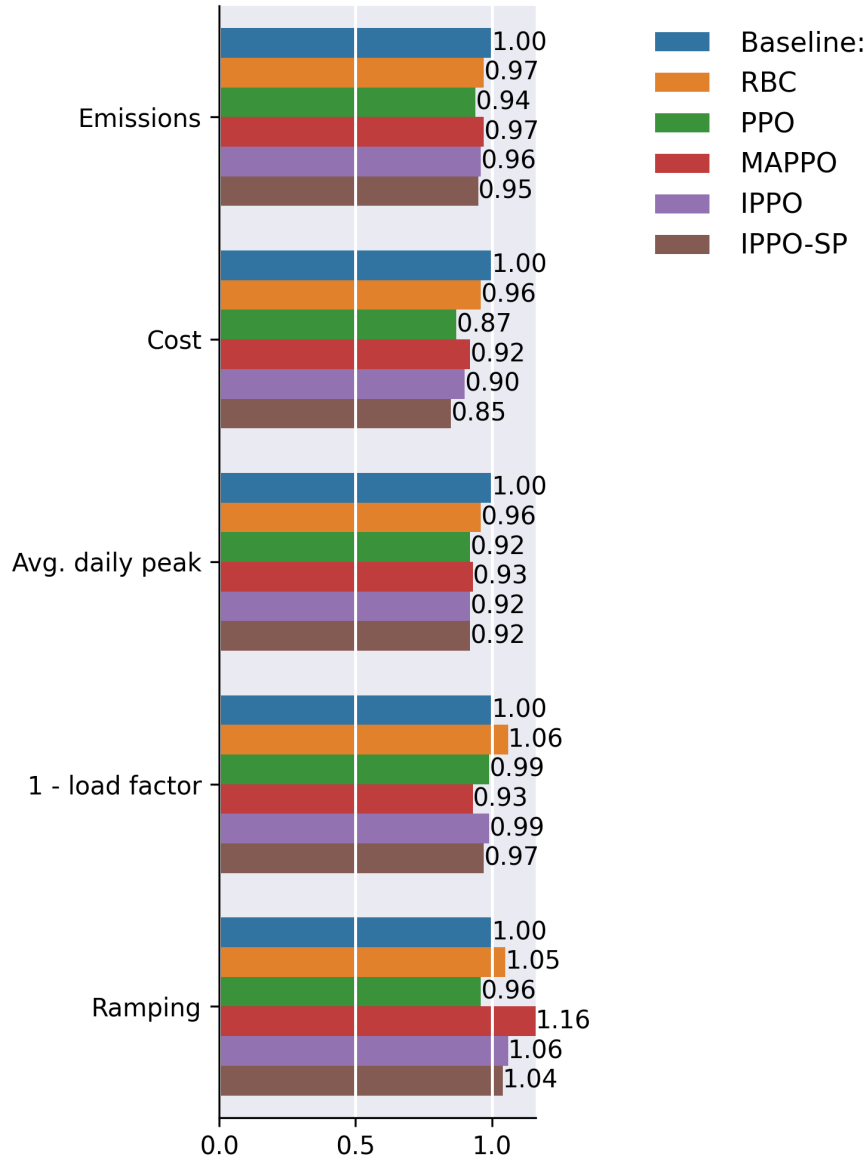


Figure 5.11: Winter - District KPIs

Building Net Load Profile In the figure 5.12 the building net load profile is presented.

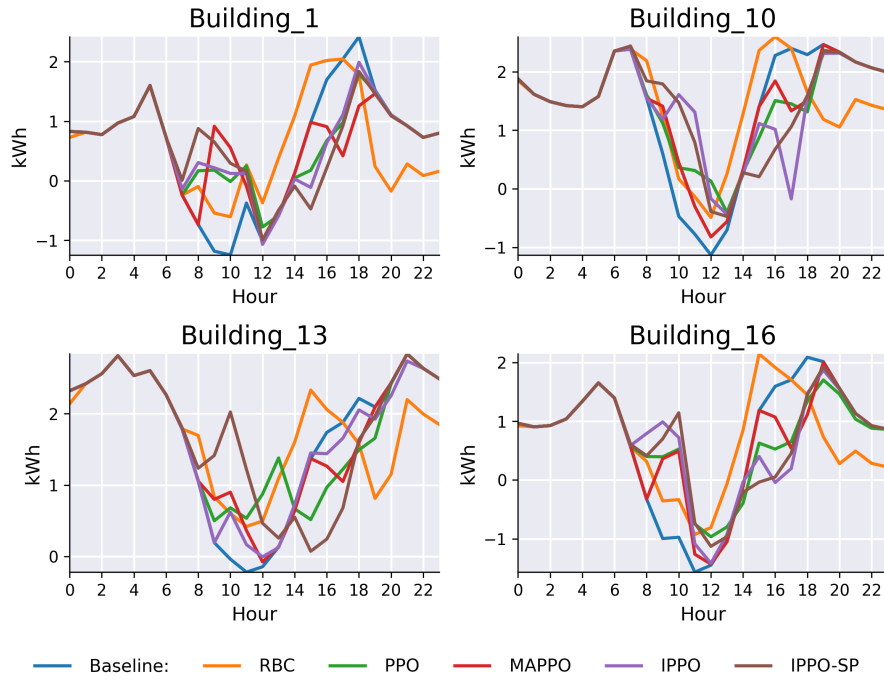


Figure 5.12: Winter - Average Daily profile per building

As done in the previous section, the flatness of each profile is measured. The table 5.3 shows the results:

Table 5.3: Winter- Flatness comparison

Building	Agent	Std_Dev	Improvement_ %
Building_1	Baseline	0,99	0,00
Building_1	RBC	0,80	19,65
Building_1	PPO	0,63	36,24
Building_1	MAPPO	0,66	33,21
Building_1	IPPO	0,68	31,31
Building_1	IPPO-SP	0,66	33,50

Building_10	Baseline	1,10	0,00
Building_10	RBC	0,80	27,68
Building_10	PPO	0,77	30,14
Building_10	MAPPO	0,90	17,94
Building_10	IPPO	0,80	27,44
Building_10	IPPO-SP	0,82	25,19

Building_13	Baseline	1,02	0,00
Building_13	RBC	0,71	30,87
Building_13	PPO	0,82	19,17
Building_13	MAPPO	0,90	12,11
Building_13	IPPO	0,92	9,55
Building_13	IPPO-SP	0,90	11,70

Building_16	Baseline	1,10	0,00
Building_16	RBC	0,81	26,24
Building_16	PPO	0,73	33,30
Building_16	MAPPO	0,88	19,58
Building_16	IPPO	0,84	23,53
Building_16	IPPO-SP	0,80	26,81

A building-by-building analysis reveals the following observations:

- **Building 1** The best result is achieved by PPO, followed by IPPO-SP, MAPPO, and IPPO. All learning-based agents outperform the RBC strategy in terms of load flattening.
- **Building 10** Also in this case, the best result is achieved by the single-agent PPO. The second-best performance is obtained by RBC, followed by IPPO, IPPO-SP, and finally MAPPO, which shows the lowest flattening improvement for this building.
- **Building 13** For this building, RBC clearly achieves the best performance. PPO ranks second, followed by MAPPO and IPPO-SP. IPPO shows the lowest flattening improvement within this building when compared to the baseline profile.
- **Building 16** PPO achieves the best result, followed by IPPO-SP. MAPPO and IPPO both perform worse than RBC in this case.

Overall, all control strategies lead to a reduction in load variability compared to the baseline profile across all buildings. However, the magnitude of improvement varies significantly depending on the building and the control architecture.

The single-agent PPO consistently demonstrates strong and stable performance, achieving the highest or near-highest flattening improvements in most cases. The rule-based controller (RBC) performs competitively in certain buildings, particularly Building 13,

where it outperforms all learning-based approaches.

The multi-agent methods (MAPPO, IPPO, and IPPO-SP) show building-dependent behavior, with performance levels generally comparable to the single-agent approach but exhibiting greater variability across buildings. These results suggest that the effectiveness of the control strategy is strongly influenced by building-specific characteristics rather than by the algorithmic class alone.

District Net Load Profiles In this paragraph the attention is toward the district load profile. It can be seen in figure 5.13.

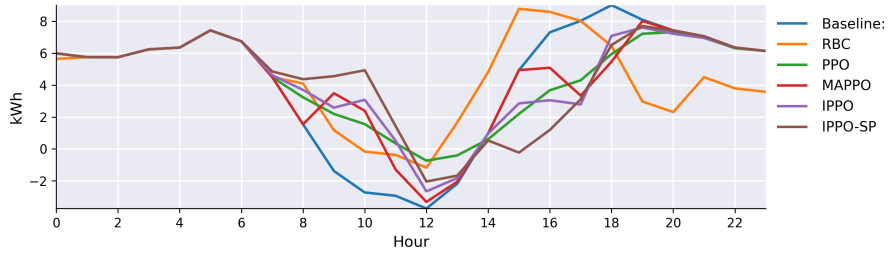


Figure 5.13: Winter - District Load Profile

In table 5.4 the comparison between the different profiles shows the best result is achieved by single-agent PPO. RBC follows while the multi-agent approaches are placed in the last places: IPPO, IPPO-SP, and MAPPO, respectively.

Table 5.4: District Flatness comparison

Agent	Std_Dev	Improvement_%
Baseline	3,95	0,00
RBC	2,75	30,46
PPO	2,58	34,65
MAPPO	3,06	22,45
IPPO	2,85	27,92
IPPO-SP	2,90	26,52

Building Daily SoC Profile In the figure 5.14 the average state of charge profiles are presented for each building.

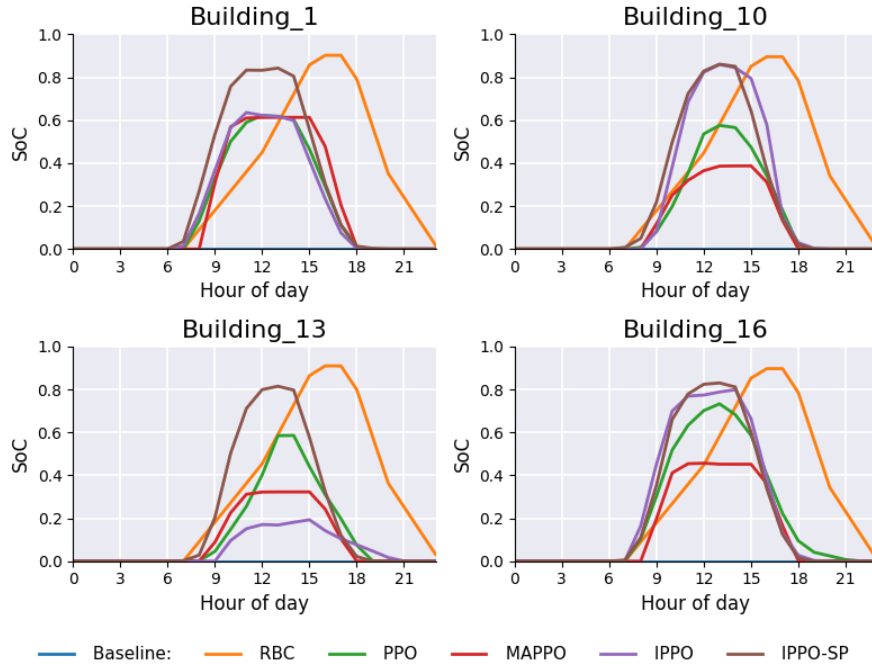


Figure 5.14: Winter - 24h Average Daily SoC profile

All the profiles of the reinforcement learning methods have the maximum value below the RBC. The IPPO-SP shows the biggest values in the charge for all four buildings. The 100% charge is never reached. For building 1 the PPO, MAPPO and IPPO-SP shows similar profiles. For building 10 IPPO and IPPO-SP show similar profiles with similar maximum state of charge. For building 13 all the profiles are different with the IPPO showing a maximum average charge of 20% but a flattened consumption. For building 16 the IPPO and the IPPO-SP shows similar profiles reaching on average 80% of the battery capacity. MAPPO shows a flat profile on average for mid day, with a value below 50% of capacity.

Chapter 6

Conclusion

This thesis addressed two complementary objectives in the field of reinforcement learning applied to building energy management: the development of a taxonomy for classifying multi-agent reinforcement learning (MARL) control strategies, and an experimental comparative study of single-agent and multi-agent PPO-based algorithms within the CityLearn simulation environment.

6.1 Taxonomy

The first contribution of this work is the proposal of a structured taxonomy for reinforcement learning control strategies in building energy management systems. Based on a systematic review of the existing literature and drawing from established MARL classifications, the taxonomy provides a coherent framework to identify, describe, and compare algorithms along key dimensions such as the degree of agent cooperation, the training and execution paradigm (centralized vs. decentralized), and the information sharing scheme. This classification serves as a reference tool to guide future algorithm selection and development, making it easier for researchers and practitioners to position new approaches within the broader landscape of MARL methods.

6.2 Experimental Evaluation

The second contribution is a systematic experimental evaluation comparing four control strategies: a single-agent Proximal Policy Optimization (PPO) controller, and three multi-agent extensions, Independent PPO (IPPO), IPPO with shared policy parameters (IPPO-SP), and Multi-Agent PPO (MAPPO). All methods were benchmark against a Rule-Based Controller (RBC) and a no-storage, no-PV baseline across four residential buildings in the CityLearn environment, evaluated separately over summer and winter periods.

The results consistently show that all reinforcement learning strategies outperform the no-storage baseline in terms of electricity cost and carbon emissions. However, the relative performance across RL methods varies notably depending on the season and the specific KPI considered.

In the **summer** period, the single-agent PPO demonstrated the strongest overall performance. At the district level, PPO achieved the highest reduction in load variability, improving the flatness of the net load profile by approximately 46.8% over the baseline, compared to 39.0% for RBC and approximately 29.6-35.0% for the multi-agent methods. MAPPO consistently ranked last among RL approaches in this season, while IPPO-SP and IPPO showed intermediate performance.

In the **winter** period, the ranking shifted. IPPO-SP emerged as the best-performing algorithm in terms of electricity cost reduction at the building level, outperforming all other approaches across all four buildings. At the district level, PPO still achieved the highest load profile flattening (34.7%) and was the only algorithm to improve the ramping metric, while MAPPO again posted the weakest performance among RL agents. Notably, MAPPO exhibited a decrease in episode reward during the final phase of winter training, suggesting policy oscillation or training instability, a behavior that is not uncommon in on-policy methods but that warrants attention in future developments.

Across both seasons, the impact of building-specific characteristics on performance emerged as a consistent finding. The variability in results across buildings, particularly in multi-agent configurations, indicates that the choice of control strategy cannot be decoupled from the load profile and structural properties of individual buildings. The single-agent PPO proved to be the most robust and generalizable approach, consistently achieving strong performance at the district level. The IPPO-SP demonstrated that parameter sharing across agents can accelerate convergence and improve cost-related outcomes, particularly in winter. MAPPO, despite its theoretical advantages in enabling explicit cooperation through centralized training, did not translate this into consistent gains in the evaluated scenarios, possibly due to the added complexity of coordinating a shared critic across heterogeneous buildings.

6.3 Future Work

Several directions can extend the work presented in this thesis. First, the evaluation could be broadened to include additional seasons, longer testing windows, and a larger number of buildings to improve the statistical robustness of the conclusions. Second, more sophisticated reward shaping strategies or multi-objective formulations could be explored to better balance cost, emissions, and grid-level metrics simultaneously. Third, incorporating load forecasting as an additional input to the agents, as explored in related literature, could further improve the quality of battery scheduling decisions. Finally, investigating the scalability of MARL approaches to larger districts with heterogeneous building portfolios and more complex grid topologies represents a natural and practically relevant extension of this work.

Bibliography

- [1] Simon Wenninger, Can Kaymakci, and Christian Wiethe. Explainable long-term building energy consumption prediction using QLattice. *Applied Energy*, 308:118300, 2022.
- [2] Bin Zhang, Weihao Hu, Amer M. Y. M. Ghias, Xiao Xu, and Zhe Chen. Multi-agent deep reinforcement learning-based coordination control for grid-aware multi-buildings. *Applied Energy*, 338:120215, 2023.
- [3] Hans Christian Gils. Assessment of the theoretical demand response potential in europe. *Energy*, 67:1–18, 2014.
- [4] Anjukan Kathirgamanathan, Mattia De Rosa, Eleni Mangina, and Donal P. Finn. Data-driven predictive control for unlocking building energy flexibility: A review. *Renewable and Sustainable Energy Reviews*, 155:111843, 2022.
- [5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.
- [6] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.
- [7] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021.
- [8] Kai Hu, Mingyang Li, Zhiqiang Song, Keer Xu, Qingfeng Xia, Ning Sun, Peng Zhou, and Min Xia. A review of research on reinforcement learning algorithms for multi-agents. *Neurocomputing*, 599:128068, 2024.
- [9] Kingsley Nweye, Kathryn Kaspar, Giacomo Buscemi, Tiago Fonseca, Giuseppe Pinto, Dipanjan Ghose, Satvik Duddukuru, Pavani Pratapa, Han Li, Javad Mohammadi, Luis Lino Ferreira, Tianzhen Hong, Mohamed Ouf, Alfonso Capozzoli, and Zoltan Nagy. Citylearn v2: energy-flexible, resilient, occupant-centric, and carbon-aware management of grid-interactive communities. *Journal of Building Performance Simulation*, pages 1–22, 2024.
- [10] Chao Yu, Akash Velu, Eugene Vinitzky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.
- [11] Hepeng Li and Haibo He. Multi-agent trust region policy optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 31(12):5138–5150, 2020.
- [12] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun

- Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2023.
- [13] Allen Wu, Kingsley Nweye, and Zoltan Nagy. Heterogeneous multi-agent reinforcement learning for grid-interactive communities. In *BuildSys '23: Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. ACM, 2023.
- [14] Zixuan Wang, Fu Xiao, Yi Ran, Yanxue Li, and Yang Xu. Scalable energy management approach of residential hybrid energy system using multi-agent deep reinforcement learning. *Applied Energy*, 341:121033, 2023.
- [15] Chao Yu, Akash Velu, Eugene Vinitzky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, volume 35, 2022.
- [16] Cheng Yang, Jihai Zhang, Fangquan Lin, Li Wang, Wei Jiang, and Hanwei Zhang. Combining forecasting and multi-agent reinforcement learning techniques on power grid scheduling task. In *2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, pages 1576–1580, 2023.
- [17] OpenAI. Spinning up in deep reinforcement learning: An introduction to reinforcement learning. https://spinningup.openai.com/en/latest/spinningup/rl_intro.html, 2018.
- [18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2 edition, 2018.
- [19] Richard E. Bellman and Stuart E. Dreyfus. *Applied Dynamic Programming*, volume 2050. Princeton University Press, 2015.
- [20] Christopher Watkins. Learning from delayed rewards. 01 1989.
- [21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [24] Fei-Fei Li, Justin Johnson, Ehsan Adeli, and Zane Durante. Cs231n: Convolutional neural networks for visual recognition. <http://cs231n.github.io/>, 2024. Stanford University.
- [25] Dumitru Erhan, Yere Yere, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 01 2009.
- [26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

- [27] David Silver, Guy Lever, Nicolas Manfred Otto Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.
- [28] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [30] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, NY, USA, 2006.
- [31] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.
- [32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.
- [34] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.
- [35] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.
- [36] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. Neural Networks for Machine Learning, Coursera, 2012. Course lecture.
- [37] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.
- [38] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 12, pages 1008–1014. MIT Press, 2000.
- [39] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.
- [40] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.
- [41] OpenAI. Proximal policy optimization. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>, 2018. Accessed: 2026-01-27.
- [42] Hongming Zhang and Tianyang Yu. Taxonomy of reinforcement learning algorithms. In *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 125–133. Springer, 2020.
- [43] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.
- [44] MARL Book Authors. Slides for Multi-Agent Reinforcement Learning: Foundations and modern approaches. <https://github.com/marl-book/slides>, 2020. GitHub repository, accessed 2026-01-27.

-
- [45] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100, 1953.
 - [46] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Computer Science. Springer, 2016.
 - [47] Christopher Amato. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning, 2024. Manuscript / lecture notes, Northeastern University, December 22, 2024.
 - [48] Christopher Amato. An initial introduction to cooperative multi-agent reinforcement learning, 2025.
 - [49] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviy-chuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? In *International Conference on Learning Representations (ICLR)*, 2021.
 - [50] Kingsley Nweye, Allen Wu, and Zoltan Nagy. Merlin: Multi-agent offline and transfer learning for occupant-centric energy flexible operation of grid-interactive communities using smart meter data and citylearn, 2023.
 - [51] Chia E. Tungom, Hong Wang, Kamuya Beata, and Ben Niu. Swoam: Swarm optimized agents for energy management in grid-interactive connected buildings. *Applied Energy*, 346:121366, 2023.
 - [52] Chia E. Tungom, Ben Niu, and Hong Wang. Swapp: Swarm precision policy optimization with dynamic action bound adjustment for energy management in smart cities. *Energy and Buildings*, 310:114125, 2024.
 - [53] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
 - [54] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 2623–2631. ACM, 2019.
 - [55] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 561–577, 2018.
 - [56] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018.
 - [57] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning, 2020.
 - [58] Climate Change AI Tutorials. Citylearn tutorial notebook. <https://citylearn-tutorial.github.io/>

[//colab.research.google.com/github/climatechange-ai-tutorials/citylearn/blob/main/citylearn_ccai_tutorial.ipynb](https://colab.research.google.com/github/climatechange-ai-tutorials/citylearn/blob/main/citylearn_ccai_tutorial.ipynb), 2023. Google Colab notebook, accessed 2026-01-27.