A Novel Safe Deep Reinforcement Learning Approach for Optimal Dispatch of Energy Hubs with Compressed Air Energy Storage

Ali Reza Daneshvar Garmroodi

A Thesis

in

The Department

of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Building, Civil, and Environmental Engineering) at

Concordia University

Montréal, Québec, Canada

March 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Ali Reza Daneshvar Garmroodi**

Entitled: **A Novel Safe Deep Reinforcement Learning Approach for Optimal Dispatch of Energy Hubs with Compressed Air Energy Storage**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Building, Civil, and Environmental Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Ursula Eicker

_____Examiner
Dr. Ursula Eicker

_____ Examiner
Dr. Mohamed Ouf

_____ Co-supervisor
Dr. Fariborz Haghighat

_____ Co-supervisor
Dr. Fuzhan Nasiri

Approved by _____
Dr. Mazdak Nik-Bakht, Graduate Program Director

March 27,2023

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

A Novel Safe Deep Reinforcement Learning Approach for Optimal Dispatch of Energy Hubs with Compressed Air Energy Storage

Ali Reza Daneshvar Garmroodi

The development of renewable energy and energy storage technologies has resulted in the emergence of Energy Hubs (EHs) in recent years. Due to the uncertainty associated with energy supply and load, scheduling EH presents a challenging task. Current model-based optimization approaches have limitations in terms of solution accuracy and computational efficiency, which hamper their application. Deep Reinforcement Learning (DRL) is a model-free approach that has demonstrated superior performance over model-based approaches. The current DRL algorithms, however, perform poorly in terms of constraint handling and global optimality. The purpose of this study is to propose a model-free, safe deep reinforcement learning approach, combining primal-dual optimization and imitation learning, for the optimal scheduling of an EH with an Advanced Adiabatic Compressed Air Energy Storage (AA-CAES). First, the operation of an AA-CAES under off-design conditions is modeled and linearized using Mixed Integer Linear Programming (MILP). Then, a safe DRL approach is proposed with training and testing steps considering a case study. The performance of the proposed approach in reducing operational cost and satisfying constraints is compared to state-of-the-art DRL algorithms as well as a deterministic MILP approach. Additionally, a test set is used to examine the generalizability of the proposed approach. Finally, the effect of off-design conditions of a tri-generative AA-CAES on the optimal dispatch strategy is investigated. Furthermore, a sensitivity analysis indicates that the proposed approach is reproducible and reliable. The results indicate that the proposed approach can effectively reduce the operational cost and satisfy the operational constraints.

Acknowledgements

Initially, I would like to express my sincere gratitude and recognize the valuable contributions of my supervisors, Prof. Fariborz Haghighat and Prof. Fuzhan Nasiri, whose unwavering support and guidance have been instrumental in enabling the completion of this thesis. I am indebted to them for providing me with exceptional mentorship, comprehensive knowledge, and insightful advice from the inception of this academic endeavor.

Furthermore, apart from my supervisors, I wish to express my appreciation to my parents, whom I profoundly miss, for their boundless love and encouragement. Finally, I extend my thanks to my brother, friends, and all individuals who have supported and accompanied me during my academic journey.

This work was financially supported by the India-Canada Centre for Innovative Multidis- ciplinary Partnerships to Accelerate Community Transformation and Sustainability (IC-IMPACTS), dedicated to the development of scientific collaborations between Canada and India.

Contents

List of Figures

List of Tables

List of Abbrevations

AC	Absorption Chiller
A-CAES	Adiabatic Compressed Air Energy Storage
AA-CAES	Advanced Adiabatic Compressed Air Energy Storage
ANN	Artificial Neural Network
BC	Behavioral Cloning
CAES	Compressed Air Energy Storage
COP	Coefficient of Performance
CCHP	Combined Cooling, Heating, and Power
CMDP	Constrained Markov Decision Process
CVaR	Constraint Value at Risk
CWT	Cold-Water Tank
DDPG	Deep Deterministic Policy Gradient
DDPGfD	Deep Deterministic Policy Gradient from Demonstrations
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
D-CAES	Diabatic Compressed Air Energy Storage
EHP	Electric Heat Pump
EH	Energy Hub
EMESS	Electrical Mechanical Energy Storage System
ESS	Energy Storage System
FESS	Flywheel Energy Storage System
GA	Genetic Algorithm
GB	Gas Boiler
GHG	Greenhouse Gas
HE	Heat Exchanger
IL	Imitation Learning
IEMS	Intelligent Energy Management System
LP	Linear Programming
MC	Monte Carlo
MDP	Markov Decision Process
MESS	Mechanical Energy Storage System
MILP	Mixed Integer Linear Programming

MORL	Multi-Objective Reinforcement Learning
MSE	Mean Squared Error
MT	Micro Turbine
PV	Photovoltaic
PD-DDPG	Primal-Dual Deep Deterministic Policy Gradient
PD-DDPGfD	Primal-Dual Deep Deterministic Policy Gradient from Demonstrations
PDF	Probability Distribution Function
PHES	Pumped Hydro Energy Storage
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
RIES	Regional Integrated Energy System
RL	Reinforcement Learning
RES	Renewable Energy Sources
RO	Robust Optimization
SDRL	Safe Deep Reinforcement Learning
SOC	State of Charge
SP	Stochastic Programming
SVR	Support Vector Regression
TES	Thermal Energy Storage
WT	Wind Turbine

Chapter 1

Introduction

Global warming and Greenhouse Gas (GHG) emissions caused by an unprecedented increase in worldwide energy consumption have led to a growing interest in developing self-sufficient communities and EHs [1]. An EH is comprised of dispatchable thermal and electrical energy sources along with Renewable Energy Sources (RES) [2]. Energy Storage Systems (ESSs) play a crucial role in EHs, providing a buffer capacity that enhances resiliency, flexibility, and reliability of energy supply. Tri-generative AA-CAES is an emerging energy storage with the ability to generate heating, cooling, and electricity, acting as a Combined Cooling, Heating, and Power (CCHP) unit [3]. AA-CAES offers several advantages over other ESSs, including low capital costs, long lifespans, and environmental friendliness [4]. However, it is important to note that since AA-CAES is a Mechanical Energy Storage System (MESS), its design and operation present greater challenges than those of conventional electricity in- electricity out ESS. Particularly, part-load and off-design operating conditions could have a significant effect on AA-CAES performance [5]. As a result, it is crucial to take into account the off-design characteristics of AA-CAES when establishing a dispatch strategy, since ignoring them could result in scheduling and control lags and demand-supply mismatches. Identifying a optimal dispatch strategy for an EH can be a challenging task due to the presence of various uncertainties, such as variations in load demand, volatility in the electricity and gas markets, and intermittent nature of renewable energy sources [6]. There are two methods that are commonly used to deal with these uncertainties: Stochastic Programming (SP) [7] and Robust Optimization (RO) [8]. However, these approaches have limitations in terms of computational cost

(intensity of computational requirements) and optimality (tendency towards conservative solutions) [9].

1.1 Problem Statement

Currently, the approaches in order to determine the optimal dispatch of EHs with AA-CAES under uncertainties rely on a model of uncertainties to deal with them. Reinforcement Learning (RL), however, is a model-free approach that takes uncertainty into account without the use of any models [9]. In fact, RL learns from encountering a variety of scenarios during the course of its learning process in order to understand the behavior and transition of uncertainties. In this regard, RL has demonstrated its superiority as a model-free approach for optimal dispatch of EHs. Nevertheless, since this method is based on trial and error, there are a number of limitations to this solution:

(1) In large search spaces, the RL appraoch may become stuck in local optima and result in a suboptimal solution.

(2) Due to the lack of a mathematical framework in RL, constraint handling remains a significant challenge.

There are important constraints in the operation of an EH that should be considered in the dispatch strategy since neglecting them can lead to catestrophic situations. Constraint handling in RL has drawn attention to itself which has led to the development of a new field of study called Safe Deep Reinforcement Learning (SDRL) [10]. However, even SDRL algorithms can not guarantee the global optimality of the solution and can easily lead to sub-optimal solutions in large scale problems like optimal scheduling of an EH with many energy flows. In this regard, this thesis proposes a framework to tackle the above mentioned challenges in operation of an EH with presence of a tri-generative AA-CAES using a generalizable model-free approach stemmed in SDRL.

1.2 Research Objectives and Contributions

As mentioned above, even though model-free approaches, such as DRL approaches, are perfect for considering uncertainties in optimization problems, some limitations remain in terms of handling constraints and finding the global optimum. As a result, the following are the major contributions of this thesis:

(1) In this thesis, a novel SDRL approach based on primal-dual optimization and Imitation Learning (IL) is proposed. This algorithm considers the operational constraints of the EH by utilizing cost networks. Unlike the traditional DRL approaches that add constraints to the reward function as a penalty term, the proposed algorithm considers the constraints as independent cost functions. To avoid local optima during exploration, expert demonstrations are utilized to help the agent for more efficient learning process.

(2) The off-design operation of a tri-generative AA-CAES in a CCHP energy hub is modelled and linearized. As the partial-load operation of AA-CAES affects the charging and discharging efficiencies of the system, the effect of off-design operation of the trigenerative AA-CAES on the scheduling of EH equipment is studied.

(3) The performance of the proposed safe DRL approach in terms of operational cost, constraint violation, and using the potential of AA-CAES as ESS is compared to other DRL and IL algorithms and the theoretical benchmark for the scheduling problem.

1.3 Thesis Organization

The remainder of the thesis is organized as follows:

- In Section 2, a literature review is presented to review the current approaches to modeling the AA-CAES dispatch problem and solution approaches, and to highlight gaps in the previous studies.
- Section 3 explains the mathematical modeling of the EH dispatch problem as well as the proposed model-free approach as a solution.

- Section 4 presents the case study and the hyperparameters used for the proposed and benchmark algorithms.
- Section 5 presents the results of the study along with a detailed discussion of the findings.
- Section 6 draws conclusions and a discussion on limitations and future avenues of research.

1.4 Publications

The proposed methodology in this thesis is published as a journal paper in *Journal of Energy Storage*:

A. D. Garmroodi, F. Nasiri, and F. Haghighat, "Optimal dispatch of an energy hub with compressed air energy storage: A safe reinforcement learning approach," *Journal of Energy Storage*, vol. 57, p. 106147, 2023

Chapter 2

Background and Literature Review

2.1 Energy Storage Systems

Burning fossil fuels has devastating effects on our environment, which is why the world is looking for alternative sources of energy [12]. The limited sources of their products and the instability of their prices are also important reasons that have led us to this decision. There is no doubt that the energy and climate change crisis are among the major problems facing humanity today. As a result of population growth and industrialization, total energy demand has increased dramatically [13]. It is apparent from a closer examination of global energy consumption from 2000 to 2018 that primary energy consumption is increasing every year (Fig. 2.1). Since fossil fuels such as natural gas, oil, and coal are the most reliable sources to generate energy, they have the most share in energy generation sources. As a result of our high dependence on fossil fuels, we are currently experiencing serious climate change. In order to reduce this dependency, renewable energy is considered to be the most pragmatic approach [14]. This is due to the fact that these sources of energy are both abundant and environmentally friendly. Therefore, the amount of electricity generated from renewable energy sources is increasing rapidly every year, which has led to a decrease in the amount of energy generated from burning fossil fuels [12].

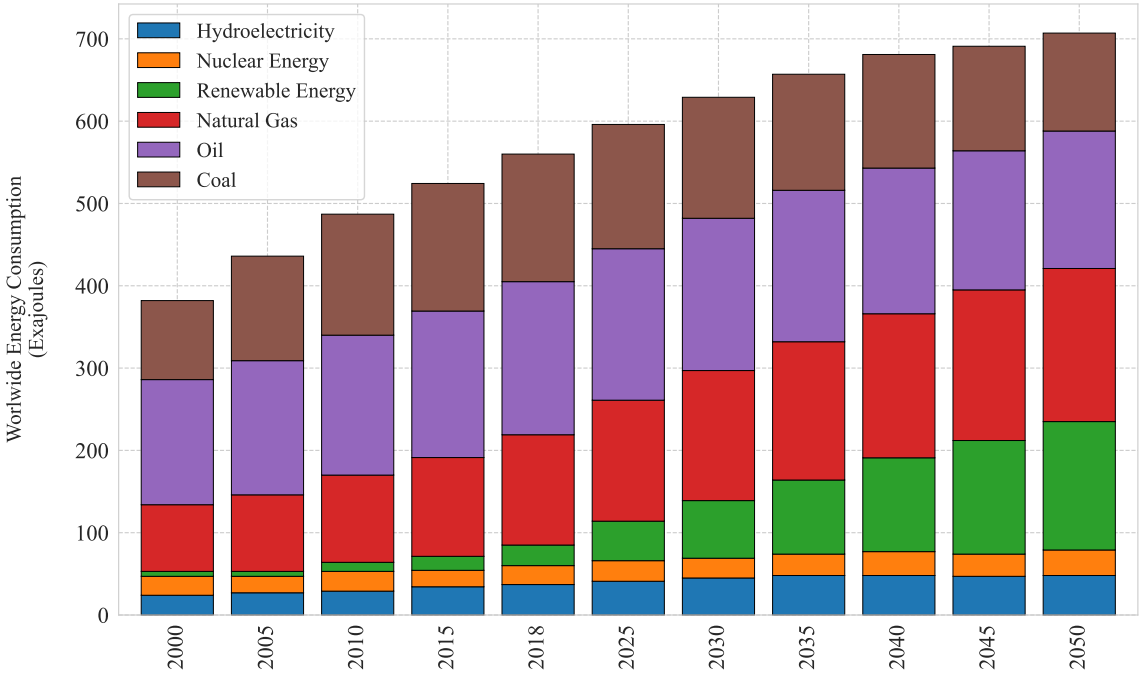


Figure 2.1: Worldwide energy consumption and production sources trend [12].

Despite the many benefits associated with using renewable sources to generate electricity, there are still challenges with their widespread adoption. The main challenge of these technologies is their intermittency in energy supply, resulting in a lower level of reliability. This will increase uncertainty and the importance of provisions for the use of energy storage systems. Consequently, energy storage systems play a crucial role in smart grids by improving the flexibility of the electric power system and helping to maintain the balance between supply and demand while coping with uncertainty arising from a variety of sources. Energy storage performance has a significant effect on both the efficiency and the overall cost of the system. Today, a wide variety of storage applications are being investigated, with the majority aiming to lower costs while ensuring long-term viability. Currently, the most significant challenge to the use of electrical energy storage systems is the capital requirement as well as the operational costs. Another ultimate goal of researchers is to ensure that these storage devices do not have a negative impact on the environment [15].

2.1.1 Energy Storage Systems Advantages

ESSs have a number of advantages in addition to their primary purpose of storing surplus energy. Since a power plant cannot exclusively rely on renewable energy without an ESS, using an ESS improves the penetration of renewable energy and reduces curtailment. As a result, fuel consumption and carbon dioxide emissions will decrease [16]. Furthermore, since RES are intermittent, it is essential to balance supply and demand in order to smooth out fluctuations. This will also help to mitigate issues with power generation's electrical systems [17]. Shaving peak energy loads will reduce the risk of load shedding, particularly if a large storage capacity is considered [18]. Another benefit of ESSs is that they improve the overall efficiency of a power plant, resulting in a reduction in long-term operating costs [19]. Furthermore, the versatility of ESSs allows them to cover rural locations that are frequently out of energy [20].

2.1.2 Energy Storage Systems Types

The development of ESSs has always been in progress, and different storage systems have developed to accommodate the various uses for energy and the wide range of applications. ESSs are mainly classified into three main categories as presented in Fig. 2.2

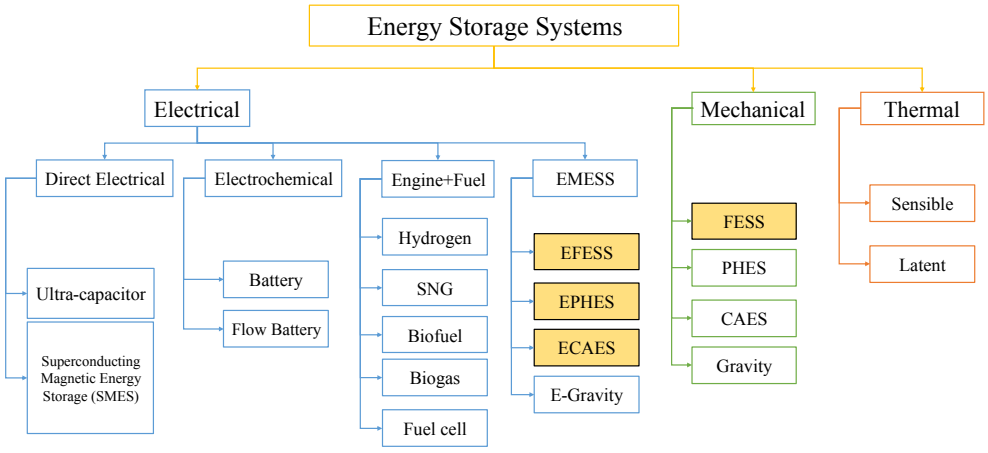


Figure 2.2: Energy storage systems classifications; most commonly used mechanical energy storage systems are highlighted [18].

There are two types of mechanical energy storage systems: pure mechanical MESS and mixed

Electrical Mechanical Energy Storage System (EMESS). A key distinction is whether stored energy is used directly or via a motor-generator. MESSs are advantageous in a number of ways over conventional ESSs, including their environmental impact, cost, and long-term viability. Flywheel Energy Storage System (FESS), Pumped Hydro Energy Storage (PHES), and Compressed Air Energy Storage (CAES) are the three basic types of MESSs, as depicted in Fig. 2.2. The most significant consideration in selecting the right system among them is energy source, load type, and available space. It's also worth noting that there are certain common benefits among the many types of MESSs, such as a faster response time and no environmental effects. These ESSs create fewer pollutants at both the operating and construction stages, which is a key component of improving air quality [18].

Among the many energy storage technologies available, PHES and CAES can support large-scale energy storage applications. Despite the fact that pumped hydro is a well-known and often used energy storage method, its heavy dependence on certain geographical factors and environmental issues makes new innovations and advances difficult. CAES is a potential ESS with high reliability, commercial feasibility, and low environmental impact. Even though large-scale CAES plants are still in operation, the technology is not widely used due to the high heat of compression loss. Scientists and researchers are working hard to increase the overall efficiency of CAES to give a better solution for grid stability, especially with the rise of wind and solar-based power generation in recent years.

2.1.3 Compressed Air Energy Storage

CAES is among the most affordable storage options in terms of both initial and operational costs [21]. As a large-scale energy storage device, it can store electrical energy for several hours. Moreover, since CAES stores compressed air in a tank, it is not vulnerable to the effects of self-discharge and degradation [22]. CAES is therefore a great utility-scale storage technology for large-scale applications such as facilitating the integration of RES into a power system. Unlike PHES, which requires certain geographic locations for utilization, CAES is much more flexible because the air can be stored in both underground caverns and large high-pressure air tanks. CAES is therefore the only commercially feasible alternative for utility-scale energy storage given current energy storage

costs [22].

CAES is one of the most promising technologies for combining renewable energy systems-based plants with electricity supply, and it has a lot of potentials to compensate for renewable energy's fluctuating nature [23]. When there is no demand in a CAES system, off-peak grid power or electricity generated from RES is used to run the compressors and store high-pressure air in a sealed underground cavern or a large storage tank. When electricity is needed, this high-pressure air from the underground reservoir is collected and employed to drive the turbine, and power is generated by the coupled generator. Compressed air is usually mixed with natural gas and burned together in the same way as a conventionally operated turbine plant does. Due to the higher temperature of the inlet air for turbines, this approach produces more electricity [23].

CAES concepts exist today at various stages of development, targeting different applications and possessing different cons and pros. Fig. 2.3 shows a general classification of different types of CAES systems based on heat utilization (handling the generated heat in the compression process):

(1) Diabatic

(2) Adiabatic

(3) Isothermal

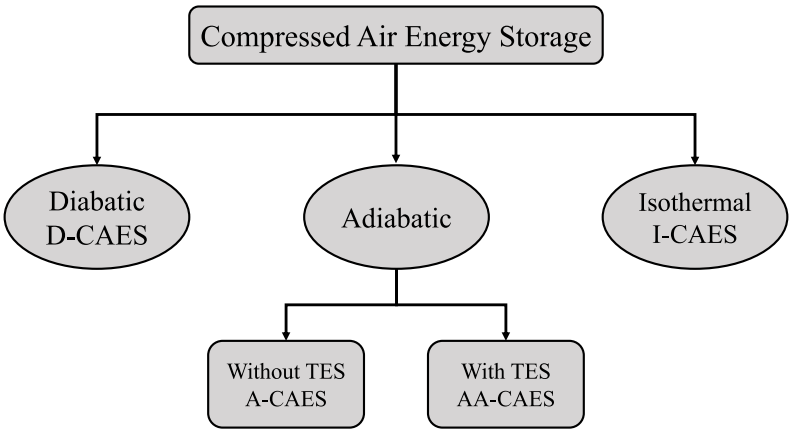


Figure 2.3: Compressed air energy storage general classification

In a Diabatic Compressed Air Energy Storage (D-CAES), to reduce the power consumption of compressors, the air leaving the compressors needs to be cooled to a lower temperature. The heat

of compression is dissipated into the atmosphere as waste without being utilized. Consequently, a large amount of energy is lost in the charging process, which makes the CAES system less efficient. Generally, if the temperature of pressurized air is too low for the expansion process, condensation and icing are likely to occur in turbines. Therefore, in a D-CAES, an external heat source like fossil fuel is needed to preheat air to a specific temperature before entering the turbine to not only prevent icing but also produce more power. Using fuels in combustion chambers will produce GHGs which will cause pollution to the environment. To overcome the shortcomings of D-CAES systems, research has been done on novel concepts of CAES systems. Fig. 2.4 and 2.5 show Huntorf and McIntosh plants as an example of D-CAES systems. Besides the number of compression stages, another difference between Huntorf and McIntosh plants is in using a recuperator to utilize the heat of exhaust air to preheat air before entering turbines which results in an 8% efficiency boost [24].

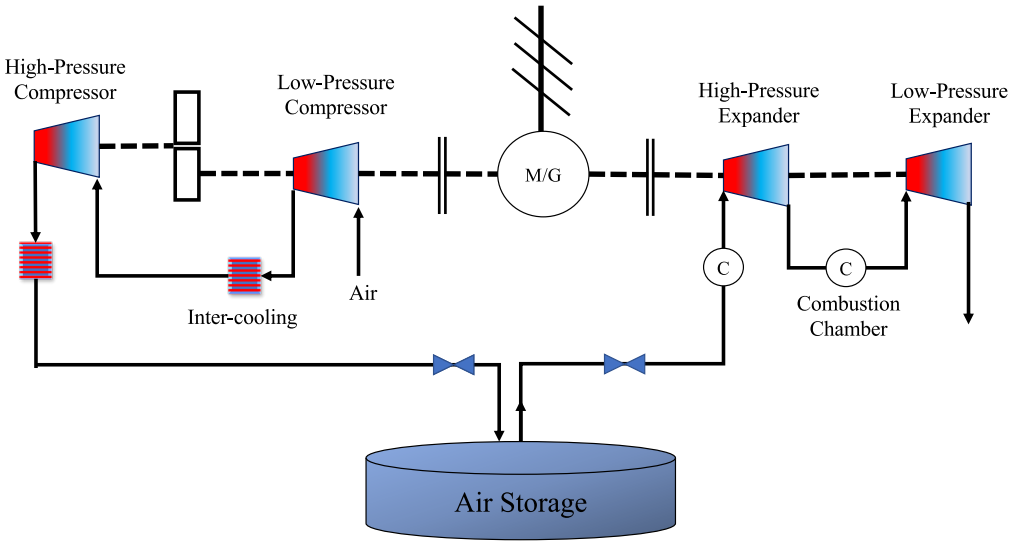


Figure 2.4: Huntorf plant compressed air energy storage simplified scheme [24]

In contrast to the D-CAES, the Adiabatic Compressed Air Energy Storage (A-CAES) system utilizes the heat of compression. In an A-CAES, the heat of compression is stored within the air itself, while in an AA-CAES, the captured heat will be stored in an additional Thermal Energy Storage (TES). During the discharge process, the stored energy can be used to preheat the air before expansion, or it can be used for heating purposes in buildings or industries. When compression

heat is not captured through TES, the air must be stored at a higher temperature, resulting in a lower energy density. Therefore, compressed air must be stored in a reservoir with a higher volume and greater thermal durability [23]. Fig. 2.6 shows the configuration for A-CAES and AA-CAES. Moreover, there are many different configurations for AA-CAES based on the temperature of compressed air storage.

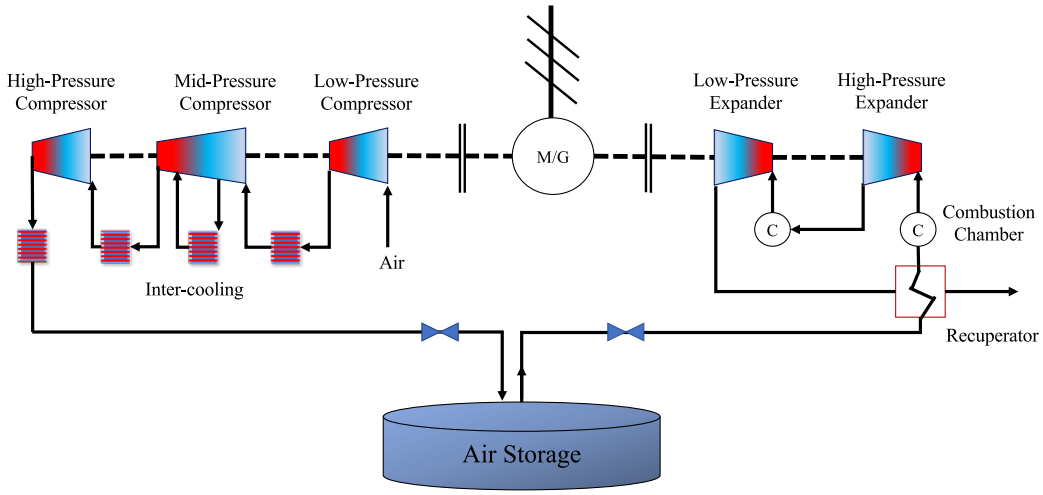
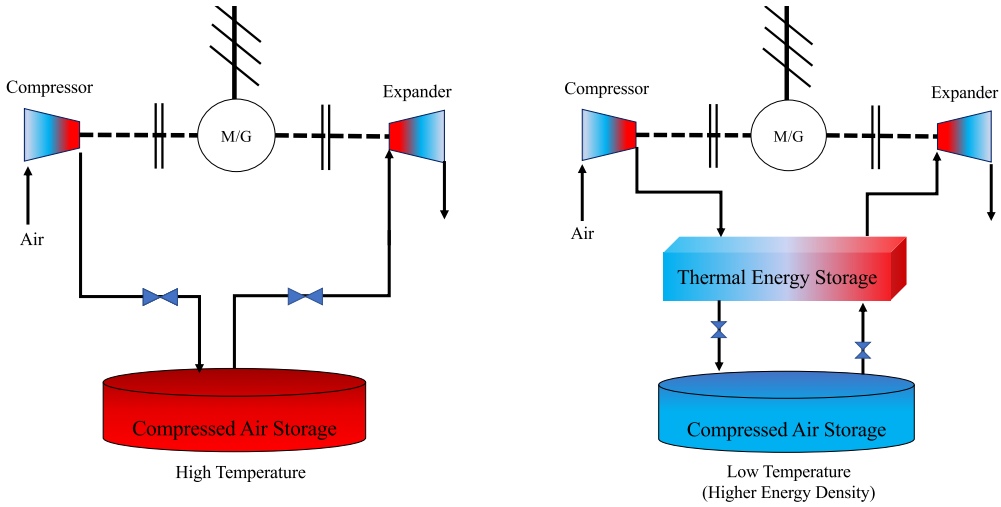


Figure 2.5: McIntosh plant compressed air energy storage simplified scheme [24]

Unlike conventional energy storage systems, such as batteries and PHES, CAES is capable of heat, power, and cooling tri-generation and storage, allowing it to function as a CCHP unit and play a critical role in integrated heat-power distribution systems. To have such a tri-generative system, the air leaving the expanders must have a lower temperature than the ambient air to be able to produce cooling energy and be used in buildings or industrial cooling purposes. This value depends on many different parameters such as the expanders' pressure ratio, the inlet temperature of the expanders, and the isentropic efficiency of the expander. These parameters can affect the final temperature of the air leaving the expander. It is possible to use the captured thermal energy from the compression process for heating purposes. On the other hand, in an isothermal system, the goal is to minimize or even prevent the heat of compression by using different techniques such as spraying water droplets in the compressor or turbine, or surrounding the air storage with a water tank to maintain a constant

temperature in the air storage system. This type of compressed air storage is not yet built in the real world and it is under investigation [23].



(a) Adiabatic CAES (b) Advanced Adiabatic CAES

Figure 2.6: A-CAES and AA-CAES configurations

2.2 Off-design Operation of CAES

CAES systems consist of three major components: compressors, air storage, and expanders. Unlike battery devices that have constant values for their charge and discharge efficiency, the charging and discharge efficiencies of CAES depend on the efficiency of the compressor and expander. These two components have two different types of efficiency: isentropic efficiency and mechanical efficiency. Mechanical efficiency is a dimensionless parameter that indicates how well a machine converts input power into output power. This performance indicator is affected by mechanical losses such as friction which exist in all mechanical components.

In thermodynamics, if a process is both adiabatic and reversible, it can be considered an isentropic process. In such a process, there is no net heat and/or mass transfer. In engineering, this process is used as a reference point for real processes because considering a process both adiabatic and reversible would result in the same initial and final entropies, which is not realistic. The isentropic process is the ideal process for steady-flow devices like compressors and turbines since they perform under adiabatic conditions. Isentropic or adiabatic efficiency is a parameter that describes

how well a device resembles a matching isentropic device. The following formulas are used to indicate the isentropic efficiencies of compressors and turbines:

$$\eta_c = \frac{W_s}{W_a} = \frac{h_{2s} - h_1}{h_{2a} - h_1} \tag{1}$$

$$\eta_t = \frac{W_a}{W_s} = \frac{h_1 - h_{2a}}{h_1 - h_{2s}} \tag{2}$$

In which η_c and η_t are the isentropic efficiencies of compressor and turbine, respectively; h_1 is the enthalpy of air before entering the compressor or turbine, h_{2s}, and h_{2a} are the isentropic and actual enthalpy of air leaving the compressor or turbine, respectively. The isentropic efficiency of compressors and expanders (turbines) varies based on different parameters such as mass flow rate, inlet air temperature, and rotational speed of the shaft. If the compressor or turbine is operating under different conditions than its design conditions, the process is called off-design conditions. The isentropic efficiency of an off-design process is lower than the design-conditioned process and there are performance curves to estimate the isentropic efficiency if the device is operating off-design. As mentioned before, if the temperature of the air leaving expanders is significantly lower than the ambient air temperature, the CAES can be used as a tri-generative system, just like a CCHP system. Therefore, the variation of CAES operation from design to the off-design condition can strongly affect the tri-generation functionality of the AA-CAES system.

2.3 Optimal Dispatch of Energy Hubs with ESS

As previously mentioned, energy storage systems play an important role in the efficiency and reliability of power systems. This technology can be applied to a wide range of applications, ranging from large-scale microgrids or energy hubs to small-scale applications, such as a house. The intermittent and fluctuating nature of renewable energies in power generation will affect the stability and security of the power system. EHs are a solution to utilize renewable energy effectively by integrating different components such as CCHP unit and ESS . Since there are many components in an energy hub, different energy flows can be present. One of the main challenges in utilizing ESS

in EHs is how to optimally control them. Controlling, in this case, means maximizing the efficiency and reliability of the power system. To be more specific, for a grid-connected energy hub with ESSs, the main objective is to minimize the total cost, the total fuel consumption, and the corresponding negative environmental impacts (emissions), while considering uncertainties like power generation from renewable sources, demand loads, and fluctuating electricity and gas prices and the limitations of each device in the energy hub. The problem here is to decide the amount of energy that should be stored or released and the time that a component must be operating. These control problems are not easily solved since they have high complexities, and the decisions must be taken in each time-step to find the optimal sets of decisions. Moreover, these optimization problems are not usually convex, which means that they might have several local minima points. Therefore, gradient-based optimization methods are not suitable for solving them, and most of the time they converge to local minima.

The optimal control methods can be classified in two categories [25]:

(1) Classical optimization techniques:

These techniques are the traditional methods that have been used for many years. Here we will have a brief explanation of each method.

- Linear Programming (LP) method:

  In situations where the objective function is linear and the system constraints are linear equalities or inequalities, linear programming can be used to solve these optimization problems [26]. It is possible to find a globally optimal solution quickly by using linear programming techniques because of their simplicity, numerical simplicity, and promising way to quickly converge to a global optimal solution when the objective function and the constraints are linear. Linear programming is not able to take the uncertainties of the system into account which is essential in practical problems.

- Dynamic Programming (DP) method:

  In the DP method, the complex problem is divided into a series of simpler sub-problems by using a multi-stage decision process. This method is very general and can be applied to both linear and non-linear objective functions and constraints. DP algorithms

will converge to the global optimal solution regardless of whether the objective function is convex or non-convex [25]. One disadvantage of this method is that it can be used only if the objective function can be solved with a recursive formula. Moreover, prior knowledge of all past and future signals (historical and forecast data) is required. The number of stages has a linear relationship with the complexity of dynamic programming algorithms but as the number of state variables increases, the complexity grows exponentially. This phenomenon is called the "curse of dimensionality" [27]. To speak more vividly, when the number of components in an EH increases, the increase in the number of equations and states makes the algorithm much more complex than before.

- Stochastic control strategies:

  These controlling methods are suitable for optimization problems that involve uncertainties. As an example, for controlling the energy storage devices, we can formulate the problem as a Markov Decision Process (MDP) and solve it by using stochastic DP. The advantage of this method is that unlike classical DP, which finds the optimal policy for predetermined signals, the stochastic approach optimizes the control policy over a family of possible signals [28]. Therefore, the output of this approach is not a single optimal solution for the problem, but an optimal control strategy that can be used for real-time operation of the system. One of the disadvantages of this method compared to the novel advanced methods like RL is that the stochastic optimization uses a specific Probability Distribution Function (PDF) for the uncertainties, while in RL, the PDF is learned through the historical data.

(2) Advanced Optimization Techniques:

There are two types of advanced optimal control methods: metaheuristics and machine learning.

- Metaheuristic Techniques:

  These methods are mainly used when there is a high-dimensional solution space, and the objective function is nonlinear or non-convex . They might have low computational

complexity, and converge to the optimal solution, but their performance is not guaranteed. Most of these algorithms work based on the same principle. Several random solutions are generated first. After that, in each iteration, based on what the objective function is, each of these solutions is ranked and is updated based on a specific mechanism. For instance, this mechanism can be inspired by natural selection, social behavior, the dynamics of ant colonies. Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Artificial Bee Colony, and Firefly Algorithm are among the metaheuristic methods used to solve energy management problems [25].

- Machine Learning Techniques:

  The machine learning techniques are mainly divided into three branches: Supervised Learning, Unsupervised Learning, and Reinforcement Learning [29]. In supervised learning, the main goal is to find a mathematical expression or function to map the inputs to outputs. In this case, both the inputs and outputs are known for the learning process (outputs are labeled). In unsupervised learning, the output data are not available. The main objective here is to find a pattern in input data. RL, which is much more complicated than the other branches of machine learning, is mainly used for control purposes. In RL, an agent is interacting with the dynamic environment to find the optimal sets of actions that lead to the maximum cumulative reward [30]. For energy management purposes, the agent receives input signals from the environment that indicate the current state of it and then takes actions (amount of power flow from/to each component) which will affect the next state of the environment. After taking the actions, the agent receives the new states and a reward from the environment. This process is done based on trial and error by the agent. After the learning process is done, the agent can make decisions for real-time energy management of the energy hub. One of the advantages of this approach is that the agent will consider the uncertainties based on the historical data that is used in the learning process. Therefore, we do not need to consider a specific probability distribution function for uncertainties and control systems. Moreover, since the learning process is done before executing the model in real-time operation, the agent can make decisions in a very short time (0.001 sec), compared to other methods

and techniques that need to solve the optimization problem in each iteration [9].

2.4 Reinforcement Learning-based Energy Management

Considering the nature of learning, the concept of learning by interacting with the environment is generally the first that comes to mind. The person is intensely aware of how their environment responds to what they do, whether they are learning to drive a car or hold a conversation, and they want to affect what happens through their behavior. Learning from interaction is a fundamental concept that underpins practically all learning and intelligence theories.

In RL, the agent learns what to do and how to map observed situations to specific actions, in order to maximize a numerical reward signal. One of the most significant differences in reinforcement learning is that the learner is not told what to do and what actions to take. It must realize which actions are likely to yield the greatest reward. In most cases, the actions do not only affect the immediate reward, but also the future rewards as well. Therefore, the two most significant distinguishing properties of reinforcement learning are trial-and-error search and delayed reward [31].

The concept of RL differs from supervised learning, which is the focus of the majority of current research on machine learning. The process of learning from a set of labeled data is known as supervised learning. The main goal of this type of learning is to extrapolate or generalize its response to the data that are not present in the dataset. Although this is an important type of learning, it is insufficient for learning through interacting with the environment. Moreover, RL is also different from unsupervised learning in which tries to find a hidden pattern in unlabeled datasets.

In order to provide better illustrations, it is helpful to have some understanding of reinforcement learning terminology:

(1) **Agent**: The entity that uses a policy to maximize expected return from transitioning between states of the environment in RL. (In energy management problems: the control system that chooses the amount of energy being utilized by each source)

(2) **Environment**: The environment is the surrounding with which the agent interacts (for example the whole microgrid system including renewables, customers, and energy market).

(3) **State**: The parameter values that describe the current condition of the environment and that is used by the agent to decide. (For example, renewables generation, load demands, state of charge of the energy storage systems, etc. in each time step)

(4) **Action**: The decision(s) that the agent makes at the current time step (for example the amount of energy to be stored/released from energy storage)

(5) **Policy**: The thought behind choosing an action by the agent is called policy, which maps the states to actions.

Basically, RL is a game between an agent and the environment. The environment is modeled as a MDP which is a discrete-time stochastic control process. In a MDP, the outcomes are partly random and partly controlled by the decision-maker. MDPs can be used to study optimization problems that are solved using DP or RL. The main goal of this game is to find the optimal policy that maximizes cumulative future rewards. As depicted in Fig. 2.7, the game between the agent and the environment at each time step is as follow:

(1) Agent observes the environment's state s_t.

(2) Agent chooses action a_t based on policy π.

(3) Agent receives the reward $r_{t+1} = R(s_t, a_t)$ from the evironment.

(4) Agent lands in the next state $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$ in which $T(s'|s, a)$ is the probability of landing in state s' by taking action a in state s.

In energy management problems, many uncertainties such as renewables generation, fluctuating load demands, and varying prices for electricity and natural gas are present. Since the EH controller can only control some parameters and sources like ESSs and imported/exported energy to the grid system and has no control over uncertainties and fluctuations in renewables and loads, this problem can be modeled as a MDP and solved by RL. The transition probability $T(s'|s, a)$ is influenced by uncertainties. In model-based approaches, these uncertainties are predicted using short-term forecasting models or estimated using Monte Carlo (MC) simulations. However, in model-free approaches, this transition probability can be learned through massive historical data that represent the

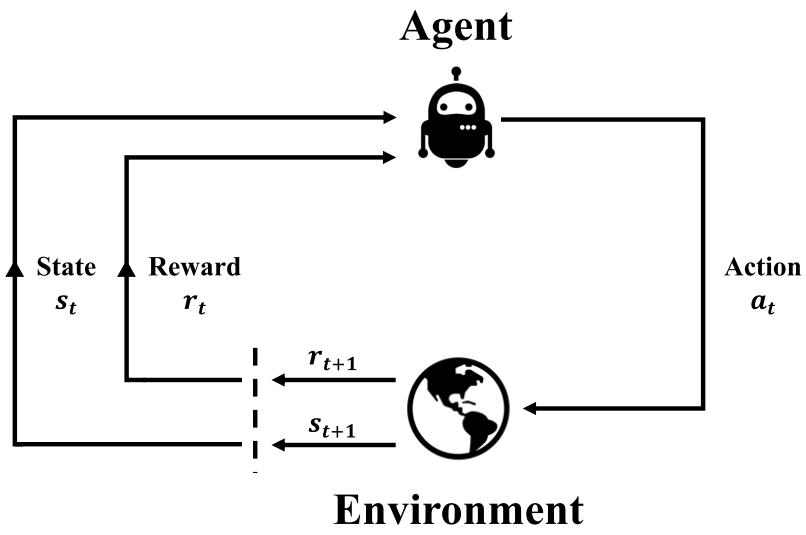


Figure 2.7: Reinforcement learning MDP framework [32]

uncertainties themselves, instead of considering a distribution function. After the learning process, the agent can choose the optimal actions in real-time operation of the energy hub to minimize or maximized the determined objective function.

One of the main disadvantages of RL is its inability to handle existing constraints in the environment. There is no mathematical framework for the RL agents, unlike model-based approaches, which makes it difficult to handle the constraints. Additionally, since RL agents learn policies through trial and error, the optimality of the learned policy cannot be guaranteed. Section 3.3 discusses the methods for dealing with constraints and global optimization.

2.5 CAES Optimal Dispatch

Numerous probabilistic and deterministic approaches have been proposed for identifying an optimal scheduling of EH in the presence of CAES. A number of significant contributions have been made in this area. The dispatch problem of a non-linear CAES model was investigated by Men et al. [33]. They considered a co-generative AA-CAES as the ESS which contributes to the heat and electricity networks. A new metaheuristic approach for optimizing the dispatch problem for a tri-generative AA-CAES was proposed by Li et al. [34]. Bai et al. [35] optimized AA-CAES operation in EHs using a tri-state and off-design model. In this study, they considered an AA-CAES

that produced only electricity. Li et al. [36] and Ma et al. [37] investigated the optimal dispatch problem of an EH with a tri-generative CAES without considering the effect of partial-load operation. Zhang et al. [38] proposed an optimal scheduling strategy for a Regional Integrated Energy System (RIES), taking into account the trading strategy as well as the applications of cogenerating AA-CAES in heating and spinning reverse services. Li et al. [39] studied a MILP mathematical model of AA-CAES, in which its off-design characteristics were considered. It was concluded that the AA-CAES could mitigate wind curtailment and reduce operating costs. Their proposed AA-CAES, however, was a co-generative system, and the effect of off-design conditions on its tri-generative functionality was not investigated.

Despite the fact that the studies mentioned above significantly contributed to the optimal dispatch of AA-CAES in EH, none of them addressed uncertainties in the load and renewable energy generation. In order to address the uncertainty associated with EH dispatch, several methods have been developed. To determine the optimal day-ahead dispatch strategy for an EH, Jalili et al. [40, 41] and Wen et al. [42] used a SP approach using PDFs for load demands and renewables generation. The EH was consisted of four energy flows (electricity, heating, cooling, and natural gas) and a solar-powered CAES was used as energy storage. Bai et al. [43] used a stochastic DP algorithm to solve a nonlinear self-dispatch problem for a small grid-connected EH comprised of an AA-CAES and a Electric Heat Pump (EHP). They considered the off-design operation of the AA-CAES and EHP. Yang et al. [44] investigated the optimal dispatch strategy for an integrated energy system with CAES and a demand response program. In order to consider different scenarios for uncertainty, SP was used. Nevertheless, their CAES system was modeled as a simple (black-box) input-output energy storage system that does not consider the characteristics of its sub-components. Jadidbonab et al. [45] developed a Constraint Value at Risk (CVaR) scheduling model for an EH with D-CAES with the uncertainties modeled using a SP approach. Zeynali et al. [6] modeled multi-objective optimal EH management with hybrid battery and CAES integration. To deal with uncertainties, a RO approach was used, taking into account the worst-case scenario. A study by Mirzaei et al. [46] focused on optimizing the utilization of energy carriers in an integrated EH by optimizing power-to-gas and D-CAES. Furthermore, MC simulations and SP were used to generate scenarios and address uncertainties. Jadidbonab et al. [47] investigated the optimal dispatch of

an EH with a conventional CAES as energy storage to meet the required electricity, heating, and cooling demands. To model the risk associated with variation in the EH's operation costs, CVaR method was applied, and to consider uncertainties, scenario-based SP was applied. According to a study by Oskouei et al. [48], a novel strategy based on robust-stochastic optimization was developed for maximizing market participation of a virtual EH that consists of integrated EHs and multi-energy industrial consumers. The part-load characteristics of a D-CAES and the uncertainty associated with power regulation were examined by Li et al. [49]. They concluded that CAES scheduling could be infeasible in actual power system operation without considering the off-design and part-load characteristics.

As mentioned, earlier studies have investigated the optimal scheduling of different CAES types in an EH with different energy flows for the day ahead. In order to deal with uncertainties, SP method is widely used. However, there are several disadvantages to using the SP method, such as the high computational cost associated with the consideration of a large number of scenarios. Furthermore, the uncertain parameters are assumed to fit into specific PDFs, which is not the case in practice. To address this issue, recently, model-free strategies, which do not rely on developing a model to incorporate uncertainties of the environment, are shown to be advantageous in solving complex decision-making problems involving high levels of variability and uncertainty [50]. Model-free techniques such as DRL have emerged in optimization (for design and operation) of EHs and microgrids.

Different DRL algorithms have been implemented for the optimal dispatch of EHs and microgrids. Many studies [51–55] have implemented a Deep Q-Network (DQN) model to optimize the dispatch of isolated and grid-tied microgrids with battery storage systems. Since the DQN algorithm can only be applied to discrete action spaces, it could result in suboptimal policies when applied to large-scale problems. It should be noted that in the mentioned studies, the microgrid operational constraints are modeled as penalty functions which are then added to the reward function. To deal with the continuous action space, some studies [56–58, 9, 59] have utilized Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO) to find the optimal policy for controlling microgrids. In spite of this, they employ the same approach as previous studies in order to address the constraints. Ding et al. [60] and Qiu et al. [61], on the other hand, have used modified

DDPG algorithms, such as Primal-Dual Deep Deterministic Policy Gradient (PD-DDPG), in order to control the EHs and smart homes.

In addition to considering uncertainty in optimizing the EH dispatch problem, it is also essential that the off-design operation of the CAES be taken into account. As a result of the uncertain nature of energy demand, as well as variations in renewables energy generation, CAES may operate in off-design and partial-load conditions [5]. In contrast to a battery energy storage system, CAES is a mechanical energy storage system that cannot be modeled as a simple input-output system. Therefore, some studies have considered the part-load operation of different types of CAES in dispatch problems. Part-load operation of CAES is taken into account in modeling electricity-generating CAES [35, 49] as well as co-generative (electricity and heating) [39, 43]. However, despite its importance, most of the studies have not considered the effect of part-load operation.

2.6 Gaps and Limitations

Summary of the literature review indicates the following gaps in the existing studies on optimal dispatch of EH with AA-CAES:

(1) SP is the most common method advocated in the literature for dealing with uncertainties. This approach utilizes a variety of stochastic scenarios to model the dispatch problem under uncertainty. The number of scenarios considered for a problem is a major determinant of the accuracy and performance of the SP approach. Due to the significant increase in computation costs when there are multiple scenarios, SP is not suitable for large-scale problems, such as management of a CCHP EH with many energy sources. Furthermore, SP considers specific probability distribution functions for representing uncertainties, which is not realistic for real-time optimization. In most cases, real-world data do not follow a particular distribution function. Consequently, it is necessary to develop a controller that can handle these uncertainties without relying on predetermined probability distributions. This realistics approach could be made possible using a model-free DRL. Several studies, [9, 61, 59, 62] have shown that a model-free DRL approach is superior to the SP approach both in terms of computational cost and optimality in case of scheduling of EHs and microgrids.

(2) A major problem in the existing DRL algorithms is their inability to handle constraints in dispatching (optimization) problems. In most studies , for instance in [63–65], the constraints are added to a reward function in form of a penalty, which could result in sub-optimal (local optimal) solutions when the search space of the problem is big. In a few studies, more advanced algorithms are utilized to handle constraints more effectively, but these algorithms might not work well for global optimization in large-scale problems [61, 66].

(3) The effect of off-design characteristics is not considered in optimal dispatch of EH for trigenerative AA-CAES, but it has been considered in some studies for co-generative AA-CAES and electricity only.

This thesis addresses the shortcomings of previous studies by proposing a safe deep reinforcement learning algorithm based on primal-dual optimization and IL to be used to develop an adaptive controller for the optimal dispatch problem under uncertainty in EH. In this algorithm, the operational constraints of real-time scheduling of EH are taken into account by using cost networks. In contrast to the traditional DRL approaches, in which the constraints are added to the reward function as a penalty term, the proposed algorithm treats the constraints as independent cost functions. In order to help the search agent follow a more efficient learning process, expert demonstrations are used to avoid getting stuck in a local optima during the learning process. The expert demonstrations are referring to the optimal dispatch results from a MILP solver, for training days only, as a baseline solution. Moreover, as the partial-load operation of AA-CAES affects the charging and discharging efficiencies of the system, the effect of off-design operation of the trigenerative AA-CAESon the scheduling of EH equipment is considered. Finally, the performance of the proposed safe DRL algorithm in terms of operational cost, constraint violation, and using the potential of AA-CAES is compared with a base DRL algorithm (DDPG) and a benchmark. The proposed methodology is summarized in figure 2.8 and will be described in details in section 3.3.4.

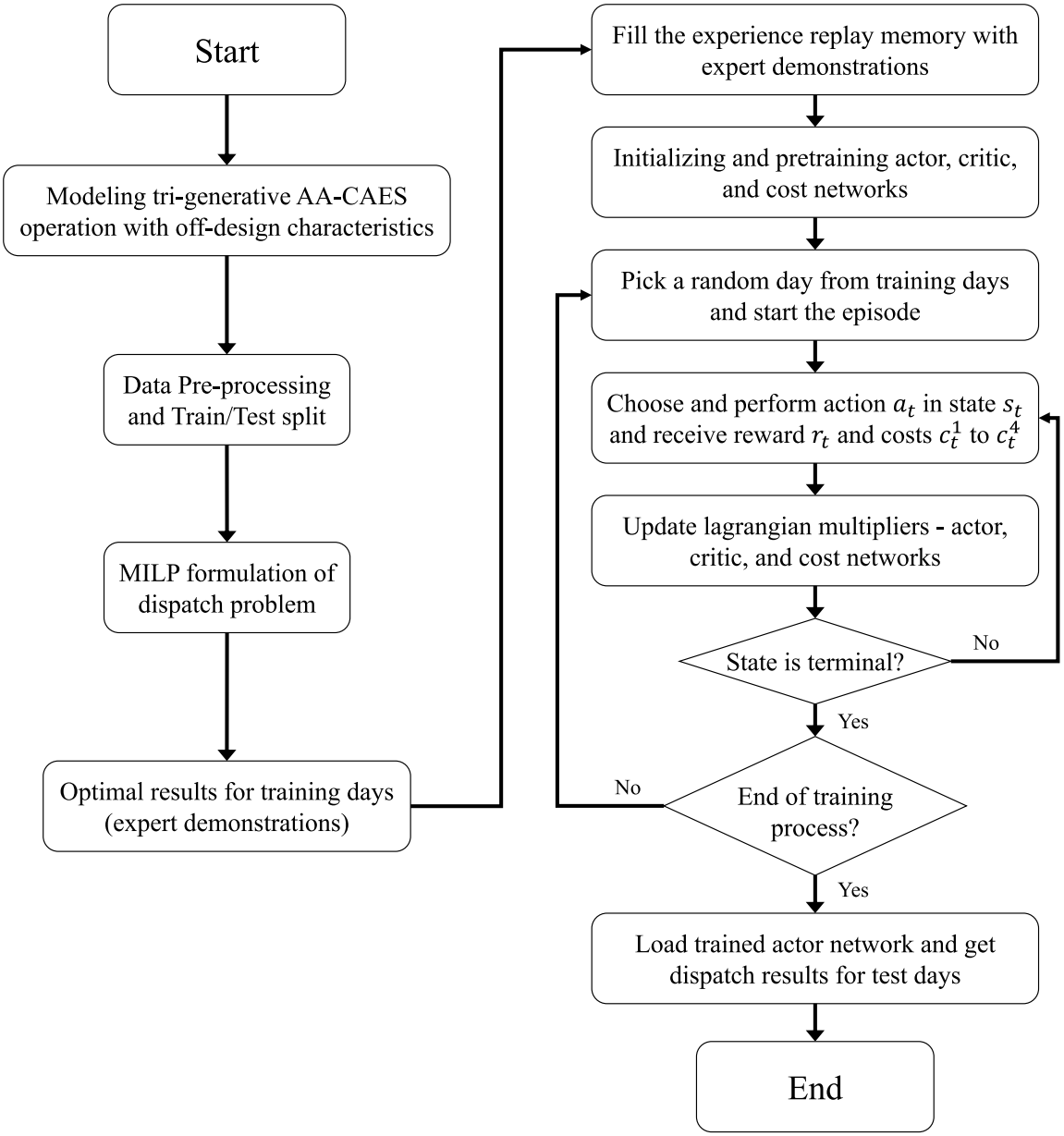


Figure 2.8: Proposed methodology for energy hub dispatching

Chapter 3

Methodology

This chapter starts with a description of the problem and the mathematical formulation of the EH in Section 3.1. Next, in Section 3.2, the optimal scheduling problem is modeled as a CMDP. Afterwards, our proposed safe DRL approach and other DRL algorithms are explained in detail in Section 3.3.

3.1 Energy Hub Modeling

3.1.1 Energy Hub Description

Fig. 3.1 shows the EH in this study which is connected to both the electric grid system and the natural gas network. The users in the EH under study have electrical, heating, and cooling demands. As a RES, the EH consists of Wind Turbine (WT) and Photovoltaic (PV) panels. Moreover, a Micro Turbine (MT) is utilized for producing both electricity and heating energies. For energy storage system, an AA-CAES is considered. Other equipment such as an EHP, a Gas Boiler (GB), and an Absorption Chiller (AC) are also present in EH. Among the mentioned equipment, MT and GB consume natural gas to produce electricity and heating. In the EH, electricity is supplied by a utility company, and for the MT and GB, natural gas is supplied by a gas company. Heating energy is generated by EHP, GB, and the MT's heat output, while coooling energy is generated by AC and EHP. In the present study, AA-CAES is a tri-generative ESS, which means that it can produce heating energy in the charging process and cooling energy (based on the ambient air temperature)

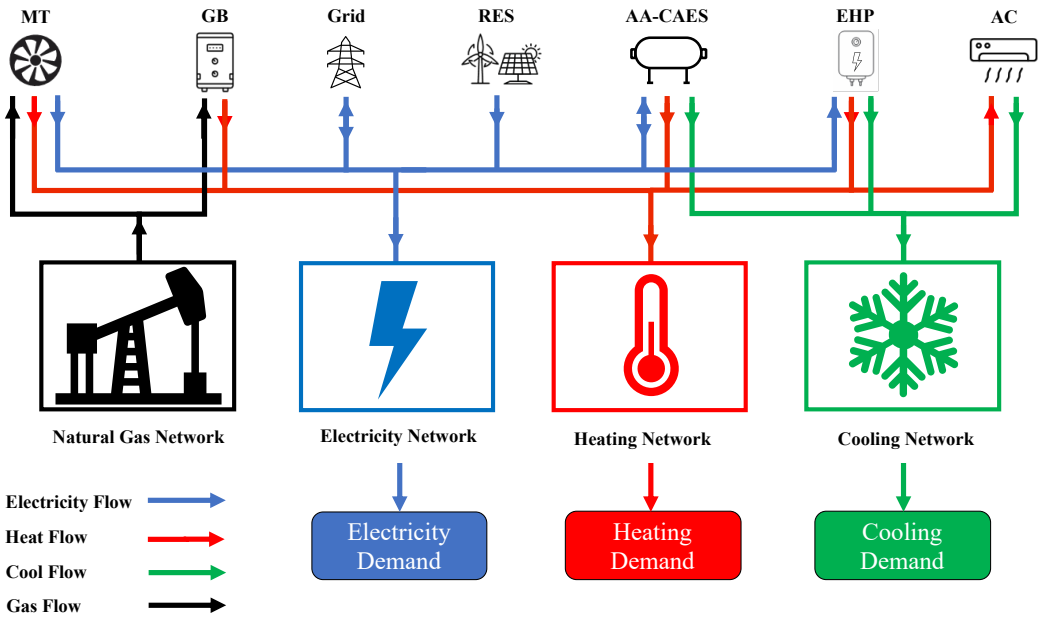


Figure 3.1: Energy hub's equipment and configuration.

in the discharging process.

3.1.2 System Modeling

The purpose of this section is to describe the operation of the EH equipment and to present the mathematical equations that govern its operation. For all equipment except AA-CAES, an input-output (black-box) model is considered, while AA-CAES is modeled in more detail. Each power term is expressed in kW.

3.1.2.1 MT Modeling

In addition to producing electricity, the MT generates a substantial amount of heating energy, which can be reused through heat recovery. Modeling the operation of the MT is as follows:

$$P_{MT}\,(t) = \; G_{MT}\,(t) \cdot \eta_{MT,p} \tag{3}$$

$$H_{MT}\,(t) = \; G_{MT}\,(t) \cdot \eta_{MT,h} \tag{4}$$

where $P_{MT}(t)$ is the electric power generated by the MT and $G_{MT}(t)$ is the amount of power that corresponds to the amount of natural gas consumed by the MT at time step t. $H_{MT}(t)$ represents the recovered heat from the MT; $\eta_{MT,h}$ represents the heat output efficiency of the MT; and $\eta_{MT,p}$ represents the electric output efficiency of the MT.

3.1.2.2 GB, AC, and EHP Modeling

Using the EHP, electricity can be converted into heat to satisfy the heating demand of the consumers. It can be modeled as follow:

$$H_{HP}\,(t) = \; P_{HP,h}\,(t) \cdot COP_{HP} \tag{5}$$

where $H_{HP}(t)$ is the heating energy generated and $P_{HP,h}(t)$ is the electricity consumed by the EHP for producing heating energy. COP_{HP} is the Coefficient of Performance (COP) of the electric heat pump. GB is another equipment that participates in meeting the heating load demand. It can be modeled as:

$$H_{GB}\,(t) = \; G_{GB}\,(t) \cdot \eta_{GB} \tag{6}$$

where $H_{GB}(t)$ represents the heating energy generated by the GB; $G_{GB}(t)$ represents the amount of power consumed by the GB; and η_{GB} represents the GB's heating efficiency. In order to meet the cooling load demands of the consumers, AC and EHP are utilized. An AC is a lithium bromide refrigerator that produces cooling energy by utilizing heating energy. Meanwhile, the EHP consumes electricity to operate its compressor and provide cooling power. In simple terms, they can be modeled as follows:

$$C_{AC}\,(t) = \; H_{AC}\,(t) \cdot COP_{AC} \tag{7}$$

$$C_{HP}\,(t) = \; P_{HP,c}\,(t) \cdot COP_{HP} \tag{8}$$

in which $C_{AC}(t)$ and $C_{HP}(t)$ are the cooling energy generated by AC and EHP, respectively.

$H_{AC}(t)$ is the heating energy fed to AC and $P_{HP,c}(t)$ is the electricity consumed by EHP to produce cooling energy.

3.1.2.3 AA-CAES Modeling

In this study, AA-CAES is considered as ESS operating in the EH. The heat of the compression in an AA-CAES is stored in a hot tank by the working fluid, which can be used either to preheat the air before expansion or to provide heating to industrial and residential buildings. Depending on the configuration and design parameters of the AA-CAES, the air leaving the air expanders may be used for cooling purposes if it is lower in temperature than the ambient air temperature.

In modeling of the AA-CAES system, the following assumptions are considered:

- The air is assumed to be an ideal gas.
- The temperature of the air storage is assumed to be constant (20C).
- Both the compression and expansion processes have two stages.
- The temperature of hot and cold tanks is assumed to be constant (do not change over time).
- The pressure drop in heat exchangers and pipes are considered as negligible.

Fig. 3.2 and 3.3 illustrate the AA-CAES configuration and thermal storage subsystem, respectively. Table 3.1 shows the properties of each point in the proposed AA-CAES system in design condition, and the design parameters are presented in Table 3.2. For the off-design conditions, the following assumptions are considered:

- The compressor and expander isentropic efficiencies are a function of the partial load ratio.
- The compression ratio of low- and high-pressure compressors are assumed to be constant over time.
- The expansion ratio of low- and high-pressure expanders are a function of partial load ratio.

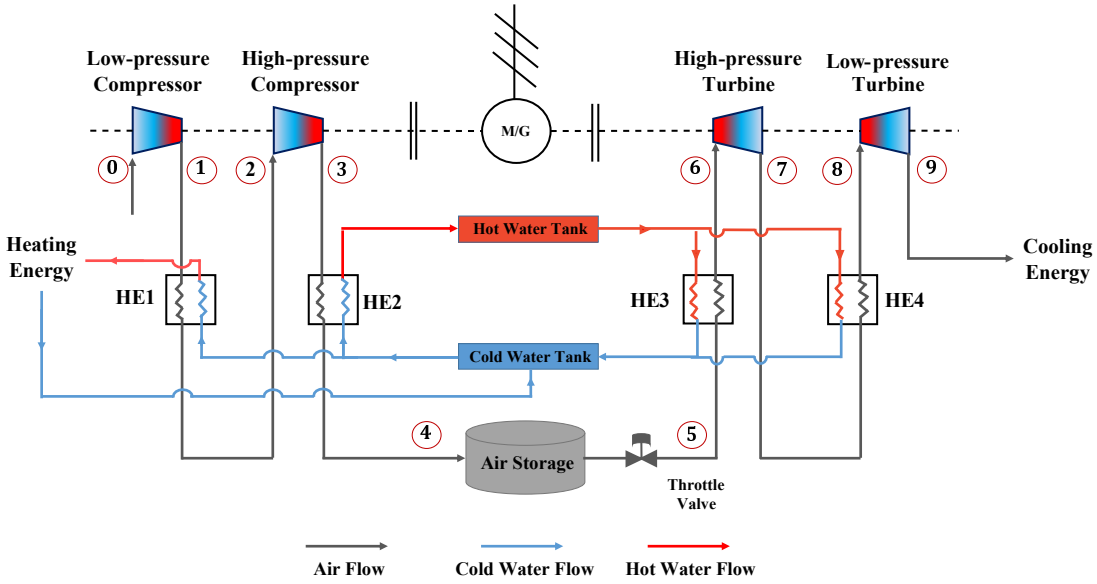


Figure 3.2: Tri-generative AA-CAES configuration.

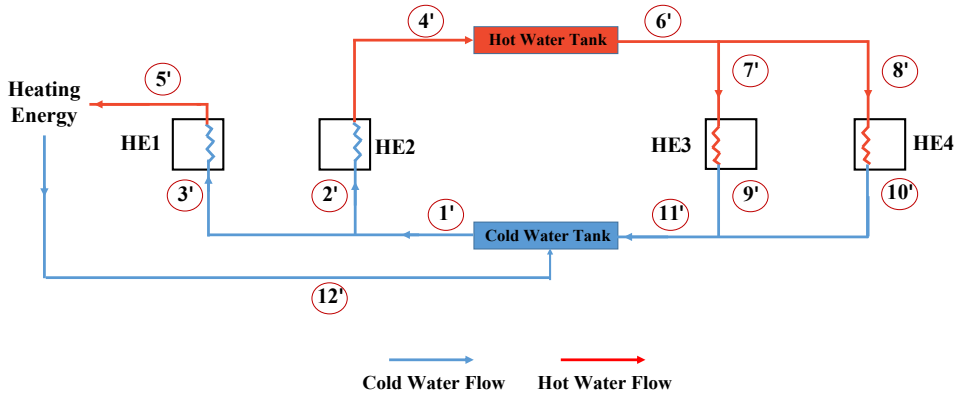


Figure 3.3: AA-CAES thermal storage subsystem.

According to the assumption that air is an ideal gas, its characteristics are as follows:

$$p = \rho \cdot R \cdot T \tag{9}$$

which demonstrates the relationship between the air pressure $p(Pa)$, air density ρ $(\frac{kg}{m^3})$, universal air constant $R(\frac{J}{kg \cdot K})$, and the air temperature $T(K)$. As air enters the first compressor from ambient conditions, its temperature and pressure are the same as those of the ambient environment

Table 3.1: Properties of each point in AA-CAES system at design conditions.

State	**Pressure** (bar)	**Temperature** (K)	**MFR** (kg/s)	**State**	**Temperature** (K)	**MFR** (kg/s)
0	1.0	293.00	1.257	$1'$	293	2.539
1	8.5	598.96	1.257	$2'$	293	1.312
2	8.5	323.59	1.257	$3'$	293	1.227
3	72.0	650.63	1.257	$4'$	368	1.312
4	72.0	328.76	1.257	$5'$	368	1.227
5	42.0	293.00	3.674	$6'$	368	1.881
6	42.0	360.50	3.674	$7'$	368	0.925
7	10.6	278.55	3.674	$8'$	368	0.956
8	10.6	359.05	3.674	$9'$	303	0.925
9	1.0	227.16	3.674	$10'$	293	0.956
				$11'$	298	1.881
				$12'$	293	1.227

***MFR** : Mass Flow Rate

($T^1_{c,in} = T_{amb}$ and $p^1_{c,in} = p_{amb}$). The pressure and temperature at the inlet of the next compressor depends on the outlet conditions of the Heat Exchanger (HE) 1. Inlet and outlet temperatures of the compressors are as follows:

$$T^i_{c,in}\,(t) = \begin{cases} T_{amb}, & i = 1 \\ T^1_{ahec,out}(t) & i = 2 \end{cases} \tag{10}$$

$$p^i_{c,in}\,(t) = \begin{cases} p_{amb}, & i = 1 \\ p^1_{ahec,out}(t) & i = 2 \end{cases} \tag{11}$$

in which $T^i_{c,in}\,(t)$ and $p^i_{c,in}\,(t)$ are the temperature and pressure of air entering the ith compressor at timestep t, respectively; T_{amb} and p_{amb} are the ambient air temperature and pressure, respectively; $T^1_{ahec,out}(t)$ and $p^1_{ahec,out}(t)$ are the air outlet temperature and pressure after the first HE in the charging process at timestep t, respectively.

As the air is compressed, its temperature and pressure increase in accordance with Eqs. (12) and (13):

$$T^i_{c,out}(t) = T^i_{c,in}(t) \cdot \left(1 + \frac{(\pi^i_c)^{\frac{\gamma_a - 1}{\gamma_a}} - 1}{\eta_c(t)}\right) \tag{12}$$

$$p^i_{c,out}(t) = p^i_{c,in}(t) \cdot \pi^i_c \tag{13}$$

Table 3.2: Design parameters of AA-CAES system.

Maximum Compressors Input Power	800 kW
Maximum Expanders Output Power	800 kW
Air Storage Volume	1100 m^3
Charging Time	$\simeq 7.5hr$
Discharging Time	$\simeq 3.5hr$
Compressors Isentropic Efficiency	0.83
Expanders Isentropic Efficiency	0.8

In these formulas, $T^i_{c,out}(t)$ is the air outlet temperature after compression in the ith compressor, γ_a is the isentropic expansion factor, which represents the ratio between the specific heat capacity of air at constant pressure and constant volume ($\gamma_a = \frac{c_p}{c_v}$); π^i_c is the compression ratio of the ith compressor and is assumed to be constant over time; and $\eta_c(t)$ is the compressors' isentropic efficiency at timestep t. As shown in Fig. 3.4 , the isentropic efficiency of the compressors varies with partial load.

The air mass flow rate of the compressors (charging mode) is then calculated using following equation:

$$\dot{m}_c(t) = \frac{P_c(t)}{w^1_c(t) + w^2_c(t)} \tag{14}$$

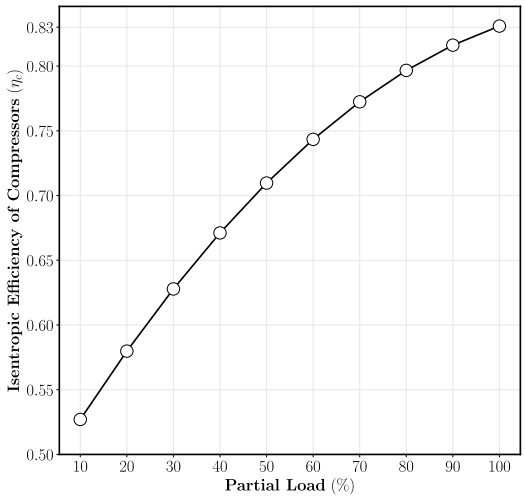


Figure 3.4: Variation of compressors' isentropic efficiency with partial load percentage [67]

in which $P_c(t)$ is the charging power at timestep t; $w^1_c(t)$ and $w^2_c(t)$ are the work of compressors

1 and 2 per unit of mass at timestep t, respectively:

$$w_c^1(t) = c_p \cdot \left(T_{c,out}^1(t) - T_{amb}\right) \tag{15}$$

$$w_c^2(t) = c_p \cdot \left(T_{c,out}^2(t) - T_{c,in}^2(t)\right) \tag{16}$$

Following compression in the first compressor, the air is cooled through HE 1 and the captured heat is used to meet the demand for heating. The amount of this heating power can be calculated as follow:

$$H_{AA-CAES}^{max}(t) = \dot{m}_c \cdot c_p \cdot \left(T_{c,out}^1(t) - T_{ahec,out}^1(t)\right) \tag{17}$$

During the charging process, the outlet air temperature of the ith HE is equal to the inlet air temperature of the $i+1$ th compressor. The temperature of air and water after intercooling process is calculated based on the effectiveness of the heat exchangers. It is assumed that the temperature of Cold-Water Tank (CWT) is equal to the ambient temperature, and is constant over time. Additionally, it is assumed that the heat capacity of the working water is higher than that of air in all HEs, and this can be achieved by controlling the mass flow rate of the water. Using the assumptions mentioned above, it is possible to determine the temperature of the air leaving the HEs during charging and discharging processes:

$$T_{ahec,out}^i(t) = (1-\epsilon)T_{c,out}^i(t) + \epsilon T_{cwt} \tag{18}$$

$$T_{e,in}^j(t) = (1-\epsilon)T_{ahed,in}^j(t) + \epsilon T_{hwt} \tag{19}$$

$$T_{ahed,in}^j(t) = \begin{cases} T_{as}, & j = 1 \\ T_{e,out}^1(t) & j = 2 \end{cases} \tag{20}$$

where T_{cwt} and T_{hwt} are the temperature of the cold and hot water tanks, respectively; $T_{e,in}^j(t)$ and $T_{e,out}^j(t)$ are the inlet and outlet air temperatures of the jth expander at timestep t, respectively; and T_{as} is the temperature of the air storage. The following equation is used for calculating the

outlet air temperature of the expanders:

$$T^{j}_{e,out}(t) = T^{j}_{e,in}(t) \cdot \left(1 - \eta_e(t) \cdot \left(1 - \left(\beta^{j}_{e}(t)\right)^{\frac{1-\gamma_a}{\gamma_a}}\right)\right) \tag{21}$$

where $\beta^{j}_{e}(t)$ is the expansion ratio of the jth expander at timestep t; and $\eta_e(t)$ is the isentropic efficiency of expanders. Figs. 3.5 and 3.6 show the variation of isentropic efficiency and expansion ratios of the expanders with partial load ratio. The expansion work or the generated electricity per unit mass by each expander is calculated by Eq. (22) and (23) and follow by that, the mass flow rate in discharging process is calculated:

$$w^{1}_{e}(t) = c_p \cdot \left(T_{as}(t) - T^{1}_{e,out}(t)\right) \tag{22}$$

$$w^{2}_{e}(t) = c_p \cdot \left(T^{2}_{c,in}(t) - T^{2}_{e,out}\right) \tag{23}$$

$$\dot{m}_e(t) = \frac{P_d(t)}{w^{1}_{e}(t) + w^{2}_{e}(t)} \tag{24}$$

in which $\dot{m}_e(t)$ is the discharging mass flow rate at timestep t; $P_d(t)$ is the discharging power at timestep t; $w^{1}_{e}(t)$ and $w^{2}_{e}(t)$ are the generated work of expanders 1 and 2 per unit of mass at timestep t, respectively. If the system is operating at nominal load, the air leaving the last expander is cold enough to be used for cooling purposes. Equation (25) calculate the amount of available cooling/heating power of the tri generative AA-CAES. Whenever the air leaving the second expander has a lower temperature than the ambient air, it can be used for cooling purposes. If this is not the case, the AA-CAES system will only be able to provide heating and electricity.

$$C^{max}_{AA-CAES}(t) = \begin{cases} Cooling : \dot{m}_e(t) \cdot c_p \cdot \left(T_{amb} - T^{2}_{e,out}(t)\right) & if\ \left(T^{2}_{e,out}(t) \leq T_{amb}\right) \\ Heating : \dot{m}_e(t) \cdot c_p \cdot \left(T^{2}_{e,out}(t) - T_{amb}\right) & if\ \left(T^{2}_{e,out}(t) > T_{amb}\right) \end{cases} \tag{25}$$

The power to power cycle efficiency of the proposed AA-CAES system can be calculated as follow:

$$\text{CE} = \frac{P_d \times t_{dch}}{P_c \times t_{ch}} \tag{26}$$

in which t_{ch} and t_{dch} are the charging and discharging times, respectively. Moreover, the power to heat and power to cold cycle efficiencies of the proposed AA-CAES system can be calculated as follow:

$$\text{PH} = \frac{H^{max}_{AA-CAES} \times t_{ch}}{P_c \times t_{ch}} \tag{27}$$

$$\text{PC} = \frac{C^{max}_{AA-CAES} \times t_{dch}}{P_c \times t_{ch}} \tag{28}$$

After running simulations, the efficiencies of the proposed AA-CAES is compared with a reference for validations:

Table 3.3: Efficiency of the proposed AA-CAES system

Efficiency	Proposed AA-CAES	Reference [68]
CE	41%	44%
PH	38%	31%
PC	11.2%	8.5%

Due to the constant temperature of the AA-CAES air reservoir, its pressure in each time step can be calculated by Eq. (29). Moreover, the instantaneous State of Charge (SOC) for the air storage is defined as a function of the current pressure (p_{as}), minimum (p^{min}_{as}), and maximum (p^{max}_{as}) allowable pressures of the air storage 30 :

$$p_{as}(t+1) = p_{as}(t) + \left(\frac{(\dot{m}_c(t) - \dot{m}_e(t)) \cdot \Delta t}{V_{as}} \right) \cdot R \cdot T_{as} \tag{29}$$

$$SOC_{AA-CAES}(t) = \frac{p_{as}(t) - p^{min}_{as}}{p^{min}_{as} - p^{max}_{as}} \tag{30}$$

As a result of the off-design characteristics of the AA-CAES, Eqs. (14) and (24) are non-linear. The equations mentioned above need to be converted to a linear form in order to ensure global optimality by MILP solver. In this regard, mass flow rates in charging and discharging processes are calculated for various input/output powers. Fig. 3.7 illustrate the relationship between mass flow rates and charging/discharging powers. A linear regression model is fitted to each curve to achieve

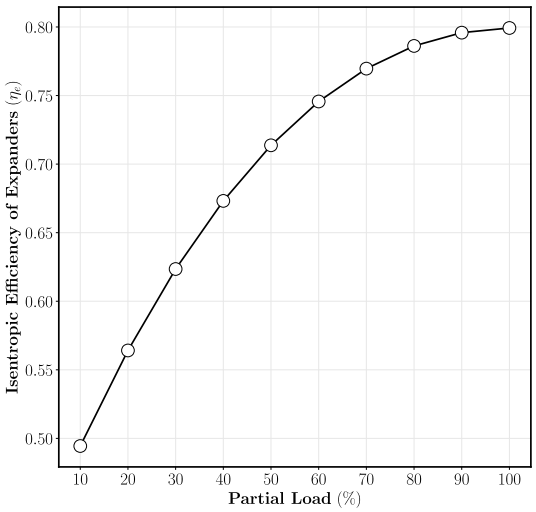


Figure 3.5: Variation of expanders' isentropic efficiency with partial load ratio [69]

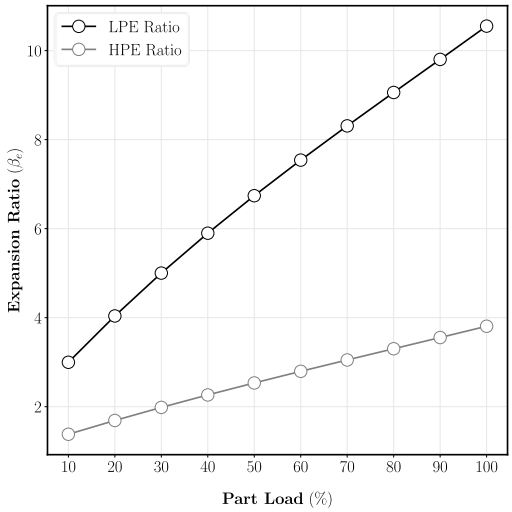


Figure 3.6: Variation of high – and low-pressure expanders' expansion ratio with partial load ratio [69]

a linear equation for mass flow rates. Thus, the mass flow rates in charging and discharging can be expressed as follows:

$$\dot{m}_{c/e} = a \cdot P_{c/e} + b \tag{31}$$

in which $\dot{m}_{c/e}$ is the mass flow rate of compressors or expanders, a and b are the coefficient and intercept of the regression model, respectively, and $P_{c/e}$ is the charging or discharging power.

3.1.2.4 Renewable Energy Sources Modeling

The proposed EH includes WTs and PV modules as RES. In addition to wind speed and direction, turbine position, turbine size, generator dynamic performance, and load distribution among parallel turbines, many factors can affect the amount of electricity generated by WTs [41]. Electrical output of the WTs are modeled using the following equation [70]:

$$P_{WT}(t) = \begin{cases} 0 & v(t) \leq v_{ci} \\ P_{WT}^{rated} \cdot \frac{v(t)^3 - {v_{ci}}^3}{{v_r}^3 - {v_{ci}}^3} & v_{ci} < v(t) \leq v_r \\ P_{WT}^{rated} & v_r \leq v(t) < v_{co} \\ 0 & v_{co} \leq v(t) \end{cases} \qquad \forall t \in H \tag{32}$$

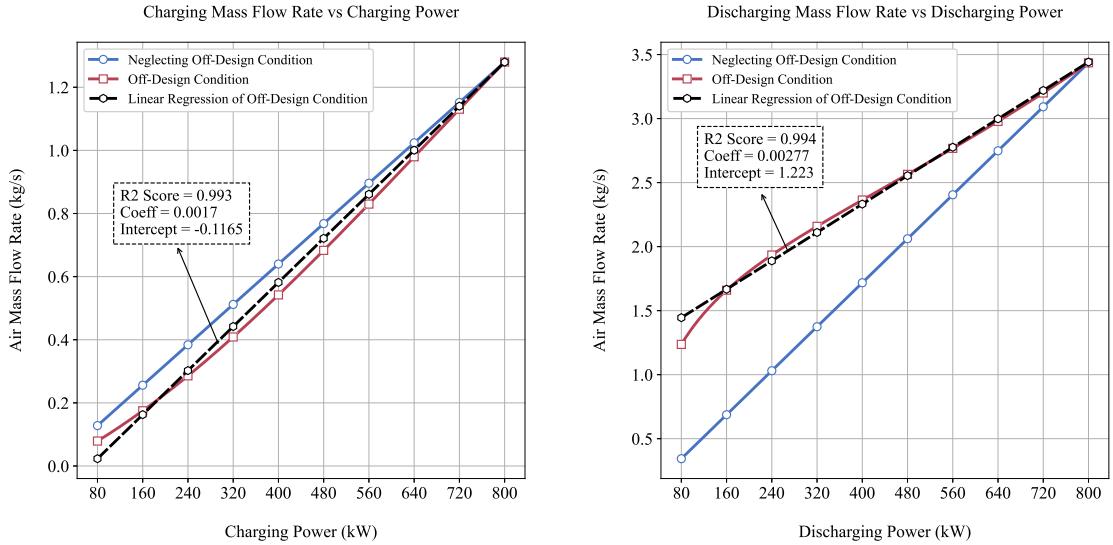


Figure 3.7: Variation of charging and discharging mass flow rate with power; the coefficient and intercept of fitted linear regression models.

where $P_{WT}(t)$ is the power generated by WT in time step t, which is a function of the wind velocity. $v(t)$, v_{ci}, v_{co}, and v_r are the wind velocity, cut in velocity, cut out velocity, and rated velocity ($\frac{m}{s}$), respectively. P_{WT}^{rated} is the rated power of WT if the wind velocity in equal to the rated velocity v_r. The cut-in speed for a wind turbine refers to the speed at which it begins to run. Once the wind speed reaches the wind turbine's rated speed, the turbine's power generation increases as well. As long as the wind speed is between the turbine rated speed and cut-out speed, the turbine generates its rated power. Whenever the wind speed exceeds the cut-out speed, the turbine stops working. Eq. (33) shows the output power of PV system as a function of the solar irradiation and ambient air temperature [71]:

$$P_{PV}(t) = \eta^{PV} \cdot S^{PV} \cdot G(t) \cdot \big(1 - 0.005(T_{amb}(t) - 25)\big) \qquad \forall t \in H \tag{33}$$

in which $P_{PV}(t)$, η^{PV}, S^{PV}, $G(t)$, and $T_{amb}(t)$ are the output power of the PV system, efficiency of the PV system, total surface area of PV system (m^2), total solar radiation ($\frac{kW}{m^2}$), and the ambient air temperature ($^{\circ}C$).

3.1.3 Objective Function

An Intelligent Energy Management System (IEMS) aims to maximize the economic benefits of the EH, which means that the output of each device at each timestep is efficiently arranged so that a maximum economic benefit will be achieved, considering operational constraints.

There are two elements to this objective, which is represented by Eq. (34) . The first term represents the cost of purchasing from/selling electricity to the grid system, and the second term is the cost of purchasing natural gas from the gas network. Compared with the daily operational costs, maintenance and emissions costs of the equipment are relatively small [56]. As a result, they are not considered in this study. The objective function can be written as follows:

$$min \sum_{t=1}^{H} \left(C_{pe}\ (t) +\ C_{pg}\ (t) \right) \tag{34}$$

$$C_{pe}(t) = \varepsilon_{e,p} \cdot P_{grid,p}(t) - \varepsilon_{e,s} \cdot P_{grid,s}(t) \qquad \forall t \in H \tag{35}$$

$$C_{pg}(t) = \varepsilon_{g}(t) \cdot \left(\frac{P_{MT}(t)}{\eta_{MT,p}} + \frac{H_{GB}(t)}{\eta_{GB}} \right) \cdot \Delta t \qquad \forall t \in H \tag{36}$$

in which $P_{grid,p}(t)$ and $P_{grid,s}(t)$ are the amount of electricity purchased from/sold to the grid system; $\varepsilon_{e,p}$ and $\varepsilon_{e,s}$ represent the purchasing and selling prices of electricity from/to grid system ($\frac{\$}{MWh}$), ε_g is the natural gas price ($\frac{\$}{MWh}$); and Δt is the time interval which is an hour.

3.1.4 Constraints

There are a number of constraints associated with the energy management system, including the energy balance equations and operational limits of the equipment. These constraints are presented in this section:

3.1.4.1 Energy Balance Constraints

Electrical network stability and safety depend on the balance between supply and demand of electricity, as shown in the following equation:

$$P_{grid,p}(t) + P_{PV}(t) + P_{WT}(t) + P_{AA-CAES,dch} + P_{MT}(t) = P_{load}(t) + P_{HP,h} + P_{HP,c} + P_{AA-CAES,ch} + P_{grid,s}(t) \tag{37}$$

where $P_{AA-CAES,ch}(t)$ and $P_{AA-CAES,dch}(t)$ are the charging/discharging power of the AA-CAES, respectively; and $P_{load}(t)$ is the electricity demand of the EH. Furthermore, there must be a balance in heating power at the heating network:

$$H_{GB}\,(t) + H_{MT}\,(t) + H_{AA-CAES}\,(t) + H_{HP}\,(t) \;= H_{load}\,(t) + H_{AC}\,(t) \tag{38}$$

in which $H_{Load}(t)$ is the heating demand of the EH. In the same way, the cooling balance is as follows:

$$C_{AC}\,(t) + C_{HP}\,(t) + C_{AA-CAES}\,(t) = C_{load}\,(t) \tag{39}$$

where $C_{load}(t)$ represents the cooling balance of the EH at timestep t.

3.1.4.2 Operational Constraints

There are operational constraints associated with each piece of equipment in the EH. The operational constraints of each equipment are discussed in this section.

- **AA-CAES Operational Constraints**

The following constraints should be considered for the operation of AA-CAES:

$$u^{ch}_{AA-CAES} \cdot P^{min}_{AA-CAES,ch} \leq P_{AA-CAES,ch}(t) \qquad \forall t \in H \tag{40}$$

$$P_{AA-CAES,ch}(t) \leq u^{ch}_{AA-CAES} \cdot P^{max}_{AA-CAES,ch} \qquad \forall t \in H \tag{41}$$

$$u^{dch}_{AA-CAES}(t) \cdot P^{min}_{AA-CAES,dch} \leq P_{AA-CAES,dch}(t) \qquad \forall t \in H \tag{42}$$

$$P_{AA-CAES,dch}(t) \leq u^{dch}_{AA-CAES}(t) \cdot P^{max}_{AA-CAES,dch} \qquad \forall t \in H \tag{43}$$

$$u^{ch}_{AA-CAES}(t) + u^{dch}_{AA-CAES}(t) \leq 1 \qquad \forall t \in H \tag{44}$$

$$SOC^{min}_{AA-CAES} \leq SOC_{AA-CAES}(t) \leq SOC^{max}_{AA-CAES} \qquad \forall t \in H \tag{45}$$

$$SOC_{AA-CAES}(0) = SOC_{AA-CAES}(H) \tag{46}$$

$$0 \leq C_{AA-CAES}(t) \leq C^{max}_{AA-CAES}(t) \qquad \forall t \in H \tag{47}$$

$$0 \leq H_{AA-CAES}(t) \leq H^{max}_{AA-CAES}(t) \qquad \forall t \in H \tag{48}$$

In which constraints (40, 41, 42, 43) indicate the lower and upper bounds of charging and discharging powers of AA-CAES. Constraint (44) controls the operational mode of AA-CAES (charge or discharge mode). Moreover, constraint (45) is for controlling the SOC of AA-CAES in its valid range and constraint (46) makes sure the amount of energy in the AA-CAES at the final timestep is equal to its initial value. Since utilizing the heating and cooling power of AA-CAES is optional, its value should be less than the maximum amount that can be utilizied in each timestep. Variables $H^{max}_{AA-CAES}(t)$ and $C^{max}_{AA-CAES}(t)$ depend on the amount of charging and discharging powers in each timestep (Eqs. (17) and (25)).

- **MT, GB, AC, and EHP Operational Constraints**

The electrical and thermal power produced by MT in EH should meet the following constraints:

$$u_{MT}(t) \cdot P^{min}_{MT} \leq P_{MT}(t) \leq u_{MT}(t) \cdot P^{max}_{MT} \qquad \forall t \in H \tag{49}$$

$u_{MT}(t)$ is a binary variable to limit $P_{MT}(t)$ in its valid range. For GB, the following constraint should be met:

$$H^{min}_{GB} \leq H_{GB}(t) \leq H^{max}_{GB} \qquad \forall t \in H \tag{50}$$

Similar to GB constraint, AC should also operate in its valid range:

$$H^{min}_{AC} \leq H_{AC}(t) \leq H^{max}_{AC} \qquad \forall t \in H \tag{51}$$

For the EHP, the operational constraints are as follow:

$$u_{HP,h}(t) \cdot H_{HP}^{min} \leq H_{HP}(t) \leq u_{HP,h}(t) \cdot H_{HP}^{max} \qquad \forall t \in H \tag{52}$$

$$u_{HP,c}(t) \cdot C_{HP}^{min} \leq C_{HP}(t) \leq u_{HP,c}(t) \cdot C_{HP}^{max} \qquad \forall t \in H \tag{53}$$

$$u_{HP,h}(t) + u_{HP,c}(t) \leq 1 \qquad \forall t \in H \tag{54}$$

in which $u_{HP,h}(t)$ and $u_{HP,c}(t)$ are binary variables that determine the status of EHP (heating or cooling mode). Constraints (52) and (53) are for the upper and lower limits of EHP. Since the EHP can only operate in heating or cooling modes, constraint (54) is considered.

- **Grid Operational Constraints**

The amount of power exchanged with the grid system should be within a valid range. Moreover, selling and purchasing energy can not happen at the same time. Therefore, the following constraints need to be met:

$$u_{grid,p}(t) \cdot P_{grid}^{min} \leq P_{grid,p}(t) \leq u_{grid,p}(t) \cdot P_{grid}^{max} \qquad \forall t \in H \tag{55}$$

$$u_{grid,s}(t) \cdot P_{grid}^{min} \leq P_{grid,s}(t) \leq u_{grid,s}(t) \cdot P_{grid}^{max} \qquad \forall t \in H \tag{56}$$

$$u_{grid,p}(t) + u_{grid,s}(t) \leq 1 \qquad \forall t \in H \tag{57}$$

3.2 Constrained Markov Decision Process

Optimal dispatch of EH explained in Section 3.1 is a sequential decision-making problem that can be formulated as a finite CMDP incorporating the operational constraints for each component in Section 3.1.4. Here, the objective is to find the optimal dispatch policy for the EH components in order to minimize daily operational costs and meet the constraints at the same time. In an MDP, the agent is interacting with the stochastic environment. This process is defined as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r >$, in which $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the state transition probability, and $r(s_t, a_t)$ is the immediate scalar reward. The main goal is to find the policy

$\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ that leads to the maximum discounted future reward $R(t) = \sum_{i=t}^{H} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0, 1]$ is the discount factor that determines the importance of the immediate reward relative to the expected cumulative reward in the future. A version of MDP, CMDP adds constraints on long-term discounted safety costs to the process. Each safety cost function is defined as $C_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, which is a mapping from transitions to safety costs. Moreover, the future discounted safety cost can be formulated as $C_i(t) = \sum_{i=t}^{H} \gamma^{i-t} c_i(s_i, a_i)$ where $c(s_t, a_t)$ is the immediate scalar cost. CMDP aims to find the policy π that maximizes discounted future reward while satisfying constraints $C_i(\pi) \leq d_i \quad \forall i \in [m]$, where d_i is the constraint tolerance and m is the constraints set.

$$\pi^* = \arg\max_{\pi \in \prod_\theta} R(\pi) \quad s.t. \quad C_i(\pi) \leq d_i \quad \forall i \in [m] \tag{58}$$

As it is illustrated in Fig. 3.8, in each timestep, the agent observes the current state of the environment s_t and chooses the action a_t based on the learned policy $\pi(a|s)$. The probability of the environment going to the next state s_{t+1} depends on the chosen action a_t and dynamism in the environment. As a result, the agent receives an immediate reward r_t, some immediate safety costs $c_t(i)$, and information about the next state s_{t+1} which enables continuation of the decision-making process over the time horizon. This section describes the CMDP's main elements:

3.2.1 State

In order to allow the agent to choose a proper action, the environment should provide enough information to the agent. To be more specific, this information represents the current state of the environment which consists of load demands $P_{load}(t)$, $H_{load}(t)$, and $C_{load}(t)$, power generated by PV panels $P_{PV}(t)$, power generated by wind turbines $P_{WT}(t)$, SOC of the AA-CAES $SOC_{AA-CAES}(t)$, electricity purchase and sell real-time price $\varepsilon_{e,p}(t)$ and $\varepsilon_{e,s}(t)$, natural gas real-time price $\varepsilon_g(t)$, ambient air temperature $T_{amb}(t)$, and the current dispatch timestep t. In the case study, due to the correlation between the selling and purchasing prices of electricity and the constant gas price, only the purchasing price of electricity is considered as a state element. Hence, the state

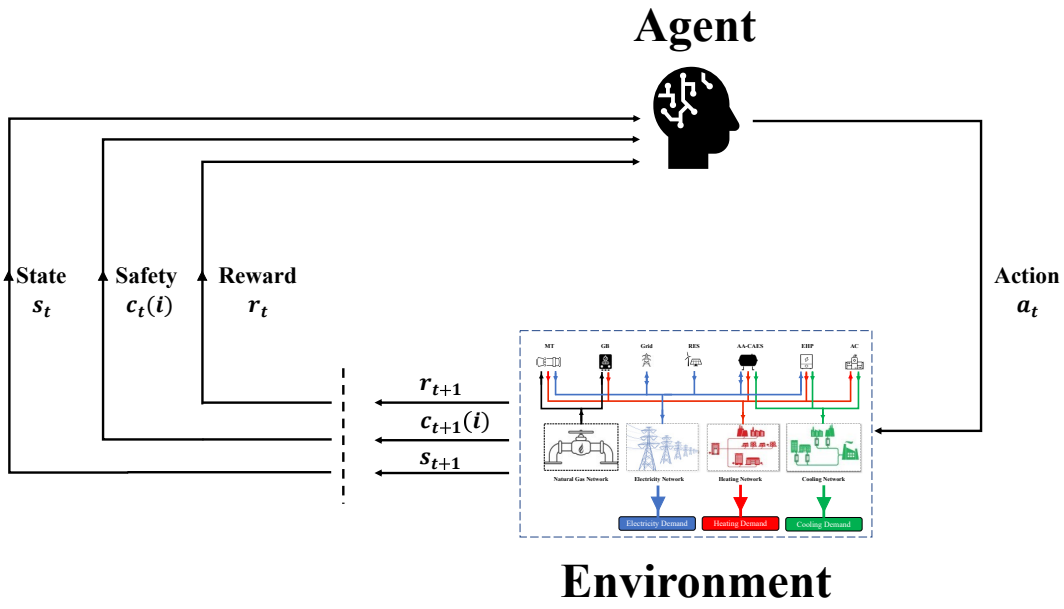


Figure 3.8: Reinforcement learning framework in CMDP

can be described as a 9-dimentional vector:

$$s_t = \left[P_{load}(t), H_{load}(t), C_{load}(t), P_{PV}(t), P_{WT}(t), SOC_{AA-CAES}(t), \varepsilon_{e,p}(t), T_{amb}(t), t\right] \quad (59)$$

3.2.2 Action

The agent performs an action based on the learned policy after observing the state of the environment. The amount of energy generated or consumed by all dispatchable equipment is considered an action to ensure the feasibility of actions $\big($Constraints (40) to (54), except (45) and (46)$\big)$. Therefore, the agent's action is consisted of the charging or discharging power of AA-CAES ($a_t^{AA-CAES}$), the electricity generated by MT (a_t^{MT}), the electricity consumed by heat pump (a_t^{HP}), the heating generated by GB (a_t^{GB}), and the heating consumed by the absorption chiller (a_t^{AC}). Moreover, $a_t^{H/C\ AA-CAES}$ determines the percentage of maximum amount of heating or cooling power generated by AA-CAES that can be utilized for dispatch (Constraints (47) and (48)). The AA-CAES and EHP actions ($a_t^{AA-CAES}, a_t^{HP}$) are scaled between [-1,1] which is the percentage of maximum charging or discharging power of the AA-CAES or the maximum power consumed by the EHP (if positive: discharging AA-CAES or EHP generates heating energy, if negative: charging

AA-CAES or EHP generates cooling energy). On the other hand, the other elements of the agent's action ($a_t^{H/C\ AA-CAES}, a_t^{MT}, a_t^{GB}, a_t^{AC}$) are scaled between [0,1] which are the percentage of their maximum capacity. Finally, the action can be described as a 6-dimentional vector:

$$a_t = \left[a_t^{AA-CAES}, a_t^{H/C\ AA-CAES}, a_t^{MT}, a_t^{GB}, a_t^{HP}, a_t^{AC}\right] \tag{60}$$

3.2.3 State transition

A number of factors contribute to the state transition in the MDP of EH. As a result, the state transition from s_t to s_{t+1} can be described as follows:

$$s_{t+1} = f(s_t, a_t, \omega_t) \tag{61}$$

in which ω_t represents the environment's uncertainties (electricity and heating and cooling demands, wind turbines and PV panels power generation). Therefore, the state transition is not only a function of the current state and action, but also a function of the existing uncertainties. However, the state transition for the SOC of AA-CAES can be determined by the charging/discharging power of AA-CAES ($a_t^{AA-CAES}$), while for $P_{load}(t)$, $H_{load}(t)$, $C_{load}(t)$, $P_{PV}(t)$, $P_{WT}(t)$, and $T_{amb}(t)$ the state transition is affected by the community's energy consumption behavior, weather conditions such as wind velocity, and solar irradiation. Thus, it is difficult to develop an accurate model for the probability distribution of randomness and uncertainty in these elements. The agent, however, can implicitly learn the transition probability between states since it observes the data over a variety of days during the training process.

3.2.4 Reward

The main goal of the IEMS agent is to minimize the operational cost as described in Section 3.1.3, while considering the operational constraints of the environment. Section 3.3.3 describes the method for addressing constraints. To minimize the operational cost of the EH, the following reward function is introduced:

$$r_t = -\left[C_{pe}(t) + C_{pg}(t)\right] \tag{62}$$

It is the agent's primary objective to maximize the cumulative reward over an episode in an MDP. Hence, in order to convert our objective function minimization into an MDP reward function, a negative sign must be added to it. To put it another way, the objective function will be minimized by maximizing the reward function.

3.2.5 Safety cost

As mentioned in Section 3.2.2, the action is consisted of the generated or consumed energy of all dispatchable equipment. Thus, there is no guarantee that the heating and cooling energy balance constraints (38) and (39) will be met. Additionally, since the exchanged power with the upstream grid system is derived from the electricity balance (37) and the power of other equipment, the electricity balance can only be satisfied if the exchanged power with the grid system does not exceed its valid range (constraints (55) and (56)). For the mentioned constraints to be considered, a penalty function is necessary. Equipment that has its input/output power defined by the agent's action does not violate the constraint. However, there is a possibility that the following constraints may not be satisfied: heating and cooling balance, power exchanged with the grid system, and the SOC of AA-CAES (constraint (45)). To deal with each constraint, the following penalty functions are considered:

$$c_1(t) = \left| H_{GB}(t) + H_{MT}(t) + H_{AA-CAES}(t) + H_{HP}(t) + H_{AC}(t) - H_{load}(t) \right| \tag{63}$$

$$c_2(t) = \left| C_{AC}(t) + C_{HP}(t) + C_{AA-CAES}(t) - C_{load}(t) \right| \tag{64}$$

$$c_3(t) = \left| P_{grid}(t) + P_{grid}^{max} \right| \cdot u_{grid}^{+} + \left| P_{grid}(t) + P_{grid}^{min} \right| \cdot u_{grid}^{-} \tag{65}$$

$$c_4(t) = \left| SOC_{AA-CAES}(t) - SOC_{AA-CAES}^{max} \right| \cdot u_{SOC}^{+} \cdot \vartheta_{SOC} + \left| SOC_{AA-CAES}(t) - SOC_{AA-CAES}^{min} \right| \cdot u_{SOC}^{-} \cdot \vartheta_{SOC} \tag{66}$$

Eqs. (63) and (64) are the violation from heating and cooling balance constraints (38) and (39), eq. (65) is the violation from the minimum/maximum allowable power exchange with the grid (55) and (56), and eq. (67) is the violation from minimum/maximum SOC of AA-CAES (45). The variables u_{grid}^{+} and u_{grid}^{-} are binary variables that indicate if there is any violation for constraint (55) and (56). More specifically, $u_{grid}^{+} = 1$ if $P_{grid}(t) > P_{grid}^{max}$ and $u_{grid}^{-} = 1$ if $P_{grid}(t) < P_{grid}^{min}$.

The same for the SOC constraint (67), $u^{+}_{SOC} = 1$ if $SOC_{AA-CAES}(t) > SOC^{max}_{AA-CAES}$ and $u^{-}_{SOC} = 1$ if $SOC_{AA-CAES}(t) < SOC^{min}_{AA-CAES}$. Moreover, ϑ_{SOC} is a penalty coefficient to scale the SOC violation to the same scale of other penalty terms. To consider the constraint (46), the following term is added to the reward function:

$$Pen_{final_{SOC}}(t) = \begin{cases} 0 & if \quad t \neq H \\ \vartheta_{SOC} \cdot \left| SOC_{AA-CAES}(H) - SOC_{AA-CAES}(0) \right| & if \quad t = H \end{cases} \qquad \forall t \in H \tag{67}$$

3.3 Deep Reinforcement Learning Algorithms

The purpose of this section will be to present novel DRL algorithms, followed by a discussion of our proposed safe reinforcement learning approach at the end of this section.

3.3.1 Q-Learning

Action-value functions are one of the most commonly used functions in reinforcement learning algorithms. As a result of taking action a_t at state s_t following policy π, the action-value function is described as the expected return [72]:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t, a_t \right] \tag{68}$$

It is well known that many reinforcement learning approaches use the Bellman equation as a recursive formula:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[r(s_t, a_t) + \gamma \, \mathbb{E}_{\pi}\left[Q^{\pi}(s_{t+1}, a_{t+1}) \right] \right] \tag{69}$$

In this case, the target policy can be described as a function $\mu \ : \mathcal{S} \leftarrow \mathcal{A}$, avoiding the inner expectation if the target policy is deterministic:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{\pi}\left[r(s_t, a_t) + \gamma \, Q^{\mu}\Big(s_{t+1}, \mu(s_{t+1}) \Big) \right] \tag{70}$$

In the RL algorithms, the action value function can be calculated iteratively based on the Bellman optimality:

$$Q_{i+1}(s_t, a_t) = Q_i(s_t, a_t) + \alpha \cdot \left(r(s_t, a_t) + \gamma \cdot \max_a Q(s_{t+1}, a) - Q_i(s_t, a_t) \right) \tag{71}$$

where $Q_{i+1}(s_t, a_t)$ and $Q_i(s_t, a_t)$ are the updated and current Q-values, $\alpha \in (0, 1]$ is the learning rate, and $\max_a Q(s_{t+1}, a)$ is the estimated optimal future value. This means that as the agent interacts with the environment, it gradually learns to improve its policy as the number of iterations increases. In conventional RL algorithms like Q-learning, the optimal Q-value is calculated by a Q-table which has all possible state-action pairs [73]. When there is a large number of possible state-action pairs, then it would be impossible to create a Q-table. Discretizing the state and actions can be a solution but will surely lead to sub-optimal policy. To address this problem, deep learning has been utilized to approximate $Q(s, a)$ by Artificial Neural Network (ANN) [50]. The data for deep learning must be independently and identically distributed (I.I.D.) in order to optimize the learning process. Due to the high temporal correlation of the transition tuples, the experiences themselves are not efficient for learning. As a solution to this problem, transitions are stored in an experience replay memory and are then randomly selected for updating the neural networks [74].

3.3.2 Deep Deterministic Policy Gradient (DDPG)

DDPG algorithm simultaneously learns a Q-function and a policy [72]. To update the Q-function and use it to learn the policy, DDPG uses the off-policy data. In DDPG, there is one network for approximating the actor function (θ^{μ}), and one network for critic function (ϕ^{Q}). Each one of these networks has its own target network (actor target $\theta^{\mu'}$ and critic target $\phi^{Q'}$)

It is the responsibility of the online critic network to provide feedback to the actor network regarding the quality of the state-action pair, in terms of the discounted future reward. The term "online" refers to the main critic network that its inputs are the current state and action pair. To update the

online critic network, the following loss function is used:

$$\mathcal{L}\left(\phi^{Q}\right)=\frac{1}{N}\sum_{j}^{N}\left(Q^{target}-Q\left(s_j,a_j|\phi^{Q}\right)\right)^2 \tag{72}$$

where $Q^{target}=r_j+\gamma\cdot Q'\left(s_{j+1},a_{j+1}=\mu'_{\theta}\left(s_{j+1}\right)|\phi^{Q'}\right)$ is the target Q-value and is equal to the current immediate reward, plus the discount factor times the output of the critic target network. The parameters of online and target critic networks are updated by:

$$\phi^{Q}\leftarrow\phi^{Q}-\alpha^{\phi}\nabla_{\phi^{Q}}\left(\mathcal{L}\left(\phi^{Q}\right)\right) \tag{73a}$$

$$\phi^{Q'}\leftarrow\tau\phi^{Q}+(1-\tau)\phi^{Q'} \tag{73b}$$

in which τ is the Poliak averaging ratio (soft update rate) and is a small value, and α^{ϕ} is the critic network's learning rate. The information from the critic network can be used to update the parameters of the actor (policy) and its target network:

$$\mathcal{L}\left(\theta^{\mu}\right)=\frac{1}{N}\sum_{j}^{N}\left(Q\left(s_j,a_j=\mu_{\theta}(s_{j+1})|\phi^{Q}\right)\right) \tag{74a}$$

$$\theta^{\mu}\leftarrow\theta^{\mu}-\alpha^{\theta}\nabla_{\theta^{\mu}}\left(\mathcal{L}\left(\theta^{\mu}\right)\right) \tag{74b}$$

$$\theta^{\mu'}\leftarrow\tau\theta^{\mu}+(1-\tau)\theta^{\mu'} \tag{74c}$$

A major challenge of learning in continuous action spaces is exploration. To deal with this problem, a mean-zero Gaussian exploration noise is added to the actor's output to increase the exploration of the agent for better action-values estimations. The learning process of DDPG agent is fully explained in Algorithm 1.

3.3.3 Primal-Dual Deep Deterministic Policy Gradient (PD-DDPG)

A traditional RL algorithm, such as DQN and DDPG, which are based on MDP, introduces environment constraints by adding penalties to the reward function [75]. There is a problem with this approach in terms of the priority and weight of the primary objective function and the penalty

Algorithm 1 Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(s, a|\phi^Q)$ and actor $\mu(s|\theta^\mu)$ with weights ϕ^Q and θ^μ.
Initialize target network Q' and θ' with weights $\phi^{Q'} \leftarrow \phi^Q,\ \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$ and dual variables λ_i
for episode = 1 to M **do**
    Receive initial observation state s_1
    for t = 1 to H **do**
        Select action $a_t = \text{clip}(\mu(s_t|\theta^\mu) + \epsilon, a_{Low}, a_{High})$ where $\epsilon \sim \mathcal{N}(\mu = 0, \sigma)$
        Execute a_t in the environment
        Observe next state s_{t+1}, reward r_t, and done signal d to indicate whether s_{t+1} is terminal
        Store $(s_t, a_t, r_t, s_{t+1}, d)$ in replay buffer $\mathcal{R}$
        If s_{t+1} is terminal, reset environment state.
        Sample a random minibatch of N transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{R}$
        Calculate Q-target value $Q_j^{target} = r_j + \gamma \cdot Q'\Big(s_{j+1}, a_{j+1} = \mu'_\theta\big(s_{j+1}\big)\big|\phi^{Q'}\Big)$
        Update critic by minimizing the loss: $\mathcal{L}\big(\phi^Q\big) = \dfrac{1}{N}\displaystyle\sum_j^N \Big(Q_j^{target} - Q\big(s_j, a_j\big|\phi^Q\big)\Big)^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_j^N \nabla_{a_j} Q\big(s_j, a_j = \mu_\theta(s_{j+1})|\phi^Q\big)\nabla_{\theta^\mu}\mu(s|\theta^\mu)$$

        Update the target networks:

$$\phi^{Q'} \leftarrow \tau\phi^Q + (1-\tau)\phi^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

terms. Considering that the agent is not aware of the constraints as independent functions, it will be guided toward the policy that maximizes the heavier term in the reward function. To consider the CMDP in DDPG algorithm, Lagrangian Multipliers are added to (58) formulation to penalize the constraint violations [66]. Therefore, the CMDP problem is reformulated as follow:

$$(\pi^*, \lambda^*) = \arg \min_{\lambda \geq 0} \max_{\pi \in \prod_\theta} \mathcal{L}(\pi, \lambda) \tag{75a}$$

$$\mathcal{L}(\pi, \lambda) = R(\pi) - \sum_i^m \lambda_i (C_i(\pi) - d_i) \qquad \forall i \in [m] \tag{75b}$$

Updating the policy network parameters and the dual variables (λ_i) is done iteratively by the following equations:

$$\theta_{k+1}^{\mu} = \theta_k^{\mu} + \alpha_k \nabla_\theta \Big(\mathcal{L}\big(\pi(\theta)\big), \lambda_i^k \Big) \Big|_{\theta=\theta_k} \qquad \forall i \in [m] \tag{76a}$$

$$\lambda_i^{k+1} = \Big[\lambda_i^k + \beta_k \big(C_i(\pi) - d_i \big) \Big]^+ \qquad \forall i \in [m] \tag{76b}$$

in which β_k is the dual variables' learning rate and the operator $[x]^+ = \max(0, x)$. This operator is used to make sure that dual variables always have non-negative values $(\lambda_i \geq 0)$.

In PD-DDPG algorithm, the online cost networks (ψ^{C_i}) exist to consider the constraint violations (cost functions). Each of these networks has its own target network $(\psi^{C_i'})$. The functionality of the cost networks is exactly the same as that of the critic networks; however, the critic networks relate to the main objective (reward function), while the cost networks relate to the constraints. In other words, both critic and cost networks are used to approximate the action value functions: critic for Q-value and cost for C-value. Therefore, updating the cost networks' parameters has the same procedure as the critic network:

$$\mathcal{L}\big(\psi^{C_i}\big)= \frac{1}{N}\sum_{j}^{N}\Big(C_i^{target} - C_i\big(s_j, a_j\big|\psi^{C_i}\big)\Big)^2 \qquad \forall i \in [m] \tag{77a}$$

$$C_i^{target} = c_{i,j} + \gamma \cdot C_i'\Big(s_{j+1}, a_{j+1} = \mu_\theta'\big(s_{j+1}\big)\big|\psi^{C_i'}\Big) \qquad \forall i \in [m] \tag{77b}$$

$$\psi^{C_i} \leftarrow \psi^{C_i} - \alpha^{\psi}\nabla_{\psi^{C_i}}\Big(\mathcal{L}\big(\psi^{C_i}\big)\Big) \qquad \forall i \in [m] \tag{77c}$$

$$\psi^{C_i'} \leftarrow \tau\psi^{C_i} + (1-\tau)\psi^{C_i'} \qquad \forall i \in [m] \tag{77d}$$

Actor network parameters should be updated in such a manner as to increase reward and reduce constraint violation. This can be achieved by considering the output of cost networks in the loss function of the actor network. Accordingly, the loss function for actor network can be written as follows and its parameters are updated using gradient ascending (74b):

$$\mathcal{L}\big(\theta^{\mu}\big)= \frac{1}{N}\sum_{j}^{N}\Bigg(Q\big(s_j, a_j\big|\phi^{Q}\big)\Big|_{a_j=\mu_\theta(s_{j+1})} - \sum_{i=1}^{m}\lambda_i C_i\big(s_j, a_j\big|\psi^{C_i}\big)\Big|_{a_j=\mu_\theta(s_{j+1})}\Bigg) \tag{78}$$

The learning process of the PD-DDPG agent is explained in Algorithm 2.

3.3.4 Proposed Deep Reinforcement Learning Algorithm

In this section, our proposed algorithm for the optimal dispatch problem in 3.2 is explained in details. The purpose of this section is to introduce a safe DRL algorithm inspired by [60, 76], referred to as Primal-Dual Deep Deterministic Policy Gradient from Demonstrations (PD-DDPGfD). The proposed algorithm is based on PD-DDPG, and IL, which uses expert demonstrations. To assist the agent with initial explorations, IL is used for pre-training the networks because the objective function is non-convex and the search space is large. The learning process of the proposed algorithm is explained in the following sub-sections.

3.3.4.1 Pre-training with Imitation Learning

In PD-DDPG, the agent performs random actions in order to gather information about the environment. Nevertheless, the initial exploration and the initialization of neural networks parameters

Algorithm 2 Primal-Dual Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(s,a|\phi^Q)$, cost networks $C_i(s,a|\psi^{C_i})$actor $\mu(s|\theta^\mu)$ with weights ϕ^Q, $\psi^{C_i}$and θ^μ.
Initialize target network Q',C_i' and μ' with weights $\phi^{Q'} \leftarrow \phi^Q, \psi^{C_i'} \leftarrow \psi^{C_i}, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for episode = 1 to M **do**
    Receive initial observation state s_1
    for t = 1 to H **do**
        Select action $a^t = \text{clip}(\mu(s_t|\theta^\mu) + \epsilon, a_{Low}, a_{High})$ where $\epsilon \sim \mathcal{N}(\mu = 0, \sigma)$
        Execute a^t in the environment
        Observe next state s^{t+1}, reward r^t, costs c_i^t, and done signal d to indicate whether s^{t+1} is terminal
        Store $(s^t, a^t, r^t, c_i^t, s^{t+1}, d)$ in replay buffer $\mathcal{R}$ ▷ $\forall i \in [m]$
        If s^{t+1} is terminal, reset environment state.
        Sample a random minibatch of N transitions $(s_j, a_j, r_j, c_{i,j}, s_{j+1}) \quad \forall i \in [m]$ from $\mathcal{R}$
        Calculate Q-target value $Q_j^{target} = r_j + \gamma \cdot Q'\Big(s_{j+1}, a_{j+1} = \mu'_\theta(s_{i+1})\Big|\phi^{Q'}\Big)$
        Calculate C-target values $C_{i,j}^{target} = c_{i,j} + \gamma \cdot C_i'\Big(s_{j+1}, a_{j+1} = \mu'_\theta(s_{j+1})\Big|\psi^{C_i'}\Big)$ ▷ $\forall i \in [m]$
        Update critic network by minimizing the loss: $\mathcal{L}\big(\phi^Q\big) = \dfrac{1}{N}\sum_j^N \Big(Q_j^{target} - Q\big(s_j, a_j\big|\phi^Q\big)\Big)^2$
        Update cost networks by minimizing the loss:

$$\mathcal{L}\big(\psi^{C_i}\big) = \frac{1}{N}\sum_j^N \Big(C_{i,j}^{target} - C_i\big(s_j, a_j\big|\psi^{C_i}\big)\Big)^2 \qquad \triangleright\ \forall i \in [m]$$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_j^N \nabla_{a_j}\left(Q\Big(s_j, a_j\Big|\phi^Q\Big) - \sum_{i=1}^m \lambda_i C_i\Big(s_j, a_j\Big|\psi^{C_i}\Big)\right)\Bigg|_{a_j = \mu^\theta(s_j)}$$

        Update dual variables using the sampled dual gradient:

$$\nabla_{\lambda_i}\mathcal{L}(\theta^\mu, \lambda_i) = \frac{1}{N}\sum_j^N \big(C_i(s_j, a_j) - d_i\big)\Big|_{a_j = \mu^\theta(s_j)} \qquad \triangleright\ \forall i \in [m]$$

$$\lambda_i^{k+1} = \Big[\lambda_i^k + \beta_k \nabla_{\lambda_i}\mathcal{L}(\theta^\mu, \lambda_i)\Big]^+ \qquad \triangleright\ \forall i \in [m]$$

        Update the target networks:

$$\begin{aligned} \phi^{Q'} &\leftarrow \tau\phi^Q + (1-\tau)\phi^{Q'} \\ \psi^{C_i'} &\leftarrow \tau\psi^{C_i} + (1-\tau)\psi^{C_i'} \qquad \triangleright\ \forall i \in [m] \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned}$$

    end for
end for

may affect the learning process, resulting in suboptimal solutions with high operating costs and low constraint violations. Therefore, it can be very difficult for the agent to come up with an acceptable dispatch policy. As a solution to this problem, imitation learning is used to pretrain the actor, critic, and cost networks. The objective of imitation learning techniques is to mimic the behavior of an expert when it comes to a particular task. In order to learn a mapping between observations and actions, the agent uses expert demonstrations that are the near optimal state-action pairs. For this study, the "Expert Demonstration" is referred to as the optimal state-action pairs obtained from the MILP solver [77] for problem 3.1. It is important to note that even actions performed by an experienced energy hub operator or solutions resulting from a preliminary optimization model could be considered expert demonstrations. The benchmark for this study is the optimal solutions obtained from an MILP solver. The process of IL can be viewed as a regression problem. Before starting the pre-training process, a dataset consisted of expert trajectories from MILP solver is created that has all the optimal state-action pairs $\mathcal{D} = [d_1, d_2, ..., d_M]$ where $d_m = \left[(s_0^m, a_0^m), (s_1^m, a_1^m), ..., (s_H^m, a_H^m)\right]$ is a demonstration for each day in the training set with the optimization horizon of (H). In the next step, a neural network with the same architecture as the actor network is created, which receives the states as input and outputs the actions. To pre-train the actor network, supervised learning is conducted to update the actor network parameters in order to minimize the following loss function:

$$\mathcal{L}^{IL}\left(\theta^{\mu}\right) = \frac{1}{M}\sum_{m=1}^{M}\Big(a_m - \theta_{\mu}(s_m|\theta^{\mu})\Big)^2 \tag{79a}$$

$$\theta^{\mu} \leftarrow \theta^{\mu} - \alpha_{\mu,IL}\nabla_{\theta^{\mu}}\mathcal{L}^{IL}\left(\theta^{\mu}\right) \tag{79b}$$

where $a_m$is the optimal action corresponding to state s_m, $\theta_{\mu}(s_m|\theta^{\mu})$ is the output of actor network, and $\alpha_{\mu,IL}$ is the IL learning rate. The critic and cost networks are function approximators for the action-value functions (Q and C) and they will output inaccurate values if not pre-trained. Consequently, they can have an adverse impact on the learning process of the actor netwrok. Therefore, the transition tuples from expert demonstrations are stored in the replay buffer memory and used to pre-train the critic and cost networks. The pre-training process is summarized in Algorithm 3.

Algorithm 3 Pre-training PD-DDPG networks

Store expert demonstrations $\mathcal{D}$ in replay buffer $\mathcal{R}$.
Randomly initialize critic network $Q(s, a|\phi^Q)$, cost networks $C_i(s, a|\psi^{C_i})$actor $\mu(s|\theta^\mu)$ with weights ϕ^Q, $\psi^{C_i}$and θ^μ.
Initialize target network Q',C_i' and μ' with weights $\phi^{Q'} \leftarrow \phi^Q, \psi^{C_i'} \leftarrow \psi^{C_i}, \theta^{\mu'} \leftarrow \theta^\mu$
for epochs = 1 to S **do**
    for iterations = 1 to I **do**
        Sample a batch of (s_m, a_m) from $\mathcal{R}$.
        Update the actor network parameters θ^μ by (79).
    end for
end for
$\theta^{\mu'} \leftarrow \theta^\mu$
for days in training-set **do**
    for t = 0 to H **do**
        Retrieve (s^t, a^t) from $\mathcal{R}$
        Execute the action a^t, and append reward, costs, and next state $r^t, c_i^t, s_{(t+1)}$ to (s_t, a_t)▷ $\forall i \in [m]$
    end for
end for
for episode =1 to P **do**
    for iteration =1 to H **do**
        Randomly sample a batch of transitions from $\mathcal{R}$.
        Update the online critic network parameters ϕ^Q by (72, 73a) and online cost network parameters ψ^{C_i} by (77a, 77c) ▷ $\forall i \in [m]$
        if iteration % policy_delay **then**
            Update target parameters by (73b, 77d) ▷ $\forall i \in [m]$
        end if
    end for
end for

3.3.4.2 Online Learning Process

After pre-training the networks, the agent begins optimizing the learned policy in terms of constraint violation and operational costs through interaction with the environment. Considering that the agent is pre-trained, exploration noise is not required to influence its choice of actions. Additionally, expert demonstrations are again used to speed up the online training process and to accommodate the following two modifications:

(1) Initially, the replay buffer contains demonstration data, and data generated by the control agent is gradually added to it as it interacts with the environment, but it should not overwrite the demonstration data. When the replay buffer reaches its maximum capacity, new transitions will overwrite earlier self-generated data.

(2) Due to the high probability that the sampled data from the replay buffer include optimal actions, an additional term is added to the actor's loss function to assist the agent in making more optimal decisions. In this case, the Mean Squared Error (MSE) between the sampled actions and the network's output should be considered [78]. By adding this term to (78), the loss term for the actor network would be:

$$\mathcal{L}(\theta^{\mu}) = \frac{1}{N}\sum_{j}^{N}\left(Q\left(s_j, a\middle|\phi^{Q}\right) - \left(a_j - a\right)^2 - \sum_{i=1}^{m}\lambda_i C_i\left(s_j, a\middle|\psi^{C_i}\right)\right)\Bigg|_{a=\mu_\theta(s_j)} \tag{80}$$

where a_j is the sampled action from replay buffer $\mathcal{R}$, and $a = \mu_\theta(s_j)$ is the actor network's output. The added term directs the agent to optimal actions during the online learning process.The online training process is summarized in Algorithm 4

Fig. 3.9 represents the whole training process of the proposed approach. It is worth mentioning that the demonstrations used for pre-training and online learning processes are the optimal solutions from MILP solver for the training set only. Moreover, since there is no model to help the agent in approximating the future states, the developed approach is model-free.

Algorithm 4 Online Training Process of PD-DDPGfD

for episode =1 to K **do**
  Pick a random day from the training set.
  for t=0 to H **do**
    Observe state s^t and execute action $a^t = \mu_\theta(s_j)$
    Collect the reward r^t and costs c_i^t from the environment and proceed to state s^{t+1} ▷ $\forall i \in [m]$
    Store transition $(s^t, a^t, r^t, c_i^t, s^{t+1})$ to replay memory B, overwrite the old self-generated transitions if the memory is full.
    if $t = H$, reset the environment.
  end for
  for iteration 0 = H **do**
    Randomly sample a batch of transitions from $\mathcal{R}$
    Update the online critic network parameters ϕ^Q by (72, 73a) and online cost network parameters ψ^{C_i} by (77a, 77c) ▷ $\forall i \in [m]$
    if iteration % policy_delay **then**
      Update actor network parameters by (80) ▷ $\forall i \in [m]$
      Update target parameters by (73b, 77d, and 74c) ▷ $\forall i \in [m]$
      Update dual variables using the sampled dual gradient:

$$\nabla_{\lambda_i} \mathcal{L}(\theta^\mu, \lambda_i) = \frac{1}{N} \sum_j^N \big(C_i(s_j, a_j) - d_i\big)\Big|_{a_j = \mu^\theta(s_j)} \quad \triangleright\ \forall i \in [m]$$

$$\lambda_i^{k+1} = \Big[\lambda_i^k + \beta_k \nabla_{\lambda_i} \mathcal{L}(\theta^\mu, \lambda_i)\Big]^+ \quad \triangleright\ \forall i \in [m]$$

    end if
  end for
end for

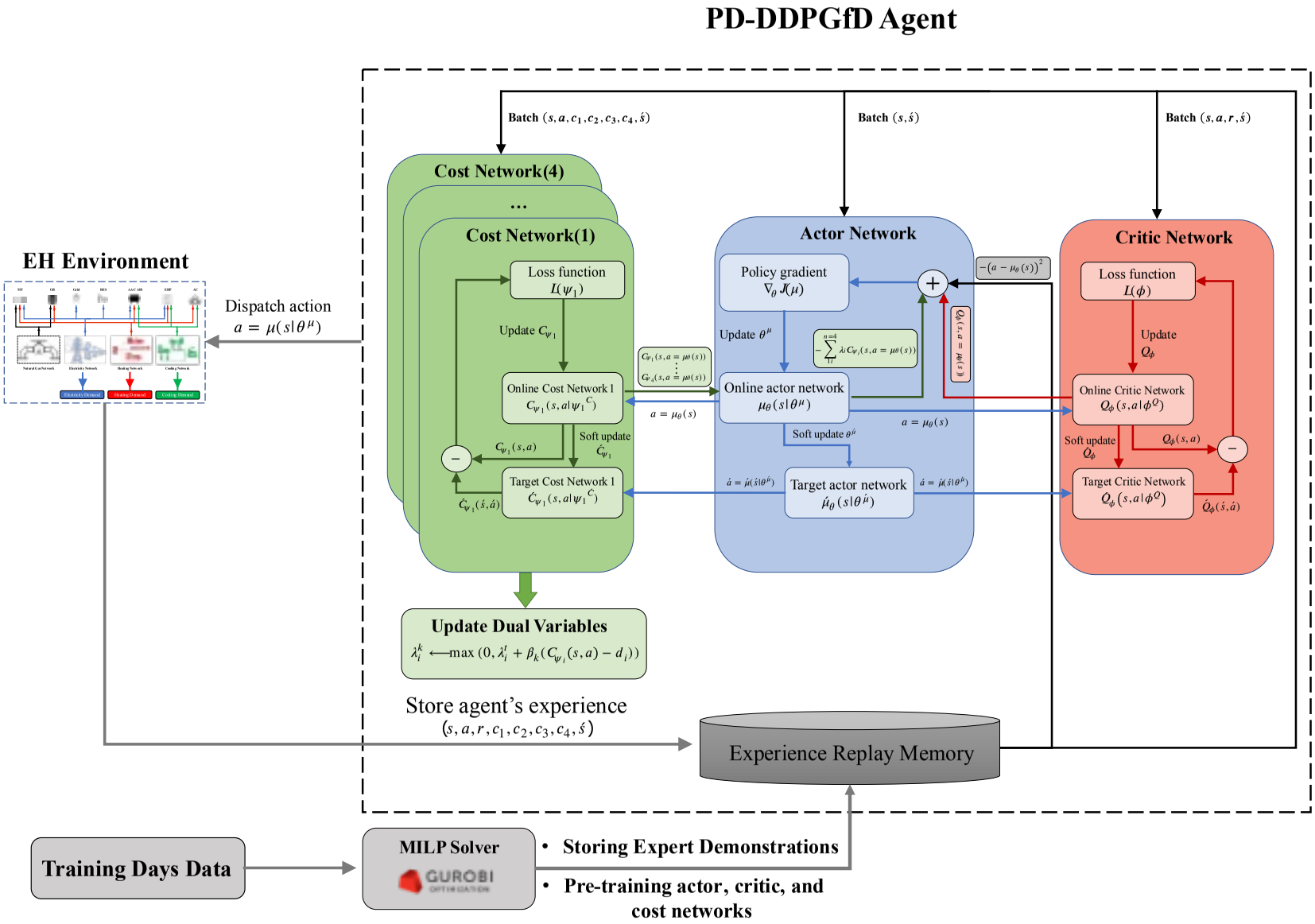


Figure 3.9: The proposed PD-DDPGfD algorithm's learning process

Chapter 4

Case study

This chapter presents in a brief way the dataset which is being used as a case study, the energy hub's specification, and the inputs to the simulations in Section 4.1. Further, the implementation of the proposed and benchmark algorithms are explained in Section 4.2.

4.1 Input Data and Pre-processing

The data from a community containing residential and commercial buildings in Santa Coloma de Gramenet, Barcelona, Spain [79] is used to evaluate the performance of the proposed algorithm. The load consumption data is generated by running simulations in commercial software TRNSYS 18 and the renewables power generation is calculated based on the weather data used for the simulations. Fig. 4.1 shows the hourly electricity, heating, and cooling demand and the ambient air temperature of the community over a year. The yearly data is split into training and test days with the test days being every fifth day (73 days in total) and the training days being the remaining (292 days). The statistic characteristics of loads and renewables generation is presented in Table 4.1 . Due to the high standard deviation of the data, training the agent is quite challenging. A Min-Max normalization method is used to normalize the data in order to enhance the learning process. In this method, all data points are scaled from 0 to 1 in order to ensure that all the features have the same scale.

The electricity price profile is obtained from previous study [56] which is based on time-of-use price. Off-peak electricity prices are 39.9 ($\frac{\$}{MWh}$) (23:00-07:00), on-peak prices are 199.9($\frac{\$}{MWh}$)

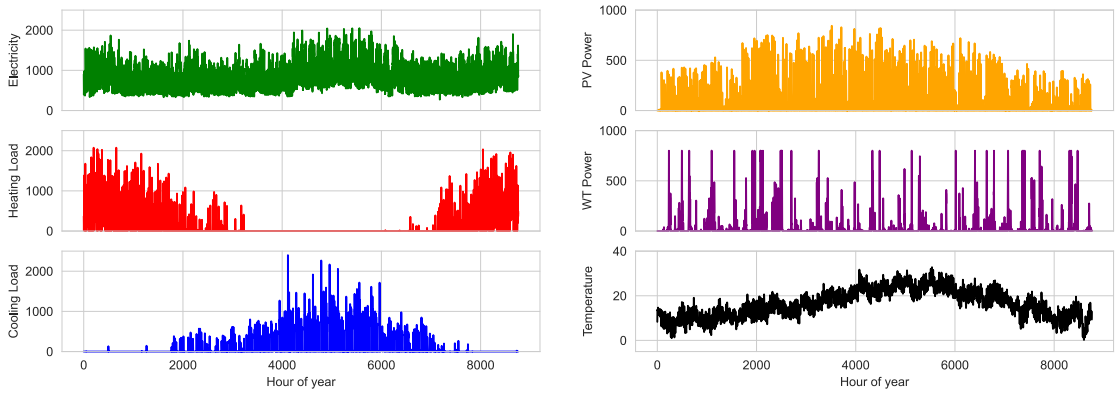


Figure 4.1: Hourly load demands, WT and PV power output, and ambient temperature of the community in the studied year

Table 4.1: Case study community loads and renewable generations information

	Min (kWh)	**Max** (kWh)	**Mean** (kWh)	**Stdv.** (kWh)
Electricity	272.20	2049.60	842.11	320.88
Heating	0	2073.80	125.44	282.63
Cooling	0	2400.00	99.01	248.86
PV Generation	0	843.59	104.60	179.43
WT Generation	0	800.00	34.39	114.39

(12:00-19:00), and mid-peak prices are 119.9 ($\frac{\$}{MWh}$) (08:00-12:00 and 20:00-23:00). It is assumed that electricity is sold for 0.5 times its purchase price and the natural gas price is fixed at 32.1 ($\frac{\$}{MWh}$) [56]. Moreover, the EH equipment specifications and operational constraints are provided in Table 4.2.

Table 4.2: Energy hub's equipment specifications and operational constraints

MT, AC, HP, and GB					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
P_{MT}^{max}	1500 kW	C_{AC}^{max}	1000 kW	P_{HP}^{max}	1000 kW
P_{MT}^{min}	200 kW	C_{AC}^{min}	0 kW	P_{HP}^{min}	0 kW
$\eta_{MT,p}$	0.35	COP_{AC}	0.9	COP_{HP}	3
$\eta_{MT,h}$	0.42	H_{GB}^{max}	1500 kW	H_{GB}^{min}	0 kW
				η_{GB}	0.8
AA-CAES					
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
$P_{AA-CAES}^{max}$	800 kW	$SOC_{AA-CAES}^{min}$	0.2	$SOC_{AA-CAES}^{max}$	0.9
$P_{AA-CAES}^{min}$	80 kW	$SOC_{AA-CAES}^{initial}$	0.5		
Grid		**WT** [80]		**PV** [81]	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
P_{grid}^{max}	2000 kW	P_{WT}^{rated}	400 kW	η^{PV}	0.186
P_{grid}^{min}	-2000 kW	v_{ci}	3.5 $\frac{m}{s}$	S^{PV}	5000 m^2
		v_r	12 $\frac{m}{s}$		
		v_{co}	25 $\frac{m}{s}$		

4.2 Implementation of The Proposed PD-DDPGfD Algorithm

The proposed algorithm is implemented using PyTorch [82] and the computations are done on a computer with Core i5-10500 CPU and Nvidia GTX 1050Ti GPU. Table 4.3 shows the parameters of the networks and the algorithms. A trial and error approach has been adopted for the parameter tuning process. Fig. 4.2 depicts the architecture of the networks with their input, hidden, and output layers. For the actor network, two hidden layers with 256 and 128 neurons are considered. A similar set of hidden layers is implemented for the critic and cost networks.

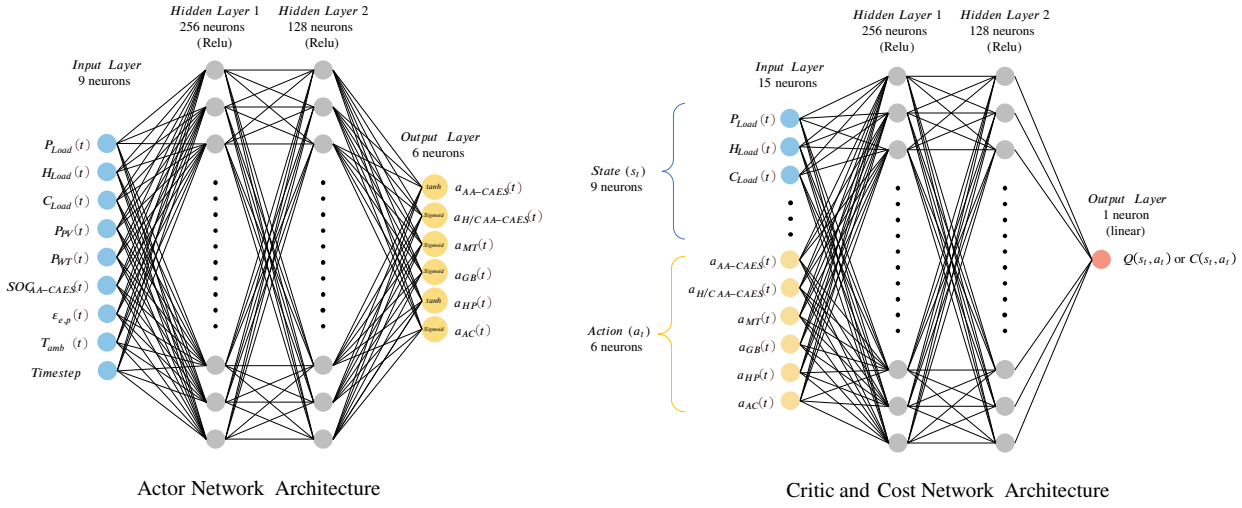


Figure 4.2: Actor, critic, and cost networks' architecture.
(activation function for each action element depends on its range)

Adam optimizer is utilized for hyperparameter optimization of all neural networks. The learning rate for actor network is 0.0005 and for critic and cost networks, 0.001 is considered. For the proposed algorithm, the batch size and replay memory size are 128 and 15,000, respectively. The target networks are updated using soft-update rate of 0.001. To stabilize the learning process, the actor and target networks are updated every 4 steps (the online critic and cost networks are updated in every step).

Since the SOC and grid constraint violations are treated as hard constraints, their constraint violation tolerance is zero. On the other hand, the heating and cooling constraints are treated as soft constraints that have the tolerance of one (kWh). For the pre-training process, the actor network

is pre-trained using expert demonstrations for 1000 epochs, while the critic and cost networks are pre-trained for 100 episodes. Fig. 4.3 shows the pre-training MSE of the actor, critic, and cost networks of the proposed algorithm.

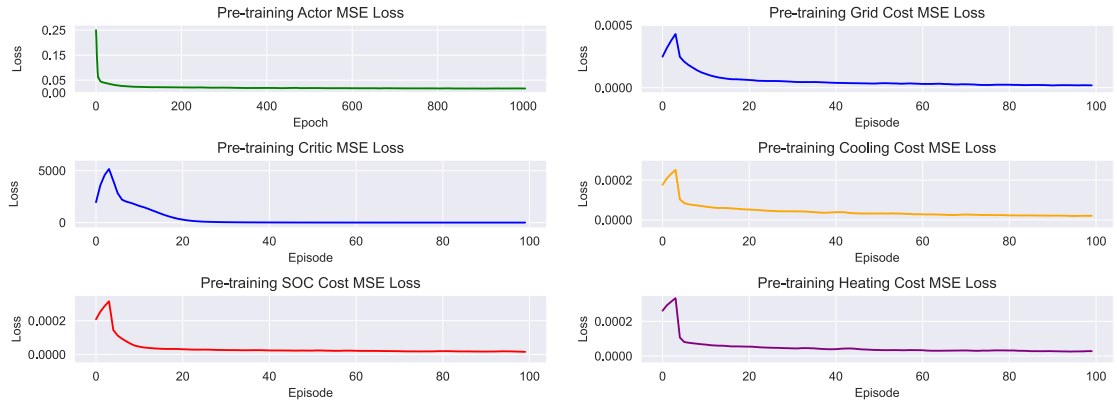


Figure 4.3: Actor, critic, and cost networks pre-training MSE loss

4.3 Implementation of The Benchmark Algorithms

In order to evaluate the performance of the proposed algorithm on both training and testing datasets in terms of operational cost and constraint violation, the results are compared to the existing state-of-the-art model-free algorithms and Support Vector Regression (SVR). For a fair comparison of performance, all algorithms are trained for 3000 episodes only. The benchmark algorithms are as follows:

(1) DDPG: the traditional existing DDPG algorithm in which the constraints of the EH are considered as a penalty and are added to the reward function. To update the actor network, only the critic network is involved.

(2) Deep Deterministic Policy Gradient from Demonstrations (DDPGfD): like DDPG, the constraints are added to the reward function. However, the actor and critic networks are pre-trained using the expert demonstrations. Moreover, the MSE between the sampled actions and the output of the actor network is added to the actor's loss function in DDPG.

(3) PD-DDPG: in this algorithm, the constraints have their own cost networks, and the actor network is updated using the output of the critic and cost networks with lagrangian multiplier.

(4) Behavioral Cloning (BC): In IL, the deep neural network is trained with the expert demonstrations and then used for real-time decision making. In this study, BC is the pre-trained actor network that is used in PD-DDPGfD and DDPGfD.

(5) SVR: as IL is a simple regression problem, any regression algorithm may be used for decision making. Support Vector Regression which works based on Support Vector Machines, is a powerful algorithm for prediction and is also used as a benchmark algorithm.

(6) Deterministic MILP: The daily dispatch problem in Section 3.1 is modeled and solved as a deterministic MILP problem, assuming perfect information about the EH mathematical models and the uncertainties in loads. This study uses the result of this approach as a theoretical benchmark.

Table 4.3: Algorithms' parameters

Parameter	**Value**
Actor learning rate	0.0005
Critic and cost networks' learning rate	0.001
Hidden layers	256/128
Reward and cost discount factor	0.995
Number of training episodes	3000
Soft-update rate	0.001
Initial exploration episodes (for PD-DDPG and DDPG)	200
Experience replay memory size (for PD-DDPGfD and DDPGfD)	15,000
Experience replay memory size (for PD-DDPG and DDPG)	100,000
Mini-batch size	128
Delayed update steps(policy_delay)	4
Initial exploration noise Stdv.	0.15
Minimum exploration noise Stdv.	0.05
Exploration noise decay rate (per episode)	0.0005
Initial dual-variables value	0
Dual-variables' learning rate	0.0001
SOC and grid constraint tolerance	0
Heating and cooling constraint tolerance	1
SOC penalty scaling coefficient ϑ_{SOC}	1500

Chapter 5

Results and Discussion

5.1 Performance Evaluation During Training Process

This section will compare the performance of the proposed PD-DDPGfD algorithm with the performance of the model-free algorithms mentioned earlier in Section 4.3 . Fig. 5.1 shows the reward learning curves of the studied DRL algorithms. It should be noted that the agents with the same reward functions are plotted on the same figure. There are solid lines on the graph that represent the sliding average of rewards over 50 episodes. As PD-DDPGfD and DDPGfD agents are pre-trained with expert policy, they have an immediate jump during the initial episode, and they have higher episodic rewards.

Considering that the agent performs action on a random day in each episode, the evaluations are performed every 500 timesteps over all the training days, and their average value is reported on a daily basis. The average daily operational costs and total constraint violations for each evaluation are shown in Fig. 5.2 . Since the BC and SVR agents have no interaction with the environment, their performance is constant over time. As shown in Fig. 5.2 , PD-DDPG and DDPG agents have high operational costs in the initial evaluation (4738$ and 7133$), whereas PD-DDPGfD and DDPGfD have lower values (2760$ and 2813$) since they are pre-trained. There is a noticeable difference between the operational costs of PD-DDPGfD and DDPGfD in each evaluation when compared with other agents. As a result of the training process, both the PD-DDPGfD and DDPGfD agents achieve almost optimal operational costs based on the theoretical benchmark. In the final evaluation,

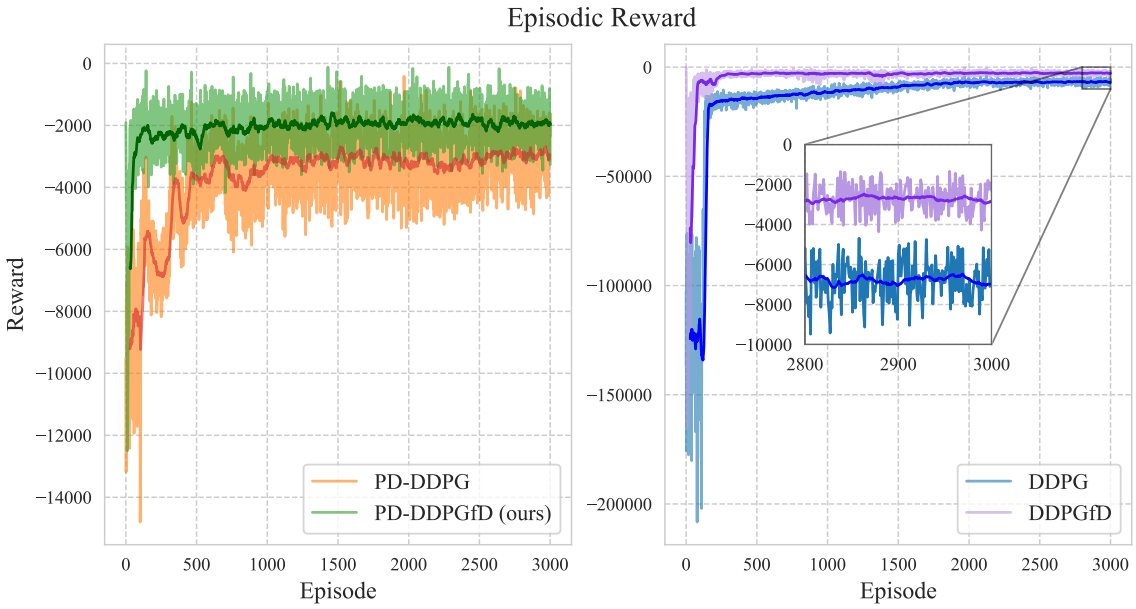


Figure 5.1: Episodic reward for different DRL algorithms over training process

the PD-DDPGfD and DDPGfD agents reach the average daily operational cost of 1826$ and 1847$, respectively, while PD-DDPG and DDPG agents reach the values of 2499$ and 2611$, respectively. This can be explained by the fact that PD-DDPG and DDPG agents are stuck in local optima, resulting in high operating costs. A further advantage of PD-DDPGfD and DDPGfD is that they have fewer fluctuations and are more stable. Therefore, expert demonstrations may contribute to lowering operating costs once they are incorporated into the learning process.

As shown in Fig. 5.2 , the PD-DDPG and PD-DDPGfD agents are capable of handling constraints more effectively than other algorithms. The actor network in these agents is penalized when it violates constraints due to the utilization of primal-dual optimization with dual variables. As compared to the BC agent, the performance of the agents DDPG and DDPGfD is almost the same. Since SVR agent is using a regression model for each piece of equipment, it has a poor performance in handling constraints. As a result, each model can have errors in predicting the optimal action, resulting in high total constraint violations.

Fig. 5.3 shows the constraint violations in the training process of PD-DDPGfD and the evolution of dual variables over time. Despite the fact that the cost networks are pre-trained, there is a high

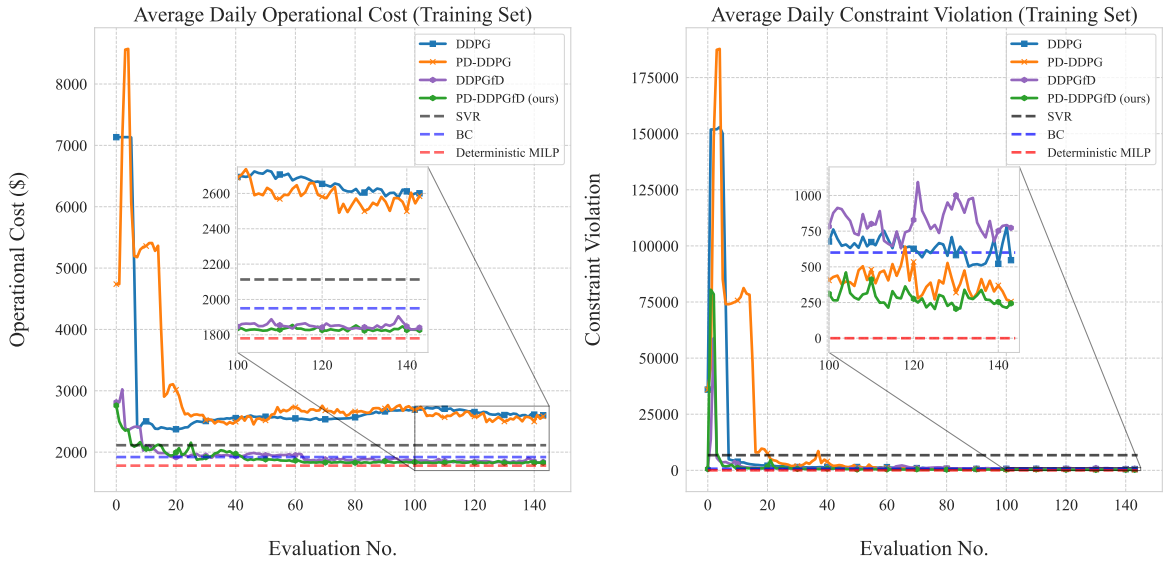


Figure 5.2: Average daily operational cost and total constraint violation over training process evaluations.

constraint violation right after the learning process begins. As the agent interacts more with the environment and updates the networks, the agent's ability to handle constraints improves. There are two possible reasons for this:

(1) The cost networks have been pre-trained with only optimal actions. This problem arises from the fact that an optimal action does not violate any constraints, and the cost networks have never encountered a state-action pair that violates a constraint. As a result, when the cost networks observe a state-action pair that is not optimal, they produce the wrong estimation of the corresponding C-value for the pair. This results in the agent being guided to actions with high constraint violations. Fig. 5.4 shows the loss value for cost networks in the first 300 episodes, which supports the explanation provided.

(2) The initial value of the dual variables is zero at the beginning of the learning process. The reason for this is that we want the agent to improve itself in terms of operational cost at the beginning. Choosing a higher initial value means that the agent should consider the output of the critic and cost networks in updating the actor network at the beginning. It is however notable that the agent's performance in terms of both operational cost and constraint handling is improved at the end of the training process.

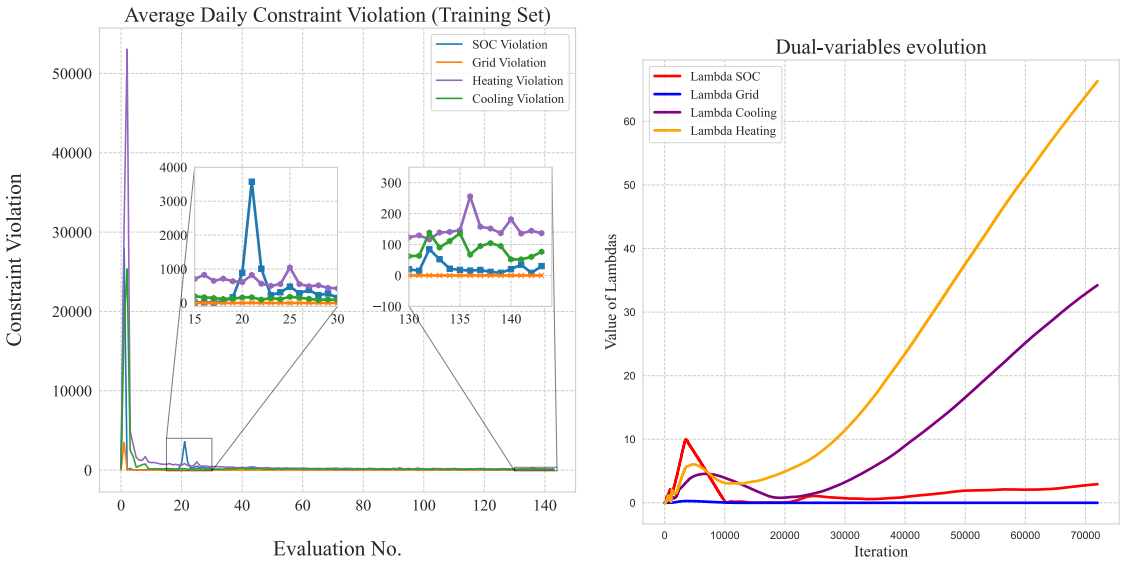


Figure 5.3: Constraint violations and dual-variables evolution in PD-DDPGfD learning.

According to Fig. 5.3 , there is a negative correlation between the dual variables and the constraints violations. Since lambda values are relatively small when costs are high, there exists a balance between maximization of rewards and minimization of violations. As a result of the increment in lambda values, even small violations of constraints will be prevented.

Table 5.1 presents operational cost and constraint violation values of each algorithm over the evaluations. According to Table 5.1, DDPG and PD-DDPG do not have any SOC constraint violations. This is primarily due to the fact that the AA-CAES is not used frequently in EH dispatch strategy. It is possible that the PD-DDPGfD and DDPGfD agents end up in a situation in which they exceed the maximum or minimum allowable SOC for the AA-CAES. Therefore, they have a small SOC violation of 20 and 76, respectively. It should be noted that this value is the absolute difference from the minimum and maximum SOC multiplied by penalty scaling coefficient ϑ_{SOC}. In terms of the grid power violation, all algorithms perform safe actions, except DDPG agent that has 13 kWh of daily average grid violation. Considering violations of heating and cooling constraints, both PD-DDPGfD and PD-DDPG have acceptable performance due to the cost networks that help the actor network to deal with the constraints. For PD-DDPGfD and PD-DDPG, the heating violations are 165 kWh/day and 265 kWh/day and cooling violations are 52 kWh/day and 165 kWh/day, while these values for DDPG and DDPGfD are 301kWh/day and 557 kWh/day for heating and 206

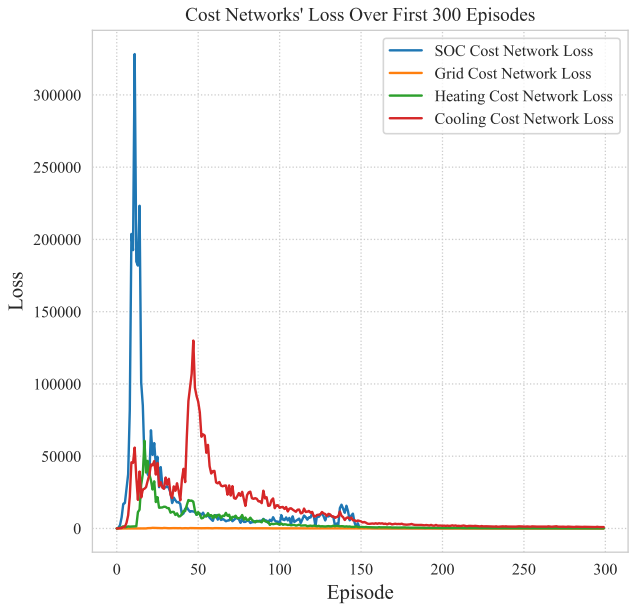


Figure 5.4: Loss value for cost networks over the first 300 episodes of learning process.

kWh/day and 117 kWh/day for cooling.

Table 5.1: Average daily operational cost and constraint violations during the training days

Method	Metric	Evaluation No.					
		1	30	60	90	120	140
DDPG	Operational Cost ($)	7133	2506	2550	2663	2654	2611
	SOC Violation	66656	0	0	0	0	0
	Grid Violation (kWh)	12168	6	12	13	13	13
	Heating Violation (kWh)	70637	821	1023	360	379	301
	Cooling Violation (kWh)	2385	669	387	279	234	206
DDPGfD	Operational Cost ($)	2813	1943	1950	1859	1842	1847
	SOC Violation	6836	6	43	94	76	76
	Grid Violation (kWh)	18	0	0	0	0	0
	Heating Violation (kWh)	32139	792	533	536	595	557
	Cooling Violation (kWh)	3539	154	153	158	156	117
PD-DDPG	Operational Cost ($)	4738	2522	2731	2712	2581	2499
	SOC Violation	11038	0	0	0	0	0
	Grid Violation (kWh)	1054	3	1	0	0	0
	Heating Violation (kWh)	30966	1629	849	480	328	265
	Cooling Violation (kWh)	2396	255	204	139	196	100
PD-DDPGfD	Operational Cost ($)	2760	1917	1863	1851	1828	1826
	SOC Violation	27953	148	61	80	21	20
	Grid Violation (kWh)	3479	0	0	0	0	0
	Heating Violation (kWh)	30908	430	247	193	180	165
	Cooling Violation (kWh)	18141	88	106	194	88	52

5.2 Generalisation Evaluation

This section focuses on evaluating the performance of the trained agents in dealing with unseen scenarios from test days data. Once the training process of the agents is done, the trained network is utilized for dynamic dispatch of EH. The trained online actor network observes the current state of the EH at each timestep and outputs the action for the equipment.

5.2.1 Performance Over Test Days

In this section, performance of the proposed PD-DDPGfD method is compared with the benchmark methods as discussed in Section 4.3 during the test days by different criteria. Fig.5.5 shows the cumulative daily operational costs of different algorithms in which the theoretical benchmark has the lowest cumulative operational cost of 128,946$. In comparison to other algorithms, PD-DDPGfD and DDPGfD have the lowest cumulative operational costs (133,657$ and 133,917$, respectively).

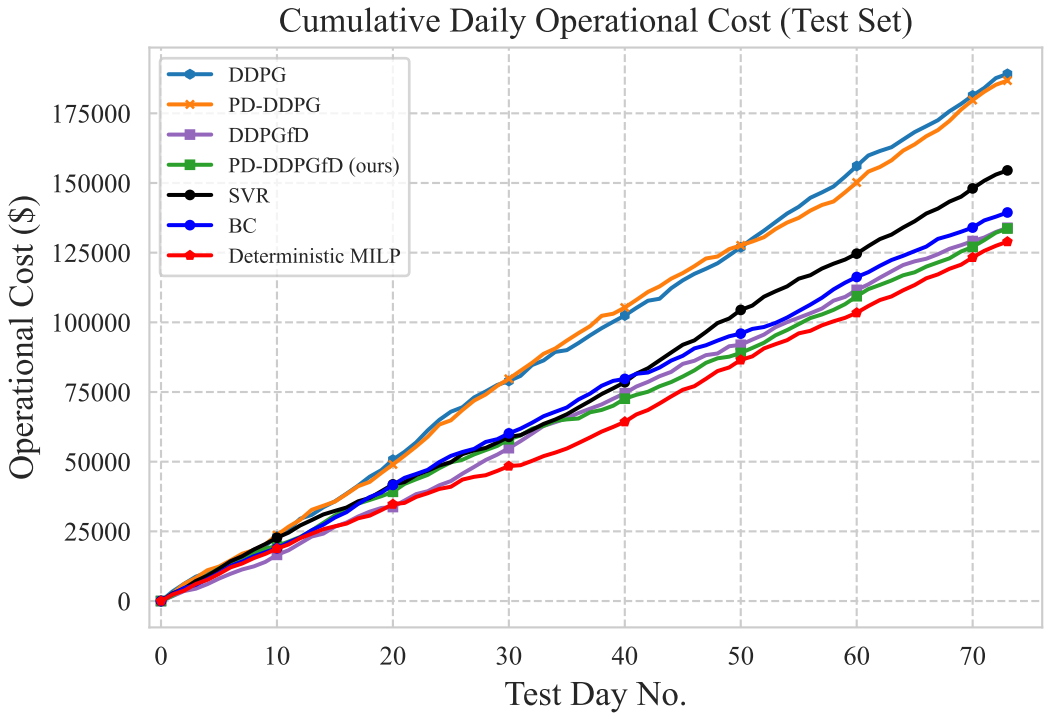


Figure 5.5: Cumulative daily operational cost of different algorithms over test days

Table 5.2 shows the average daily operational costs and constraint violations during the test period (it should be noted that since the model-based optimization approaches are always guaranteed to ensure the heat and cool demand–supply balance and electricity grid exchange limits, the deterministic MILP has no record of constraint violations). SVR agent has the highest heating and cooling balance constraint violation due to giving a similar action for GB and AC units (Fig. 5.6), even when there is no cooling or heating demand. DDPG, DDPGfD, and BC agents have almost the

same performance in terms of constraint violation, since they do not utilize primal-dual optimization. However, PD-DDPGfD agent outperforms other agents in terms of both operational cost with 2.74% error compared to theoretical benchmark and constraint violation.

Table 5.2: Average daily operational cost and constraint violations during the test days

Method	**Operational Cost**	**Violation** (kWh)			
	(Error with optimal)	**SOC**	**Grid**	**Heating**	**Cooling**
Behavioral Cloning	1915$ (7.22%)	108	0	643	321
SVR	2125$ (18.98%)	96	0	4018	913
DDPG	2606$ (45.91%)	9	0	483	240
DDPGfD	1839$ (2.96%)	67	0	563	147
PD-DDPG	2572$ (44.0%)	0	0	201	67
PD-DDPGfD (ours)	1835$ (2.74%)	24	0	135	65
Theoretical Benchmark	1786$ (0%)	0	0	0	0

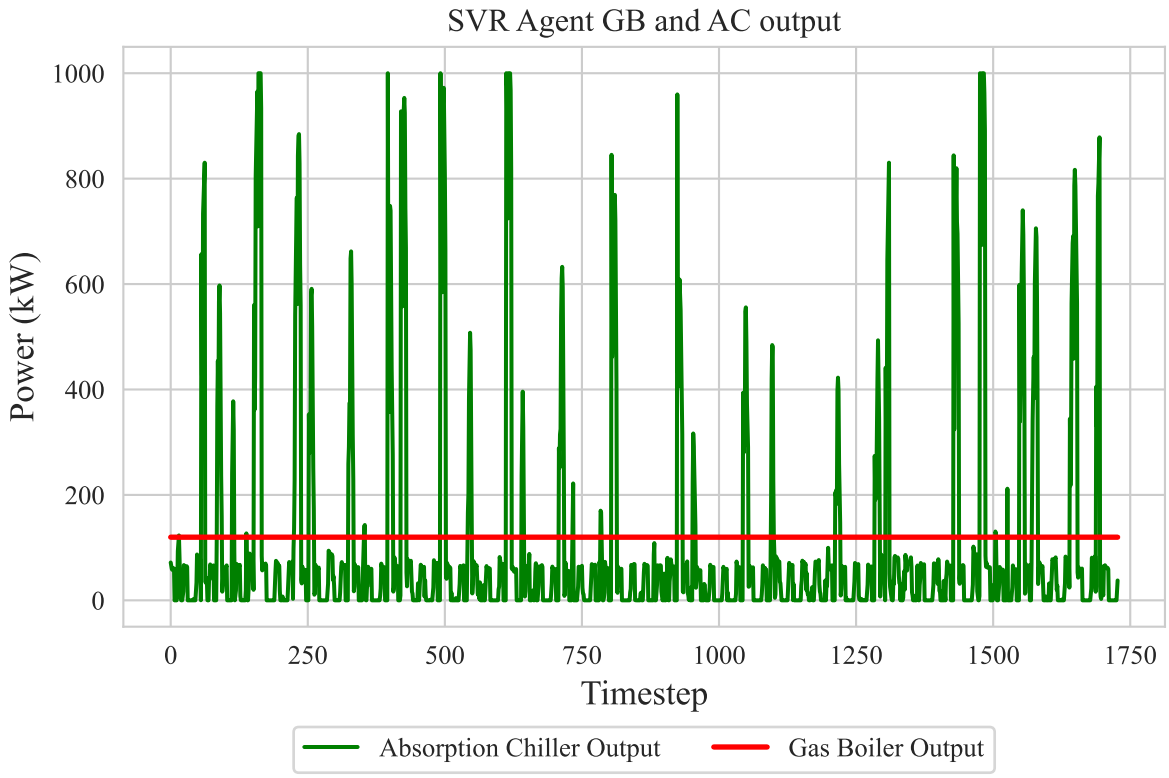


Figure 5.6: Gas boiler and absorption chiller power outputs for SVR agent over test days

Figs. 5.7 to 5.9 show the electrical, heating, and cooling balance of all agents over the 73 test days. Based on Fig. 5.7, it can be seen that the majority of the electrical demand of the community is met by importing power from the grid system. Some agents with lower operational costs have however taken advantage of their opportunity to sell electricity to the grid system (similar to the optimal strategy). Moreover, PD-DDPGfD and DDPGfD act like the theoretical benchmark since they have a pre-training stage before online learning.

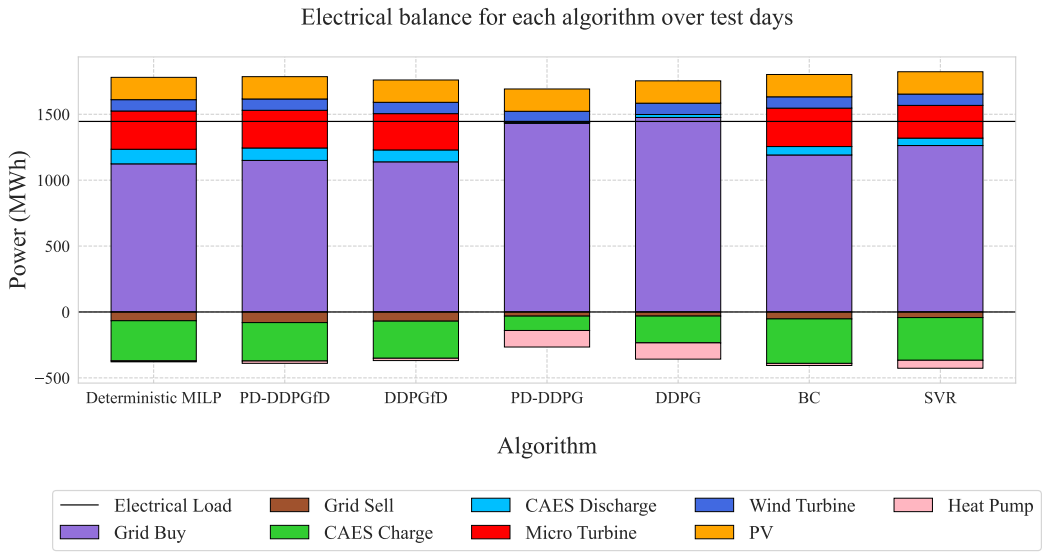


Figure 5.7: Electrical balance of all agents over test days

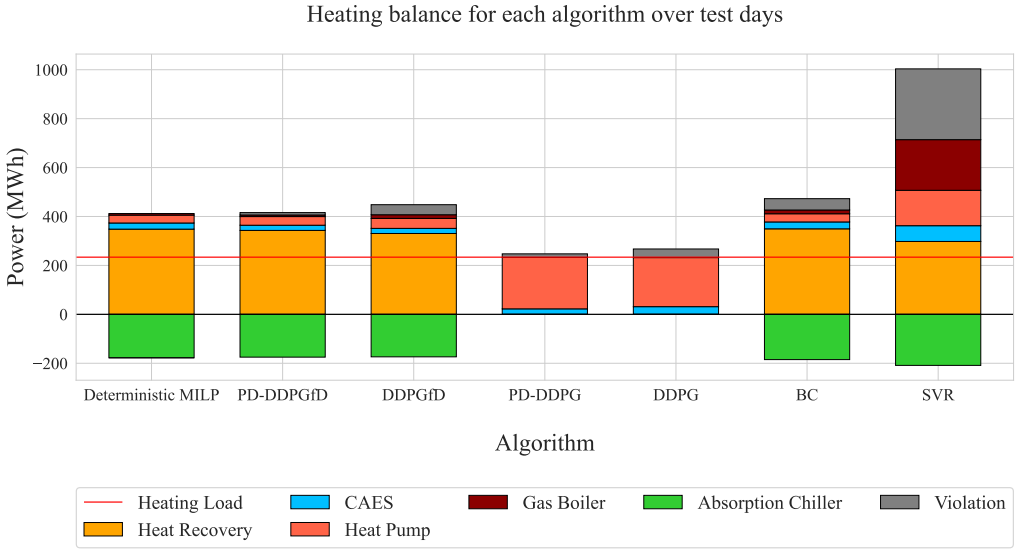


Figure 5.8: Heating balance of all agents over test days

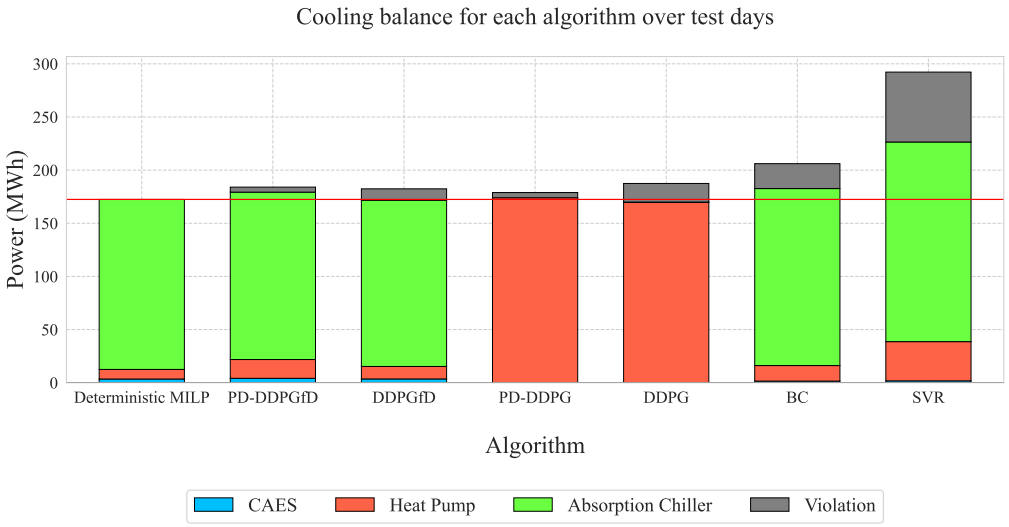
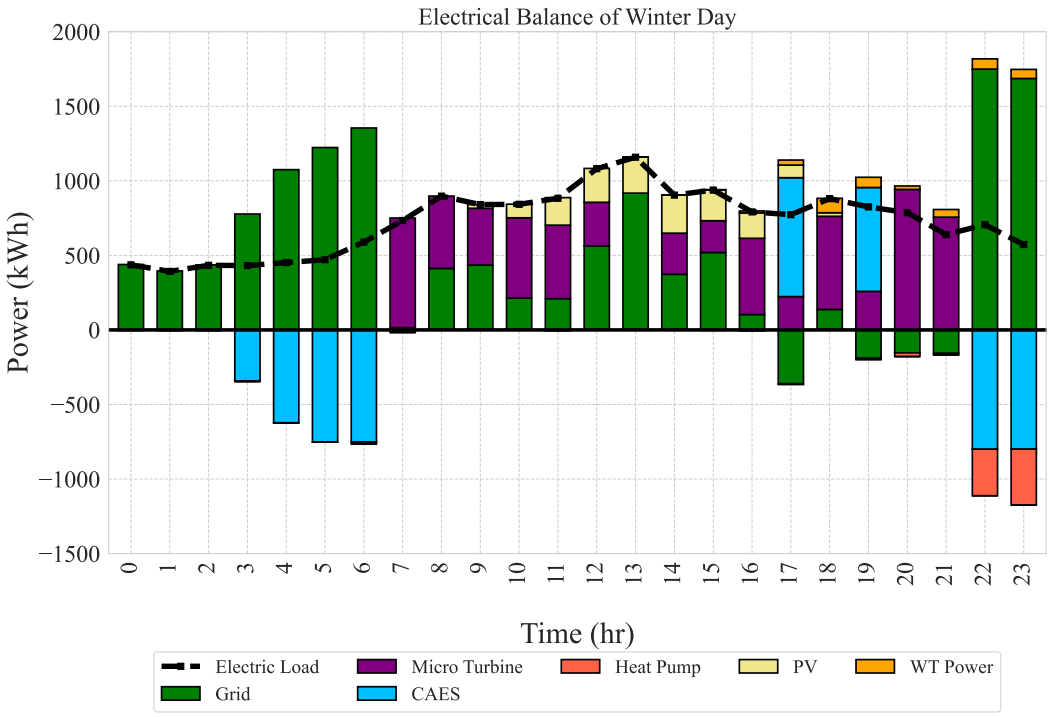


Figure 5.9: Cooling balance of all agents over test days

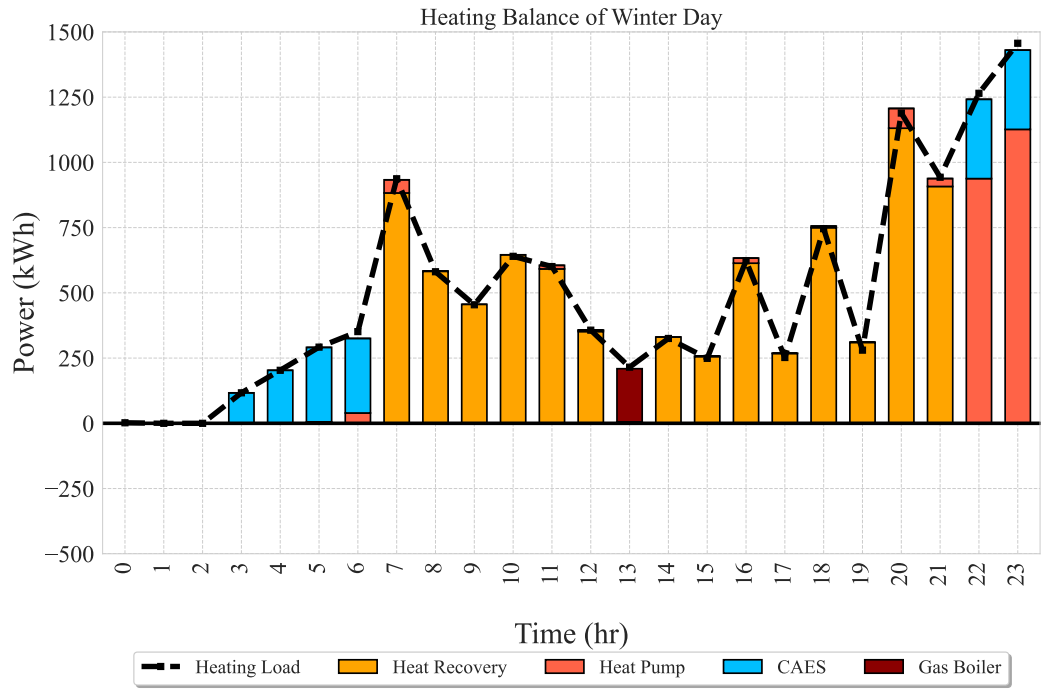
According to Fig. 5.8, heat from the heat recovery process of microturbine is mainly satisfying the heating demand. AA-CAES satisfies a small portion of total heating demand since the charging process of AA-CAES happens in the hours with low heating demand. PD-DDPGfD and DDPGfD agents have a similar dispatch strategy as the deterministic MILP approach for the heating demand, with higher violations for DDPGfD. As discussed in Section 5.1, DDPG and PD-DDPG agents get stuck in a local optima. The reason for this is that they rely exclusively on heat pumps for heating and cooling and never use MTs or ACs. By using a specific source for the supply of demand, they can minimize the constraints violations from heating and cooling balance, without paying attention to the high operating costs.

5.2.2 Dispatch Strategy Over Two Scenarios

The proposed PD-DDPGfD approach is applied to different scenarios to determine the optimal dispatch results and analyze generalization performance. Two scenarios from the test set are selected to test the performance of the PD-DDPGfD, a typical winter day (February 16) and a typical summer day (August 18). On these two days, the load curves and renewable energy generation are completely different, which illustrates the generalization capabilities of the proposed model. In winter, there is a high demand for heating, low PV generation, and no cooling demand, whereas during the summer there is the opposite situation.



(a) Electrical Balance



(b) Heating Balance

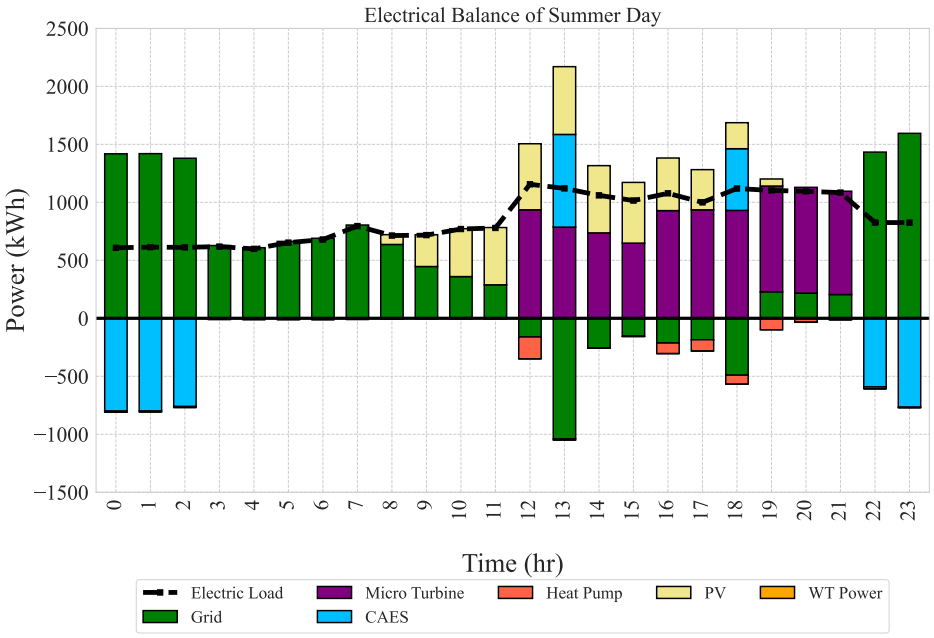
Figure 5.10: Dispatch results of PD-DDPGfD for a typical winter day from test set

PD-DDPGfD agent's dispatch results for a typical winter and summer day are shown in Figs. 5.10 and 5.11. Figs. 5.10(a) and 5.11(a) show the electrical balance results of the two scenarios, including power exchanged with the grid system, power generated by MT, power consumed by EHP, charge/discharge power of AA-CAES, and renewable generation PV and WT. In electrical balance, the heat pump has a negative value since it consumes electricity to generate heating or cooling. Furthermore, the CAES system's power is negative when it is charging, and positive when it is discharging. Fig. 5.10(b) shows the heating balance of the winter day including the heating energies from heat recovery unit, EHP, AA-CAES charging process, and gas boiler. Moreover, Fig. 5.11(b) presents the cooling balance of the summer day including absorption chiller, heat pump, and cooling from AA-CAES discharging process.

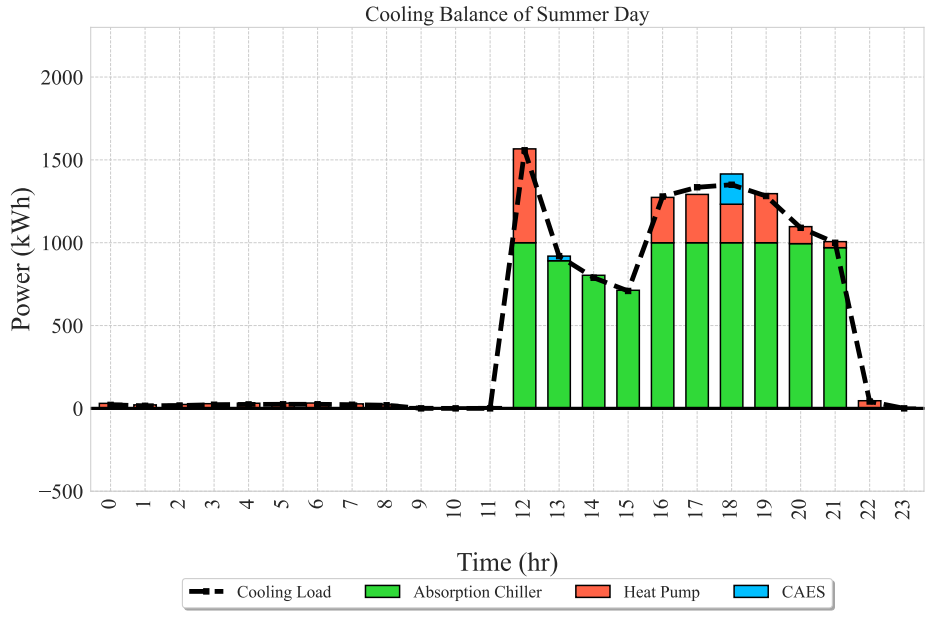
According to Fig. 5.10(a), during off-peak hours, electric load demands are met by importing energy from the grid system. The MT consumes natural gas to generate electricity during mid-peak and peak hours, and the rest is supplied by the grid system and AA-CAES. A limit on the amount of electricity generated by the MT in each hour is imposed by the heating demand, i.e., if the MT generates more electricity, the heating balance will be violated. Moreover, the charging of AA-CAES takes place at two different times, firstly during those off-peak hours (6 to 9) when there is a heating demand, and secondly during those last two hours of the day when the price is low and the SOC of AA-CAES needs to be recovered to its initial value.

Fig. 5.10(b) illustrates that the early morning heating demand is mostly met by AA-CAES charging and the heat pump as a result of the low electricity price during off-peak hours. The remaining portion of the heat is provided by a heat pump due to insufficient heating from AA-CAES (6AM). The agent decides to charge AA-CAES in the hours of 3 and 4 with partial loads, although charging AA-CAES with a lower power results in a lower charging efficiency. This is because AA-CAES can satisfy the heating demand in these two hours without any additional cost, whereas if AA-CAES was charged with nominal load at hour 3, additional electricity would have to be imported from the grid system to run the heat pump at hour 4, which would increase the operational cost. To meet the heating demand during mid-peak and on-peak hours, the heat recovered from MT is used. Due to the fact that the heating demand in hour 13 is lower than the corresponding heat recovered from the minimum operational power of the MT, the GB is used to cover the heating

load. Additionally, AA-CAES attempts to recover its SOC for the next day in the last two hours of the day. As a consequence, the heating demand in these hours is mainly met by AA-CAES charging heat and the EHP.



(a) Electrical Balance



(b) Cooling Balance

Figure 5.11: Dispatch results of PD-DDPGfD for a typical summer day from test set

Fig. 5.11(a) illustrates the electrical balance for the summer day. Similar to a winter day, the grid system supplies electrical demand during off-peak hours. However, the AA-CAES is charged in hours 0 to 3 with nominal load. Since PV generation is higher in summer, there is an opportunity for the agent to sell the excess energy to the grid system and reduce the cost. As a result, AA-CAES is discharged during peak hours to not only sell excess energy, but also cover a small portion of the cooling load (hour 18). The amount of electricity MT generates during the summer is limited by the cooling load because the recovered heat is consumed to generate cooling energy in AC. Thus, during peak hours with cooling loads, the MT generates electricity and sells it to the grid. The cooling balance of the summer day is shown in Fig. 5.11(b). In the illustration, the AC is primarily used during peak hours to cover the cooling load, while the EHP is utilized during off-peak hours. Even though the AA-CAES is discharged at hour 13, no use is made of its cooling capacity. It occurs because the agent decides to generate more electricity using MT, sell it to the grid system, and use the generated heat to run the AC in that hour. In this way, operational costs will be reduced. Moreover, a small amount of violation from cooling balance can be seen in hours 17 and 18.

5.2.3 AA-CAES Utilization

The purpose of this section is to analyze the performance of each algorithm with regard to utilizing the AA-CAES in the dispatch strategy. Fig. 5.12 shows the power frequency of the AA-CAES over the test days (the idle mode is not considered). The deterministic MILP approach illustrates how AA-CAES should operate optimally. It is shown that AA-CAES charges at nominal load most of the time. In some cases, however, charging with partial load is more cost-effective than charging with full load. As a result of considering the off-design characteristics, AA-CAES mostly discharges with nominal loads to have a higher discharge efficiency. It can be seen that AA-CAES in PD-DDPGfD and DDPGfD approaches have a strategy that is almost the same as the optimal strategy. However, the PD-DDPG and DDPG only charge the AA-CAES in most hours since charging with partial loads and barely discharging the system is the best way to satisfy the SOC constraint. BC and SVR agents have prediction errors since they attempt to mimic the deterministic MILP approach. Consequently, they always discharge the AA-CAES with less than 50% of the nominal load.

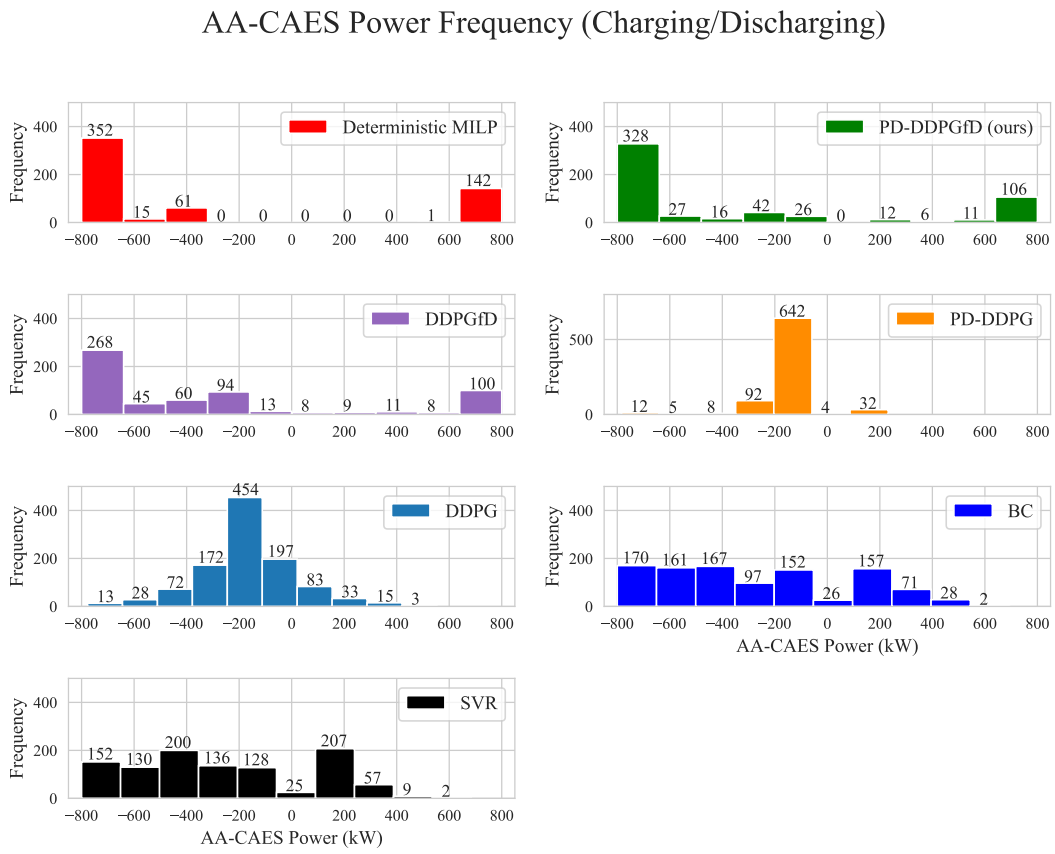


Figure 5.12: AA-CAES power frequency over test days (only charging/discharging modes)

Fig. 5.13 illustrates the total consumed and generated energies from AA-CAES over the test days. The total efficiency of the AA-CAES in this period is determined by the following equation:

$$Eff_{AA-CAES} = \frac{Discharged\ Energy + Heating\ Supplied + Cooling\ Supplied}{Charged\ Energy} \times 100\% \qquad (81)$$

The PD-DDPGfD and DDPGfD, like the deterministic approach, result in high total efficiencies of 41.01% and 40.72%, respectively. Other approaches, however, have lower efficiency because they barely discharge the AA-CAES and charge it in most hours. Additionally, since PD-DDPG and DDPG agents discharge the AA-CAES with small partial loads, the air leaving the last expander has a higher temperature than the ambient air. It is therefore not possible for AA-CAES to generate any cooling energy. Additionally, AA-CAES is mainly in charging mode in PD-DDPG and DDPG approaches, which generates more heat. In comparison to the deterministic approach, PD-DDPGfD and DDPGfD utilize more cooling energies from the AA-CAES discharge process. This is because

these agents are not able to understand that using MT and AC is more economically beneficial than AA-CAES to satisfy cooling loads. It can be concluded that in this EH, the AA-CAES mainly contributes in satisfying the electrical and heating demands, not the cooling demand.

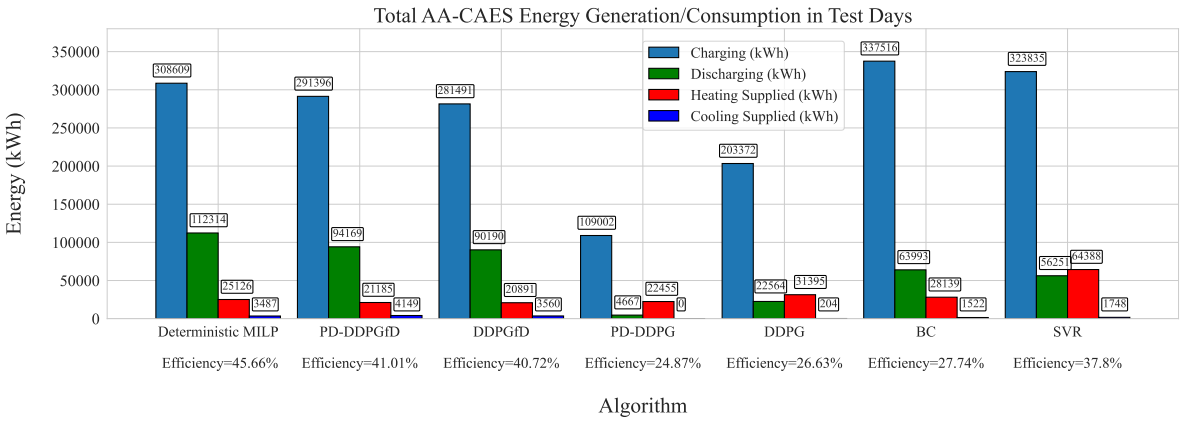


Figure 5.13: Total AA-CAES energy generation/consumption and total efficiency in test days

Detailed information about each algorithm's SOC violation during test days can be found in Table 5.3. BC, SVR, and DDPG agents have the highest violations against the minimum and maximum allowable SOCs. As DDPG agent uses AA-CAES less frequently, it violates the constraint in fewer days than BC and SVR agents. With PD-DDPG agent, AA-CAES is always operated within the valid range due to the less frequent operation. PD-DDPGfD and DDPGfD agents only violate from the maximum SOC. The PD-DDPGfD agent, however, violates in less days, compared to the DDPGfD agent. There is no doubt that the PD-DDPGfD agent utilizes nearly the full potential of AA-CAES. There is however a small deviation (0.022) from the maximum allowable SOC (0.9), which is still safe for air tank's pressure. PD-DDPGfD's SOC violation in Table 5.2 is caused by this small violation in 14 days, including the following timesteps where AA-CAES is idle. Although our proposed algorithm considers the constraints, there is always a trade-off between maximizing reward and minimizing constraint violation [66]. It can explain the very small violations from even the hard constraints like SOC constraint.

Table 5.3: Minimum and maximum AA-CAES SOC and No. of unique days with SOC violation over test days in each algorithm

Method	Min SOC	Max SOC	Days with Violation
Behavioral Cloning	0.137	0.995	66
SVR	0.119	0.956	38
DDPG	0.188	1.057	2
DDPGfD	0.222	0.933	38
PD-DDPG	0.250	0.708	0
PD-DDPGfD	0.207	0.922	14

5.3 Impact of Partial-load Operation of AA-CAES

This section examines the effects of partial-load operation of AA-CAES on the optimal dispatch of EH. On Fig. 5.14, the power frequency of the AA-CAES over the test days is shown in two modes, one in which off-design characteristics are considered and one in which they are neglected. By neglecting the off-design characteristics, AA-CAES operates as a simple input-output energy storage system with constant charging/discharging efficiencies. This results in several periods of discharging with a low partial load ratio.

Table 5.4 shows the results of EH dispatch with charge/discharge powers of AA-CAES while neglecting the off-design characteristics. Several issues can arise as a result of neglecting the off-design characteristics, such as violations of the SOC limits and a considerable mismatch between electricity and cooling loads and the supply of power. Since the isentropic efficiency and expansion ratio of the expanders is lower than design values in part-load operation, a higher amount of compressed air is needed to be discharged to generate the same amount of power. As a result, the air storage will be emptied sooner because a lot of air will be discharged for a small amount of power. Therefore, there will be some timesteps that the AA-CAES receives discharge signals, although the AA-CAES tank is empty. This leads to a considerable mismatch between electricity supply and demand. The cooling mismatch is due to the fact that the output air from expanders has a higher

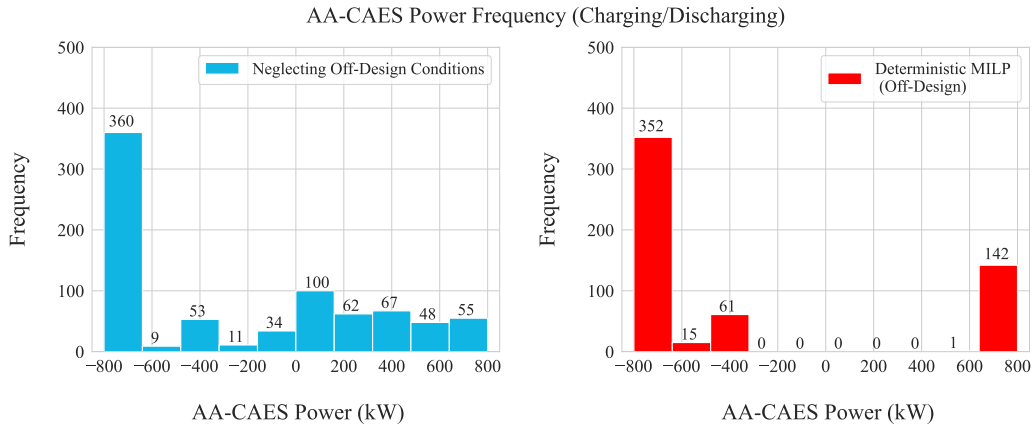


Figure 5.14: AA-CAES power frequency over test days (only charging/discharging modes) in two modes of considering and neglecting off-design characteristics

temperature than its designed value when discharged with low part-loads. However, neglecting the off-design conditions has no considerable impact on the amount of generated heating during the charging process. This is primarily due to the way compressors work for a given power input. In fact, at lower part-loads, the given power is used mainly to increase the temperature of the airflow rather than to achieve a higher mass flow rate. In case of the amount of heat generated, off-design conditions does not make much of a difference since the higher airflow temperature compensates for the reduction in compressed air flow rate. Moreover, the SOC of AA-CAES is violated in all test days due to not considering the off-design characteristics in discharging process.

Table 5.4: SOC violation and power mismatch by neglecting the off-design conditions of AA-CAES

Days with SOC violation	**Power Mismatch**		
	Electricity	Heating	Cooling
73	12.72 (MWh)	1(kWh)	2.01(MWh)

5.4 Sensitivity Analysis

The purpose of this section is to provide a brief sensitivity analysis of the hyperparameters of the proposed approach. There are a number of factors that can have an impact on the performance of black-box approaches, and examining the effect of all the hyperparameters may not give a reliable and meaningful result. Consequently, this section will only address the effect of the following hyperparameters:

(1) Random Seed

(2) Discount Factor

5.4.1 Random Seed

In order to generate the same results from a random generator, a random seed is used. The parameter is used in the initialization of neural networks in deep learning or while creating training and test datasets. In order to evaluate the performance of different models, the same training and validation data sets should be obtained while using different hyperparameters or machine learning algorithms. In this case, a random seed value is necessary. However, in reinforcement learning, random seed is only used for the initialization of an agent's neural network in order to ensure that the results are not obtained by accident. Therefore, in this section, different agents are trained for five different random seeds to make sure of the reproducablity of the results.

Figs. 5.15 and 5.16 show the operational cost and constraint violation of the agents in the learning process for five random seeds. Since the results have fluctuations, for better illustration, An average sliding over 20 evaluations is shown. As illustrated, even with different random seeds, our proposed approach converges to a near-optimal policy with the lowest operational cost and constraint violations compared to other approaches. Accordingly, the results presented in the previous sections can be reached with a very small error in terms of the accuracy. In contrast, the DDPGfD approach exhibits different policies depending on the random seed used. According to Fig. 5.15, compared to our proposed approach, the operational cost of different seeds for DDPGfD has more fluctuations. As an example, running with random seed 0 results in even worse performance than

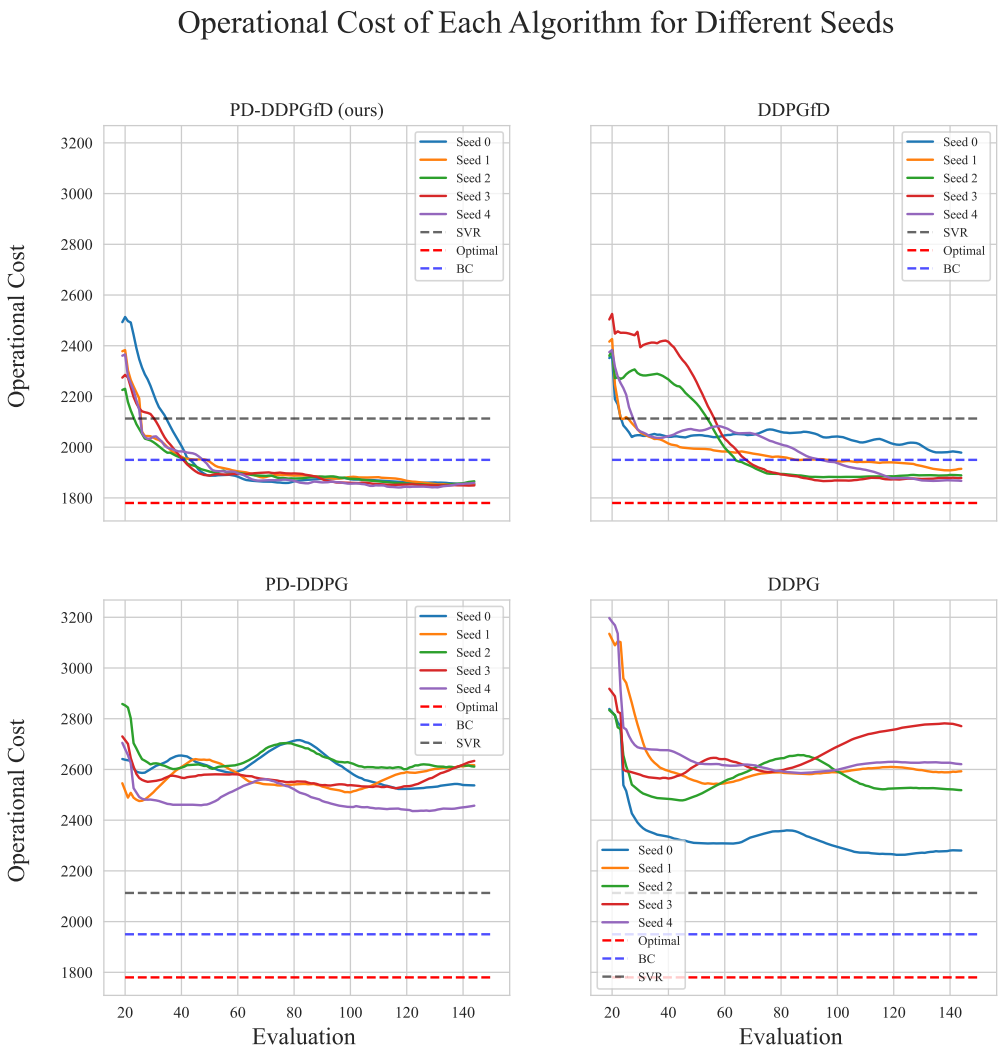


Figure 5.15: Average daily operational cost over training process evaluations for five random seeds

the BC agent. The operational cost of PD-DDPG and DDPG agents is always higher than that of BC and SVR agents, and they are always stuck in a local optimal state even with different random seeds. It should be noted that, regardless of the random seed used, the dispatch policy achieved by the DDPGfD agent is always worse than the BC agent in terms of constraint violation (Fig. 5.16). In general, the PD-DDPG and PD-DDPGfD agents have low constraint violations with different random seeds.

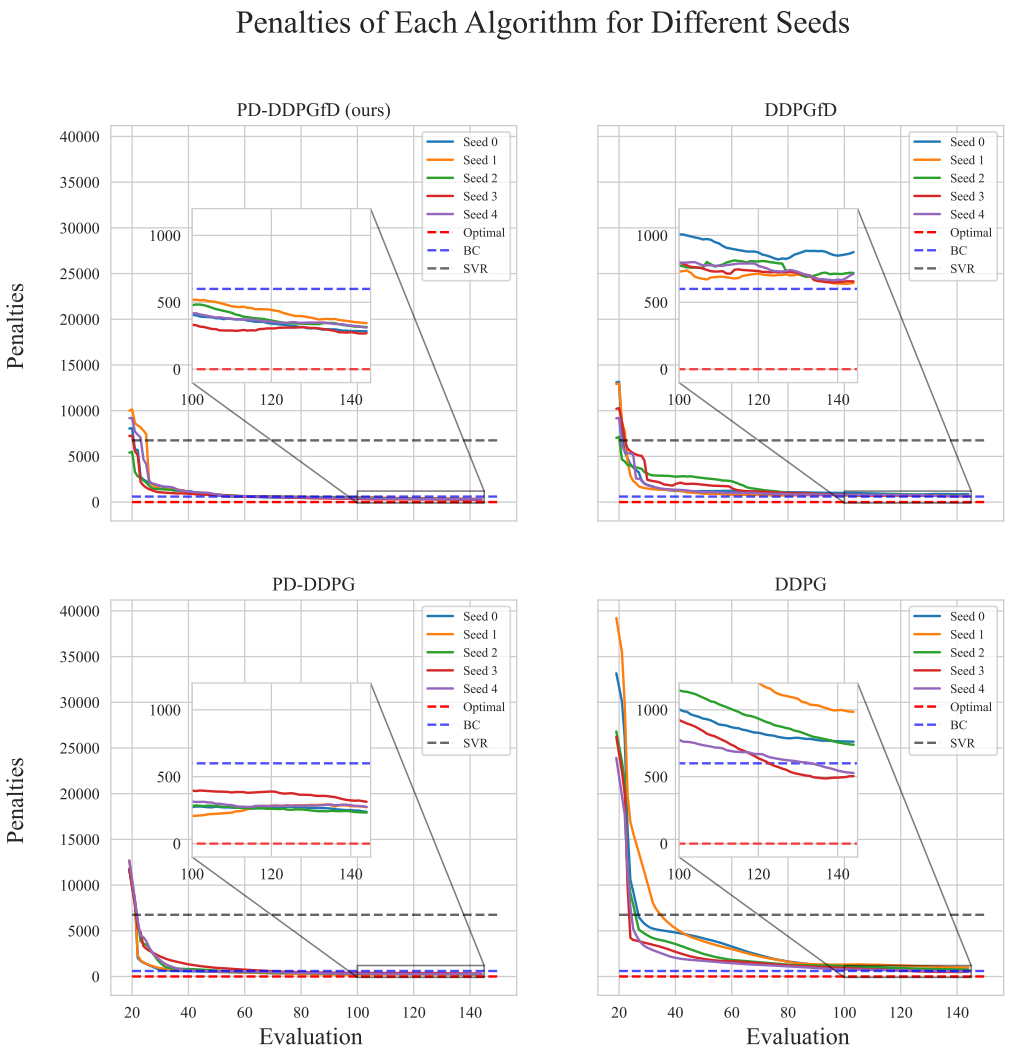


Figure 5.16: Average daily total constraint violation over training process evaluations for five random seeds

5.4.2 Discount Factor

Reinforcement learning aims to maximize long-term rewards weighted by a discount factor. Essentially, the discount factor determines how much reinforcement learning agents care about rewards in the distant future compared with the immediate reward. For instance, if $\gamma = 0$ then the agent will be completely myopic and will only be able to learn about actions that have an immediate reward associated with them. In the case of $\gamma = 1$, the agent will be equally concerned with future rewards as with the current immediate reward. Fig. 5.17 shows the weight value for different

discount factor values in the horizon of $t = 24$.

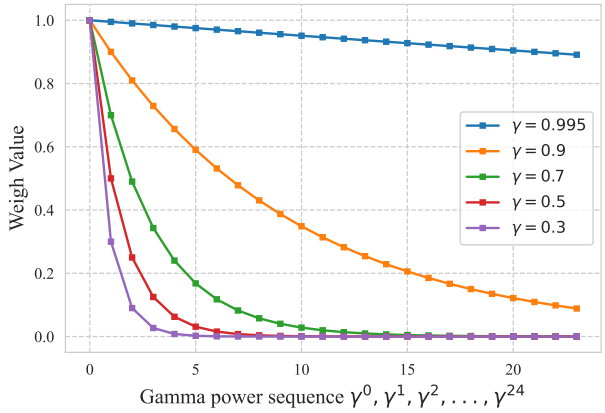


Figure 5.17: Importance of discount factor in considering the future rewards in a finite horizon

This section examines the effect of different discount factors on the performance of our proposed approach. Fig. 5.18 illustrates the average daily operational cost of the proposed approach for different discount factor values. The results indicate that in general, a higher discount factor value will result in lower operating costs. There is, however, a slight difference in the operational costs for different discount factors. An analysis of the case study in more detail may be able to explain why this is the case.

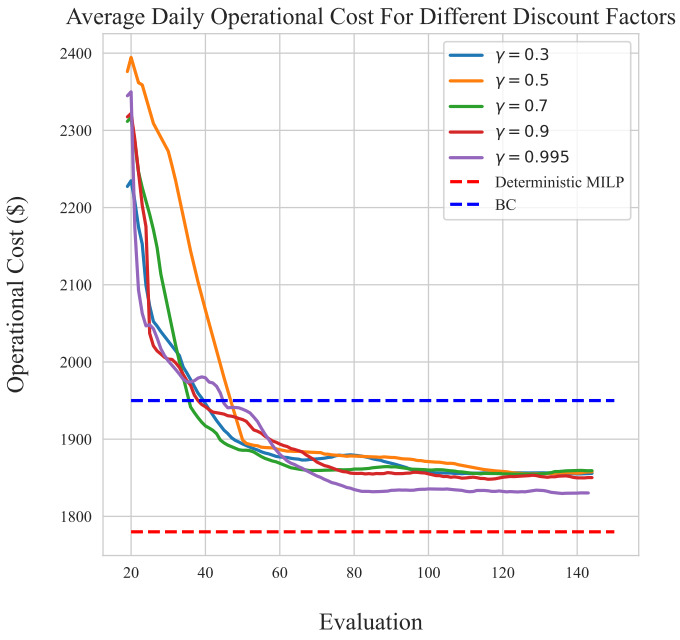


Figure 5.18: Average daily operational cost of PD-DDPGfD over test days for different discount factors

In the case study, discount factor will only affect how AA-CAES is used by the agent, as other equipment has an impact only on the immediate reward or cost fuction. As an example, using the MT in a timestep has no impact on the reward the agent will receive in the future, but this is not the case with the energy storage. Since the power must be purchased from the grid, charging the AA-CAES will increase the operational cost. As a result, the agent receives a smaller reward in this timestep, but if the agent discharges the AA-CAES at a later time (in hours with a high sell price), it will receive a larger reward that will compensate for the smaller reward received during the charging process. Fig. 5.19 shows the power frequency of AA-CAES over the test days for different discount factors. The explanation above can be verified by observing that with a higher discount factor, agents understand that it is beneficial to charge and discharge CAES more frequently.

Figure 5.19: AA-CAES power frequency over test days (only charging/discharging modes) for different discount factors

Chapter 6

Conclusion and Future Work

6.1 Summary

In this thesis, a novel SDRL approach that employs primal-dual optimizations and IL is proposed. We aimed at enhancing the performance of DRL algorithms in optimal scheduling of a trigenerative AA-CAES based EH. In doing so, the effect of considering off-design characteristics of the trigenerative AA-CAES as energy storage was investigated. In order to ensure a global optimization, the operation of the trigenerative AA-CAES in the presence of off-design conditions was modeled and linearized. Additionally, the results from an experimental case study revealed that utilizing only IL as part of DDPG algorithm can effectively reduce the operational cost of traditional DDPG algorithm, while still violating operational constraints at the same time. Alternatively, primal-dual optimization could effectively satisfy constraints but could get stuck in local optima with a high operational cost due to utilizing only a certain source of energy for satisfying heating and cooling demands. Taking this into account, it can be concluded that the proposed approach, by combining the two methods mentioned above, was able to accomplish both objectives at the same time. To be more specific, the average daily operational cost of the proposed approach (PD-DDPGfD) is only 2.74% higher than the theoretical benchmark, as opposed to BC, SVR, DDPG, PD-DDPG, and DDPGfD approaches with 7.22%,18.98%, 45.91%, 44.0%, and 2.96%, respectively. Based on the proposed approach, there were fewer average daily heating and cooling violations of 135kWh and 65kWh, respectively, and very few SOC violations. A notable feature of the proposed approach is

that, unlike the SP approach, it requires no prior model to formulate the uncertainties. Furthermore, it has been demonstrated that ignoring the off-design characteristics of AA-CAES could result in a large mismatch between the supply and demand of electricity and cooling at the EH, due to the discharge of AA-CAES at low partial loads. Moreover, a sensitivity analysis of the random seeds demonstrated that the proposed method is reliable and gives the same performance over different runs, whereas other approaches give different results and are not reliable.

6.2 Contributions

The main contributions of this thesis can be summarized as follow:

(1) The operation of a tri-generative AA-CAES with off-design conditions in an EH was modelled and linearized.

(2) The effect of neglecting the off-design operation of a tri-generative AA-CAES in a case study EH was investigated.

(3) A novel SDRL approach by combining two existing DRL approaches (PD-DDPG and IL) was proposed to enhance the performance of the traditional DDPG algorithm in terms of minimizing the operational cost while satisfying the constraints of EH dispatch problem.

(4) The performance of the proposed approach in terms of operational cost and constraint violation was compared to the existing DRL, SDRL, and IL approaches.

(5) A sensitivity analysis on the random seeds was performed to ensure the reliability and reproducibility of the results.

6.3 Assumptions and Limitations

The present study has a number of limitations. Due to the absence of a mathematical framework for handling constraints in DRL algorithms, it remains challenging to model the operation of EHs under a wide range of operational constraints. Therefore, a simplified model of the EH operation is

considered. For instance, the ramp-up and ramp-down constraints for different equipment such as MT is not considered.

In regard to the operation of AA-CAES, a simple model is presented for its off-design operation and thermal storage. It is assumed that the thermal storage is only for the AA-CAES system itself nad has no contribution in satisfying the heating demand of the community. Moreover, the dynamic operation and temperature variation of the thermal storage in AA-CAES is not studied. Furthermore, since most of the EH equipment are mechanical components, their response time should be considered in real-time scheduling. However, in this study, the timestep is assumed to be 1 hour, which is not realistic for real-time optimization.

Moreover, due to the lack of data for simulations, it is assumed that the gas price and buy/sell electricity price have the same profile over the whole year. However, in practice, these profiles may vary from day to day. The cooling and heating consumption data for some buildings are missing in the case study. Since there is no TES in the EH to store the excess heat of compression in AA-CAES when there is no demand, this can lead to inaccurate evaluation of the AA-CAES total efficiency.

It is also worth mentioning that in this study, the specification of the EH equipment are pre-assumed. It is observed in the results section that the sizing of the AC can affect the amount of utilized cooling energy from AA-CAES. Therefore, in order to have a better insight about the scheduling probelm, the sizing of the EH should be optimized first.

6.4 Future Works

The following suggestions are provided for future studies:

(1) Considering more operational constraints of the EH equipment and real-time operation of AA-CAES as mentioned in the limitations.

(2) An analysis of the operational impact of uncertainties and dynamism in the price of electricity and gas.

(3) Performing data analysis on the train-test split of data in order to determine the generalization of the proposed approach. In existing studies, it is not clear how test days are chosen from the

dataset for RL studies. For a better understanding of how to split the dataset for training and testing, it is recommended to use unsupervised learning and clustering algorithms.

(4) Considering the GHG emission alongside the operational cost of the EH as a multi-objective problem. Multi-Objective Reinforcement Learning (MORL) is achieving more attentions in the recent years. Therefore, it would be a good idea to consider it as a field for future studies.

Bibliography

[1] M. E. Hassanzadeh, M. Nayeripour, S. Hasanvand, and E. Waffenschmidt, "Decentralized control strategy to improve dynamic performance of micro-grid and reduce regional interactions using bess in the presence of renewable energy resources," *Journal of Energy Storage*, vol. 31, p. 101520, 2020.

[2] H. Chen, H. Wang, R. Li, Y. Zhang, and X. He, "Thermo-dynamic and economic analysis of sa novel near-isothermal pumped hydro compressed air energy storage system," *Journal of Energy Storage*, vol. 30, p. 101487, 2020.

[3] A. L. Facci, D. Sánchez, E. Jannelli, and S. Ubertini, "Trigenerative micro compressed air energy storage: Concept and thermodynamic assessment," *Applied Energy*, vol. 158, pp. 243–254, 2015.

[4] E. Bazdar, M. Sameti, F. Nasiri, and F. Haghighat, "Compressed air energy storage in integrated energy systems: A review," *Renewable and Sustainable Energy Reviews*, vol. 167, p. 112701, 2022.

[5] A. Arabkoohsar, H. R. Rahrabi, A. S. Alsagri, and A. A. Alrobaian, "Impact of off-design operation on the effectiveness of a low-temperature compressed air energy storage system," *Energy*, vol. 197, p. 117176, 2020.

[6] S. Zeynali, N. Rostami, A. Ahmadian, and A. Elkamel, "Robust multi-objective thermal and electrical energy hub management integrating hybrid battery-compressed air energy storage systems and plug-in-electric-vehicle-based demand response," *Journal of Energy Storage*, vol. 35, p. 102265, 2021.

[7] I. Gomes, R. Melicio, and V. Mendes, "A novel microgrid support management system based on stochastic mixed-integer linear programming," *Energy*, vol. 223, p. 120030, 2021.

[8] J. Yang and C. Su, "Robust optimization of microgrid based on renewable distributed power generation and load demand uncertainty," *Energy*, vol. 223, p. 120043, 2021.

[9] C. Guo, X. Wang, Y. Zheng, and F. Zhang, "Real-time optimal energy management of microgrid with uncertainties based on deep reinforcement learning," *Energy*, vol. 238, p. 121873, 2022.

[10] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, "A review of safe reinforcement learning: Methods, theory and applications," *arXiv preprint arXiv:2205.10330*, 2022.

[11] A. D. Garmroodi, F. Nasiri, and F. Haghighat, "Optimal dispatch of an energy hub with compressed air energy storage: A safe reinforcement learning approach," *Journal of Energy Storage*, vol. 57, p. 106147, 2023.

[12] S. Dale *et al.*, "Bp statistical review of world energy," *BP Plc: London, UK*, pp. 14–16, 2021.

[13] A. Olabi, T. Wilberforce, M. Ramadan, M. A. Abdelkareem, and A. H. Alami, "Compressed air energy storage systems: Components and operating parameters–a review," *Journal of Energy Storage*, vol. 34, p. 102000, 2021.

[14] R. Ren21 *et al.*, "Global status report," *REN21 secretariat, Paris*, 2016.

[15] A. Olabi, C. Onumaegbu, T. Wilberforce, M. Ramadan, M. A. Abdelkareem, and A. H. Al-Alami, "Critical review of energy storage systems," *Energy*, vol. 214, p. 118987, 2021.

[16] A. A. Eladl, M. I. El-Afifi, M. A. Saeed, and M. M. El-Saadawi, "Optimal operation of energy hubs integrated with renewable energy sources and storage devices considering co2 emissions," *International Journal of Electrical Power & Energy Systems*, vol. 117, p. 105719, 2020.

[17] S. Aznavi, P. Fajri, R. Sabzehgar, and A. Asrari, "Optimal management of residential energy storage systems in presence of intermittencies," *Journal of Building Engineering*, vol. 29, p. 101149, 2020.

[18] M. Mahmoud, M. Ramadan, A.-G. Olabi, K. Pullen, and S. Naher, "A review of mechanical energy storage systems combined with wind and solar applications," *Energy Conversion and Management*, vol. 210, p. 112670, 2020.

[19] L. Al-Ghussain, R. Samu, O. Taylan, and M. Fahrioglu, "Sizing renewable energy systems with energy storage systems in microgrids for maximum cost-efficient utilization of renewable energy resources," *Sustainable Cities and Society*, vol. 55, p. 102059, 2020.

[20] A. Baldinelli, L. Barelli, G. Bidini, and G. Discepoli, "Economics of innovative high capacity-to-power energy storage technologies pointing at 100% renewable micro-grids," *Journal of Energy Storage*, vol. 28, p. 101198, 2020.

[21] X. Luo, J. Wang, M. Dooner, and J. Clarke, "Overview of current development in electrical energy storage technologies and the application potential in power system operation," *Applied energy*, vol. 137, pp. 511–536, 2015.

[22] F. Barnes and J. Levine, "Large energy storage systems," *NY: Taylor & Francis Group*, 2011.

[23] G. Venkataramani, P. Parankusam, V. Ramalingam, and J. Wang, "A review on compressed air energy storage–a pathway for smart grid and polygeneration," *Renewable and sustainable energy reviews*, vol. 62, pp. 895–907, 2016.

[24] M. Budt, D. Wolf, R. Span, and J. Yan, "A review on compressed air energy storage: Basic principles, past milestones and recent developments," *Applied energy*, vol. 170, pp. 250–268, 2016.

[25] R. Machlev, N. Zargari, N. Chowdhury, J. Belikov, and Y. Levron, "A review of optimal control methods for energy storage systems-energy trading, energy balancing and electric vehicles," *Journal of Energy Storage*, vol. 32, p. 101787, 2020.

[26] J. Usaola, "Operation of concentrating solar power plants with storage in spot electricity markets," *IET renewable power generation*, vol. 6, no. 1, pp. 59–66, 2012.

[27] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703. John Wiley & Sons, 2007.

[28] W. Enang and C. Bannister, "Modelling and control of hybrid electric vehicles (a comprehensive review)," *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 1210–1239, 2017.

[29] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[30] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[32] A. Perera and P. Kamalaruban, "Applications of reinforcement learning in energy systems," *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110618, 2021.

[33] J. Men, J. Qiu, X. Chen, Z. Wang, and X. Yao, "Optimization method for a class of integrated energy system with compressed air energy storage," in *2021 40th Chinese Control Conference (CCC)*, pp. 6868–6872, IEEE, 2021.

[34] Z. Li, P. Li, J. Xia, and X. Liu, "Optimization for micro-energy grid dispatch based on non-supplementary fired compressed air energy storage aided energy hub and hybrid hyperspherical search," *Journal of Modern Power Systems and Clean Energy*, 2021.

[35] J. Bai, W. Wei, L. Chen, and S. Mei, "Modeling and dispatch of advanced adiabatic compressed air energy storage under wide operating range in distribution systems with renewable generation," *Energy*, vol. 206, p. 118051, 2020.

[36] R. Li, L. Chen, T. Yuan, and C. Li, "Optimal dispatch of zero-carbon-emission micro energy internet integrated with non-supplementary fired compressed air energy storage system," *Journal of Modern Power Systems and Clean Energy*, vol. 4, no. 4, pp. 566–580, 2016.

[37] X. Ma, C. Zhang, K. Li, F. Li, H. Wang, and J. Chen, "Optimal dispatching strategy of regional micro energy system with compressed air energy storage," *Energy*, vol. 212, p. 118557, 2020.

[38] S. Zhang, S. Miao, Y. Li, B. Yin, and C. Li, "Regional integrated energy system dispatch strategy considering advanced adiabatic compressed air energy storage device," *International Journal of Electrical Power & Energy Systems*, vol. 125, p. 106519, 2021.

[39] Y. Li, S. Miao, B. Yin, J. Han, S. Zhang, J. Wang, and X. Luo, "Combined heat and power dispatch considering advanced adiabatic compressed air energy storage for wind power accommodation," *Energy Conversion and Management*, vol. 200, p. 112091, 2019.

[40] M. Jalili, M. Sedighizadeh, and A. S. Fini, "Optimal operation of the coastal energy hub considering seawater desalination and compressed air energy storage system," *Thermal Science and Engineering Progress*, vol. 25, p. 101020, 2021.

[41] M. Jalili, M. Sedighizadeh, and A. S. Fini, "Stochastic optimal operation of a microgrid based on energy hub including a solar-powered compressed air energy storage system and an ice storage conditioner," *Journal of Energy Storage*, vol. 33, p. 102089, 2021.

[42] P. Wen, Y. Xie, L. Huo, and A. Tohidi, "Optimal and stochastic performance of an energy hub-based microgrid consisting of a solar-powered compressed-air energy storage system and cooling storage system by modified grasshopper optimization algorithm," *International Journal of Hydrogen Energy*, vol. 47, no. 27, pp. 13351–13370, 2022.

[43] J. Bai, W. Wei, L. Chen, and S. Mei, "Rolling-horizon dispatch of advanced adiabatic compressed air energy storage based energy hub via data-driven stochastic dynamic programming," *Energy Conversion and Management*, vol. 243, p. 114322, 2021.

[44] D. Yang, M. Wang, R. Yang, Y. Zheng, and H. Pandzic, "Optimal dispatching of an energy system with integrated compressed air energy storage and demand response," *Energy*, vol. 234, p. 121232, 2021.

[45] M. Jadidbonab, E. Babaei, and B. Mohammadi-ivatloo, "Cvar-constrained scheduling strategy

for smart multi carrier energy hub considering demand response and compressed air energy storage," *Energy*, vol. 174, pp. 1238–1250, 2019.

[46] M. A. Mirzaei, M. Zare Oskouei, B. Mohammadi-Ivatloo, A. Loni, K. Zare, M. Marzband, and M. Shafiee, "Integrated energy hub system based on power-to-gas and compressed air energy storage technologies in the presence of multiple shiftable loads," *IET Generation, Transmission & Distribution*, vol. 14, no. 13, pp. 2510–2519, 2020.

[47] M. Jadidbonab, A. Dolatabadi, B. Mohammadi-Ivatloo, M. Abapour, and S. Asadi, "Risk-constrained energy management of pv integrated smart energy hub in the presence of demand response program and compressed air energy storage," *IET Renewable Power Generation*, vol. 13, no. 6, pp. 998–1008, 2019.

[48] M. Z. Oskouei, B. Mohammadi-Ivatloo, M. Abapour, M. Shafiee, and A. Anvari-Moghaddam, "Strategic operation of a virtual energy hub with the provision of advanced ancillary services in industrial parks," *IEEE Transactions on Sustainable Energy*, vol. 12, no. 4, pp. 2062–2073, 2021.

[49] Y. Li, S. Miao, B. Yin, W. Yang, S. Zhang, X. Luo, and J. Wang, "A real-time dispatch model of caes with considering the part-load characteristics and the power regulation uncertainty," *International Journal of Electrical Power & Energy Systems*, vol. 105, pp. 179–190, 2019.

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[51] M. H. Alabdullah and M. A. Abido, "Microgrid energy management using deep q-network reinforcement learning," *Alexandria Engineering Journal*, vol. 61, no. 11, pp. 9069–9078, 2022.

[52] Y. Ji, J. Wang, J. Xu, X. Fang, and H. Zhang, "Real-time energy management of a microgrid using deep reinforcement learning," *Energies*, vol. 12, no. 12, p. 2291, 2019.

[53] R. Leo, R. Milton, and S. Sibi, "Reinforcement learning for optimal energy management of a solar microgrid," in *2014 IEEE global humanitarian technology conference-south asia satellite (GHTC-SAS)*, pp. 183–188, IEEE, 2014.

[54] B. V. Mbuwir, F. Ruelens, F. Spiessens, and G. Deconinck, "Battery energy management in a microgrid using batch reinforcement learning," *Energies*, vol. 10, no. 11, p. 1846, 2017.

[55] G. Muriithi and S. Chowdhury, "Optimal energy management of a grid-tied solar pv-battery microgrid: A reinforcement learning approach," *Energies*, vol. 14, no. 9, p. 2700, 2021.

[56] T. Yang, L. Zhao, W. Li, and A. Y. Zomaya, "Dynamic energy dispatch strategy for integrated energy system based on improved deep reinforcement learning," *Energy*, vol. 235, p. 121377, 2021.

[57] T. A. Nakabi and P. Toivanen, "Deep reinforcement learning for energy management in a microgrid with flexible demand," *Sustainable Energy, Grids and Networks*, vol. 25, p. 100413, 2021.

[58] Z. Zhu, Z. Weng, and H. Zheng, "Optimal operation of a microgrid with hydrogen storage based on deep reinforcement learning," *Electronics*, vol. 11, no. 2, p. 196, 2022.

[59] A. Dolatabadi, H. H. Abdeltawab, and Y. A.-R. I. Mohamed, "Deep reinforcement learning-based self-scheduling strategy for a caes-pv system using accurate sky images-based forecasting," *IEEE Transactions on Power Systems*, 2022.

[60] H. Ding, Y. Xu, B. C. S. Hao, Q. Li, and A. Lentzakis, "A safe reinforcement learning approach for multi-energy management of smart home," *Electric Power Systems Research*, vol. 210, p. 108120, 2022.

[61] D. Qiu, Z. Dong, X. Zhang, Y. Wang, and G. Strbac, "Safe reinforcement learning for real-time automatic control in a smart energy-hub," *Applied Energy*, vol. 309, p. 118403, 2022.

[62] A. Dolatabadi, H. Abdeltawab, and Y. A.-R. I. Mohamed, "A novel model-free deep reinforcement learning framework for energy management of a pv integrated energy hub," *IEEE Transactions on Power Systems*, 2022.

[63] Y. Ji, J. Wang, J. Xu, and D. Li, "Data-driven online energy scheduling of a microgrid based on deep reinforcement learning," *Energies*, vol. 14, no. 8, p. 2120, 2021.

[64] H. Li, Z. Wang, L. Li, and H. He, "Online microgrid energy management based on safe deep reinforcement learning," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2021.

[65] G. Zhang, W. Hu, D. Cao, Z. Zhang, Q. Huang, Z. Chen, and F. Blaabjerg, "A multi-agent deep reinforcement learning approach enabled distributed energy management schedule for the coordinate control of multi-energy hub with gas, electricity, and freshwater," *Energy Conversion and Management*, vol. 255, p. 115340, 2022.

[66] Q. Liang, F. Que, and E. Modiano, "Accelerated primal-dual policy optimization for safe reinforcement learning," *arXiv preprint arXiv:1802.06480*, 2018.

[67] A. S. Alsagri, A. Arabkoohsar, H. R. Rahbari, and A. A. Alrobaian, "Partial load operation analysis of trigeneration subcooled compressed air energy storage system," *Journal of Cleaner Production*, vol. 238, p. 117948, 2019.

[68] Z. Han and S. Guo, "Investigation of operation strategy of combined cooling, heating and power (cchp) system based on advanced adiabatic compressed air energy storage," *Energy*, vol. 160, pp. 290–308, 2018.

[69] P. Zhao, L. Gao, J. Wang, and Y. Dai, "Energy efficiency analysis and off-design analysis of two different discharge modes for compressed air energy storage system using axial turbines," *Renewable Energy*, vol. 85, pp. 1164–1177, 2016.

[70] M. Q. Wang and H. Gooi, "Spinning reserve estimation in microgrids," *IEEE Transactions on Power Systems*, vol. 26, no. 3, pp. 1164–1174, 2011.

[71] A. Ghasemi, M. Banejad, and M. Rahimiyan, "Integrated energy scheduling under uncertainty in a micro energy grid," *IET Generation, Transmission & Distribution*, vol. 12, no. 12, pp. 2887–2896, 2018.

[72] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[73] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proceedings of SAI Intelligent Systems Conference*, pp. 426–440, Springer, 2016.

[74] L.-J. Lin, *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[75] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[76] Y. Du and D. Wu, "Deep reinforcement learning from demonstrations to assist service restoration in islanded microgrids," *IEEE Transactions on Sustainable Energy*, vol. 13, no. 2, pp. 1062–1072, 2022.

[77] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022.

[78] L. He, N. Aouf, J. F. Whidborne, and B. Song, "Deep reinforcement learning based local planner for uav obstacle avoidance using demonstration data," *arXiv preprint arXiv:2008.02521*, 2020.

[79] I. Andresen, T. H. Trulsrud, L. Finocchiaro, A. Nocente, M. Tamm, J. Ortiz, J. Salom, A. Magyari, L. Hoes-van Oeffelen, W. Borsboom, *et al.*, "Design and performance predictions of plus energy neighbourhoods–case studies of demonstration projects in four different european climates," *Energy and Buildings*, vol. 274, p. 112447, 2022.

[80] Y. Atwa, E. El-Saadany, M. Salama, and R. Seethapathy, "Optimal renewable resources mix for distribution system energy loss minimization," *IEEE Transactions on Power Systems*, vol. 25, no. 1, pp. 360–370, 2009.

[81] M. Sedighizadeh, M. Esmaili, A. Jamshidi, and M.-H. Ghaderi, "Stochastic multi-objective

economic-environmental energy and reserve scheduling of microgrids considering battery energy storage system," *International Journal of Electrical Power & Energy Systems*, vol. 106, pp. 1–16, 2019.

[82] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

Appendix A

Operation Optimization Python Codes

Listing A.1: CAES Off-design Modeling Python Code

```
#Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#AA-CAES Specifications:
P_CAES_ch_max = 800     #Maximum Charging Power
P_CAES_dch_max = 800    #Maximum Discharging Power
gamma = 1.4             #Air Isentropic Expansion Factor
cp = 1.005              #Air Specific Heat Capacity
T_amb = 293             #Ambient Air Temperature
T_tank = 293            #Air Tank Temperature
V_tank = 1100.19        #Air Tank Volume
timestep = 3600         #1-hour Timestep
eff = 0.9               #Heat Exchanger Effectiveness
T_cold_w = 273+20       #Cold Water Tank Temperature
T_hot_w = 273+95        #Hot Water Tank Temperature

#Functions For Operation:

#Off-desing Curves:
def P_ratio_exp_1_offdesign (P_discharge):
    a1 = 1.0526686612247751e+000
    a2 = 3.3837646597101853e-002
    a3 = -1.0594350616851346e-004
```

```
    a4 = 4.3037073018992756e-007
    a = P_discharge/P_CAES_dch_max* 100
    HPT_R = a1 + a2 * (a) + a3 * (a**2) + a4 * (a**3)
    return HPT_R

def P_ratio_exp_2_offdesign (P_discharge):
    a1 = 1.8675819367444066e+000
    a2 = 1.1792419504272698e-001
    a3 = -5.0802207851639934e-004
    a4 = 1.9700035951777700e-006
    a = P_discharge/P_CAES_dch_max * 100
    LPT_R = a1 + a2 * (a) + a3 * (a**2) + a4 * (a**3)
    return LPT_R

def Isentropic_comp_offdesign (P_charge):
    a1 =  4.6948804252492010e+001
    a2 =  5.9916135114652325e-001
    a3 = -2.3775603099293894e-003
    a = P_charge/P_CAES_ch_max* 100
    eff_comp = a1 + a2 * (a) + a3 * (a**2)
    return (eff_comp/100)

def Isentropic_exp_offdesign (P_discharge):
    a1 = 4.1390729581326283e+001
    a2 = 8.6154009677822008e-001
    a3 = -5.7158107778840558e-003
    a4 = 9.5398673887267854e-006
    a = P_discharge/P_CAES_dch_max* 100
    eff_exp = a1 + a2 * (a) + a3 * (a**2) + a4 * (a**3)
    return (eff_exp/100)


#Outlet Temperature of Compressor 1:
def T_out_comp_1_offdesign (P_charge):

    T_out_comp_1 = ((T_amb) * (1 + (((P_ratio_comp_1_offdesign(P_tank))**((gamma-1)/gamma)
    )-1)/Isentropic_comp_offdesign(P_charge)))
    return T_out_comp_1

#Outlet Temperature of Compressor 2:
def T_out_comp_2_offdesign (P_charge):
```

```
    T_out_HE1 = T_out_HE1_offdesign(P_charge,P_tank)
    T_out_comp_2 = ((T_out_HE1) * (1 + (((P_ratio_comp_2_offdesign(P_tank))**((gamma-1)/
    gamma))-1)/Isentropic_comp_offdesign(P_charge)))

    return T_out_comp_2

#Outlet Temperature of Heat Exchanger 1:
def T_out_HE1_offdesign (P_charge,P_tank):
    T_out_HE1 = T_out_comp_1_offdesign (P_charge, P_tank) * (1-eff) + T_cold_w * eff

    return T_out_HE1

#Charging Mass Flow Rate Off Design:
def charging_mfr_offdesign (P_charge, P_tank):
    work = cp * (T_out_comp_1_offdesign(P_charge, P_tank) - T_amb +
    T_out_comp_2_offdesign(P_charge,P_tank) -
                        T_out_HE1_offdesign (P_charge,P_tank))
    charging_mfr = P_charge/work
    return charging_mfr

#Charging Heat Generation Off Design:
def charging_heat_offdesign (P_charge, P_tank):
    heat = charging_mfr_offdesign (P_charge, P_tank) * (T_out_comp_1_offdesign(P_charge,
    P_tank) - T_out_HE1_offdesign (P_charge,P_tank))
    return heat

#Outlet Temperature of Expander 1:
def T_out_exp_1_offdesign (P_discharge):
    T_out_HE3 = T_out_HE3_offdesign (P_discharge)
    T_out_exp_1 = ((T_out_HE3) * (1 - (Isentropic_exp_offdesign(P_discharge) *
                                       (1 - ((1/P_ratio_exp_1_offdesign(P_discharge))**((
    gamma-1)/gamma))))))
    return T_out_exp_1

#Outlet Temperature of Expander 2:
def T_out_exp_2_offdesign (P_discharge):
    T_out_HE4 = T_out_HE4_offdesign (P_discharge)
    T_out_exp_2 = ((T_out_HE4) * (1 - (Isentropic_exp_offdesign(P_discharge) *
                                       (1 - ((1/P_ratio_exp_2_offdesign(P_discharge))**((
    gamma-1)/gamma))))))
    return T_out_exp_2
```

```
102
103 #Outlet Temperature of Heat Exchanger 3:
104 def T_out_HE3_offdesign (P_discharge):
105     T_out_HE3 = T_amb * (1-eff) + T_hot_w * eff
106
107     return T_out_HE3
108
109 #Outlet Temperature of Heat Exchanger 4:
110 def T_out_HE4_offdesign (P_discharge):
111     T_out_HE4 = T_out_exp_1_offdesign (P_discharge) * (1-eff) + T_hot_w * eff
112
113     return T_out_HE4
114
115 #Updating the Pressure of Air Storage:
116 def new_pressure_offdesign(P_charge,P_discharge,P_tank):
117     timestep = 3600
118     old_density = P_tank * 100000 / (287 * T_tank)
119     new_density = (old_density * V_tank + (charging_mfr_offdesign(P_charge,P_tank)-
120                                            discharging_mfr_offdesign(P_discharge))*
    timestep)/V_tank
121
122     new_pressure = new_density * (T_tank)*287 /100000
123
124     return new_pressure
125
126 #Discharging Mass Flow Rate Off Design:
127 def discharging_mfr_offdesign(P_discharge):
128     output_work = cp * (T_out_HE3_offdesign (P_discharge) - T_out_exp_1_offdesign(
    P_discharge) +  T_out_HE4_offdesign (P_discharge) -T_out_exp_2_offdesign(P_discharge))
129
130     discharging_mfr = P_discharge/output_work
131
132     return discharging_mfr
133
134 #Discharging Cooling Generation Off Design:
135 def discharging_cool_offdesign(P_discharge):
136     cool = discharging_mfr_offdesign(P_discharge) * cp * (T_out_exp_2_offdesign(
    P_discharge))
137     return cool
138
139 #Extra Functions For Linearizing Cooling Power Generation:
```

```
def discharging_ambient_cool_offdesign(P_discharge):
    cool = discharging_mfr_offdesign(P_discharge) * cp
    return cool

def cooling_load_off_design(P_discharge,t_amb):
    cool = discharging_mfr_offdesign(P_discharge) * cp * (t_amb - T_out_exp_2_offdesign(
    P_discharge))
    return cool



#Linearizing The Operation:

# x is the charing and discharging power nad y is the mass flow rate
x = np.arange(80,800.1,0.1)
y1 = list()
for i in x:
    y1.append(discharging_mfr(i))

y2 = list()
for i in x:
    y2.append(discharging_mfr_offdesign(i))

x_ratio = [i/P_CAES_ch_max *100 for i in x]


from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=1, include_bias=False)
poly_features = poly.fit_transform(x.reshape(-1, 1))
from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y2)
poly_reg_y_predicted_lin = poly_reg_model.predict(poly_features)
from sklearn.metrics import mean_squared_error
poly_reg_rmse_lin = np.sqrt(mean_squared_error(y2, poly_reg_y_predicted_lin))

#Reading coefficient and intercept of linear regression
coeff = poly_reg_model.coef_
intercept = poly_reg_model.intercept_
```

Listing A.2: MILP Python Code

```
#Importing Libraries
import numpy as np
import pandas as pd
import pyomo.environ as pyo
from pyomo.environ import *
from pyomo.opt import SolverFactory
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import cycle, islice
import random

#Reading the consumption, PV, and Wind data
data = pd.read_csv("Dataset.csv").drop('Unnamed: 0', axis=1)

#Defining the gas and electricity price
buy_price = 24*[0]
buy_price[0:7] = 7*[3.99]
buy_price[22] = 3.99
buy_price[23] = 3.99
buy_price[11:19] = 8*[19.99]
buy_price[7:11] = 4*[11.99]
buy_price[19:22] = 3*[11.99]

sell_price = [i/2 for i in buy_price]

gas_price = 3.21

#Defining AA-CAES functions:
P_CAES_ch_max = 800
P_CAES_dch_max = 800
gamma = 1.4
cp = 1.005
T_amb = 298
T_tank = 298
V_tank = 1100.19

#Discharge Mass Flow Rate Linear Model
def discharging_mfr(P_discharge):
    term2 = 0.00278671 * P_discharge + 1.2292957997285587
```

```
40
41      return term2
42
43  #Discharge Cooling Load Linear Model
44  def cooling_load_predicted(P_discharge,t_amb):
45      term1 = 0.39792147*P_discharge + 432.3243866973607
46      term2 = 0.00278671 * P_discharge + 1.2292957997285587
47      cooling_load = term1 - term2 *t_amb
48
49      return -1 *cooling_load
50
51  #Charging Mass Flow Rate Linear Model
52  def charging_mfr(P_discharge):
53      term2 = 0.0017455 * P_discharge -0.11658996400
54
55      return term2
56
57  #Updating the air storage pressure
58  def new_pressure(P_charge,P_discharge,P_tank,bin_charge,bin_discharge):
59      timestep = 3600
60      old_density = P_tank * 100000 / (287 * T_tank)
61      new_density = (old_density * V_tank + (charging_mfr(P_charge)*bin_charge-
62                                             discharging_mfr(P_discharge)*bin_discharge)*
      timestep)/V_tank
63
64      new_pressure = new_density * (T_tank)*287 /100000
65
66      return new_pressure
67
68  #Defining empty lists and dataframes for optimal policy:
69  df_optimal = pd.DataFrame()
70  picked_days = []
71  operation_costs = []
72
73  #Defining Training and Test Days:
74  test_days = np.arange(5,366,5)
75  all_days = np.arange(1,366,1)
76  training_days = [i for i in all_days if i not in test_days]
77
78  #Running the Model For Training Days:
79  for a in training_days:
```

```
80
81      #Reading Data For Day (a):
82      t = 24
83      e_load = data[data["Day"] == a]["Electricity"].values
84      h_load = data[data["Day"] == a]["Heating"].values
85      c_load = data[data["Day"] == a]["Cooling"].values
86      wind_data = data[data["Day"] == a]["WT_power"].values
87      PV_data = data[data["Day"] == a]["PV power"].values
88      temp = data[data["Day"] == a]["Temperature"].values
89
90      # Energy Hub Equipment
91      t= range(24)
92      t2 = range(25)
93
94      #Micro Turbine:
95      P_mt_max = 1500       # Maximum Electricity Power Generated by Micro Turbine
96      P_mt_min = 200        # Minimum Electricity Power Generated by Micro Turbine
97      n_mt = 0.35           # Electrical Efficiency of Micro Turbine
98      n_hru = 0.42          # Efficiency of Heat Recovery Unit
99      H_hru_max =P_mt_max*(n_hru/n_mt)
100     H_hru_min =P_mt_min*(n_hru/n_mt)
101     gas_max = 10000
102     gas_min = 0
103
104     # Gas Boiler
105     H_gb_min = 0          # Minimum Heating Power Generated by Gas Boiler
106     H_gb_max = 1500       # Maximum Heating Power Generated by Gas Boiler
107     n_gb = 0.8            # Efficiency of Gas Boiler
108
109     #Grid System
110     P_grid_min = 0        # Minimum Electricity Imported/Exported from Grid System
111     P_grid_max = 2000     # Maximum Electricity Imported/Exported from Grid System
112
113     #Absorption Chiller
114     H_ac_max = 1000/0.9   # Maximum Heating Power Consumed by Absorption Chiller
115     H_ac_min = 0          # Minimum Heating Power Consumed by Absorption Chiller
116     n_ac = 0.9            # Efficiency of Absorption Chiller
117
118     #Electric Chiller
119     C_ec_max = 1500       # Maximum Cooling Power Generated by Electric Chiller
120     C_ec_min = 0          # Minimum Cooling Power Generated by Electric Chiller
```

```
    COP_ec = 3            # COP of electric chiller

    #Heat Pump:
    H_hp_max = 3000       # Maximum Heating Power Generated by Heat Pump
    C_hp_max = 3000       # Maximum Cooling Power Generated by Heat Pump
    H_hp_min = 0          # Minimum Heating Power Generated by Heat Pump
    C_hp_min = 0          # Minimum Cooling Power Generated by Heat Pump
    COP_hp   = 3          # COP of Heat Pump

    #AA-CAES
    P_CAES_ch_max = 800       # Maximum Charging Power of AA-CAES
    P_CAES_ch_min = 80        # Minimum Charging Power of AA-CAES

    P_CAES_dch_max = 800      # Maximum Discharging Power of AA-CAES
    P_CAES_dch_min = 80       # Minimum Discharging Power of AA-CAES

    #Decision Variables:
    model = pyo.ConcreteModel()
    model.time = pyo.Set(initialize=(i for i in t))
    model.timestep = pyo.Param(model.time,initialize=3600)
    model.time_soc = pyo.Set(initialize=(i for i in t2))

    #Micro Turbine
    model.P_gt = pyo.Var(model.time,bounds=(0,P_gt_max))
    model.bin_P_gt = pyo.Var(model.time, bounds=(0,1), domain=Binary)
    model.H_hru = pyo.Var(model.time, bounds= (0,H_hru_max))

    #Grid System
    model.P_grid_buy = pyo.Var(model.time, bounds= (P_grid_min,P_grid_max))
    model.P_grid_sell = pyo.Var(model.time, bounds= (P_grid_min,P_grid_max))
    model.bin_grid_buy = pyo.Var(model.time, bounds=(0,1),domain=Binary)
    model.bin_grid_sell = pyo.Var(model.time, bounds=(0,1),domain=Binary)

    #AA-CAES System
    model.P_CAES_ch = pyo.Var(model.time,bounds = (0,P_CAES_ch_max))
    model.P_CAES_dch = pyo.Var(model.time,bounds = (0,P_CAES_dch_max))
    model.Pressure_CAES = pyo.Var(model.time_soc, bounds=(45,69))
    model.bin_CAES_ch = pyo.Var(model.time, bounds=(0,1), domain=Binary)
    model.bin_CAES_dch = pyo.Var(model.time, bounds=(0,1), domain=Binary)
    model.heat_caes = pyo.Var(model.time,bounds = (0,P_CAES_ch_max))
    model.cold_caes = pyo.Var(model.time,bounds = (0,P_CAES_ch_max))
```

```
    model.maximum_cold_caes = pyo.Var(model.time,bounds = (0,P_CAES_ch_max))
    model.maximum_heat_caes = pyo.Var(model.time,bounds = (0,P_CAES_ch_max))

    #Gas Boiler
    model.H_gb = pyo.Var(model.time,bounds = (H_gb_min,H_gb_max))

    #Absorption Chiller
    model.H_ac = pyo.Var(model.time,bounds = (H_ac_min,H_ac_max))
    model.C_ac = pyo.Var(model.time,bounds = (n_ac*H_ac_min,n_ac*H_ac_max))

    #Heat Pump
    model.H_hp = pyo.Var(model.time,bounds = (H_hp_min,H_hp_max))
    model.P_hp_H = pyo.Var(model.time,bounds = (H_hp_min/COP_hp,H_hp_max/COP_hp))
    model.P_hp_C = pyo.Var(model.time,bounds = (C_hp_min/COP_hp,C_hp_max/COP_hp))
    model.C_hp = pyo.Var(model.time,bounds = (C_hp_min,C_hp_max))
    model.bin_HP_H = pyo.Var(model.time, bounds=(0,1), domain=Binary)
    model.bin_HP_C = pyo.Var(model.time, bounds=(0,1), domain=Binary)

    #Re-assiging for simplification
    bin_P_mt=model.bin_P_mt

    P_hp_H = model.P_hp_H
    P_hp_C = model.P_hp_C
    H_hp = model.H_hp
    C_hp = model.C_hp
    bin_HP_H = model.bin_HP_H
    bin_HP_C =model.bin_HP_C

    C_ac = model.C_ac
    H_hru = model.H_hru
    timestep = model.timestep
    P_mt = model.P_mt
    P_grid_buy = model.P_grid_buy
    P_grid_sell = model.P_grid_sell
    H_gb = model.H_gb
    H_ac = model.H_ac

    bin_grid_buy = model.bin_grid_buy
    bin_grid_sell = model.bin_grid_sell


```

```
    P_CAES_ch =model.P_CAES_ch
    P_CAES_dch=model.P_CAES_dch
    Pressure_CAES=model.Pressure_CAES
    bin_CAES_ch=model.bin_CAES_ch
    bin_CAES_dch=model.bin_CAES_dch
    cold_caes = model.cold_caes
    heat_caes = model.heat_caes

    #Constraints:

    #AA-CAES:
    def CAES_charge_lower(model,i):
        return model.bin_CAES_ch[i]*P_CAES_ch_min <= model.P_CAES_ch[i]
    model.CAES_charge_limit_lower = Constraint(model.time, rule=CAES_charge_lower)

    def CAES_charge_upper(model,i):
        return model.P_CAES_ch[i] <= model.bin_CAES_ch[i]*P_CAES_ch_max
    model.CAES_charge_limit_upper = Constraint(model.time, rule=CAES_charge_upper)

    def CAES_discharge_lower(model,i):
        return model.bin_CAES_dch[i]*P_CAES_dch_min <= model.P_CAES_dch[i]
    model.CAES_discharge_limit_lower = Constraint(model.time, rule=CAES_discharge_lower)

    def CAES_discharge_upper(model,i):
        return model.P_CAES_dch[i] <= model.bin_CAES_dch[i]*P_CAES_dch_max
    model.CAES_discharge_limit_upper = Constraint(model.time, rule=CAES_discharge_upper)

    def CAES_operation_mode(model,i):
        return bin_CAES_ch[i]+bin_CAES_dch[i]<=1
    model.CAES_operation_mode_limit = Constraint(model.time, rule=CAES_operation_mode)

    def SOC_initial(model):
        return Pressure_CAES[0] == 57
    model.Pressure_CAES_initial = Constraint(rule= SOC_initial)

    def SOC_final(model):
        return Pressure_CAES[24] == Pressure_CAES[0]
    model.Pressure_CAES_final = Constraint(rule= SOC_final)


    model.Pressure_CAES_new = pyo.ConstraintList()
```

```
    for i in range(24):
        model.Pressure_CAES_new.add(expr = Pressure_CAES[i+1] ==
                                    new_pressure(P_CAES_ch[i],P_CAES_dch[i],Pressure_CAES[
    i],bin_CAES_ch[i],bin_CAES_dch[i]))

    def CAES_heat_mode(model,i):
        return heat_caes[i] <= P_CAES_ch[i]*bin_CAES_ch[i]*0.38
    model.CAES_heat_mode = Constraint(model.time, rule=CAES_heat_mode)

    def CAES_cold_mode(model,i):
        return cold_caes[i] <= cooling_load_predicted(P_CAES_dch[i],temp[i] + 273)*model.
    bin_CAES_dch[i]
    model.CAES_cold_mode = Constraint(model.time, rule=CAES_cold_mode)

    def CAES_maximum_heat_caes(model,i):
        return model.maximum_heat_caes[i] == P_CAES_ch[i]*bin_CAES_ch[i]*0.38
    model.CAES_maximum_heat_caes = Constraint(model.time, rule=CAES_maximum_heat_caes)


    def CAES_maximum_cold_caes(model,i):
        return model.maximum_cold_caes[i] == cooling_load_predicted(P_CAES_dch[i],temp[i]
    + 273)*model.bin_CAES_dch[i]
    model.CAES_maximum_cold_caes = Constraint(model.time, rule=CAES_maximum_cold_caes)


    #Grid Constraints
    def Grid_buy_limit(model,i):
        return P_grid_buy[i]<= bin_grid_buy[i]*P_grid_max
    model.Grid_buy_limit = Constraint(model.time, rule=Grid_buy_limit)

    def Grid_sell_limit(model,i):
        return P_grid_sell[i]<= bin_grid_sell[i]*P_grid_max
    model.Grid_sell_limit = Constraint(model.time, rule=Grid_sell_limit)

    def Grid_exchange_mode(model,i):
        return bin_grid_sell[i]+bin_grid_buy[i]<=1
    model.Grid_exchange_mode_limit = Constraint(model.time, rule=Grid_exchange_mode)

    #Gas Network Constraints:
    def gas_max_use(model,i):
        return (P_mt[i] /(n_mt) + H_gb[i]/n_gb)<=gas_max
```

```
282     model.gas_max_use = Constraint(model.time, rule=gas_max_use)
283
284     def gas_min_use(model,i):
285         return (P_mt[i] /(n_mt) + H_gb[i]/n_gb)>=gas_min
286     model.gas_min_use = Constraint(model.time, rule=gas_min_use)
287
288
289     #Heat Recovery Unit Constraints:
290     def HRU_convert(model,i):
291         return H_hru[i] == P_mt[i]*(n_hru/n_mt)
292     model.HRU_heat = Constraint(model.time, rule=HRU_convert)
293
294     #AC Constraints:
295     def AC_convert(model,i):
296         return C_ac[i] == n_ac*H_ac[i]
297     model.AC_convert = Constraint(model.time, rule=AC_convert)
298
299     #Heat Pump Constraints:
300     def HP_convert_1(model,i):
301         return P_hp_H[i] == H_hp[i]/COP_hp
302     model.HP_convert_1 = Constraint(model.time, rule=HP_convert_1)
303
304     def HP_convert_2(model,i):
305         return P_hp_C[i] == C_hp[i]/COP_hp
306     model.HP_convert_2 = Constraint(model.time, rule=HP_convert_2)
307
308     def HP_operation(model,i):
309         return bin_HP_C[i]+bin_HP_H[i]<=1
310     model.HP_operation = Constraint(model.time, rule=HP_operation)
311
312     def HP_heating(model,i):
313         return model.H_hp[i] <= model.bin_HP_H[i]*H_hp_max
314     model.HP_heating = Constraint(model.time, rule=HP_heating)
315
316     def HP_cooling(model,i):
317         return model.C_hp[i] <= model.bin_HP_C[i]*C_hp_max
318     model.HP_cooling = Constraint(model.time, rule=HP_cooling)
319
320     #Micro Turbine Constraints:
321     def MT_limit_upper(model,i):
322         return P_mt[i] <= bin_P_mt[i]*P_mt_max
```

```
model.MT_limit_upper = Constraint(model.time, rule=MT_limit_upper)
def MT_limit_lower(model,i):
    return P_mt[i] >= bin_P_mt[i]*P_mt_min
model.MT_limit_lower = Constraint(model.time, rule=MT_limit_lower)

#Energy Balance Constraints:
def elec_balance(model,i):
    return e_load[i] + P_CAES_ch[i] + P_hp_C[i] + P_hp_H[i] +P_grid_sell[i] == (
wind_data[i] + PV_data[i] + P_grid_buy[i] + P_mt[i] +  P_CAES_dch[i])
model.elec_balance = pyo.Constraint(model.time, rule = elec_balance)

def heat_balance(model,i):
    return H_hp[i] + H_gb[i] + H_hru[i] + heat_caes[i] == h_load[i] +H_ac[i]
model.heat_balance = pyo.Constraint(model.time, rule = heat_balance)


def cold_balance(model,i):
    return C_ac[i]+ C_hp[i] + cold_caes[i] == c_load[i]
model.cold_balance = pyo.Constraint(model.time, rule = cold_balance)


#Objective Function :
def objective_function(model):
    return sum(P_grid_buy[i]*(buy_price[i])- (P_grid_sell[i] * sell_price[i]) +
                (gas_price) * (P_mt[i] /(n_mt) + H_gb[i]/n_gb) for i in model.time)
model.objective_function = pyo.Objective(rule = objective_function, sense=pyo.minimize
)

#Solver:

opt = SolverFactory('gurobi')
results = opt.solve(model,tee=True)
result = results.Problem._list[0].lower_bound

#Reading Results
df = pd.DataFrame()
for v in model.component_objects(pyo.Var, active=True):
    if v == Pressure_CAES:
        Pressure_values = [value(v[key]) for key in v]
        df[str(v)] = Pressure_values[0:24]
        pass
```

```
362
363         else:
364             optimal_values = [value(v[key]) for key in v]
365             df[str(v)]=optimal_values
366
367     grid_buy=df["P_grid_buy"]
368     grid_sell=df["P_grid_sell"]
369     grid = []
370     charge_caes = df["P_CAES_ch"]
371     discharge_caes = df["P_CAES_dch"]
372     CAES = []
373     for i in range(24):
374         if grid_buy[i]==0:
375             grid.append(grid_sell[i]*-1)
376         else:
377             grid.append(grid_buy[i])
378         if round(discharge_caes[i])==0:
379             CAES.append(charge_caes[i]*-1)
380         else:
381             CAES.append(discharge_caes[i])
382
383
384     df["Grid"] = grid
385     df["CAES"] = CAES
386     df["P_hp"] = df["P_hp_C"]* -1+ df["P_hp_H"]
387     df["Wind Power"] = wind_data
388     df["PV Power"] = PV_data
389     df["Electricity"] = e_load
390     df["Heating"] = h_load
391     df["Cooling"] = c_load
392     df["Temperature"] = temp
393
394     df.drop(axis=1, columns=['P_grid_buy', 'P_grid_sell', 'bin_grid_buy',
395            'bin_grid_sell',
396            'P_CAES_ch', 'P_CAES_dch','bin_CAES_ch', 'bin_CAES_dch',
397            ],inplace=True)
398
399     df["timestep"] = np.arange(0,24,1)
400     df["Day"] = 24*[a]
401     df_optimal = pd.concat([df_optimal, df])
402
```

```
    days_pd.append(a)
    operation_costs.append(result)

#Converting the cooling and heating energy of AA-CAES to action (based on maximum heating and cooling)
cold_ratio = []
for i in range(len(df_optimal["cold_caes"])):
    if df_optimal["maximum_cold_caes"].values[i] == 0:
        cold_ratio.append(0)
    else:
        cold_ratio.append(df_optimal["cold_caes"].values[i]/df_optimal["maximum_cold_caes"].values[i])

heat_ratio = []
for i in range(len(df_optimal["heat_caes"])):
    if df_optimal["maximum_heat_caes"].values[i] == 0:
        heat_ratio.append(0)
    else:
        heat_ratio.append(df_optimal["heat_caes"].values[i]/df_optimal["maximum_heat_caes"].values[i])

heat_cold_ratio = np.array(cold_ratio)+np.array(heat_ratio)
df_optimal["caes_heat_cool"] = heat_cold_ratio
```

Appendix B

Reinforcement Learning Python Codes

Listing B.1: EH CMDP Environment Python Code

```
#Importing Libraries:
import math
import random
from typing import Optional
import pandas as pd
import numpy as np
import gym
from gym import spaces
from gym.utils import seeding

#Reading Data:
data = pd.read_csv("Training Set.csv")
all_data = pd.read_csv("All_answers.csv")
days_to_train = data["Day"].unique()
all_days = all_data["Day"].unique()

#maximum and minimum:
max_elec = max(all_data["Electricity"])
min_elec = min(all_data["Electricity"])
max_heat_load = max(all_data["Heating"])
min_heat_load = min(all_data["Heating"])
max_cold_load = max(all_data["Cooling"])
min_cold_load = min(all_data["Cooling"])
max_pv = max(all_data["PV Power"])
min_pv = min(all_data["PV Power"])
```

```
max_wind = max(all_data["Wind Power"])
min_wind = min(all_data["Wind Power"])
max_temp = max(all_data["Temperature"])
min_temp = min(all_data["Temperature"])

#Defining The CAES Object:
class CAES_Class(object):

    cp = 1.005
    P_amb = 1
    T_amb = 298
    P_min = 42
    P_max = 72
    time_step = 3600
    T_tank=298
    R = 287
    gamma = 1.4

    P_CAES_ch_max = 800
    P_CAES_dch_max = 800
    gamma = 1.4
    cp = 1.005
    T_amb = 298
    T_tank = 298
    V_tank = 1100.19

    def __init__(self,initial=57):
        self.pressure = initial
        self.SOC = round((self.pressure-self.P_min)/(self.P_max - self.P_min),ndigits=3)

    def discharging_mfr(self,P_discharge):
        term2 = 0.00278671 * P_discharge + 1.2292957997285587

        return term2


    def cooling_load_predicted(self,P_discharge,t_amb):
        term1 = 0.39792147*P_discharge + 432.3243866973607
        term2 = 0.00278671 * P_discharge + 1.2292957997285587
        cooling_load = term1 - term2 *t_amb

```

```
67          return -1 *cooling_load
68
69      def charging_mfr(self,P_discharge):
70          term2 = 0.0017455 * P_discharge -0.11658996400
71
72          return term2
73
74      def new_pressure(self,P_charge,P_discharge,P_tank,bin_charge,bin_discharge):
75          timestep = 3600
76          old_density = P_tank * 100000 / (287 * self.T_tank)
77          new_density = (old_density * self.V_tank + (self.charging_mfr(P_charge)*bin_charge
    -
78                                                  self.discharging_mfr(P_discharge)*
    bin_discharge)*self.time_step)/self.V_tank
79
80          new_pressure = new_density * (self.T_tank)*287 /100000
81
82          return new_pressure
83
84      def update(self,P_charge, P_discharge):
85          if P_charge >=80:
86              self.pressure = self.new_pressure(P_charge,0,self.pressure,1,0)
87              return self.pressure
88          elif P_discharge >=80:
89              self.pressure = self.new_pressure(0,P_discharge,self.pressure,0,1)
90              return self.pressure
91          else:
92              self.pressure = self.new_pressure(0,0,self.pressure,0,0)
93              return self.pressure
94
95
96      def Charge(self, P_ch, T_amb):
97
98          self.update(P_ch,0)
99
100         self.SOC = round((self.pressure-self.P_min)/(self.P_max - self.P_min),ndigits=3)
101         if P_ch>=80:
102             P_ch_heat = P_ch * 0.38
103         else:
104             P_ch_heat = 0
105
```

```
        return (P_ch_heat)


    def Discharge(self,P_dch, T_amb):

        self.update(0,P_dch)

        self.SOC = round((self.pressure-self.P_min)/(self.P_max - self.P_min),ndigits=3)
        if P_dch>=80:
            P_dch_cool = self.cooling_load_predicted(P_dch,T_amb + 273)
        else:
            P_dch_cool = 0
        return P_dch_cool


#Defining Electricity and Gas Prices:
elec_price = 25*[0]
elec_price[0:7] = 7*[3.99]
elec_price[22] = 3.99
elec_price[23] = 3.99
elec_price[11:19] = 8*[19.9]
elec_price[7:11] = 4*[11.99]
elec_price[19:22] = 3*[11.99]
elec_price[24] = 3.99
gas_price = 3.21

#Defining The Environment
class Train_Env (gym.Env):
"""
  ###Decription

  The environment is consisted of Grid, WT, PV, MT, HP, AC, GB, CAES and loads.

  The state space is : Electricity Load, Heating Load, Cooling Load, PV Power, WT Power,
    SOC of AA-CAES, Buy Price Electricity, Ambient Temp, Timestep = 9 Elements

  The action space is : CAES, MT, HP, AC, GB, Heat/Cool Ratio of CAES

"""

  # Defining The Initial Parameters of Energy Hub
```

```
    def __init__ (self, p_grid_max=2000, p_caes_max=800, p_mt_max=2000,
                p_ec_max = 1000, p_hp_max = 1000, h_gb_max = 2000,
                n_mt = 0.35, h_ac_max = 1000, n_hr = 0.42, n_gb = 0.8, cop_hp = 3, cop_ac
    = 0.9,
                iterations = 24,c = 1000):

        self.penalty_coeff = c            #Penalty Coefficient
        self.iterations = iterations      #Iterations
        self.p_grid_max = p_grid_max      #Grid Max Power
        self.p_caes_max = p_caes_max      #CAES Max Power
        self.p_mt_max = p_mt_max          #MT Max Power
        self.p_hp_max = p_hp_max          #HP Max Power
        self.h_gb_max = h_gb_max          #GB Max Power
        self.h_ac_max = h_ac_max          #AC Max Power
        self.n_hr = n_hr                  #Heat Recovery Efficiency
        self.n_gb = n_gb                  #GB Efficiency
        self.cop_hp = cop_hp              #COP of HP
        self.cop_ac = cop_ac              #COP of AC
        self.time_step = 0                #Timestep
        self.n_mt = n_mt                  #MT Electric Efficiency
        self.op_cost = []                 #List for Operational Costs
        self.elecs_hist = [[],[],[],[]]   #Lists for Storing Electricity Components Powers
        self.heats_hist = [[],[],[],[],[]]#Lists for Storing Heating Components Powers
        self.colds_hist = [[],[],[]]      #Lists for Storing Cooling Components Powers
        self.picked_days = []             #List for Picked Random Days
        self.penalty_SOC_caes=[]          #List to Store the SOC Violation
        self.penalty_c_load = []          #List to Store Cooling Violation
        self.penalty_h_load = []          #List to Store Heating Violation
        self.penalty_p_grid=[]            #List to Store Grid Violation
        self.caes_SOC_hist = []           #List to Store SOC History
        self.penalty_cost_1 = []          #List to Store Penalty Cost 1
        self.penalty_cost_2 = []          #List to Store Penalty Cost 2
        self.penalty_cost_3 = []          #List to Store Penalty Cost 3
        self.penalty_cost_4 = []          #List to Store Penalty Cost 4


    #Defining State and Action Spaces:
    self.action_space = spaces.Box(low = np.array([-1,-1,0,0,0,0]), high = np.array
    ([1,1,1,1,1,1]), dtype = np.float32,
                                    shape = (6,))
    self.observation_space = spaces.Box(low = -np.inf, high = np.inf, dtype=np.float32,
```

```
                                        shape = (9,))


    #Creating an AA-CAES System
      def _create_caes (self):
          self.caes = CAES_Class()
          self.initial_SOC_caes = self.caes.SOC
          return self.caes

      #Defining a Function to Output the Current State of The Environment
      def _build_state (self):

          SOC_caes = self.caes.SOC

          self.wind_data = self.train["Wind Power"].reset_index(drop=True)
          wind = (self.wind_data[self.time_step] - min_wind)/ (max_wind- min_wind)

          self.temp = self.train["Temperature"].reset_index(drop=True)
          temp = (self.temp[self.time_step]- min_temp)/ (max_temp - min_temp)

          self.PV_data = self.train["PV Power"].reset_index(drop=True)
          pv = (self.PV_data[self.time_step]- min_pv)/ (max_pv - min_pv)

          self.elec_load = self.train["Electricity"].reset_index(drop=True)
          e_load = (self.elec_load[self.time_step]- min_elec)/ (max_elec - min_elec)

          self.heat_load = self.train["Heating"].reset_index(drop=True)
          h_load = (self.heat_load[self.time_step]- min_heat_load)/ (max_heat_load -
      min_heat_load)

          self.cold_load = self.train["Cooling"].reset_index(drop=True)
          c_load = (self.cold_load[self.time_step]- min_cold_load)/ (max_cold_load -
      min_cold_load)

          buy_price = (elec_price[self.time_step]- min(elec_price))/ (max (elec_price)- min(
      elec_price))

          state = np.array([SOC_caes, wind, pv, e_load, h_load, c_load, temp, buy_price,
      self.time_step/24])

          return state
```

```
222
223     #Defining a Function to Calculate the Operational Cost of a timestep
224     def operation_cost(self, p_grid, p_mt, h_gb):
225         grid_prices_1 = 0.75 * elec_price[self.time_step]
226         grid_prices_2 = 0.25 * elec_price[self.time_step]
227         op_cost_grid = grid_prices_1 * p_grid + grid_prices_2 * abs(p_grid)
228         op_cost_gb = h_gb/self.n_gb * gas_price
229         op_cost_mt = (p_mt / self.n_mt) * gas_price
230 
231         return (op_cost_mt + op_cost_grid + op_cost_gb)/100
232 
233     #Defining Step Function for Performing an Action and Move to Next State
234     def step(self, action):
235 
236         #Initializing reward and costs
237         reward = 0
238         cost_1 = 0
239         cost_2 = 0
240         cost_3 = 0
241         cost_4 = 0
242 
243         #Reading the data from an action
244         caes_action = action[0]
245         hp_action = action[1]
246         mt_action = action[2]
247         ac_action = action[3]
248         gb_action = action[4]
249         caes_heat_cool_action = action[5]
250 
251         #Converting the action to power terms
252         p_caes = self.p_caes_max * caes_action
253         p_mt = self.p_mt_max * mt_action
254         p_hp = self.p_hp_max * hp_action
255         h_hr = p_mt * (self.n_hr / self.n_mt)
256         h_ac = ac_action * self.h_ac_max
257         c_ac = h_ac * self.cop_ac
258         h_gb = gb_action * self.h_gb_max
259 
260         #Making Sure that Power of MT is less than its Minimum value = 200
261         if p_mt<199.9999:
262             p_mt=0
```

```
            h_hr=0

        #Making Sure that Power of CAES is less than its Minimum value = 80:
        if abs(p_caes)<79.999:
            p_caes = 0

        #CAES Charging or Discharging:
        if (p_caes > 0):
            c_caes = self.caes.Discharge(p_caes,self.temp[self.time_step]) *
    caes_heat_cool_action

        if c_caes <0:
            h_caes = abs(c_caes) * caes_heat_cool_action
            c_caes = 0

        else:
            h_caes = 0


        if (p_caes < 0):
            h_caes = self.caes.Charge(-p_caes,self.temp[self.time_step]) *
    caes_heat_cool_action
            c_caes = 0

        else:
            c_caes = 0
            h_caes = 0
            pass

        #Heat Pump Operation COoling or Heating:
        if p_hp <0:
            c_hp = abs(p_hp) * self.cop_hp
            h_hp =0
        else:
            h_hp = p_hp * self.cop_hp
            c_hp = 0


        #Heating and Cooling Power Generation:
        cool_generated = c_hp + c_caes + c_ac

```

```
302         heat_generated = h_gb + h_hr + h_hp + h_caes
303 
304         #Calculating Power Exchanged With Grid:
305         p_grid = self.elec_load[self.time_step] + abs(p_hp) - (self.wind_data[self.
    time_step] + self.PV_data[self.time_step]+ p_caes + p_mt)
306 
307 
308         #Creating Dataframes of Powers
309         self.elecs_hist[0].append(p_grid)
310         self.elecs_hist[1].append(p_caes)
311         self.elecs_hist[2].append(p_mt)
312         self.elecs_hist[3].append(abs(p_hp))
313 
314 
315         self.heats_hist[0].append(h_gb)
316         self.heats_hist[1].append(h_hr)
317         self.heats_hist[2].append(h_hp)
318         self.heats_hist[3].append(h_ac)
319         self.heats_hist[4].append(h_caes)
320 
321 
322         self.colds_hist[0].append(c_ac)
323         self.colds_hist[1].append(c_hp)
324         self.colds_hist[2].append(c_caes)
325 
326         self.caes_SOC_hist.append(self.caes.SOC)
327 
328 
329         # Adding operation cost to reward function
330         reward -= self.operation_cost(p_grid, p_mt, h_gb)
331         self.op_cost.append(self.operation_cost(p_grid, p_mt, h_gb))
332 
333         #Constraints Calculation
334         u_caes_up = 0
335         u_caes_low = 0
336         u_grid_up = 0
337         u_grid_low = 0
338 
339         if self.caes.SOC < 0.2:
340             u_caes_low = 1
341 
```

```
        elif self.caes.SOC > 0.9:
            u_caes_up = 1

        if p_grid > self.p_grid_max:
            u_grid_up = 1

        elif p_grid < -self.p_grid_max:
            u_grid_low = 1

        penalty_caes = (abs(self.caes.SOC - 0.9)*u_caes_up + abs(self.caes.SOC - 0.2)*
    u_caes_low)*self.penalty_coeff
        penalty_grid = (abs(p_grid - self.p_grid_max)*u_grid_up + abs(p_grid + self.
    p_grid_max)*u_grid_low)
        penalty_h_load = abs (self.heat_load[self.time_step]- heat_generated)
        penalty_c_load = abs (self.cold_load[self.time_step] - (cool_generated))

        penalty_cost = penalty_grid + penalty_caes + penalty_h_load + penalty_c_load

        self.penalty_costs.append(penalty_cost)
        self.penalty_SOC_caes.append(penalty_caes)
        self.penalty_p_grid.append(penalty_grid)
        self.penalty_c_load.append(penalty_c_load)
        self.penalty_h_load.append(penalty_h_load)

        cost_1 += penalty_caes
        cost_2 += penalty_grid
        cost_3 += penalty_c_load
        cost_4 += penalty_h_load

        self.penalty_cost_1.append(penalty_caes)
        self.penalty_cost_2.append(penalty_grid)
        self.penalty_cost_3.append(penalty_c_load)
        self.penalty_cost_4.append(penalty_h_load)

        #Moving to next state
        self.time_step += 1
        done = self.time_step == 24

        #Final SOC of AA-CAES Constraint
        if done:
            penalt = abs(self.caes.SOC - 0.5) *self.penalty_coeff
```

```
            if penalt <1e-3:
                penalt = 0
            reward -= penalt

        #Build the Next State of The Environment
        state = self._build_state()
        info = None

        return state, reward, cost_1, cost_2, cost_3, cost_4, done, info

    #Defining a Funciton to reset the Environment at the End of a Day
    def reset (self):

        self.time_step = 0
        self.caes = self._create_caes()
        self.op_cost = []
        self.penalty_costs = []
        self.penalty_cost_1 = []
        self.penalty_cost_2 = []
        self.penalty_cost_3 = []
        self.penalty_cost_4 = []

        self.elecs_hist = [[],[],[],[]]
        self.heats_hist = [[],[],[],[],[]]
        self.colds_hist = [[],[],[]]
        self.penalty_SOC_caes = []
        self.penalty_h_load = []
        self.penalty_gas_consump = []
        self.penalty_p_grid = []
        self.penalty_c_load = []
        self.caes_SOC_hist = []
        self.get_day()
        return self._build_state()


    #Getting the Information of a Random Day in Dataset
    def get_day (self):
        if len(self.picked_days)==len(days_to_train):
            self.picked_days = list()

```

```
421         available_days = [day for day in list(days_to_train) if day not in self.
        picked_days ]
422         day = available_days[0]
423 
424         if day == 365:
425             self.train = (data[data["Day"]==day]).append(all_data[all_data["Day"]==1].iloc
        [0])
426 
427         elif day+1 not in list(days_to_train):
428             self.train = (data[data["Day"]==day]).append(all_data[all_data["Day"]==(day+1)
        ].iloc[0])
429         else:
430             self.train = (data[data["Day"]==day]).append(data[data["Day"]==(day+1)].iloc
        [0])
431 
432         self.picked_days.append(day)
433         return self.train
434 
435     #Defining a Random Seed For Environment
436     def seed(self, seed):
437         random.seed(seed)
438         np.random.seed(seed)
```

Listing B.2: Imitation Learning Python Code

```
#Importing Libraries:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import numpy as np
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

#Reading Optimal Answers For Training Days:
optimal_answers = pd.read_csv("Optimal Answers.csv")

#Electricity and Gas Price:
buy_price = 24*[0]
buy_price[0:7] = 7*[3.99]
buy_price[22] = 3.99
buy_price[23] = 3.99
buy_price[11:19] = 8*[19.9]
buy_price[7:11] = 4*[11.99]
buy_price[19:22] = 3*[11.99]
gas_price = 3.21

#Defining a class for Actor Network:

class Actor(nn.Module):

    def __init__(self, state_dim, action_dim, max_action):
        super(Actor, self).__init__()
        self.layer_1 = nn.Linear(state_dim, 256)
        self.layer_2 = nn.Linear(256, 128)
        self.layer_3 = nn.Linear(128, 2)
        self.layer_4 = nn.Linear(128, action_dim-2)

        self.max_action = max_action

    def forward(self, x):
```

```
        x = F.relu(self.layer_1(x))
        x = F.relu(self.layer_2(x))
        x1 = self.max_action * torch.tanh(self.layer_3(x))
        x2 = self.max_action * torch.sigmoid(self.layer_4(x))
        x = torch.cat([x1,x2],1)
        return x

#Defining the Actions from Optimal Answers:
y_true = optimal_answers[["CAES","P_hp","P_gt","H_ac","H_gb","caes_heat_cool"]]

#Normalizing the actions based on maximum capacity of each equipment:
y_true["CAES"] = y_true["CAES"]/800
y_true["P_hp"] = y_true["P_hp"]/1000
y_true["P_gt"] = y_true["P_gt"]/1500
y_true["H_ac"] = y_true["H_ac"]/(1000/0.9)
y_true["H_gb"] = y_true["H_gb"]/1500

#Defining the input features (States of the Problem) and also Normalizing them:

x = pd.DataFrame()

x["SOC_CAES"] = (optimal_answers["Pressure_CAES"] - 42)/30

x["Wind"] = (optimal_answers["Wind Power"] - min(optimal_answers["Wind Power"])) / (max(
    optimal_answers["Wind Power"])- min(optimal_answers["Wind Power"]))

x["PV"] = (optimal_answers["PV Power"] - min(optimal_answers["PV Power"])) / (max(
    optimal_answers["PV Power"])- min(optimal_answers["PV Power"]))

x["Electricity"] = (optimal_answers["Electricity"] - min(optimal_answers["Electricity"]))
    / (max(optimal_answers["Electricity"])- min(optimal_answers["Electricity"]))

x["Heating"] = (optimal_answers["Heating"] - min(optimal_answers["Heating"])) / (max(
    optimal_answers["Heating"])- min(optimal_answers["Heating"]))

x["Cooling"] = (optimal_answers["Cooling"] - min(optimal_answers["Cooling"])) / (max(
    optimal_answers["Cooling"])- min(optimal_answers["Cooling"]))

x["Temperature"] = (optimal_answers["Temperature"] - min(optimal_answers["Temperature"]))
    / (max(optimal_answers["Temperature"])- min(optimal_answers["Temperature"]))

```

```
x["Buy Price"] = (buy_price * 365)
x["Buy Price"] = [(i - (min(buy_price))) / (max(buy_price) - min(buy_price)) for i in x["
    Buy Price"].values]

x["Time Step"] = optimal_answers["timestep"] / 24

#Converting X and Y to Tensors:
x_train = torch.tensor(x.values).float()
y_train = torch.tensor(y_true.values).float()

#Creating a Dataset From Tensors:
from torch.utils.data import TensorDataset
dataset = TensorDataset(x_train, y_train)

#Defining MSE Loss and Random batching:
from torch.utils.data import DataLoader

batch_size = 64
train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)


def mse_loss(predictions, targets):
    difference = predictions - targets
    return torch.sum(difference * difference)/ difference.numel()


#Creating an Actor Network:
actor = Actor(9,6,1)

#Defining The Adam Optimizer:
optimizer = torch.optim.Adam(params = actor.parameters(),lr=0.0001)

#Saving The Loss Functions:
loss_hist=[]


#Starting Learning Process:
epochs = 1000
from math import sqrt
for i in range(epochs):
    # Iterate through training dataloader
```

```
    for x,y in train_loader:
        # Generate Prediction
        preds = actor(x)
        # Get the loss and perform backpropagation
        loss = mse_loss(preds, y)
        optimizer.zero_grad()
        loss.backward()
        # Let's update the weights
        optimizer.step()
        loss_hist.append((loss.detach().item()))
    if loss.detach().item()<0.007: #Stop If loss is lower than 0.007
        break
    print(f"Epoch {i}/{epochs}: Loss: {(loss)}")


#Saving The Learned Actor Network (Behavioral Cloning Network):
torch.save(actor.state_dict(), 'pretrained_actor.pth')
```

Listing B.3: PD-DDPGfD Python Code

```
#Importing Libraries:
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

#Using Cuda if Available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#Defining Actor, Critic, and Cost Networks:
#Actor Network:
class Actor(nn.Module):

    #Initial Layers and Dimentions:
    def __init__(self, state_dim, action_dim, max_action):
        super(Actor, self).__init__()
        self.layer_1 = nn.Linear(state_dim, 256)     #First Layer
        self.layer_2 = nn.Linear(256, 128)           #Second Layer
        self.layer_3 = nn.Linear(128,2)              #Third Layer for Those actions between [-1,1]
        self.layer_4 = nn.Linear(128, action_dim-2)  #Third Layer for Those actions between [0,1]
        self.max_action = max_action

    #Forward Propagation:
    def forward(self, x):
        x = F.relu(self.layer_1(x))                              #Forward Passing With ReLU Function Layer 1
        x = F.relu(self.layer_2(x))                              #Forward Passing With ReLU Function Layer 2
        x1 = self.max_action * torch.tanh(self.layer_3(x))       #Forward Passing With Tanh Function for [-1,1]
        x2 = self.max_action * torch.sigmoid(self.layer_4(x))    #Forward Passing With ReLU Function for [0,1]
        x = torch.cat([x1,x2],1)                                 #Concatenating Layer 3 and 4 for the Action Output
        return x

#Critic Network:
```

```
class Critic(nn.Module):

    def __init__(self, state_dim, action_dim):
        super(Critic, self).__init__()
        #Defining the critic neural network
        self.layer_1 = nn.Linear(state_dim + action_dim, 256)
        self.layer_2 = nn.Linear(256, 128)
        self.layer_3 = nn.Linear(128, 1)


    def forward(self, x, u):
        xu = torch.cat([x,u], 1)
        #forward propagation on the first critic neural network
        x1 = F.relu(self.layer_1(xu))
        x1 = F.relu(self.layer_2(x1))
        x1 = self.layer_3(x1)
        return x1

    def Q1(self, x, u):
        xu = torch.cat([x,u], 1)
        x1 = F.relu(self.layer_1(xu))
        x1 = F.relu(self.layer_2(x1))
        x1 = self.layer_3(x1)
        return x1

#Cost Network:
class Cost(nn.Module):

    def __init__(self, state_dim, action_dim):
        super(Cost, self).__init__()
        self.layer_1 = nn.Linear(state_dim + action_dim, 256)
        self.layer_2 = nn.Linear(256, 128)
        self.layer_3 = nn.Linear(128, 1)


    def forward(self, x, u):
        xu = torch.cat([x,u], 1)
        x1 = F.relu(self.layer_1(xu))
        x1 = F.relu(self.layer_2(x1))
        x1 = self.layer_3(x1)
        return x1
```

```
74
75      def C1(self, x, u):
76          xu = torch.cat([x,u], 1)
77          x1 = F.relu(self.layer_1(xu))
78          x1 = F.relu(self.layer_2(x1))
79          x1 = self.layer_3(x1)
80          return x1
81
82
83  #Defining PD-DDPGfD Algorithm:
84  class PD_DDPGfD(object):
85
86      #Initializing the network parameters:
87      def __init__(self, state_dim, action_dim, max_action, lambda_, lambda_step,
        constraint_limit_1, constraint_limit_2,
88                   constraint_limit_3, constraint_limit_4,lr_critics, lr_actor):
89
90          self.actor = Actor(state_dim, action_dim, max_action).to(device)
        #Creating an Actor Model
91          self.actor.load_state_dict(torch.load('../Pre-training/pretrained_actor.pth'))
        #Loading the Pre-trained Model
92          self.actor_target = Actor(state_dim, action_dim, max_action).to(device)
        #Creating an Actor-target Model
93          self.actor_target.load_state_dict(self.actor.state_dict())
        #Copying Actor to Actor-target
94          self.actor_optimizer = torch.optim.Adam(self.actor.parameters(), lr=lr_actor)
        #Defining Optimizer for Actor Model
95
96          self.critic = Critic(state_dim, action_dim).to(device)
        #Creating a Critic Model
97          self.critic.load_state_dict(torch.load('pretrained_critic.pth'))
        #Loading Pre-trained Critic Model
98          self.critic_target = Critic(state_dim, action_dim).to(device)
        #Creating Critic-target Model
99          self.critic_target.load_state_dict(self.critic.state_dict())
        #Copying Critic to Critic-target
100         self.critic_optimizer = torch.optim.Adam(self.critic.parameters(), lr=lr_critics)
        #Defining Optimizer for Actor Model
101
102         #Cost Networks are similar to Critic networks:
103         self.cost_1 = Cost(state_dim, action_dim).to(device)
```

```
        self.cost_1.load_state_dict(torch.load('pretrained_cost_1.pth'))
        self.cost_1_target = Cost(state_dim, action_dim).to(device)
        self.cost_1_target.load_state_dict(self.cost_1.state_dict())
        self.cost_1_optimizer = torch.optim.Adam(self.cost_1.parameters(), lr=lr_critics)

        self.cost_2 = Cost(state_dim, action_dim).to(device)
        self.cost_2.load_state_dict(torch.load('pretrained_cost_2.pth'))
        self.cost_2_target = Cost(state_dim, action_dim).to(device)
        self.cost_2_target.load_state_dict(self.cost_2.state_dict())
        self.cost_2_optimizer = torch.optim.Adam(self.cost_2.parameters(), lr=lr_critics)

        self.cost_3 = Cost(state_dim, action_dim).to(device)
        self.cost_3.load_state_dict(torch.load('pretrained_cost_3.pth'))
        self.cost_3_target = Cost(state_dim, action_dim).to(device)
        self.cost_3_target.load_state_dict(self.cost_3.state_dict())
        self.cost_3_optimizer = torch.optim.Adam(self.cost_3.parameters(), lr=lr_critics)

        self.cost_4 = Cost(state_dim, action_dim).to(device)
        self.cost_4.load_state_dict(torch.load('pretrained_cost_4.pth'))
        self.cost_4_target = Cost(state_dim, action_dim).to(device)
        self.cost_4_target.load_state_dict(self.cost_4.state_dict())
        self.cost_4_optimizer = torch.optim.Adam(self.cost_4.parameters(), lr=lr_critics)

        self.max_action = max_action

        #Defining dual-variables for each cost network
        self.lambda_1 = torch.tensor([lambda_], requires_grad=False).to(device)
        self.lambda_2 = torch.tensor([lambda_], requires_grad=False).to(device)
        self.lambda_3 = torch.tensor([lambda_], requires_grad=False).to(device)
        self.lambda_4 = torch.tensor([lambda_], requires_grad=False).to(device)

        #Defining dual-variables learning rate:
        self.lambda_step = lambda_step

        #defining constraint toleration for each constraint:
        self.constraint_limit_1 = constraint_limit_1
        self.constraint_limit_2 = constraint_limit_2
        self.constraint_limit_3 = constraint_limit_3
        self.constraint_limit_4 = constraint_limit_4

        #Storing Loss For Each Network:
```

```
        self.Q_loss = []
        self.C1_loss = []
        self.C2_loss = []
        self.C3_loss = []
        self.C4_loss = []
        self.C4_value = []
        self.actor_losss = []
        self.Q_value = []
        self.Q_value_target = []

    #Defining a function for output Action for a given State
    def select_action(self, state):
        state = torch.Tensor(state.reshape(1, -1)).to(device)
        return self.actor(state).cpu().data.numpy().flatten()

    #Defining a function for training process:
    def train(self, replaybuffer_pretrain, iterations, batch_size=100, discount=0.99, tau
    =0.005, policy_noise=0.2,
              noise_clip=0.5, policy_freq=2):

        for it in range(iterations):

            # Step 1: We sample a batch of transitions (s, s', a, r, c1, c2, c3, c4) from
    the memory
            batch_states, batch_next_states, batch_actions, batch_next_actions,
    batch_rewards,
            batch_costs_1, batch_costs_2, batch_costs_3, batch_costs_4, batch_dones =
    replaybuffer_pretrain.sample(batch_size)

            # Step 2: Converting numpy arrays to tensors
            state = torch.Tensor(batch_states).to(device)
            next_state = torch.Tensor(batch_next_states).to(device)
            action = torch.Tensor(batch_actions).to(device)
            reward = torch.Tensor(batch_rewards).to(device)
            cost_1 = torch.Tensor(batch_costs_1).to(device)
            cost_2 = torch.Tensor(batch_costs_2).to(device)
            cost_3 = torch.Tensor(batch_costs_3).to(device)
            cost_4 = torch.Tensor(batch_costs_4).to(device)
            done = torch.Tensor(batch_dones).to(device)

            # Step 3: From the next state s', the Actor target plays the next action a'
```

```
182             next_action = self.actor_target(next_state)
183 
184             # Step 4: The  Critic and Cost targets take each the couple (s', a') as input
    and return two Q-values Qt1(s',a') and Ct(s',a') as outputs:
185             target_Q = self.critic_target(next_state, next_action)
186             target_C_1 = self.cost_1_target(next_state, next_action)
187             target_C_2 = self.cost_2_target(next_state, next_action)
188             target_C_3 = self.cost_3_target(next_state, next_action)
189             target_C_4 = self.cost_4_target(next_state, next_action)
190 
191             # Step 5: We get the final target of the Critic and Cost models, which is: Qt
    = r + gamma * min(Qt1, Qt2), where gamma is the discount factor
192             target_Q = reward + ((1 - done) * discount * target_Q).detach()
193             target_C_1 = cost_1 + ((1 - done) * discount * target_C_1).detach()
194             target_C_2 = cost_2 + ((1 - done) * discount * target_C_2).detach()
195             target_C_3 = cost_3 + ((1 - done) * discount * target_C_3).detach()
196             target_C_4 = cost_4 + ((1 - done) * discount * target_C_4).detach()
197 
198             # Step 6: The Critic model takes the couple (s, a) as input and return two Q-
    values Q1(s,a) as output
199             current_Q = self.critic(state, action)
200             self.Q_value.append(current_Q.sum().detach())
201             self.Q_value_target.append(target_Q.sum().detach())
202 
203             # Step 7: We compute the loss coming from the Critic model: Critic Loss =
    MSE_Loss(Q(s,a), Qt)
204             critic_loss = F.mse_loss(current_Q, target_Q)
205 
206             # Step 8: We backpropagate this Critic loss and update the parameters of
    Critic model with an Adam optimizer
207             self.critic_optimizer.zero_grad()
208             critic_loss.backward()
209             self.critic_optimizer.step()
210             self.Q_loss.append(critic_loss.detach())
211 
212             # Step 9: The Cost models take each the couple (s, a) as input and return two
    C-values C(s,a) as outputs
213             current_C_1 = self.cost_1(state, action)
214             current_C_2 = self.cost_2(state, action)
215             current_C_3 = self.cost_3(state, action)
216             current_C_4 = self.cost_4(state, action)
```

```
217
218             # Step 10: We compute the loss coming from the Cost models: Cost Loss =
        MSE_Loss(C(s,a), Ct) for each network:
219             cost_loss_1 = F.mse_loss(current_C_1, target_C_1)
220             self.cost_1_optimizer.zero_grad()
221             cost_loss_1.backward()
222             self.cost_1_optimizer.step()
223
224             cost_loss_2 = F.mse_loss(current_C_2, target_C_2)
225             self.cost_2_optimizer.zero_grad()
226             cost_loss_2.backward()
227             self.cost_2_optimizer.step()
228
229             cost_loss_3 = F.mse_loss(current_C_3, target_C_3)
230             self.cost_3_optimizer.zero_grad()
231             cost_loss_3.backward()
232             self.cost_3_optimizer.step()
233
234             cost_loss_4 = F.mse_loss(current_C_4, target_C_4)
235             self.cost_4_optimizer.zero_grad()
236             cost_loss_4.backward()
237             self.cost_4_optimizer.step()
238
239             #Storing C Loss:
240             self.C1_loss.append(cost_loss_1.detach())
241             self.C2_loss.append(cost_loss_2.detach())
242             self.C3_loss.append(cost_loss_3.detach())
243             self.C4_loss.append(cost_loss_4.detach())
244
245             # Step 11: Once every some iterations, we update our Actor model by performing
         gradient ascent on the output of the Critic and Cost models:
246             if it % policy_freq == 0:
247                 Q1 = self.critic.Q1(state, self.actor(state))
248                 C1 = self.cost_1.C1(state, self.actor(state))
249                 C2 = self.cost_2.C1(state, self.actor(state))
250                 C3 = self.cost_3.C1(state, self.actor(state))
251                 C4 = self.cost_4.C1(state, self.actor(state))
252
253                 #Adding an extra term for imitating the expert demonstrations:
254                 action_loss = F.mse_loss(action, self.actor(state))
255
```

```
                actor_loss = - (Q1 - self.lambda_1.detach() * C1 - self.lambda_2.detach()
        * C2 - self.lambda_3.detach() * C3 - self.lambda_4.detach() * C4 - action_loss).mean()

                self.actor_optimizer.zero_grad()
                actor_loss.backward()
                self.actor_optimizer.step()

                # Step 12: Still once every some iterations, we update the weights of the
        Actor target by polyak averaging and dual-variables:
                for param, target_param in zip(self.actor.parameters(), self.actor_target.
        parameters()):
                    target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
        data)

                lambda_gradient_1 = (self.cost_1.C1(state, self.actor(state)) - self.
        constraint_limit_1).mean()
                self.lambda_1 = max(torch.tensor([0]).to(device),
                                    self.lambda_1 + self.lambda_step * lambda_gradient_1).
        detach()

                lambda_gradient_2 = (self.cost_2.C1(state, self.actor(state)) - self.
        constraint_limit_2).mean()
                self.lambda_2 = max(torch.tensor([0]).to(device),
                                    self.lambda_2 + self.lambda_step * lambda_gradient_2).
        detach()

                lambda_gradient_3 = (self.cost_3.C1(state, self.actor(state)) - self.
        constraint_limit_3).mean()
                self.lambda_3 = max(torch.tensor([0]).to(device),
                                    self.lambda_3 + self.lambda_step * lambda_gradient_3).
        detach()

                lambda_gradient_4 = (self.cost_4.C1(state, self.actor(state)) - self.
        constraint_limit_4).mean()
                self.lambda_4 = max(torch.tensor([0]).to(device),
                                    self.lambda_4 + self.lambda_step * lambda_gradient_4).
        detach()

                # Step 13: Still once every some iterations, we update the weights of the
        Critic and Costs target by polyak averaging:
```

```
283                 for param, target_param in zip(self.critic.parameters(), self.
    critic_target.parameters()):
284                     target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
    data)
285 
286                 for param, target_param in zip(self.cost_1.parameters(), self.
    cost_1_target.parameters()):
287                     target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
    data)
288                 for param, target_param in zip(self.cost_2.parameters(), self.
    cost_2_target.parameters()):
289                     target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
    data)
290                 for param, target_param in zip(self.cost_3.parameters(), self.
    cost_3_target.parameters()):
291                     target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
    data)
292                 for param, target_param in zip(self.cost_4.parameters(), self.
    cost_4_target.parameters()):
293                     target_param.data.copy_(tau * param.data + (1 - tau) * target_param.
    data)
294 
295     # Making a save method to save a trained model
296     def save(self, filename, directory):
297         torch.save(self.actor.state_dict(), '%s/%s_actor.pth' % (directory, filename))
298         torch.save(self.critic.state_dict(), '%s/%s_critic.pth' % (directory, filename))
299         torch.save(self.cost_1.state_dict(), '%s/%s_cost_1.pth' % (directory, filename))
300         torch.save(self.cost_2.state_dict(), '%s/%s_cost_2.pth' % (directory, filename))
301         torch.save(self.cost_3.state_dict(), '%s/%s_cost_3.pth' % (directory, filename))
302         torch.save(self.cost_4.state_dict(), '%s/%s_cost_4.pth' % (directory, filename))
303 
304     # Making a load method to load a pre-trained model
305     def load(self, filename, directory):
306         self.actor.load_state_dict(torch.load('%s/%s_actor.pth' % (directory, filename)))
307         self.critic.load_state_dict(torch.load('%s/%s_critic.pth' % (directory, filename))
    )
308         self.cost_1.load_state_dict(torch.load('%s/%s_cost_1.pth' % (directory, filename))
    )
309         self.cost_2.load_state_dict(torch.load('%s/%s_cost_2.pth' % (directory, filename))
    )
```

```
310             self.cost_3.load_state_dict(torch.load('%s/%s_cost_3.pth' % (directory, filename))
        )
311             self.cost_4.load_state_dict(torch.load('%s/%s_cost_4.pth' % (directory, filename))
        )
```