Reinforcement Learning for Multicarrier Energy Management:

A Computationally Efficient Solution for TU Delft's Green Village

SET3901: Graduation Project

Víctor Andrés Rodríguez de Trío

Delft University of Technology

TU Delft

Reinforcement Learning for Multicarrier Energy Management:

A Computationally Efficient Solution for TU Delft's Green Village

by

Víctor Andrés Rodríguez de Trío

Student Surname	Student Number
Andrés Rodríguez de Trío	5608368

Supervisor (Main):	Dr. Gautham Ram Chandra Mouli
Supervisor:	Dr. Pedro Vergara Barrios
Daily Supervisor:	Darío Slaifstein
Project Duration:	December, 2023 - January, 2025
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover:	House in the Westfjords Region (Iceland) by Luke Stackpoole
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

Thank you to everyone who patiently walked with me in 2024, and especially to those who invited me on walks.

Víctor Andrés Rodríguez de Trío
Delft, January 2025

Abstract

Decarbonisation and efforts to reduce living expenses are driving interest in multicarrier energy systems (MCES) that integrate electricity, heat and e-mobility. These integrated networks require sophisticated energy management strategies to address uncertainties in energy supply, demand, and weather conditions.

This thesis seeks to develop a reinforcement learning (RL)-based energy management system (EMS) for a multicarrier residential building at TU Delft's Green Village. The case study household integrates photovoltaic and solar thermal systems, an electric vehicle (EV), lithium-ion battery, heat pump, and thermal storage. Current management uses white-box model predictive control (MPC) which, while effective, demands significant computational resources. In addition, its implementation requires expertise in optimal control and physics-based modelling.

The developed RL-based EMS provides a computationally efficient alternative, leveraging data-driven methods to learn system dynamics. The RL approach was benchmarked against the existing Expert, a day-ahead MPC planner. Performance was evaluated in terms of operational safety, grid energy exchange costs, and satisfaction of EV state of charge (SoC) demands.

The RL agent achieved performance comparable to that of the Expert in managing the MCES, while improving accessibility for developers lacking control theory expertise. The RL agent displayed consistent and near-optimal performance, resulting in only a 4% increase in grid exchange costs while improving both EV charging compliance and safety constraint adherence.

A literature review of RL applications for residential EMS is presented alongside an investigation of advanced policy update algorithms, deep neural network architectures, temporal feature engineering, and reward shaping strategies to analyse their impact on EMS performance.

The implemented RL-based control solution has been shown to manage electrical and thermal subsystems while maintaining safety and minimising costs. The computational efficiency and reduced modelling requirements of an RL agent highlight its suitability for small-scale MCES applications, such as residential or office buildings, where MPC may not be practical.

***Keywords**: Reinforcement Learning, Energy Management System, Multi-Carrier Residential Energy System, Computational Efficiency, Low-Cost Optimization, MPC Benchmark, Hybrid Energy Storage, Proximal Policy Optimization (PPO), Multi-Objective Optimization, Julia Programming*

Code Repository

The complete source code employed to address the research objectives is accessible at:
https://github.com/Victor-Andres-RdeTrio/RL4MCES

Summary

The problem tackled in this Master Thesis is the development of an RL-based energy management system (EMS) for a multicarrier energy system (MCES) situated in the Green Village of Delft University of Technology. The household integrates various energy resources, including PV and solar thermal systems, an electric vehicle (EV), a battery energy storage system (BESS), a heat pump (HP), and a thermal energy storage system (TESS). These assets interact dynamically through a power electronic interface, which provides flexibility to the system and, most importantly, grid connectivity.

A day-ahead planner white-box Model Predictive Control (MPC), known as the Expert, serves as the benchmark EMS for the current MCES, while also being frequently used in other energy systems for balancing energy use, operational safety, and economic considerations. However, MPC's reliance on specialised expertise in control theory and physics-based modelling, coupled with its high computational demands, limits its accessibility for widespread residential deployment.

This thesis investigates reinforcement learning (RL) as a computationally efficient alternative to MPC, while addressing some of the gaps found in current research. These include the unexplored potential of RL in hybrid energy storage systems, challenges in multi-objective reward function design and the underutilisation of advanced policy gradient algorithms like Proximal Policy Optimization (PPO). The work seeks to answer a central question: *To what extent can RL enable non-experts in control theory to manage MCESs efficiently while maintaining safety and performance comparable to MPC?*

The developed RL agent is tasked with minimising a multi-objective function that incorporates grid energy costs, penalties for unmet EV state of charge (SoC) requirements, and penalties for constraint violations. The MCES consists of interdependent components and is affected by uncertain exogenous information, all of which should be considered when making decisions. In the electrical subsystem, the agent must manage bidirectional power flows between the grid, the BESS, and the EV battery, while accounting for energy production from PV panels. In the thermal subsystem, the agent must allocate energy between heating demands and storage, by directing the HP and leveraging the buffering capacity of the TESS.

The process to reach the final RL agent presented in this thesis is built upon different reward functions, feature vectors, and deep neural network (DNN) architectures, which are treated as hyperparameters to optimise. Systematically exploring these options allowed the discovery of high-performing configurations.

Reward formulations were designed to reflect the multi-objective nature of the sequential decision problem (SDP). Eight distinct reward combinations were selected, arising from combining two approaches for grid cost reduction, two EV penalty methods, safety violation penalties and an optional margin reward component (which discourages moving towards out-of-bounds states). The combinations were chosen to prioritise significant paradigm shifts, with adjustable weights treated as hyperparameters to fine-tune the importance of each objective.

To further enhance the RL agent's adaptability to the uncertain future, a correlation study was conducted on the Expert EMS, which revealed meaningful time lags and relationships between state variables and decisions. What was learned became the foundation for various feature vector configurations, each offering the agent different past information about the state variables. For contrast, configurations relying on periodic patterns or randomised time lags were also included. This expansion of the feature space provided the agent with a broader range of possible inputs, thus reducing reliance on trial-and-error in feature selection.

The standard DNN design featured separate Multi-Layer Perceptrons (MLPs) for the Actor and Critic, each with two hidden layers of uniform width, employing Tanh activation functions and orthogo-

nal weight initialisation. Eleven additional architectures were introduced, including variations in depth, width, parallelisation and neuron interconnectivity, aiming to uncover configurations capable of greater abstraction and improved learning efficiency, potentially with lower computational demands.

To evaluate the proposed hyperparameter configurations, three stochastic policy gradient algorithms –Vanilla Policy Gradient with Critic (VPG-C), Advantage Actor-Critic with generalized advantage estimation (A2CGAE), and PPO– were implemented, representing a range of algorithmic complexity. This comprehensive evaluation framework enabled thorough testing of the reward functions, feature vectors, and neural network designs, ultimately facilitating the development of a lightweight and high-performing RL-based EMS.

The performance metric serves as the foundation of the hyperparameter optimisation process, ensuring alignment with the system's objectives by integrating three essential components: grid operational cost, EV charging satisfaction, and safety constraints. These elements are combined into a comprehensive metric that prioritises safety above all, followed by EV charging requirements, and, only when these are satisfied, grid cost reductions. This approach helps evaluate RL agents and guides computational resource allocation during optimisation. The metric's non-linear design ensures small behavioural differences translate to meaningful results.

The hyperparameter optimisation unfolds in two stages: a basic and an extended phase. The basic optimisation explores common parameters, identifying the best performing hyperparameter configurations (known as *profiles*) for each of the three policy gradient algorithms (VPG-C, A2CGAE, and PPO). The extended optimisation expands this process by incorporating additional parameters, the most relevant of which are those discussed above: neural network architectures, reward formulations, and feature vector configurations. Both phases rely on the BOHB algorithm, which iteratively samples the search space, trains agents using these configurations, and evaluates them based on the performance metric. Multiple agents are trained per configuration to account for randomisation effects, with the hyperparameter profiles selected based on averaged outcomes.

Key insights emerged during optimisation. The inclusion of the margin reward proved to be a crucial development, substantially improving performance consistency while delivering remarkable enhancements in both median and peak performance metrics. Regarding DNN architectures, the *3CW* actor architecture, consisting of three parallel branches of uniform width (one per decision), showed promising results. High-performing outliers suggest that specific hyperparameter configurations may be able to leverage the more complex design. The critic architecture analysis revealed a different pattern, with both *Bottleneck* and *Residual* architectures presenting notable improvements in performance.

Surprisingly, simpler feature vectors –that only consider the last timestep– outperformed more complex alternatives, likely due to their robustness against overfitting. PPO exhibited the highest median performance and algorithmic robustness, while VPG-C, unexpectedly, achieved remarkable results through outlier configurations.

The pipeline culminates in the identification of the *top RL agent (TRLA)*, derived from the extended optimisation phase. This agent implements the *3CW* actor architecture and a bottleneck critic. Its reward structure incorporates the margin reward, which proved pivotal for stability and success. While its grid cost is slightly higher than other candidates, its superior handling of EV charging demands solidifies its position as the most balanced and effective EMS.

The performance of the *TRLA* was validated through a comprehensive comparison with the Expert, using metrics from the latter's objective function to ensure a fair evaluation. Despite the inherent stochasticity of its decision-making, the *TRLA* demonstrated consistency across the testing dataset. While the Expert optimised grid exchange costs more effectively, this was sometimes achieved at the expense of penalties for unmet EV SoC requirements. Temporal analyses confirmed the *TRLA*'s ability to deliver reliable outcomes, with aggregate performance aligning with the Expert's, even though they made different decisions at each step.

In terms of safety, the *TRLA* benefitted from the integration of the *safe projection* mechanism, which dynamically adjusts action limits at each timestep based on system component constraints. This approach ensured any infractions remained negligible and within acceptable limits, thus protecting the

integrity of the MCES. Moreover, the *TRLA* showed better safety performance than the Expert in certain areas.

The *TRLA*'s overall behaviour shows its capacity to balance conflicting long-term objectives. In the electrical subsystem, the *TRLA* achieved comparable outcomes to the Expert, effectively balancing loads and ensuring optimal EV SoC before departure. Within the thermal domain, the agent successfully managed resources while respecting TESS constraints and minimising grid energy costs. These results show that the *TRLA* is ready for deployment, where its efficiency and safety should be further tested.

The research validates RL as an effective tool for non-experts to manage MCESs, achieving performance comparable to MPC while maintaining safety and significantly reducing computational requirements during deployment.

Code Repository

The complete source code employed to address the research objectives is accessible at:
https://github.com/Victor-Andres-RdeTrio/RL4MCES

The author may be reached by email: vandres.trio@proton.me

Contents

Preface i

Abstract ii

Summary iii

1 Introduction 1

1.1 Problem Definition . . . 2
1.2 Research Goal . . . 2
1.3 Main Contribution . . . 3
1.4 Statement of Research Questions . . . 3
1.5 Structure of the Thesis . . . 3

2 Literature Review 5

2.1 Introduction . . . 5
2.2 Overview of Sequential Decision Problems . . . 5
2.3 Policies . . . 6
2.3.1 Policy Search . . . 6
2.3.2 Lookahead Policies . . . 7
2.4 Introduction to Reinforcement Learning . . . 7
2.4.1 Modelling Framework . . . 7
2.4.2 Towards a Definition . . . 8
2.4.3 Learning . . . 8
2.4.4 On or Off Policy . . . 8
2.5 Is a model useful? . . . 9
2.5.1 Model-Free . . . 9
2.5.2 Model-Based . . . 10
2.6 Defining a Model-Free Algorithm . . . 10
2.6.1 Class Selection: Actor-Critic . . . 10
2.6.2 Stochastic Policy Gradient . . . 11
2.7 Updating the Agent . . . 12
2.7.1 Rewards . . . 12

2.7.2 Actor loss . . . 15
2.7.3 Critic loss . . . 16
2.7.4 Loss Gradients . . . 17
2.8 Activation Functions . . . 20
2.9 RL in Energy Management Systems . . . 23
2.9.1 Strengths . . . 23
2.9.2 Weaknesses . . . 23
2.9.3 Residential Buildings . . . 24
2.9.4 PPO in practice . . . 25
2.9.5 MCES Modelling . . . 26
2.9.6 Direct Deployment in MCES: Safe Learning . . . 26
2.10 Research Gaps . . . 27
2.11 Research Questions . . . 28

3 Energy Management System **29**
3.1 Introduction . . . 29
3.2 System Components . . . 29
3.3 Sequential Decision Problem . . . 30
3.3.1 Objective function . . . 30
3.3.2 State variable . . . 32
3.3.3 Decision variable . . . 33
3.3.4 Exogenous information . . . 34
3.3.5 Transition function . . . 34
3.4 Safe Behaviour . . . 36
3.4.1 Safety Projection . . . 36
3.4.2 Safety during training . . . 38
3.4.3 Safety after deployment . . . 38
3.5 Expert EMS . . . 38
3.6 Data: Training and Testing . . . 39
3.6.1 Expanding the Data . . . 40

4 RL Agent Construction **42**
4.1 Introduction . . . 42
4.2 Reward Functions . . . 42
4.2.1 Grid Cost . . . 43

4.2.2 EV Penalty at Departure . . . 44
4.2.3 Projection Penalty . . . 45
4.2.4 Margin Reward . . . 46
4.2.5 Final Reward . . . 47
4.3 Feature vector . . . 49
4.4 Neural Network Design . . . 51
4.4.1 Dimensions of the Deep Neural Network . . . 51
4.4.2 Policy Output . . . 52
4.4.3 Weight Initialisation . . . 52
4.4.4 Architecture . . . 53
4.5 Implementation of Policy Gradient Algorithms . . . 57
4.5.1 Vanilla Policy Gradient with Critic (VPG-C) . . . 57
4.5.2 Advantage Actor Critic with Generalized Advantage Estimation (A2CGAE) . . . 58
4.5.3 Proximal Policy Optimization (PPO) . . . 59
4.6 Data Collection . . . 60

5 Hyperparameter Optimisation and Agent Evaluation 62
5.1 Introduction . . . 62
5.2 Performance Metrics . . . 62
5.2.1 Grid Cost Metric . . . 63
5.2.2 EV Penalty Metric . . . 64
5.2.3 Projection Penalty Metric . . . 64
5.2.4 Final Metric . . . 64
5.3 Hyperparameter Optimisation . . . 65
5.3.1 Hyperparameter Set . . . 66
5.3.2 Basic Range . . . 68
5.3.3 Extended Range . . . 68
5.3.4 Procedure . . . 69
5.3.5 Results . . . 71
5.4 Finding a Near-Optimal Policy . . . 75
5.4.1 Hyperparameter Profile Performance Analysis . . . 75
5.4.2 Comparison of Best Agents . . . 76

6 RL EMS Validation and Conclusions 78
6.1 Introduction . . . 78

6.2 Comparison with Expert . . . 78
6.2.1 Objective Function Metrics . . . 79
6.2.2 Agent Behaviour . . . 79
6.2.3 Safety Performance . . . 84
6.2.4 Validation . . . 85
6.3 Conclusions . . . 85
6.3.1 Further Work . . . 88

References **89**

A Safety, Data Augmentation and Exploding Gradients **93**
A.1 Exploding Gradients . . . 93
A.2 Safety Projection Model . . . 93
A.2.1 Objective Function . . . 94
A.2.2 Equality constraints: . . . 94
A.2.3 Variables and Bound Constraints . . . 95
A.2.4 Constants . . . 96
A.2.5 Safety Projection Details . . . 97
A.2.6 Safety Projection in Operation . . . 97
A.3 Synthetic Expansion of Training Data . . . 98

B Hyperparameter Optimisation: Extra Information **100**
B.1 Finding the Best Hyperparameters: BOHB Algorithm . . . 100
B.2 Auxiliary Hyperparameter Ranges . . . 101
B.2.1 Feature Vectors . . . 101
B.2.2 Reward Functions . . . 104
B.2.3 Reward Weights . . . 104
B.2.4 Neural Network Architectures . . . 104
B.3 Supplementary Hyperparameter Optimisation Results . . . 106
B.3.1 Architectures Comparison . . . 107
B.3.2 Feature Vectors Comparison . . . 108
B.4 System Specifications for Training RL Agents . . . 109
B.5 Working with Julia's RL Package . . . 110
B.5.1 Code Repository . . . 110

List of Acronyms

3CW 3 Constant Width Branches. 56, 71, 74, 80
3PYR 3 Pyramid Branches. 56, 71, 74

A2C Advantage Actor-Critic. 9, 11, 15, 25, 26, 58, 59
A2CGAE Advantage Actor-Critic with generalized advantage estimation. iv, 25, 42, 58–60, 62, 69, 71–73, 75–77
A3C Asynchronous Advantage Actor-Critic. 58
ACKTR Actor-Critic using Kronecker-Factored Trust Region. 25
AD automatic differentiation. 17, 18
ADP Approximate Dynamic Programming. 7

BESS battery energy storage system. iii, 29, 32–37, 39, 47, 48, 52, 80–84, 87
BOHB Bayesian Optimisation and Hyperband. iv, 62, 66, 68–71, 86, 100, 101

COP coefficient of performance. 35
CW Constant Width. 53, 56, 69, 71, 74, 86

DCW Deep Constant Width. 71, 74, 87
DDPG Deep Deterministic Policy Gradient. 11
DLA direct lookahead approximation. 38
DNN deep neural network. iii, iv, 3, 6, 9, 10, 12–15, 17–22, 25, 28, 36–38, 42, 43, 48–58, 66–68, 74, 86, 87, 93, 104, 106–109
DPG Deterministic Policy Gradient. 9

EMA exponential moving average. 20
EMS energy management system. iii, iv, 1–3, 5, 23–28, 30, 31, 38
EV electric vehicle. iii–v, 2, 3, 24, 25, 27, 29, 31–36, 39, 40, 44, 45, 47, 48, 52, 62–65, 72–74, 77, 79–85, 87

GAE generalized advantage estimation. 3, 14–17, 28, 58–60, 66, 86

HP heat pump. iii, 29, 33, 35, 36

MCES multicarrier energy system. iii, v, 1–3, 5, 26, 29–31, 34–39, 41–43, 45–48, 64, 68, 73, 78, 83–88
MDP Markov decision process. 5, 7, 8, 30
MILP Mixed-Integer Linear Programming. 25
MIQCP Mixed-Integer Quadratically Constrained Program. 39
MPC Model Predictive Control. iii, v, 1–3, 5, 8, 10, 23, 27, 28, 38, 39, 78, 86–88
MSE mean squared error. 16, 17, 58

NN neural network. 10–12, 16, 19, 55, 56, 60

PEI Power Electronic Interface. 29, 33
PFA policy function approximation. 6, 9

PPO Proximal Policy Optimization. iii, iv, 9, 11, 13, 15, 25–28, 42, 59, 60, 62, 67, 69, 71–73, 75–77, 86

PV photovoltaic. iii, 24, 26, 29, 34, 39

RL reinforcement learning. iii–v, 1–5, 7–12, 15, 22–33, 36, 38–43, 45, 47, 51, 52, 56–58, 62, 64, 66, 75–79, 84–88, 109, 110

RMSE root mean squared error. 79

RNG random number generator. 41

SAC Soft Actor-Critic. 11, 25, 26

SDP sequential decision problem. iii, 3, 5, 6, 8, 10, 12, 29, 30, 33, 34, 37, 39, 42, 62

SGD stochastic gradient descent. 18–20

SoC state of charge. iii–v, 3, 31, 32, 34–37, 42, 44, 47, 64, 72, 78–80, 82, 83, 85, 87

ST solar thermal. iii, 29, 79

TD3 Twin Delayed Deep Deterministic Policy Gradient. 11, 25, 26

TESS thermal energy storage system. iii, v, 29, 31–33, 35, 36, 38, 39, 47, 77, 79, 82–85, 87

TRLA top RL agent. iv, v, 12, 62, 76–85, 87

TRPO Trust Region Policy Optimization. 9, 11, 59, 60

VFA value function approximation. 7–9

VPG Vanilla Policy Gradient. 11, 25, 57, 58

VPG-C Vanilla Policy Gradient with Critic. iv, 25, 42, 58, 59, 62, 66, 67, 69, 71–73, 75–77, 86

1

Introduction

The global push toward decarbonisation, coupled with efforts to reduce living expenses, has accelerated the integration of diverse energy systems, where electricity, mobility, and heat converge. Known as multicarrier energy systems (MCES), they require coordinated resource management over daily, monthly, and annual timescales. Beyond cost savings, they commonly demand resilience, sustainability, and energy autonomy, all of which call for the development of sophisticated strategies that can address forecast uncertainties in energy supply, demand, and weather conditions.

The incorporation of diverse energy **generation** and **storage** methods, such as electric and thermal systems, along with electric vehicles that allow bidirectional energy flow, adds another layer of complexity. Despite this, a locally implemented energy management system (EMS) can harness the potential of these intertwined energy resources to satisfy the needs of the residents.

The conventional approach of Model Predictive Control (MPC) excels in managing an MCES by modelling dynamic transition functions to predict future states and optimise control actions. It can account for the relationships among energy resources and integrate forecasts for demand, local generation, and market prices, which would provide a robust framework to handle uncertainties.

However, designing an effective MPC-based EMS **requires significant expertise** to accurately capture the complexities of the MCES while ensuring computational feasibility. Furthermore, the iterative nature of its optimisation process demands substantial computational resources, particularly in light of the expanding interconnected energy infrastructures prevalent in modern urban environments. For small-scale MCES, such as residential or office buildings, the adoption of conventional MPCs remains unrealistic. This limitation suggests a compelling case for more **accessible and cost-effective alternatives** to be explored and implemented.

Reinforcement Learning (RL) appears to be a promising alternative for operating energy systems [41], characterised by lower computational costs and reduced dependence on specialised expertise compared to MPC. By learning system dynamics directly from data, RL eliminates the need for detailed a priori models, minimising the technical undertaking specific to each project. Additionally, this method demonstrates enhanced adaptability to complex state spaces and changing environments [12, 57].

While RL is likely surpassed by some MPC designs, its performance can be near-optimal, and clearly superior to traditional rule-based control. The cost-efficiency and flexibility of RL make it attractive for small-scale practical applications. However, achieving robust and safe operation, both in the training phase and during real-world implementation, represents an ongoing challenge [57].

1.1. Problem Definition

The problem central to this MSc thesis is the **development of an energy management system based on reinforcement learning** for a multicarrier electrified residential building located in the Green Village of Delft University of Technology.

The household under consideration is part of the FLEXINet project. It functions as a multicarrier energy system that integrates photovoltaic and solar thermal energy, an electric vehicle (EV), a lithium-ion battery, a heat pump, and a thermal storage system. These assets are dynamically managed by a power electronic interface, enabling flexible interaction with the electrical grid.

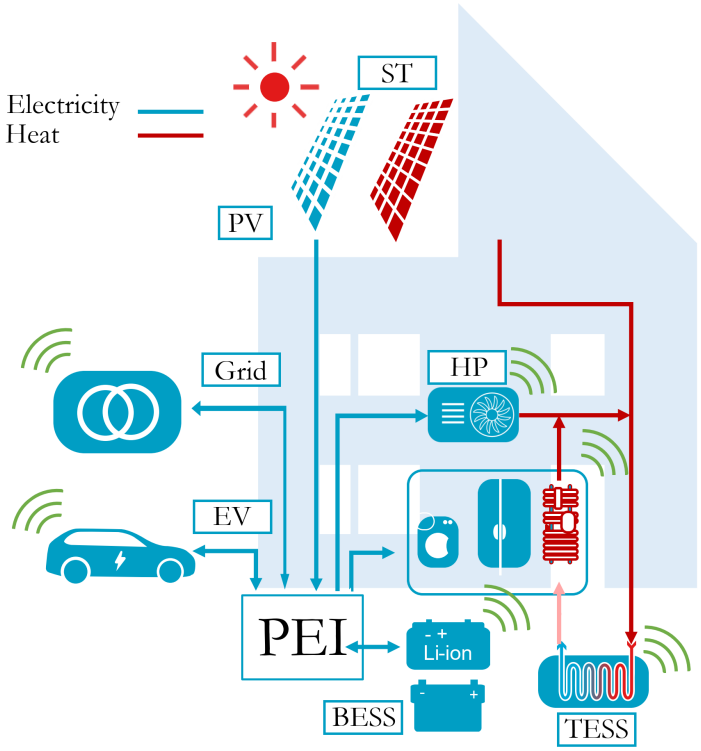

Figure 1.1: Schematic diagram of the MCES from the research by Slaifstein et al.[55]. The arrow points indicate the directions in which energy can flow through the system.

At present, a white-box MPC Expert is used to manage the MCES. The EMS designed in this thesis will aim to achieve a near-optimal policy by optimising the same objective as the Expert, which will serve as the performance benchmark. The project seeks to establish a **computationally viable alternative suitable for household deployment**. The resulting EMS will also allow an assessment of the performance that can be achieved without relying on specialised expertise in *optimal control* or *physics-based modelling*.

To accomplish this task, a literature study on reinforcement learning and its application in residential EMSs will be undertaken. In light of potential future interactions between the RL and Expert agents, the algorithms were encouraged to be developed in Julia or Python. The Julia Programming Language [8] will be selected due to its superior computational efficiency.

1.2. Research Goal

Assess the potential of reinforcement learning for non-experts in control theory by building a computationally efficient alternative to MPC for managing a multicarrier urban energy system.

1.3. Main Contribution

The main contribution of this thesis lies in the development of a computationally efficient reinforcement learning (RL)-based EMS for managing a multicarrier urban energy system at TU Delft's Green Village. This work demonstrates performance comparable to traditional MPC methods, while maintaining operational safety and allowing real time operation. Additionally, RL is shown to be an effective tool for practitioners without deep knowledge of optimal control theory or physics-based modelling.

A literature review is provided, exploring the theoretical foundations and practical applications of RL in EMS, with a focus on multicarrier energy system (MCES) in residential buildings.

This thesis provides insights into the impact on performance of advanced RL policy update algorithms, sophisticated DNN architectures, temporal feature engineering, and reward component shaping.

Furthermore, a direct comparison between MPC and RL is presented, offering an evaluation of their performance in terms of operational safety, grid exchange costs, and EV SoC demand satisfaction.

1.4. Statement of Research Questions

This thesis is centred on one overarching research question, which is further explored through three sub-questions. To provide clarity and context, these questions will be **discussed in greater depth in Section 2.11**, where their alignment with the research gaps outlined in Section 2.10 is elaborated.

Main Research Question

In the management of multicarrier urban energy systems, to what extent can practitioners without expertise in optimal control theory and physics-based modelling leverage reinforcement learning to create a computationally efficient alternative to model predictive control, while maintaining comparable safety and performance?

Sub-Research Question 1

How do progressively advanced policy update algorithms (from basic advantage estimation to GAE to trust region constraints) contribute to achieving near optimal performance?

Sub-Research Question 2

What is the impact of DNN architecture sophistication, temporal feature engineering, and reward component shaping for RL algorithms in achieving performance that is close to optimal?

Sub-Research Question 3

How do the RL and MPC approaches compare in terms of operational safety, energy storage use, and economic performance?

1.5. Structure of the Thesis

This thesis is organized to guide the reader from foundational concepts to the final validation of the proposed RL agent for energy management. The chapters are structured as follows:

- **Chapter 2:** Provides a comprehensive review of the theoretical principles and current state of RL in energy management systems, identifying research gaps that shape this work.
- **Chapter 3:** Formally defines the sequential decision problem for a multicarrier electrified residen-

tial building, introduces the Expert, and discusses data preparation techniques.

- **Chapter 4:** Focuses on constructing the RL agent, including reward design, feature vector configurations, neural network architectures, and the implementation of policy gradient algorithms.

- **Chapter 5:** Addresses hyperparameter optimisation in two stages using a systematic pipeline and evaluates the performance of candidate agents.

- **Chapter 6:** Validates the top-performing RL agent by comparing its performance to the Expert and concludes with a discussion of research findings and potential future work.

2

Literature Review

2.1. Introduction

This literature review explores the theoretical foundations and practical applications of RL in EMS. The mathematical framework required for understanding RL applications is first established through an exploration of sequential decision problems and policy formulation. The review progresses through key aspects of RL, examining both model-free and model-based approaches, before focusing on actor-critic architectures and stochastic policy gradient methods.

A detailed examination of the technical implementation literature addresses key topics such as agent updating algorithms, loss functions, reward design and processing, and activation functions. A firm grasp of these components is crucial for understanding the algorithms' degree of complexity and to devise a **computationally efficient alternative** to MPC.

The review concludes by examining the **current state of RL in EMS**, with a focus on MCES in residential buildings. Research gaps are identified after an analysis of the strengths and weaknesses of RL-based approaches. Additionally, the modelling of a MCES to create a virtual environment is discussed. These gaps inform the research questions that will guide the thesis' development of an EMS for a multicarrier electrified residential building.

2.2. Overview of Sequential Decision Problems

Energy Management Systems seek to optimise the use of energy resources by responding to changes in demand and supply conditions. This interaction requires an **agent** to make appropriate decisions based on the current known information, which will then be translated into cumulative or final rewards. This is a simple description of a sequential decision problem (SDP), which in even simpler terms involves a succession of decision-making and gathering of information, followed by another step of decision-making.

Warren B. Powell [45] has created a concise and universal framework for dealing with these problems, which includes RL and MPC, along with a broad range of disciplines such as optimal control, Markov decision processes (MDPs), stochastic or robust optimisation, approximate dynamic programming, and others. The difference between his framework and the rest is that the problem is modelled first and the policy is found afterwards, since all the policy types can be useful for producing sequential decisions. To provide a basic introduction, according to Powell, there are five fundamental elements in any SDP:

- **The state variable** S^n captures the information or beliefs after n observations. It is the information needed for making decisions and modelling the problem, nothing more. The initial state S^0

encompasses not just deterministic parameters but also may incorporate the initial distributions of uncertain parameters.

- **The decision variable** x^n is determined through the policy $X^\pi(S^n)$ and will be performed in the environment by the agent or controller. The policy is a function, likely with tunable parameters, that maps the states to decisions, either in a deterministic or a stochastic manner.
- **Exogenous information** W^{n+1}. It will arrive after the decision x^n.
- **The transition function** allows for the evolution of the environment; it contains the necessary information such that, given a state, a decision, and the exogenous information, the next state can be obtained. $S^{n+1} = S^M(S^n, x^n, W^{n+1})$ describes the transition, and includes all the dynamics of the system.
- **The objective function**. Each step will produce a reward (or cost), named $C(S^n, x^n)$. There are various ways to consider the performance of the agent, but a common one is to maximise the cumulative reward, described as: $\max_\pi \mathbb{E}\left\{\sum_{n=1}^{N} C(S^n, X^\pi(S^n)) \mid S^0\right\}$. Sometimes, only final rewards will be considered. Any SDP will aim to maximise the objective function over policies.

The indexing can also be done with t for time. It is a very useful terminology because everything is indexed following what the agent knows in the timestep t. Uncertainty in state variables or exogenous information is common and usually needs to be modelled. Once the basic elements have been described, the information process of an SDP can be written as: $\left(S^0, x^0, W^1, S^1, x^1, W^2, \ldots, S^n, x^n, W^{n+1}, \ldots, S^N\right)$ [45]. With the objective function established, the search for the optimal policy begins.

2.3. Policies

Powell's contributions [45] are particularly relevant because, according to his research, the complete body of knowledge that addresses decision-making under uncertainty can be categorised into two overarching strategies for formulating policies: **policy searching and looking ahead**.

This approach makes the distinction between fields (like active learning, approximate dynamic programming, decision trees...) a matter of choosing a style of policy. The fundamental four policies (detailed below) resulting from these strategies can also be combined into hybrid policies, showing how varied the methods to solve a problem can become. All four policy classes may result in an optimal policy [45].

2.3.1. Policy Search

This class encompasses searching over functions or tunable parameters, and it is divided into two subclasses [45]:

Policy Function Approximations (PFAs)

Analytical functions that map a state to a decision. For discrete states and actions, they may be represented as a lookup table. Alternatively, these functions can be parametrized, such as through (Artificial) Neural Networks. Additionally, locally linear approximations and other non-parametric methods, such as deep neural networks, are also included. These approaches are highly adaptable in terms of their structure and the number of parameters, enabling them to capture complex non-linear relationships within the data.

Cost Function Approximations (CFAs)

Parametrized optimisation models, usually deterministic, they are very relevant in industry but usually ignored by the SDP researchers, Powell [45] introduces them as a fundamental policy class. The main difference from policy function approximation (PFA)s is that there is an embedded optimisation that requires solving for each decision to be made.

2.3.2. Lookahead Policies

The aim of these policies is to make a good decision by considering the effect they will have on the predicted future of the system. There are again two major subclasses to consider [45]:

Value Function Approximations (VFAs)

Each state is given a value that represents the expected contribution (reward) or cost of being in that state if the agent follows a particular policy. This is the foundational approach of dynamic programming. The policy determines which action will maximise the sum of the immediate reward (or minimise the cost) and the value of the next state. For clarity, this formulation from [45] is presented:

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E}_{W_{t+1}} \left\{V_{t+1}(S_{t+1}) \mid S_t\right\}\right) \tag{2.1}$$

This approach is often impractical due to the *curse of dimensionality*, as any expansion in the state or action space would result in exponentially higher computational demands. To address this challenge, fields like approximate dynamic programming or RL have developed the idea of approximating the value function V_{t+1} (e.g., Q-learning), usually by means of machine learning.

Direct Lookahead Approximations (DLAs)

At the current state, the next decisions are planned ahead, considering their downstream effects. This sequence of orchestrated actions is optimised to increase both immediate and predicted rewards, resulting in the optimal decision in the present state. Typically, this computation will be performed at each step. In order to look ahead into the future of the system, an approximate lookahead model is created, which may be deterministic or stochastic, escalating considerably the complexity of the problem at hand.

2.4. Introduction to Reinforcement Learning

To delve into RL first it becomes necessary to define what it is, and this is no simple task. According to Sutton and Barto [59], RL embodies a problem, a class of solution methods that work well on said problem, and the field that studies both. However, the question arises: *What are the problems, and what are the solution methods to employ?* This becomes harder to answer as RL has exploded in popularity – with $50,000$ new citations from 2015 to 2020. While RL arose as a community that many would identify as using Approximate Dynamic Programming (ADP), the field now encompasses too many problems for one set of solution methods [46].

RL started with a similar idea to ADP: approximating the value of being in a state. However, instead of focusing on learning the state value function $V(s)$, the paradigm shifted to learning the state-action value function $Q(s, a)$, which indicates the expected contributions from being in a state s, taking an action a and then following the policy [45]. This approach, known as Q-learning, stood at the core of RL during the 1990s. Sutton and Barto [59] affirm that such value functions are one of the fundamental subelements of RL, but they are just a method, one class of policy [46], and other methods are used even in their own work. RL has grown beyond VFAs with the adoption of all four classes of policies (see Section 2.3).

2.4.1. Modelling Framework

Since its origins, the community of RL has used the modelling framework of **Markov Decision Processes** (MDPs) [45]:

- **State Space:** Set of states denoted by s that the system may occupy.
- **Actions:** Set of actions denoted by a that the **agent** can perform.

- **Transition matrix:** The one-step state transition matrix, with $P(s' \mid s, a)$ representing the probability of transitioning to state $S_{t+1} = s'$ given the current state $S_t = s$ and action a, follows the Markov property. This guarantees that the next state depends on the current state and the action taken, not on the past.
- **Reward Function:** $r(s, a)$ gives the immediate reward obtained after taking action a in state s.
- The transition matrix and reward function are usually contained in an abstraction called **environment**, where everything that cannot be changed arbitrarily by the agent resides [7].

Powell [45] contends that while the mathematical elegance of Markov decision process (MDP) frameworks is apparent, it lacks practical utility in real-world applications. Despite the theoretical utility of abstract state and action *spaces*, in Powell's modelling framework (inspired by Optimal Control) state and action *variables* are used, which provide concrete, measurable quantities that can be directly coded. Additionally, he highlights the limitation of the one-step transition matrix, which is often not computable. The transition function, he argues, is a more potent and versatile tool. Moreover, Powell emphasises thinking of accumulated rewards over time, as opposed to focusing on rewards obtained in a single step. As can be seen when presenting the SDP in Section 3.3, the modelling framework provided by Powell is used [45].

2.4.2. Towards a Definition

According to Sutton and Barto [59] any method appropriate for solving an MDP, is a reinforcement learning method. So, if a method is suitable, but it **does not involve learning**, like MPC, **is it RL?** [46].

Reinforcement Learning extends beyond the scope of MDP, as this is simply a framework for making decisions. RL methods have surpassed value function approximation (VFA), and learning may not be necessary under widely recognised definitions of the field. There is a lot of uncertainty around the topic, but these statements from Powell may bring some clarity [46]:

- RL problems are just SDP, and the MDP framework may be used to define them.
- RL methods can be categorised as policies, which are grouped into four main classes. However, in order for the term *learning* to have any meaning, one must assume that RL methods **involve learning**.

2.4.3. Learning

There are two common meanings for **learning** in the machine learning (ML) community, which are often a source of confusion. *Offline* learning is commonly seen in supervised learning scenarios, where a **batch optimisation** takes place, that is, all the available data is used at once to fit a usually parametrised model [45]. On the other hand, reinforcement learning frequently involves *online* learning, where the data arrives **sequentially**, so the model parameters are updated as the information arrives. Of course, there are hybrid approaches, such as imitating an expert policy with *offline* learning before fine-tuning parameters in an *online* fashion.

2.4.4. On or Off Policy

The online learning nature of RL means that information acquisition is shaped by the exploration method followed by the agent, which defines how the next state in the simulation is reached –a crucial consideration for problems employing VFA policies.

If the agent is following the latest policy for decision-making (often referred to as the *implementation policy* [45]) to guide its way through the state space, it is defined as **on policy learning**. Otherwise, if the policy for exploring (that is, the *learning policy*) is different from the *implementation policy* (e.g. epsilon-greedy approaches) it is labelled as **off policy learning** [45].

2.5. Is a model useful?

When learning from an environment, an agent may just use the rewards received to update its policy, be it by policy search or value function approximations. The agent might also consider a different strategy, to store all the available information (such as the current state, action, new state, reward...) and approximate the transition and reward functions, which will be used to plan ahead. This is the most basic difference between *model-free* and *model-based* approaches, respectively. The former acts, and the latter plans before acting.

Ground-truth models are nowhere to be found in real-world problems, so if the agent wants to model the environment, it must learn through experience, which creates considerable challenges. A very concerning one is that bias in the model will likely be exploited by the policy, so even if the agent excels at the simulation, it will likely behave suboptimally in the real world [1]. Learning models can provide numerous benefits that will be detailed later, but it is fundamentally difficult. The ease of implementation and tuning has resulted in model-free algorithms attracting many researchers and becoming more popular [1].

2.5.1. Model-Free

All main classes of policies have now been employed in some form to solve RL problems [45], but there were two classes in the spotlight, especially in model-free RL, until a hybrid of both was developed to hopefully use their best qualities.

- **Actor-Only Methods [31]:**
  - Work with PFA.
  - Estimate the gradient of performance with respect to the Actor's DNN parameters directly by simulation. The simplest representation of such a loss function can be seen in Equation 2.12.
  - May have large variance in gradient estimators and lack continuous learning, since new gradients are estimated as the policy changes that do not include past information.
  - Examples: REINFORCE, Deterministic Policy Gradient (DPG)
- **Critic-Only Methods [31]:**
  - Rely exclusively on VFA.
  - Aim to learn an approximate solution to the Bellman equation, hoping to shape a near-optimal policy.
  - Success in constructing a good VFA may not guarantee near-optimality of the resulting policy. More details on the equations behind value function estimation can be seen in Section 2.7.3.
  - Examples: Deep Q-Networks (DQN)
- **Actor-Critic Methods [31]:**
  - Combine the strengths of Actor-only and Critic-only methods.
  - The value function learned by the Critic updates the Actor's policy parameters, aiming for performance improvement. A loss function that captures this approach can be found in Equation 4.22. Convergence is typically easier to achieve with gradient-based methods, and there is often little guarantee that the VFA will converge.
  - Examples: Advantage Actor-Critic (Advantage Actor-Critic (A2C)), Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO)

The computational effort of these algorithms takes place mainly **before deployment** in real world scenarios, and **even simple systems require large amounts of computation** [5].

2.5.2. Model-Based

There are two main reasons for learning a model: sample efficiency and generalisation. Sampling efficiency refers to the ability of an agent to acquire a useful policy through a minimal number of interactions with the environment [5]. Generalisation, on the other hand, distinguishes model-based methods from model-free approaches by enabling agents to **learn** the underlying dynamics of an environment and **plan** ahead in uncertain territories [32].

Learning within model-based frameworks involves approximating an imagined model of the environment by estimating transition and objective functions. This allows predicting future negative rewards and destructive pathways [32]. NNs are commonly employed for these approximations, as they impose no restrictions on model complexity and do not require an analytical approach [5]. However, this flexibility comes at the cost of explainability, which remains a significant drawback in such systems.

To improve the accuracy of the learned dynamics, researchers often adopt techniques that minimise bias and variance. Plaat et al. [44] identify three main strategies used for this purpose: probabilistic inference, ensemble models, and latent models.

Planning, another key component, uses the learned system dynamics to guide the agent toward optimal decision-making. A common approach is to train model-free algorithms within simulated environments, where vast amounts of synthetic data are available. This is particularly useful when sampling from the actual environment is costly or impractical. Despite potential inaccuracies in the model, planning aims to produce effective policies. While brute-force methods like value iteration are sometimes used, lookahead methods such as trajectory rollouts or MPC are often preferred, especially for high-variance or complex transition models [44].

Some advanced methodologies incorporate planning into the deep learning process itself. By employing differentiable algorithms, these end-to-end approaches backpropagate rewards across all components, from actions to observations [44].

Despite its versatility and efficiency, the model-based paradigm faces notable challenges. DNNs, although powerful, often require large datasets to avoid overfitting due to their abundant degrees of freedom. Thus, a challenge of deep model-based RL is to find accurate transition models without many samples [44]. Additionally, model inaccuracies can accumulate when predicting various steps ahead, leading to unreliable policies [41]. Other challenges include the inability to account for diverse potential future outcomes and excessively confident predictions beyond the data used for training [32].

2.6. Defining a Model-Free Algorithm

In Section 2.5 two different approaches were presented: model-free and model-based. Before continuing with the Literature Review, due to the depth of both fields, a choice must be made to further the research. For this thesis, **the model-free approach is selected.**

In pursuit of the simplest possible policy that can rival MPC performance, addressing *Sub-Research Question 1* (Section 2.11), the relative simplicity and well-documented implementation of model-free algorithms make them more appropriate than model-based approaches for this study.

The model-free approach is further supported for three reasons. First, the availability of a virtual environment for agent interaction, presented in Chapter 3, mitigates the sample efficiency concerns that typically favour model-based methods. Second, robust agent selection, discussed in Section 5.4, addresses generalisation requirements. Third, the use of algorithms implemented in the Julia Language [8] enables fast and cost-effective retraining when needed.

2.6.1. Class Selection: Actor-Critic

The SDP that will be addressed in this thesis (detailed in Chapter 3) will be defined in **continuous state and action spaces**, which presents unique challenges, therefore some thought must go into

choosing the appropriate class of model-free algorithm (from among those defined in Section 2.5.1). For a continuous action output, the use of policies based only on value function approximations (e.g. Q-learning) is problematic because they will discretise the action space, leading to a loss in precision.

Discretising introduces the *curse of dimensionality*, so that for every increase in the number of possible actions, there will be an exponential cost of computation and a reduction in sample efficiency. Exploration in high-dimensional spaces becomes difficult, and suboptimal policies may be reached. A good solution to this problem, mentioned in Section 2.3 is to approximate the value function V with a neural network (NN). However, once it is approximated, a new problem emerges: Finding the maximum value action for the current state from a continuous action space is computationally very expensive.

The Actor-Critic class addresses this new problem in various ways, but in most cases the value function is used as the Critic, as a bias to reduce the variance of the policy gradient estimates and thus stabilise the updates of the Actor, which take place *without the need for discretisation*. In addition, the Actor-Critic configuration improves upon the Actor-only, and provides versatility and stability. In their comprehensive review, Fu et al. [18] examine the emerging applications of RL techniques to optimise energy efficiency in buildings, and for scenarios involving continuous action domains, they specifically recommend the use of Actor-Critic algorithms.

2.6.2. Stochastic Policy Gradient

Actor-critic methods are essentially *Policy Gradient* algorithms since their updates require computing gradients of the expected reward with respect to the policy parameters. The **policy is represented by the Actor**, and the Critic will enhance gradient computation. However, there are two main types of *Policy Gradient* (PG) algorithms:

- **Stochastic PG**: In continuous action spaces it outputs the parameters of a distribution, commonly the mean and the standard deviation of a Gaussian, from which actions are sampled. This randomness promotes exploration, and reduces the risk of overfitting or getting trapped in local maxima of the reward landscape. These methods are generally more robust to hyperparameter settings and can take into consideration the uncertainties of the environment. Some examples are: TRPO, Soft Actor-Critic (SAC), A2C and Proximal Policy Optimization (PPO).
- **Deterministic PG**: Outputs an action directly, it was conceived for continuous action spaces. Although it avoids sampling, which increases efficiency, noise must be added to force exploration. They heavily rely on the Critic's accuracy for policy updating, with errors in value estimation leading to unstable updates. Does not naturally include entropy in the objective, as it does not operate over a distribution. If regularisation is needed, it must be introduced through other means. Deterministic at deployment, producing consistent actions for the same input states. Some examples are: Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3).

With this comparison in mind, the **Stochastic Policy Gradient** was chosen for the project, and three algorithms of increasing complexity will be explored. The most basic is the Vanilla Policy Gradient [60], then the more sophisticated Advantage Actor-Critic [39] and finally the Proximal Policy Optimization [51]. These algorithms are also referred throughout the thesis as simply *Policy Gradient* or *Policy Updating Algorithms*, and the theory essential to their understanding will be found in Sections 2.7.2, 2.7.3 and 2.7.4.

The progression in algorithm complexity allows for an analysis that starts at the foundational concepts and ends with the more advanced techniques. This approach provides insights into their relative performance and applicability to the research problem. Detailed explanations of the algorithms' theory and implementation are given in Section 4.5.

2.7. Updating the Agent

The Actor and the Critic components of the RL **agents** put forward by this thesis will be **implemented as independent deep neural networks (DNNs)**, which means the approach qualifies as **deep reinforcement learning (DRL)**, though it will be referred to simply as RL. Some architectures use a shared DNN for both the Actor and the Critic, with only the final layers separating them, however, Andrychowicz et al. [4] do not recommend this approach.

Throughout the present work, the structure and internal components that make up the agent's DNNs will be altered and optimised to achieve the most effective design, identified as the *top RL agent* in Section 6.2. However, before this optimal configuration can be reached, the process by which the RL **agent learns from experience** must be described and understood.

In the RL framework, the two key components are the agent and the environment. The environment reflects the external world, allowing the agent to engage with it by observing its state and making decisions. The impact of these actions is communicated via reward signals, allowing the agent to learn a good strategy through **a process of trial and error** [7]. The following Sections will delve into the theory of reward signals, as well as detailing how their impact moves the agent towards more optimal policies.

2.7.1. Rewards

Rewards were presented as a fundamental part of any SDP (Section 2.2). Within the framework of reinforcement learning, rewards are necessary for guiding the agent to reduce (or increase) the objective function. The specific reward functions applied to train the RL agents in this project will be discussed in Section 4.2. The current Section will examine reward design research and the reward processing techniques used to improve the stability and, when feasible, the speed of learning.

To adhere to the conventions of reinforcement learning and avoid ambiguity, rewards will be represented by the letter R, rather than the generic C used by Warren Powell for *cost*. To preserve the intuition behind the terms, *costs* and *penalties* will be treated as the opposite of *rewards*. Thus, *negative rewards* will be defined as behaviours to be avoided, aligning with *positive costs or penalties*.

Reward Design

Rewards in reinforcement learning are often designed by **trial and error**. In a study by Booth et al. [9], 92% of 24 surveyed RL experts reported adhering to this practice. The authors also mention that their findings are consistent with other studies in the field. While trial and error can be effective, it falls prey to myopic strategies that RL practitioners usually favour, such as prioritising immediate state-action pair rewards rather than thinking of the cumulative reward signal [9], which is responsible for updating the policy.

Typically, reward functions are crafted separately from the policy updating algorithm and NN design. A usual approach is: first the reward is decided upon, then various NN architectures, hyperparameters, and update strategies are explored. If the results are not satisfactory, the reward may be adjusted, though more often, additional hyperparameter tuning and algorithm modifications are attempted.

Booth et al. [9] explored the consequences of this practice and determined that **reward functions can be overfitted** to the chosen hyperparameter set and algorithm. To mitigate this issue, they suggest defining two reward functions: one will be used as a **performance metric**, and the other will be treated as **a hyperparameter**. In the latter case, multiple rewards should be designed, so that the space of parameter optimisation is opened up and subject to less bias.

The performance metric should be designed to represent the desired behaviour of the agent, as the effectiveness of an optimisation algorithm ultimately depends on the quality of the performance metric it is designed to optimise [30]. Therefore, a policy obtained through RL techniques can only be optimal if the performance metric is encoded in the reward function.

The straightforward approach would be to simply use the function that measures performance as the reward function itself. However, depending on how it was designed, it might prove to be inadequate for obtaining optimal results. What **must be encoded** in the reward function is that which the performance metric measures. It is worth noting that numerous functions may be valid for encoding this desired behaviour into the agent's weights and biases.

When additional rewards are introduced to encourage behaviours that are generally advantageous but not included in the performance metric, one has entered the realm of **reward shaping** [30]. This approach attempts to accelerate convergence by guiding the agent in its path to an optimal policy. One potential danger of reward shaping is that it can decrease the upper bound on the policy's performance; fortunately, there is a field of study on *safe* reward shaping to ameliorate this issue.

Reward-to-go

At every timestep, the agent receives a reward, but this immediate feedback is not directly applied by the policy updating algorithms (such as PPO). Instead, the reward is subjected to **processing**. A central concept in this processing is the *reward-to-go*, alternatively referred to as the *return-to-go* or simply *return*. This measure represents the cumulative sum of rewards from a given timestep onwards in a trajectory (τ). It can be expressed as:

$$R_t(\tau) = \sum_k r^{\tau}_{t+k} \tag{2.2}$$

The *reward-to-go* can be conceptualised as an empirical estimate of the action state value function $Q^{\pi}(S_t, x_t)$, since an action is also associated with the reward. This function quantifies the expected return when beginning in state S, taking an arbitrary action x (which may not necessarily conform to the policy π), and then adhering to policy π for all subsequent actions [1].

The *reward-to-go* represents a single sample from this value function, leading to higher variance in estimates. In advanced policy gradient algorithms, instead of relying on empirical estimates, the *reward-to-go* is approximated by a deep neural network (DNN) acting as the state-action value function.

To mitigate the high variance associated with using the raw *reward-to-go* in gradient calculations, a baseline subtraction technique is often employed. The baseline can be used if it only depends on the state. This is because, in expectation, the baseline's impact on the policy loss is zero [1]. This property allows for the incorporation of baselines without biasing the policy gradient. A common choice for the baseline is the **state value function**, i.e., $V^{\pi}(S_t)$. This choice, if combined with the *reward-to-go* leads to an estimate of the advantage function:

$$A^{\pi}(S_t, x_t) \approx R_t - V^{\pi}(S_t) \tag{2.3}$$

In simple terms, the advantage represents how beneficial the last action was compared to the reward you could expect on average from that state, i.e. from the policy's default behaviour [50].

Discounted rewards

The idea of discounting rewards is introduced to balance immediate and future rewards. The agent aims to maximise the expected discounted return, defined as [59]:

$$G_t(\tau) = \sum_k^{\infty} \gamma^k r_{t+k} \tag{2.4}$$

This concept is crucial for two reasons:

- Allows for the convergence of rewards in continuing tasks (those without terminal states). The result of the summation will be bounded because the rewards themselves are bounded.
- Discounting incorporates the uncertainty of the future into the learning process, indicating that immediate rewards are more certain than future ones. However, it allows for less rewarding actions that may lead to higher future rewards to be considered.

This perspective of viewing rewards as something that can be delayed for greater overall benefits, a notion brought forth by the development of society, is represented by the parameter γ. A value close to 0 makes the agent concerned mostly with immediate rewards [59], and $\gamma \approx 1$ makes it take into account the future almost as much as the present. The discount parameter acts as a tool to reduce variance in the calculation of the rewards, at the cost of adding bias [50] (since it is a systematic deviation from the true undiscounted return). The value functions are mathematically expressed as follows:

$$V^{\pi,\gamma}(S_t) := \mathbb{E}_{S_{t+1:\infty}, x_{t:\infty}} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \right] \tag{2.5}$$

$$Q^{\pi,\gamma}(S_t, x_t) := \mathbb{E}_{S_{t+1:\infty}, x_{t+1:\infty}} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \right] \tag{2.6}$$

$$A^{\pi,\gamma}(S_t, x_t) := Q^{\pi,\gamma}(S_t, x_t) - V^{\pi,\gamma}(S_t) \tag{2.7}$$

The superscript π denotes that the function is associated with a policy, and the trajectory is obtained by sampling from the policy $x_t \sim \pi(x_t \mid S_t)$.

Generalized Advantage Estimation

Using the advantage as part of the policy gradient update provides nearly the lowest variance that could be achieved [50], the only problem is that it needs to be estimated, which will be the purpose of the Critic DNN (via the state value function). The next step is to determine what is a good advantage estimator; for that, the work from Schulman and colleagues [50] is very useful.

Equation 2.5 presents the expression for $V^{\pi,\gamma}(S_t)$, which is an accurate estimator of the value function for the discounted reward, since it uses the raw returns in an infinite summation. To use Schulman's terminology, $V^{\pi,\gamma}(S_t)$ is a γ-just estimator, since it does not introduce any new bias in the prediction of the discounted rewards which, to be clear, are already biased by the presence of γ.

In practical terms, it is unlikely to have access to a γ-just value or advantage function. Furthermore, while using a summation of discounted rewards provides unbiased results, it often leads to high variance. To address this, an approximation must be used, but unless it's γ-just, it will introduce bias. On one side, high variance will cause large fluctuations in gradient estimates, slow convergence, and require more samples to average out the noise. On the other hand, adding bias may cause a failure in convergence or reach a poor solution [50], regardless of the amount of samples. Attempting to find a proper balance, Schulman et al. [50] proposed advantage estimators that reduce considerably variance while keeping bias to a tolerable amount, this approach is called Generalized Advantage Estimation (GAE).

From the possible γ-just advantage estimators (e.g. $A^{\pi,\gamma}(S_t, x_t)$), a particularly useful one for code level implementation is the Temporal Difference (TD) residual of $V^{\pi,\gamma}$ [50], since only an approximation of the value function is needed:

$$\delta_t^{V^{\pi,\gamma}} = r_t + \gamma V^{\pi,\gamma}(S_{t+1}) - V^{\pi,\gamma}(S_t) \tag{2.8}$$

$$\mathbb{E}_{S_{t+1}} \left[\delta_t^{V^{\pi,\gamma}} \right] = \mathbb{E}_{S_{t+1}} \left[Q^{\pi,\gamma}(S_t, x_t) - V^{\pi,\gamma}(S_t) \right] = A^{\pi,\gamma}(S_t, x_t) \tag{2.9}$$

Above can be seen that $r_t + \gamma V^{\pi,\gamma}(S_{t+1})$ is considered equivalent to $Q^{\pi,\gamma}(S_t, x_t)$, since the action taken is implicit in the immediate reward r_t.

Instead of $V^{\pi,\gamma}(S_t)$, only an approximate value function $V(S_t)$ is available in practice, which as mentioned before, will introduce bias. However, if a sequence of steps are added together, the impact of the bias introduced by the approximation diminishes, since more empirical returns are used and the approximation is preceded by a smaller weight in the sum [50]:

$$\hat{A}_t^{(N)} := \sum_{k=0}^{N-1} \gamma^k \delta_{t+k}^V = -V(S_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{N-1} r_{t+N-1} + \gamma^N V(S_{t+N}) \tag{2.10}$$

For clarity, it is worth noting that the first term of the summation found in Equation 2.10 is not considered to introduce bias because it is a function of the state sampled before the action x_t. In its current form, Equation 2.10 is equivalent to the **N-step** advantage estimator, which offers a way to bridge the gap between TD and Monte Carlo methods, where a number of steps (N) can be used to mitigate the bias of one-step TD [59].

The main insight of generalized advantage estimation (GAE) comes into play when an exponentially weighted average of the estimators is obtained by incorporating the parameter λ, as shown below. This allows for a finer control over the bias-variance trade-off, enabling a better balance between the accuracy and stability of the advantage estimator [50]:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}^V \tag{2.11}$$

The two edge cases of GAE are useful for understanding its purpose:

- For $\lambda = 0$ only the immediate TD residual will be used (δ_t^V), which reduces considerably the variance at the cost of potentially introducing a high bias (if the value function is not γ-just), since only one step is utilised.
- For $\lambda = 1$ is equivalent to just using empirical returns, having high variance and no bias.

The role of λ is then to manipulate the steepness of the descent of the contribution that future step estimates have on the total summation. In other words, a low λ will reduce the impact of the later terms that contribute to the summation, and as it progresses further in time, the reduction becomes exponentially more noticeable. In practice, there are no infinite summations, episodes or batches have a determined length when estimating the policy gradient and so the approximation of the value for the last state will be given by $V(S_t)$, which is why GAE is a useful tool.

It should be pointed out that λ and γ are independent parameters, and will have different ranges of values [50]. γ influences the scale of the value function $V^{\pi,\gamma}(S_t)$, while $\lambda < 1$ reduces the variance and only adds bias if the value function approximation is inaccurate (not γ-just). GAE will be a crucial part of the policy updates of advanced algorithms like A2C and PPO.

GAE has been shown to provide superior performance over N-step [4], but there exists another advantage estimator called **V-trace** [15]. V-trace is slightly more modern than GAE, and its contribution is adding importance sampling weights to the summation, to take into account that the policy being updated may be a bit different from the one that generated the trajectory. However, Andrychowicz et al. [4] have compared GAE and V-Trace without finding a significant performance difference, which means that the added complexity of V-trace is not needed to obtain policy updates of similar quality.

2.7.2. Actor loss

The RL agent will be made up of an Actor and a Critic deep neural network, but the one making decisions, and thus considered to **represent the policy**, is the Actor. In this section, the focus will be

placed on how to update the parameters of the Actor Network.

If any learning is going to take place, the loss function must be differentiable with respect to the Actor's parameters (θ). This means that the policy (π) must be represented in some way as part of the loss. The other element that must be present is a way of determining how good or bad the actions taken by the agent were, and for Actor-Critic algorithms, this task belongs to the Critic.

NNs aim to maximise [50] rewards, but to do so, they can only modify the probability distribution over their actions.

Stochastic policy gradient algorithms have been chosen for this research, so their loss function will be analysed, which is fundamentally equivalent across all algorithms of this type. The idea behind the loss function began with the REINFORCE paper written by Ronald J. Williams and colleagues [65] in 1992. It presented a Monte Carlo method for estimating the policy gradient, which allowed the update of the policy directly, just making use of past trajectories, without any need for an internal model of the environment. This function has seen countless alterations, but the basic formulation to estimate the gradient [50] can be represented by this equation:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} R_t \nabla_\theta \log \pi_\theta(x_t|S_t)\right] \tag{2.12}$$

Function R_t is the *reward-to-go*, described in Section 2.7.1, which has been replaced in newer versions of the algorithm. The *reward-to-go* can be discounted, a baseline can be subtracted (to reduce variance), or it could even be normalised. However, modern implementations often employ GAE as a more sophisticated advantage estimator. Using the advantage makes intuitive sense in the context of loss function minimisation. It guides the gradient update of parameters such that better-than-average ($A^\pi(S_t, x_t) > 0$) actions become more likely, and worse-than-average actions ($A^\pi(S_t, x_t) < 0$) see their probability reduced.

Another relevant observation about the loss function is that it utilises the natural logarithm of the action probabilities, henceforth referred to as "**log probabilities**" rather than the probabilities themselves. This logarithmic transformation offers some advantages:

- **Normalization of Impact**: it normalises the impact of all actions. Without the natural logarithm, the more likely actions would have a disproportionate effect on the loss function, since they will appear more frequently in the trajectory used for updating. Even actions yielding only slightly above-average rewards would bias the policy towards them.
- **Numerical Stability**: Multiplying many small probabilities can quickly result in values too small to be properly represented with floating point precision, resulting in numerical underflow. Using log probabilities makes it into a summation, avoiding the problem.

2.7.3. Critic loss

The Critic is used to act as a baseline, to reduce the variance of the returns used by the Actor during the training steps. To be an appropriate baseline, the Critic must depend only on the state, guaranteeing that no bias will be introduced. The most common solution in policy gradient algorithms to this constraint is to make the Critic approximate the **state value function.** This has the added benefit of providing a great tool for estimating the advantage, as demonstrated with the summation of TD residuals (computed via an approximate value function) in Equation 2.10 (Section 2.7.1).

When using an N-step advantage estimator (see Section 2.7.1), which doesn't rely on the added parameter of GAE, the Critic will be made to minimise the mean squared error (MSE) loss [33]:

$$\frac{1}{|\mathcal{D}|}\sum_{\mathcal{D}}\left(\left(\sum_{k=0}^{N-1}\gamma^k r_{t+k} + \gamma^N V_\phi(S_{t+N})\right) - V_\phi(S_t)\right)^2 \tag{2.13}$$

Here, $\mathcal{D}$ is a batch of transitions that have been gathered by the Actor's interactions, and ϕ represents the parameters of the Critic's DNN. It may seem confusing to some to realise that the loss function trying to be minimised in Equation 2.13 is basically the advantage estimator shown in Equation 2.10. However, it serves a different purpose. For advantage estimation, the aim is to compare the estimated return from taking the action x_t in the state S_t with the estimated return of being in state S_t.

When the Critic is updated, by using multiple samples, the first term $\sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N V_\phi(S_{t+N})$ will effectively estimate the average value of states under the current policy, since the actions that produced the rewards were sampled from the policy. Consequently, the term approximates in expectation the true value function [33], and it will be treated as independent of ϕ. This process illustrates a relevant concept: the action value function estimates, when sampled according to the policy, **converge in expectation to a value function estimate**.

Other approaches involve the use of the Huber loss, which behaves like MSE for small differences, but it outputs the absolute error for those that go above a chosen threshold. However, Andrychowicz and colleagues [4] compared it with MSE and their results were worse for all environments when using the Huber loss, even when tuning the threshold, so they discourage its use.

In the case of using GAE (Section 2.7.1), the value function will be learned by minimising:

$$\frac{1}{|\mathcal{D}|} \sum_{\mathcal{D}} \left(\left(\hat{A}_t^{GAE}(S_t, x_t) + V_\phi(S_t) \right) - V_\phi(S_t) \right)^2 \tag{2.14}$$

Here, for the reasons stated above, the first term will also be treated as independent of ϕ.

2.7.4. Loss Gradients

The loss function serves as a quantitative measure of the performance of the model, unique to each algorithm. It is the direct result of the action distributions generated by the forward propagation of the input (the state of the environment) through the network's architecture, which consists of weight matrices, bias vectors, and activation functions. The activation functions (Section 2.8) are conventionally determined with the network's initialisation, but the weights and biases are considered tunable parameters.

Once the loss is calculated, the updating of the parameters of the DNN may begin. The first step is called backpropagation, and its purpose is to determine the impact that small variations in each parameter will have on the final loss; this is, of course, a gradient in parameter space.

However, the raw gradients are not directly applied to update the network parameters. Instead, some processing is done, with the main objective of **aiding convergence and optimisation stability**. After presenting the backpropagation procedure, techniques like gradient clipping and adaptive learning rate will be explored, which rely on altering the gradients or the learning rates.

Backpropagation

Central to any optimisation approach is minimising the loss function, which may be done by navigating along some directions of the parameter space. However, there is one direction which is steepest, and that is determined by the gradient, which may be understood as (the transpose of) the Jacobian matrix of a scalar-valued function. The gradient of interest connects the loss function directly to the DNN's parameters. To efficiently compute this gradient in terms of both memory and time, some approaches must be discarded, like the use of numerical approximations or more classic symbolic differentiation algorithms [54]. The most optimised approach is known as **automatic differentiation (AD)**.

Every operation performed during the forward pass of the inputs across the DNN can be fragmented into smaller operations for which the derivatives are already established and stored in the code. The sequence of operations are recorded on an *evaluation trace* and could be ordered on a directed acyclic graph. Then, leveraging the chain rule, all the derivatives between the loss and the network parameters can be merged. Thanks to the use of *evaluation traces*, advanced algorithms can differentiate even

when control flow elements are used in the code [54], such as conditionals and loops. Even if there were many branching paths that could have been followed during the forward pass, only one was taken, and thus the past is traced.

automatic differentiation (AD) exists in two modes: forward and reverse. For DNNs, the **reverse mode** is frequently used because its time complexity is considerably lower when dimensionality of the output space is significantly smaller than that of the input space. Reverse AD can obtain the gradient of a scalar loss with respect to all input parameters in a single backward pass (one row of the Jacobian). Even in reverse mode, the forward pass needed for inference will still take place, and all the intermediate variables will be stored, so that they can be used later to obtain the corresponding derivatives.

It will likely become more clear with a simple example. A very common activation function is the hyperbolic tangent, which will be applied to the output of one layer before it is introduced in the next. For a function $y = \tanh(z)$, the output derivative with respect to the input z is:

$$\frac{dy}{dz} = 1 - \tanh^2(z) \tag{2.15}$$

As can be seen, in this case only the result from the forward pass (the value of $\tanh(z)$) will be needed to compute the derivative. This process will start at the last layers of the DNN and each gradient will be passed through a *pullback* function. The *pullback* function applies the chain rule, as will be shown below. In this manner, frequently called **backpropagation** when applied to DNNs, the gradient of the final loss will be *propagated backwards* from the deepest to the shallowest layers. The pullback functions are an efficient implementation of the Vector Jacobian Product, because there is usually no need to calculate the whole Jacobian matrix for the task of backpropagation.

To illustrate how the pullback function works for the tanh activation, let us consider how it would affect a loss function that depends on the output y. The incoming gradient ($\partial Loss/\partial y$) would be multiplied by the local derivative of the tanh function, obtaining therefore the gradient of the loss with respect to z.

$$\frac{\partial Loss}{\partial z} = \frac{\partial Loss}{\partial y} \cdot \frac{dy}{dz} = \frac{\partial Loss}{\partial y} \cdot (1 - \tanh^2(z)) \tag{2.16}$$

When it comes to the implementation of AD, efficiency is crucial, particularly because backpropagation is one of the most computationally intensive operations within DNN optimisation. As is often the case, efficiency is usually traded for generality, and having a considerable amount of both is a significant programming challenge [48]. Some libraries attempt this with surprising success, and this thesis will utilise one such library, *Zygote* [25], which is implemented by Michael Innes and other contributors in the Julia programming language [8].

The usefulness and efficiency of *Zygote* [25] originate from its innovative attempt to improve the idea of the *evaluation trace*. When faced with dynamic code, instead of tracing a single path from the myriad of possible branches, *Zygote* employs a *source code transformation* to generate static code that encompasses all potential branches. The library then traces the branches that are traversed during execution. This feature reduces uncertainty and essentially avoids unrolling the dynamic parts of the code [48]. When cleverly implemented, its more static nature enables for low-level optimisation. It is worth mentioning that DNNs are not heavily reliant on dynamic code, and therefore the advantage offered by *Zygote* in this case is that of efficiency and convenience.

Gradient Clipping

There are various gradient-based methods used to try to minimise differentiable non-convex and potentially stochastic functions. The canonical approach is **stochastic gradient descent (SGD)**, which although simple, manages to achieve great empirical results with theoretical guarantees [67]. As can be expected, if the function is trying to be maximised, then gradient ascent will be performed. SGD updates the parameters of the Actor or Critic by using a randomly selected subset of transitions from the environment. For on-policy algorithms these subsets are frequently random subsets of an episode recently finished in the environment. Machine learning has witnessed an evolution in recent years, new

methods that improve upon SGD's results have been designed, and gradient clipping is particularly successful among them.

The formulas presented in this section will be used to illustrate the main insight behind the method, while avoiding unnecessary complexity [67].

Gradient Descent:

$$\theta_{k+1} = \theta_k - \eta \nabla f(\theta_k) \tag{2.17}$$

Clipped Gradient Descent:

$$\theta_{k+1} = \theta_k - h_c \nabla f(\theta_k), \quad \text{where} \quad h_c := \min\left(\eta_c, \frac{\gamma_{clip} \cdot \eta_c}{\|\nabla f(\theta_k)\|}\right) \tag{2.18}$$

- θ_k: The current parameter point or vector at iteration k.
- θ_{k+1}: The next parameter point or vector at iteration $k+1$.
- f: The objective function trying to be minimised.
- η: The fixed step size (learning rate) for ordinary gradient descent.
- $\nabla f(\theta_k)$: The gradient of the function f at point θ_k.
- h_c: The step size for clipped gradient descent. It may be considered a constant step (η_c) with an alteration of the gradient ($\gamma \cdot \|\nabla f(\theta)\|^{-1}$), or an adaptive step size keeping the gradient constant.
- γ_{clip}: A constant factor in the clipped gradient descent formula. May be understood as the clipping limit.
- $\|\nabla f(\theta)\|$: The norm (magnitude) of the gradient.

Through the work of Zhang and colleagues [67] it has been proven that gradient clipping (and normalised gradient) **converges arbitrarily faster than gradient descent** with a fixed step size. In addition to accelerating convergence, clipping gradients is effective in dealing with **exploding gradients**, an issue often encountered in deep neural networks, and proves fundamental for training complex models like Recurrent NNs [37].

The phenomenon of exploding gradients is more pervasive than it may initially appear. The research carried out by Philipp, Song and Carbonell [42] contributes considerably to the understanding and definition of this problem. They affirm that it is not a simple numerical anomaly, but rather a sign of a fundamentally complex optimisation problem, limiting the effective depth of DNNs. To further the understanding of this issue, exploding gradients will be discussed in detail in Appendix A.1.

Adaptive Learning Rate

Beyond altering the gradient for network updates, there are additional techniques that adjust the learning rate for each parameter. The most popular amongst them is Adam, derived from *Adaptive Moment Estimation*, which was introduced by Kingma and Ba in 2014 [29]. It became the standard optimiser, thanks to its efficiency and ease of implementation. The equations behind Adam are the following.

Moment updates:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.19}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.20}$$

Bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.21}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.22}$$

Parameter Update:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.23}$$

- g_t: Gradient at time step t
- m_t: First moment estimate (mean of gradients)
- v_t: Second moment estimate (uncentered variance of gradients)
- β_1, β_2: Exponential decay rates for moment estimates
- α: Learning rate
- ϵ: Small constant for numerical stability

As can be seen, Adam incorporates momentum, accelerating the convergence to a local minimum by using the exponential moving average (EMA) of the gradient's first and second moments. It also performs bias correction, since the initialisation of the moment estimates as vectors of zeros biases them, especially during the first timesteps [29]. It is also considered robust, since it is less sensitive to hyperparameter tuning [67] than other approaches.

Despite these virtues, Adam is not the ideal optimiser, and its limitations have been known for years, forcing researchers to find better algorithms, and the resulting Adam variants are quite diverse. AMSGrad dealt with the converge problems of Adam by using the maximum of past squared gradients, Yogi took into account the size of mini-batches, and RAdam rectifies the variance of adaptive learning rates [68].

These are just a few of the alternatives developed through the years, but they often fall short in generalisation performance compared to unaltered SGD for large-scale datasets [68]. Even so, a promising algorithm was developed by Zhuang et al. [68] in 2020 called **AdaBelief**, it provides fast convergence, good generalisation, training stability, and all with the *same computational cost* of first-order gradient methods such as Adam. The core insight behind AdaBelief is to use the EMA of the gradient's first moment as the prediction (or *belief*) of the next timestep's gradient. If the new gradient deviates significantly, the learning rate is reduced, resulting in smaller steps. AdaBelief differs from Adam in the calculation of v_t and therefore in the parameter update step. The key differences are the following.

Second moment update:

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(g_t - m_t)^2 + \epsilon \tag{2.24}$$

Bias correction for s_t:

$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t} \tag{2.25}$$

Parameter update:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon} \tag{2.26}$$

AdaBelief scales the update in proportion to the change in the gradient, which relates to the Hessian [68], and thus approximate curvature information is introduced in a first-order method, leading to better performance than Adam. A clear example that shows AdaBelief taking advantage of curvature knowledge is this: If the current gradient is large, and the difference with the previous gradient is small, then ideally the step should be large, because although it's a steep slope of the loss function, it's not noticeably close to the minimum, since its curvature is small. In this case, Adam would make a short step, whereas AdaBelief would make a large step [68].

Another relevant improvement from AdaBelief is that it does not ignore the sign of the gradient in the second moment estimate. If the gradient is changing direction frequently while having a similar magnitude, the learning rate will be small for both Adam and AdaBelief, but if the gradient sign is constant, then Adam will still perform a small update while AdaBelief will not.

2.8. Activation Functions

The architecture of DNNs consists of an input and output layer and various hidden layers between. At its core, each layer performs a linear transformation, represented mathematically as a matrix multipli-

cation between a weight matrix and the output vector from the previous layer. This cascade of linear transformations is inherently limited in the data that can be represented. To increase the complexity of the patterns that the DNN can approximate, non-linearity has to be introduced in the form of activation functions.

Activation functions are applied at the end of each hidden layer, and sometimes after the last layer as well, possibly with the purpose of normalising the output or simply bounding it. Even though they are usually simple operations, they can differ in their main properties, some are bounded and zero-centred, and others are not. However, the most used ones are commonly nonlinear, differentiable, and continuous [20].

There are three main problems that affect the decision about the possible activation functions: **vanishing or exploding gradients** and **dead neurons** [20]. When activation functions, such as the sigmoid (shown in Figure 2.1 as $\sigma(x)$), have a very limited output range, all the inputs (i.e. pre-activations) that fall on the saturated regions of the function will be compressed and the respective gradient will become close to zero. If enough gradients are small, they might **vanish exponentially** during backpropagation. The opposite case, that of exploding gradients, is discussed in Appendix A.1.

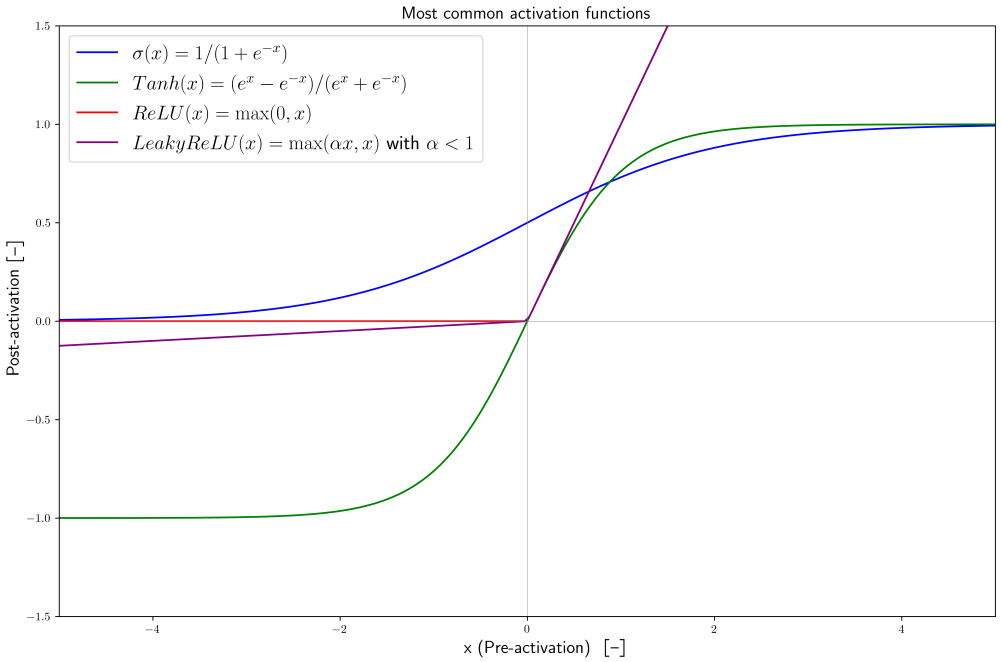

Figure 2.1: Most used activation functions. List from article: [26].

Dead neurons are those that are permanently inactive, the input is not crossing all the layers, and so no update of its weights may take place. The Rectified Linear Unit (ReLU) function shown in Figure 2.1 provides a good example, if the weights are adjusted during backpropagation in such a way that the input to the function remains negative, then there will be no activation of the neuron, and its gradient will also remain null.

While various activation functions have been developed for specific use cases, those commonly depicted in literature (shown in Figure 2.1) remain the most widely used. However, choosing these functions does not guarantee optimal learning, as numerous factors, including loss magnitude and network depth, influence performance. This situation has encouraged the research for a universally effective function that outperforms the rest in common benchmarks [49].

Arguably, the most widely used in DNN is ReLU, which has almost no computational cost and, thanks to its piecewise linear nature, offers a seamless flow for the gradients during backpropagation (when the input to ReLU was positive), this eases the optimisation process [49]. To mitigate the dead neuron problem associated with ReLU, variants like Leaky ReLU have been introduced.

A recent empirical study, by Andrychowicz et al. [4], provided insights into the performance of various activation functions in the context of on-policy RL for continuous control tasks. They tested the following activation functions: *ELU, Leaky ReLU, ReLU, Sigmoid, Swish, and Tanh*. Their final recommendation is to use the hyperbolic tangent (Tanh), challenging the conventional adoption of ReLU, which had the **worst results**. However, they note that Tanh should be used with caution in deeper networks to avoid significant impacts on gradient propagation.

Although vanishing gradients may be a problem with Tanh in certain conditions, it has some great properties. Dead neurons are not likely a concern, and it is zero-centred, which aids considerably in the optimisation dynamics, leading to less epochs of training [20]. Furthermore, the bounded output of Tanh (-1 to 1) can provide a form of implicit regularisation, preventing overfitting and so potentially enhancing the DNN's robustness to input perturbations. Tanh is computationally more expensive than ReLU, but thanks to the efficiency of new backpropagation algorithms, a faster convergence often outweighs the marginal increase in computational cost.

$$Mish(x) = x \cdot \tanh(\text{softplus}(x)) \tag{2.27}$$

Mish, created by Diganta Misra [38], presents a novel non-monotonic activation function that has recently gained attention due to several advantageous characteristics. Mish shares some properties with Swish [49], but it was not tested against Tanh by Andrychowicz et al. [4], and it has been shown to outperform *Swish, ReLU, and Leaky ReLU* [38]. One relevant change from Tanh is that Mish avoids saturation by being unbounded above, reducing the likelihood of vanishing gradients [38]. In addition, its non-monotonic nature makes the relation between input and output less predictable for the DNN, since the reduction of an input signal may cause a larger activation. When this property is effectively utilised during training, it can help extract more complex patterns from the input features. Lastly, Mish has been shown to improve the generalisation capabilities of DNNs, particularly in computer vision benchmarks [38].

2.9. RL in Energy Management Systems

Research by Alanne et al. [2] highlights the remarkable growth in RL-based studies for building EMS, which began to accelerate after 2017. Fu et al. [18] argue that the use of RL within intelligent buildings is an ”inevitable trend in the future” and Pinthurat et al. [43] express that RL-based approaches hold ”great promise for energy management” when discussing smart homes equipped with energy storage systems.

2.9.1. Strengths

After analysing a range of discussions, surveys, overviews, and state-of-the-art applications of RL in EMS, several recurring arguments have been identified that highlight the growing relevance of these algorithms. The following are among the most notable ones:

- **Data-driven solution to a complex problem:** The data provided by buildings is on the rise, as well as the complexity of the control problems they provide. RL, and similar data-driven approaches, use this information to abstract patterns, making them effective in dealing with energy-related problems in complex state and action spaces [18, 41, 57].
- **Increased computing resources:** Advances in computational power reduce the training periods of RL agents, improving the feasibility of their use [18].
- **Interaction demands learning:** While supervised and unsupervised methods prove effective for analysing and predicting patterns, they are often inadequate for managing dynamic interactions. In these situations, RL proves to be a more suitable choice [2].
- **Handling uncertainty:** RL is effective in managing the uncertainties inherent to energy demand, grid behaviour, market fluctuations, or renewable energy generation [41]. Fu et al. [18] also confirm that RL can outperform traditional methods when handling data with significant uncertainties.
- **Validated EMS:** RL is already a proven approach as a highly adaptive EMS. Leveraging real-time data, RL-based systems efficiently optimise energy use, integrate renewable sources, minimise grid dependence, and reduce energy costs for residents [43].
- **Efficient and near optimal:** Even when RL is outperformed by MPC algorithms, its performance is considered close to optimal, with reduced computational requirements. It also provides a significant improvement over widely used rule-based controls [57].
- **Cheaper models:** The cost of modelling system dynamics is replaced by the cost of acquiring quality data, which requires less engineering labour. The RL agent’s ability to explore and exploit enables it to navigate a large decision space with minimal knowledge of the system’s underlying physics [18, 41].
- **Implicit prediction capabilities:** The RL methodology integrates predictive capabilities into its learning process, enabling it to achieve superior forecasting performance compared to supervised approaches without the complexity of explicit prediction modules [53, 47].

2.9.2. Weaknesses

An overview of RL would be incomplete if only its strengths were considered. As a rapidly evolving field attracting considerable attention, its limitations are being discovered, explored and in many cases, addressed through continuous innovation.

One of the main obstacles in the current state of RL is its sample inefficiency, demanding large datasets that are costly to collect [43]. In real-world applications, the data requirements of RL agents often result in extended training durations, potentially spanning months or years [40, 57]. To mitigate this issue, the use of **simulated environments** has become a common practice, as detailed in Section 2.9.5.

Navigating high-dimensional spaces presents a significant challenge for RL, requiring advanced techniques for function approximation. The fundamental trade-off between exploration and exploitation adds another layer of complexity to effective learning [43]. Moreover, reinforcement learning algorithms are critically sensitive to hyperparameter tuning, where minor changes can significantly impact performance.

Safety remains an ongoing challenge, with researchers working to develop mechanisms that guarantee predictable behaviour during both training and deployment, especially in critical applications. Ethical concerns may also emerge, particularly in contexts involving data collection, human interaction, or the deployment of agents without established safety guarantees. Building EMSs often demand sensitive data, including occupancy patterns and the specific times of appliance usage, which poses challenges for open sourcing [41]. Furthermore, the lack of explainability inherent to black-box algorithms undermines the trust that EMS developers may have in their outcomes.

Ongoing research focuses on mitigating the current limitations of RL, such as by developing strategies to improve sample efficiency and refine hyperparameter optimisation techniques [43]. In EMS applications, safety must be prioritised to prevent equipment damage or dangerous indoor conditions. The use of fallback controllers as backups or the implementation of safety layers to shield the EMS from unsafe RL actions are crucial measures [40]. The field's transformative potential hinges on overcoming these technical barriers, with researchers aiming to expand RL's applicability and performance across various domains [43].

2.9.3. Residential Buildings

The focus of the literature review will now be placed on residential buildings, as they are the main concern of this project and represent a key focus area within EMS. The need for building-specific research originates in the distinct operational, structural, and energy consumption characteristics inherent to each building category, which is especially relevant for data-driven algorithms. Such a targeted approach allows for the identification of the limitations and opportunities of each kind of energy system. To establish a structured framework, Shaqour et al. [53] identified five building types: residential, office, university campus/school, data centres, and other commercial buildings.

The residential sector is, in fact, critical for energy research due to its substantial energy consumption and environmental impact. As of 2020, residential buildings represented 22% of global energy demand, considerably more than the 8% contribution of non-residential buildings [53]. Smart buildings are transitioning from passive energy consumers to active, adaptive grid participants, which becomes especially relevant with the incorporation of intermittent renewable energy generation in their energy systems [53].

In their 2022 systematic review, Shaqour et al. [53] identified 470 studies addressing reinforcement learning (RL) for building energy management, 105 of which targeted residential buildings. After screening for quality, clarity, and high relevance for the RL EMS, 31 papers remained. Similarly, 52 papers that targeted non-residential systems withstood the filtering. Perera and Kamalaruban [41] conducted another review in 2021, examining 89 relevant papers. These reviews, while not exhaustive, offer a relevant perspective on the field's recent direction.

In addition to the reviews discussed above, the following review articles, including some from 2024, are recommended to the interested reader [2, 18, 43, 66]. These reviews have been studied, and some of their relevant findings are reflected in the current work.

Technological Scope of the EMS

Shaqour et al. [53] found that 74% of the energy management systems in residential buildings targeted HVAC systems, which echoes the findings of Perera and Kamalaruban [41]. This focus is unsurprising as HVAC systems are often the most energy-intensive components in buildings. Additionally, 19% of the studies explored integrating electric vehicles (EV), and 77% considered demand response strategies. Most importantly to this thesis, **only 11 studies examined the integration of photovoltaic (PV)**

generation and energy storage, and just 3 extended this combination to include EVs.

Alanne et al. [2] highlighted that most research fails to consider the potential availability of rooftop PV generation, despite being increasingly deployed. Finally, only 3.2% of the papers reviewed by Perera et al. [41] addressed the integration of vehicle-to-grid and energy dispatch (i.e. generation and storage).

While residential EMS are typically designed to enhance comfort and reduce energy use and costs, they may also address factors like CO_2 levels or peak demand [53]. However, balancing conflicting objectives, such as minimising energy costs, maximising self-consumption, and ensuring occupant comfort, can complicate efforts to achieve optimal outcomes [41].

The inclusion of energy storage within residential energy systems provides opportunities for RL algorithms to improve battery performance. These agents respond to dynamic factors, such as electricity prices, energy demand, and renewable energy forecasts, in real time. These methods improve self-sufficiency and outperform rule-based approaches in terms of cost reductions [43]. Furthermore, Stoffel et al. [57] confirmed that RL algorithms are an effective and scalable solution for achieving near-optimal EV charging.

Predominant Algorithms

According to Shaqour et al. [53], DQN emerges as the most widely applied algorithm for residential buildings. Other algorithms, such as DDQN and DDPG, also appear frequently, and comparative evaluations of multiple algorithms are common in the literature. Perera and Kamalaruban [41] report similar observations, highlighting the prevalent reliance on Q-learning. In addition, the default function approximators, as noted by Shaqour and colleagues, are DNNs [53].

Actor-Critic algorithms, such as SAC, A2C, and Actor-Critic using Kronecker-Factored Trust Region (ACKTR), have been adopted by 42% of residential studies reviewed by Shaqour et al. [53], although PPO is notably absent in these cases. Perera and Kamalaruban [41] report lower adoption rates for Actor-Critic algorithms in residential EMS and also note the absence of PPO. However, in non-residential settings, Actor-Critic methods were employed more frequently, with PPO accounting for 19% of these studies [53].

Since these reviews, more advanced algorithms such as PPO and TD3 have been increasingly applied to residential building research. For instance, Kang et al. [27] compare A2C, PPO, TD3, and SAC with an Mixed-Integer Linear Programming (MILP) optimisation approach. Similarly, Pinthurat et al. [43] identify three residential studies, among 33 reviewed articles, employing PPO.

2.9.4. PPO in practice

Among the RL algorithms compared in this thesis, PPO [51] represents the most advanced, with VPG-C being the least advanced, and A2CGAE occupying a middle ground. The theory and implementation details behind PPO will be provided in Section 4.5, and its impact on RL agents is displayed in Section 5.3.5. Currently the focus will be placed on how PPO performs in the residential EMS literature, along with its practical advantages and disadvantages.

Andrychowicz et al.[4] recommend using PPO above all the other policy loss functions they tested (*AWR, VPG, RPA, V-MPO, and V-Trace*), due to its superior performance in four out of five environments used in their study. PPO appears to be less sensitive to hyperparameter variations and provides a consistent high performance on almost all conditions. The robustness to hyperparameter fluctuations is echoed by the findings of Pinthurat et al. [43] and Yu et al. [66].

PPO offers other notable benefits, including stability, efficient use of samples, scalability, broad adaptability to various tasks, and straightforward implementation. Nonetheless, it has drawbacks such as difficulty balancing exploration and exploitation, occasional slow convergence, a tendency to settle in local optima, and inconsistent performance across different runs [43, 66].

Section 2.9.3 highlighted the sparse use of PPO in residential EMS, underscoring its potential as an area for deeper investigation, as Perera and Kamalaruban [41] have suggested. One notable example addressing this gap is Kang et al. [27], who presented the Pareto front that emerges when balancing self-sufficiency and the minimisation of peak loads.

Kang and colleagues [27] consider a complex EMS, with batteries, PV generation and a connection to the grid. After comparing the A2C, PPO, TD3, and SAC algorithms, they conclude that A2C and PPO achieved stable learning with improving rewards, while PPO excelled at balancing competing objectives by considering optimisation weights. Conversely, TD3 and SAC displayed erratic behaviour and struggled to converge. However, this is just one example, other studies have reported successful applications of all these algorithms, such as Stoffel et al. with SAC [57].

Within the 2024 review by Pinthurat et al. [43], three additional papers have been found that apply PPO to a residential EMS. These studies report the successful application of PPO to energy systems incorporating battery storage and PV generation, with some verifying PPO's performance through simulations on real-world data or assessing its reliability through robustness evaluations.

2.9.5. MCES Modelling

The implementation of a reinforcement learning agent in an MCES may be done directly, or after using a simulation of the MCES to perform the training. While direct deployment in physical systems can provide real-world data, it presents significant challenges including possible equipment damage, added operational costs, and safety concerns [18], as well as potentially long training times [40]. Though safe learning can offer an alternative path, that will be explored in Section 2.9.6, there has been a widespread adoption of simulation-based approaches for development and testing.

For MCES applications focused purely on power demand management and energy storage, without consideration for detailed building physics or occupant behaviour, simplified modelling approaches may be enough. The key is striking a balance between model fidelity, development costs, and computational requirements. Although more sophisticated models offer greater precision, marginal benefits may not justify the added complexity and resource requirements.

Models can be developed through data-driven black-box models or physics-based building emulators of varying fidelity, including digital twins on the high end. Some studies have compared these approaches, and they suggest that simulation-based methods outperform purely data-driven ones. Furthermore, simulation models allow for predicting system behaviour under previously unobserved conditions [2].

Digital twins represent a promising direction, as they reflect current building conditions. Although agents trained in such environments show high potential for successful deployment in the real counterpart, the computational cost of high-fidelity digital twins remains a challenge. Future research could focus on determining the optimal level of fidelity needed for appropriate generalisation to real environments [2].

2.9.6. Direct Deployment in MCES: Safe Learning

When deploying an RL agent in a real-world MCES, safety considerations are crucial to prevent equipment damage and maintain user comfort. While direct training in the urban MCES was not an option for this thesis, it is of interest to briefly examine some **methods for safe learning.**

Three main approaches to safety in RL have emerged over time. The first type begins with an approximate model of system dynamics and gradually refines the understanding of the system and the control policy. The second category implements safety through reward penalties, though without explicit safety guarantees. Finally, the third approach enhances the agent's reliability by providing safety certificates based on real-world dynamic models developed separately [11].

For instance, one technique creates data-driven barrier certificates by incorporating physical laws

into the learning process, defining a set of safe states that restrict exploration during both the training and deployment phases [7].

As another example, Ceusters et al. [11] developed a model-free approach that combines hard-constraint satisfaction with policy optimisation while maintaining independence from the core RL formulation.

2.10. Research Gaps

While the primary focus of this work is the development of an RL energy management system for a multicarrier electrified residential building (as detailed in Section 1.1), the literature review has uncovered several secondary research opportunities that align with and enrich the main objective:

1. **Comprehensive Multicarrier Energy System:** Substantial research has been conducted on standalone components within residential energy systems. However, the current literature lacks sufficient exploration of RL-based EMS solutions that simultaneously manage multiple types of energy storage, thermal systems, renewable generation, and EV integration is understudied in the current literature [41, 53, 2].
2. **PPO Underutilisation:** While Actor-Critic methods are widely adopted in residential EMS, the application of PPO algorithms remains notably limited in this context, despite their successful implementation in non-residential settings [53, 41].
3. **Hybrid Energy Storage Systems:** The application of RL-based EMS for controlling hybrid energy storage systems in residential energy optimisation remains under explored [43].
4. **Multi-objective Optimisation:** The development of well-defined reward functions for multi-objective scenarios remains a persistent challenge in RL applications for energy management [7].
5. **Limited Comparative Analysis:** While RL shows promising results over simple control strategies, comparisons with advanced methods such as MPC are scarce [41, 2], leaving uncertainty about relative performance benefits.

Beyond the research gaps addressed in this thesis, the literature review has revealed other promising opportunities for RL-based EMSs that, while falling outside the scope of this work, are worthy of note.

Multi-agent RL is frequently identified in the literature as a compelling research direction [53, 2, 43], with some studies demonstrating its superior control efficiency compared to a single agent [18]. This approach shows promise in community-level applications, enabling coordination of distributed energy resources and vehicle-to-grid systems for peer-to-peer energy trading [53]. Another gap appears with the integration of emerging technologies, where advances in 5G networks, edge computing, and blockchain technology could improve device connectivity, enable local decision-making, and secure energy exchanges [43].

A fundamental challenge remains in bridging the gap between theory and practice as, according to Yu et al. [66], only a fifth of studies have validated their findings in real-world settings beyond simulated environments. Physics Informed RL proves to be a promising direction for tackling the *curse of dimensionality* and reducing the distance between simulation and reality [7].

Current safe RL methods for safety-critical applications are too expensive and impractical, leading to the use of simulated environments. This underscores the need for new solutions that guarantee safety throughout the learning process and use existing data effectively to reduce exploration risks in real environments [58].

The field faces additional technical hurdles that require attention: developing methods for automated hyperparameter optimisation to reduce computational burden [66]; creating algorithms that can scale with increasing system complexity [43]; enhancing the interpretability of control decisions [57]; and establishing robust frameworks for data privacy in residential energy systems [43].

2.11. Research Questions

This thesis is driven by one central research question, which is further explored through three distinct sub-questions. These inquiries have been formulated to address the gaps in the literature described in Section 2.10.

Main Research Question

In the management of multicarrier urban energy systems, to what extent can practitioners without expertise in optimal control theory and physics-based modelling leverage reinforcement learning to create a computationally efficient alternative to model predictive control, while maintaining comparable safety and performance?

Even though the Main Research Question is broad, it reflects the first and last research gaps quite clearly. The second gap, concerning the underutilisation of *PPO*, is addressed by the first Sub-Research Question. This gap may stem from the delayed adoption of more advanced stochastic policy gradient algorithms within the research community. It is also plausible that simpler algorithms have been sufficient to achieve results close to optimal in the EMSs studied so far. For this reason, the following question is asked:

Sub-Research Question 1

How do progressively advanced policy update algorithms (from basic advantage estimation to GAE to trust region constraints) contribute to achieving near optimal performance?

Performance in RL algorithms can significantly fluctuate based on changes to the DNN architecture or reward formulations [41]. As referenced by the fourth research gap, this challenge is further amplified by the difficulty of crafting effective reward functions for multi-objective scenarios.

Sub-Research Question 2

What is the impact of DNN architecture sophistication, temporal feature engineering, and reward component shaping for RL algorithms in achieving performance that is close to optimal?

This work seeks to address the fifth research gap by providing a comparative study of RL and MPC. Among the areas examined is the operation of hybrid energy storage systems, which the third research gap identifies as a promising domain for RL-based EMS. This motivates the following research question:

Sub-Research Question 3

How do the RL and MPC approaches compare in terms of operational safety, energy storage use, and economic performance?

3

Energy Management System

3.1. Introduction

The sequential decision problem that will be tackled in this thesis is the energy management of a multicarrier electrified residential building located in the Green Village of Delft University of Technology.

The household under consideration is part of the FLEXINet project. The research by Slaifstein et al.[55] will provide the basis for the equations and problem formulation of this thesis. They also created the Expert that serves as a reference benchmark for performance evaluation.

This chapter presents a description of the SDP –that concerns this thesis– by formally defining its objective function, state and decision variables, exogenous information, and transition functions. Afterwards, the chapter addresses safe behaviour considerations, mainly the algorithms devised to project the agent's actions across training and deployment phases.

The Expert system, used for validating the RL agent's decisions, is introduced in the following section. The discussion then moves to data preparation, exploring both training and testing datasets, with emphasis on the techniques used for synthetic data expansion.

3.2. System Components

The MCES of the building under study integrates thermal and electrical subsystems to meet the residents' heat and electricity requirements ($load$). A diagram illustrating this configuration is presented in Figure 3.1.

In the thermal subsystem, heat is generated by the heat pump (HP) and the solar thermal (ST) system, consumed by the loads, and stored within the thermal energy storage system (TESS). The inclusion of the TESS enables temporal decoupling of heat production and demand, improving the flexibility of the system.

The electrical subsystem has various elements that allow for **bidirectional** energy flow: the power grid ($grid$), the EV battery, and the battery energy storage system (BESS) installed in the building. Within this framework, the HP functions only as a consumer of electrical energy, converting it into heat, while the PV panels act as a source of electrical power.

The subscripts $[PV, ST, grid, load, HP, EV, BESS, TESS]$ will be used to refer to the described elements of the MCES. Additionally, superscripts $[e, th]$ will specify whether they belong to the electrical or thermal subsystems, respectively. The Power Electronic Interface (PEI) will act as a central node to manage the energy distribution, allowing for high optimisation of the system by permitting complete interactivity between the components.

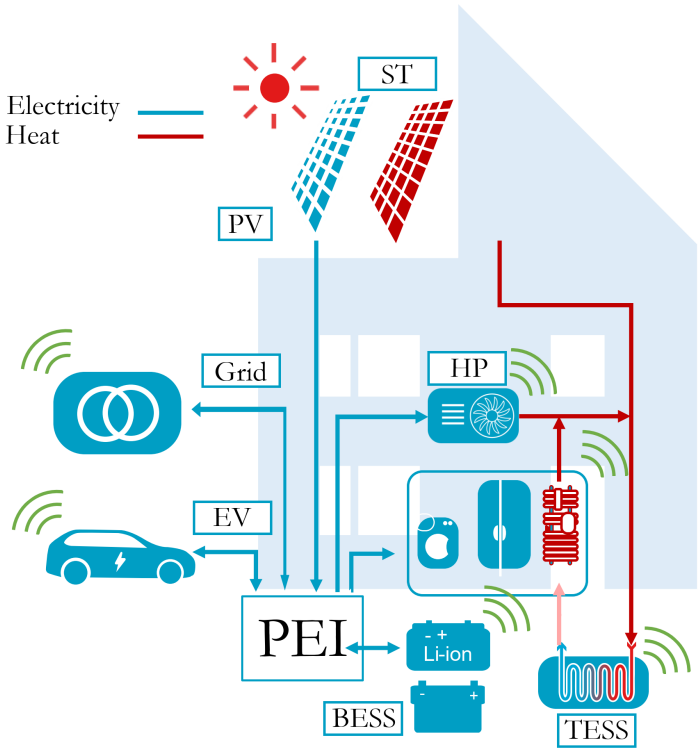

Figure 3.1: Schematic diagram of the MCES from Slaifstein et al.[55]. The arrow points indicate the directions in which energy can flow through the system.

3.3. Sequential Decision Problem

The problem will not be framed as an MDP; instead, it will follow the unified modelling framework proposed by Powell [45], outlined in Section 2.2, and represented in Figure 3.2. Although challenging slightly traditions of RL, this approach appears to present the problem more clearly, making a distinction between the state information available to the decision maker at time t and the exogenous information that will arrive before the next action is taken (Section 3.3.4). Also, the reward should not need to depend just on the current state of the system S_t.

Crucial to the success of any RL algorithm is the development of a working environment where the agent may learn a correct policy. The closer the environment is to reality, the more transferable the agent's knowledge during deployment.

In the following sections, the core components of the SDP to be solved by the EMS (i.e., RL agent or Expert) will be formulated. The equations described in these sections will be implemented in the Julia Language [8] as a dynamic RL environment, representing the MCES. This process begins with the definition of a structured type that contains the state variable, which inherits from the abstract environment type –*AbstractEnv*– of the *ReinforcementLearning.jl* framework [61]. The MCES components outlined in Section 3.2 are integrated as independent structured types within this environment. A transition function is used to update the state of each component, taking into account both agent decisions and exogenous information.

3.3.1. Objective function

The objective function of the RL agent will be based upon that of the Expert (presented in Section 3.5), accommodating for the fundamental differences between the two approaches. The Expert's objective

can be represented as follows:

$$\min_{x_t} \mathbb{E}_W \left[C_{\text{grid}} + p_{\text{SoCDep}} + Aux_{\text{TESS}} \right] \tag{3.1}$$

The optimisation seeks to minimise the expected value with respect to the exogenous variable W (representing the uncertainty) for the sum of three primary components: net grid energy costs (C_{grid}), a penalty for not charging the EV to the desired state of charge (SoC^*_{dep}) at the time of departure (t_{dep}), and the constraint violations of the thermal energy storage system (represented by the auxiliary term Aux_{TESS}).

$$C_{\text{grid}} = w^{\text{exp}}_{\text{grid}} \sum_{t=1}^{T} \left(\lambda_{\text{buy,t}} \cdot P^{+}_{\text{grid},t} + \lambda_{\text{sell,t}} \cdot P^{-}_{\text{grid},t} \right) \cdot \Delta t \tag{3.2}$$

$$p_{\text{SoCDep}} = w^{\text{exp}}_{\text{SoCDep}} \sum_{t \in \mathcal{T}_{\text{dep}}} \left(\xi_{\text{SoCDep},t} \right)^2 \tag{3.3}$$

$$Aux_{\text{TESS}} = w^{\text{exp}}_{\text{TESS}} \sum_{t=1}^{T} \max\left(0, \text{SoC}_{\text{TESS},t} - \text{SoC}^{\text{max}}_{\text{TESS}}\right) \cdot \Delta t \tag{3.4}$$

Where:

$P^{+}_{\text{grid},t}$: Positive value of grid power (energy goes from the grid into the PEI)
$P^{-}_{\text{grid},t}$: Negative value of grid power (energy is extracted from the PEI and sold to the grid)
$\lambda_{\text{buy,t}}$: Buying price of electricity from the grid at time t
$\lambda_{\text{sell,t}}$: Selling price of electricity to the grid at time t
Δt : Time step duration
$w^{\text{exp}}_{\text{grid}}$: Weight used by the Expert for the grid objective
$w^{\text{exp}}_{\text{SoCDep}}$: Weight used by the Expert for the state of charge (SoC) objective
$w^{\text{exp}}_{\text{TESS}}$: Weight used by the Expert for the TESS auxiliary objective
$\mathcal{T}_{\text{dep}}$: Set of all departure times t_{dep}
$\xi_{\text{SoCDep},t} = \text{SoC}^*_{\text{dep}} - \text{SoC}_{\text{EV},t}$
Difference between the desired SoC and the actual SoC of the EV at time t
$\text{SoC}^{\text{max}}_{\text{TESS}}$: Maximum SoC of the TESS
$\text{SoC}_{\text{TESS},t}$: State of Charge of the TESS at time t

It may be noted that charging the EV above $\text{SoC}^*_{\text{dep}}$ will also increase the objective, as the battery should be used as much as possible within the system, and overcharging the vehicle would contradict this aim.

It is also relevant to address the weights used for each component of the objective function. The expert system utilises specific weights tailored to its non-linear optimisation routine at each timestep. These coefficients **do not represent any physical quantity**, they are calibrated to ensure satisfactory system behaviour (e.g., not overcharging the TESS). This abstraction from physical quantities allows for a more flexible control strategy. The performance evaluation between the RL agent and Expert EMSs will analyse **each objective component separately** (Section 6.2.1).

The inclusion of Aux_{TESS} transforms a hard constraint into a soft one, preventing optimisation infeasibility. Section 3.4.1 provides additional context regarding this slack variable, since an analogous implementation in the RL agent's *safe projection* mechanism was introduced.

The RL agent, unlike the Expert, requires explicit training to learn and respect operational constraints, since it starts with no prior knowledge of the MCES component's limitations. Therefore, the

auxiliary TESS term (Aux_{TESS}) is replaced by a more comprehensive projection penalty term (p_{proj}), which accounts for all potential constraint violations. The RL agent's objective function thus becomes:

$$\min_{x_t} \mathbb{E}_W \left[C_{\text{grid}} + p_{\text{SoCDep}} + p_{\text{proj}} \right] \tag{3.5}$$

The projection term is defined as:

$$p_{\text{proj}} = w_{\text{proj}} \cdot (\xi^{\text{SoC}}_{\text{BESS}} + \xi^{\text{SoC}}_{\text{EV}} + \xi^{\text{SoC}}_{\text{TESS}} + \xi^{\text{p}}_{\text{grid}} + \xi^{\text{p}}_{\text{TESS}} + \xi^{\text{p}}_{\text{BESS}} + \xi^{\text{p}}_{\text{EV}} + \xi^{\text{p}}_{\text{HP}}) \tag{3.6}$$

There are two types of projections: *initial* verify that actions proposed by the agent are within bounds before implementation in the environment, while *operational* projections adjust state variables post-transition to maintain system constraints. For a description of the projection methodology and its implementation, the reader is referred to Section 3.4.1.

w_{proj} : Weight for the projection penalty in objective function.

Initial Projection (Action Validation):

$\xi^{\text{p}}_{\text{BESS}}$: Violation of BESS power constraints by RL agent's decision P_{BESS}

$\xi^{\text{p}}_{\text{EV}}$: Violation of EV power constraints by RL agent's decision P_{EV}

$\xi^{\text{p}}_{\text{HP}}$: Violation of heat pump power constraints by RL agent's decision P^{e}_{HP}

Operational Projection (State Constraints):

$\xi^{\text{SoC}}_{\text{BESS}}$: Violation of BESS SoC constraints

$\xi^{\text{SoC}}_{\text{EV}}$: Violation of EV SoC constraints

$\xi^{\text{SoC}}_{\text{TESS}}$: Violation of TESS SoC constraints

$\xi^{\text{p}}_{\text{grid}}$: Violation of grid power exchange constraints

$\xi^{\text{p}}_{\text{TESS}}$: Violation of TESS power constraints

3.3.2. State variable

The state is, by convention, considered to contain variables that evolve over time. The state vector can be further subdivided into physical state information, other deterministic information, and beliefs about the system [45]. Given that beliefs are not an explicit part of the RL framework, and considering that all state information in this problem happens to be physical, state information will be classified under a single class.

$$S_{sa,t} = [SoC_{sa}, v_{sa}, i_{sa}, OCV_{sa}]_t \tag{3.7}$$

$$S_{\text{TESS},t} = [SoC_{\text{TESS}}, P_{\text{TESS}}]_t \tag{3.8}$$

$$S_{P,t} = \left[P_{grid}, P_{\text{ST}}, P^{\text{th}}_{\text{HP}}\right]_t \tag{3.9}$$

subscript sa : Follows the notation by Slaifstein et al. [55]
indicating **storage assets** (BESS and EV battery)
$SoC_{sa,t}$: State of charge of the storage asset at time step t
$OCV_{sa,t}$: Open Circuit Voltage of the storage asset at time step t
$i_{sa,t}$: Current going through the cell of the storage asset at time step t
$v_{sa,t}$: Voltage of the storage asset at time step t
$SoC_{\text{TESS},t}$: State of charge of the TESS at time step t
$P_{\text{TESS},t}$: Average power of the TESS during time step t
$P_{\text{grid},t}$: Average power flow with the grid during time step t
$P_{\text{ST},t}$: Average thermal power provided by the Solar Thermal unit during time step t
$P^{\text{th}}_{\text{HP},t}$: Average thermal power produced by the heat pump during time step t
Proportional to P^{e}_{HP}

Another relevant parameter to consider in this modelling framework is the **initial state** (S_0), which includes the initial values of state variables (or distributions of beliefs about them) and deterministic parameters that remain unchanged. It is worth noting that the PEI, while serving as the central communication node between components, is not represented in the state variable, as it merely facilitates the exchange of power flows.

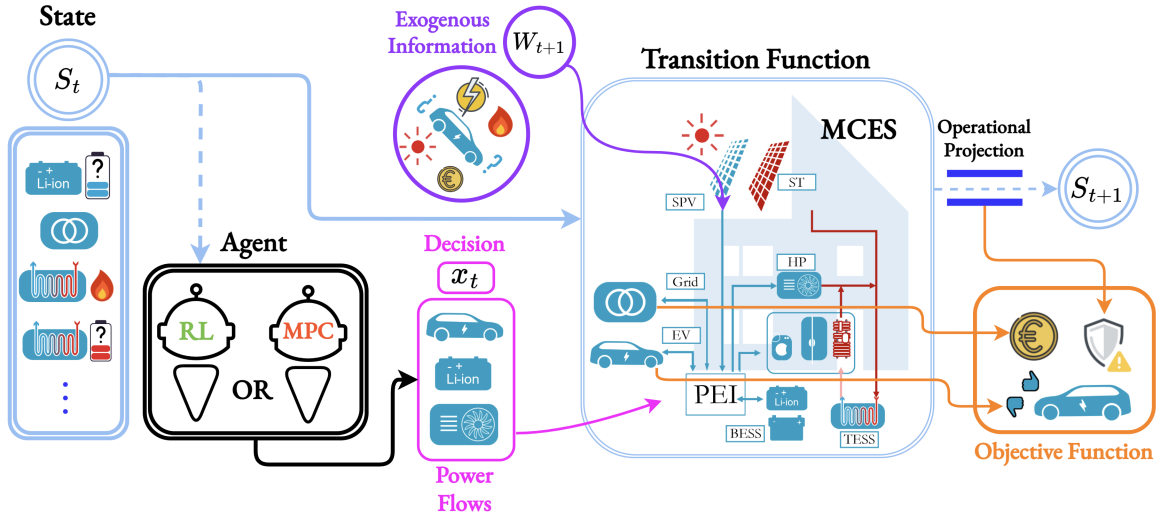

Figure 3.2: This diagram represents the main elements of the SDP addressed by the Expert or RL agent. The notation aligns with that used in Section 3.3 to formulate the sequential decision problem. Dashed lines indicate that data transformations occur between the connected nodes.

3.3.3. Decision variable

The decision variable x_t is a vector containing three continuous power setpoints, associated with the EV battery, BESS, and HP, all within the electrical subsystem:

$$x_t = X^{\pi}_t(S_t) = [P_{\text{EV}}, P_{\text{BESS}}, P^{\text{e}}_{\text{HP}}]_t \tag{3.10}$$

X^{π}_t is the policy function at timestep t, the subscript is relevant as the policy will be modified frequently during training. The decision for the EV battery and the BESS will be positive when energy flows into the PEI and negative if the agent decides to recharge the storage assets.

Decisions are usually **subject to constraints**, and in this case they will be determined by the technical limitations of the affected system components. The process of projecting the decision onto a safe space is described in detail in Section 3.4.1.

It is worth noting that the decisions are discrete power setpoints for a whole timestep, in this case with a duration of 15 minutes (although this value can be modified). Therefore, there must be an underlying mechanism for continuously minimising the difference between the chosen set point and the measured state of the system. Error handling of this manner is usually performed by proportional–integral–derivative (PID) controllers.

3.3.4. Exogenous information

After the agent has made a decision, new exogenous information is received, moving the environment to a new state. The exogenous information is represented by the variable W, following Powell's notation [45], and the subscript t indicates that the information becomes available to the agent at the specified timestep, i.e., it can affect the decisions of that timestep. The elements contained in W_t are average values during the time in between $t-1$ and t. This means that the agent will only have access to exogenous past information when making a decision. Then, when the next batch arrives (W_{t+1}), the transition function is used to update the state variable (as will be described in Section 3.3.5).

$$W = \left[P^{\mathsf{e}}_{\mathsf{load}}, P^{\mathsf{th}}_{\mathsf{load}}, P_{\mathsf{PV}}, \lambda_{\mathsf{sell}}, \lambda_{\mathsf{buy}}, \gamma_{\mathsf{EV}}, P^{\mathsf{drive}}_{\mathsf{EV}}\right] \tag{3.11}$$

$P^{\mathsf{e}}_{\mathsf{load}}$: Electrical load
$P^{\mathsf{th}}_{\mathsf{load}}$: Thermal load
P_{PV} : Power generated by PV panels
λ_{sell} : Price of selling energy to the grid
λ_{buy} : Price of buying energy from the grid
γ_{EV} : Binary variable, equal to 1 if the EV is present in the environment
$P^{\mathsf{drive}}_{\mathsf{EV}}$: Power used by the EV when not connected to the MCES

3.3.5. Transition function

The functions that update the state variable receive the current state ($S_{a,t}$), the agent's decisions $x_{a,t}$ and the exogenous information W_{t+1} (Section 3.3.4) as inputs. The subscript a shown in the equations below represents any of the *assets* described in Section 3.2. The reader is referred to Figure 3.2 for a simplified representation of the SDP.

$$S_{a,t+1} = S^M_a\left(S_{a,t}, x_{a,t}, W_{t+1}\right) \tag{3.12}$$

The equations of the transition function can be grouped into electrical or thermal. To ensure safety, two projection stages (as shown in Figure 3.3) are implemented: an *initial projection* before the transition function and an *operational projection* afterward. All details concerning this safety measures can be found in Section 3.4.1.

Electrical Subsystem

The main equation on the electrical side is the balance of all electrical energy assets with the grid.

$$P_{\mathsf{grid},t} = P^{\mathsf{e}}_{\mathsf{load},t} + P^{\mathsf{e}}_{\mathsf{HP},t} - P_{\mathsf{PV},t} - P_{\mathsf{BESS},t} - \gamma_{\mathsf{EV},t} \cdot P_{\mathsf{EV},t} \tag{3.13}$$

The EV battery and the BESS will have a SoC that needs to be updated as well. The value of $\gamma_{\mathsf{EV},t}$ will determine how the EV battery is updated, since the EV needs to be present for its battery to be useful for the MCES. If the EV is absent ($\gamma_{\mathsf{EV},t} = 0$), the agent's decision $P_{\mathsf{EV},t}$ will be replaced by the exogenous information $P^{\mathsf{drive}}_{\mathsf{EV,\ t+1}}$ for the transition.

The basic equations for modifying the SoC are:

$$\mathsf{SoC}_{sa,t+1} = \mathsf{SoC}_{sa,t} - \frac{\Delta t}{Q_{sa} \cdot 3600} \cdot \eta_c \cdot i_{sa,t} \tag{3.14}$$

$$i_{sa,t} = \frac{P_{sa,t}}{v_{sa,t} \cdot N_{s,sa} \cdot N_{p,sa}} \tag{3.15}$$

$$OCV_{sa,t} = a_{OCV,sa} + b_{OCV,sa} \cdot \mathsf{SoC}_{sa,t} \tag{3.16}$$

$$v_{sa,t} = OCV_{sa,t} \tag{3.17}$$

All the coefficients and values used within the transition function that are not part of the state, decisions or exogenous information are considered **parameters of the transition function**. The main parameters are shown below:

Δt : time step duration
Q_{sa} : cell capacity of the storage asset (Ah)
η_c : Coulombic efficiency
$N_{s,sa}$: number of cells in series for asset sa
$N_{p,sa}$: number of cells in parallel for asset sa
$a_{OCV,sa}, b_{OCV,sa}$: parameters for the Open Circuit Voltage of asset sa

The Coulombic efficiency η_c represents the ratio between the charge extracted from the battery and the charge used to restore its original capacity [34], which applies to both the EV battery and the BESS. However, using the formulas as presented above poses a slight problem when the agent demands power from the storage assets. The Coulombic efficiency is not 1, which means there will be an energy loss, and the actual current that the battery outputs will be $\eta_c \cdot i_{sa}$. To satisfy the agent's power setpoint, a higher current needs to be demanded. Therefore, the modified equation is:

$$i_{sa,t} = \frac{P_{sa,t}}{v_{sa,t} \cdot N_{s,sa} \cdot N_{p,sa} \cdot \eta_c} \tag{3.18}$$

Another consequence of this change is that, when updating the SoC –only for current extraction– there will be no need to apply η_c again, since the storage asset will indeed lose the charge proportional to the increased $i_{sa,t}$. When injecting current into the storage asset, the whole process is reversed, and Equations 3.14 and 3.15 may be used normally. It is worth noting that the same measures will apply when updating $\mathsf{SoC}_{\mathsf{TESS}}$ (Equation 3.23).

Thermal Subsystem

The heat pump will convert electrical energy to thermal, thus connecting both subsystems. The conversion is modelled by a simple scaling where the coefficient η_{HP} represents the HP's coefficient of performance (COP):

$$P^{\mathsf{th}}_{\mathsf{HP}} = \eta_{\mathsf{HP}} \cdot P^{\mathsf{e}}_{\mathsf{HP}} \tag{3.19}$$

The thermal energy storage system (TESS) is fundamental for the balancing of the thermal demand with the sources. When $P_{\mathsf{TESS}} > 0$, the TESS is providing energy to the building (i.e. the MCES). Q_{TESS} indicates the thermal energy storage capacity and η_{TESS} is the thermal transfer efficiency.

$$P_{\mathsf{ST},t} = P_{\mathsf{PV},t} \cdot \eta_{\mathsf{ST}} \tag{3.20}$$

$$\text{Subject to} \quad P_{\mathsf{ST},t} \leq P_{\mathsf{ST}}^{\mathsf{max}} \tag{3.21}$$

$$P_{\mathsf{TESS},t} = P_{\mathsf{load},t}^{\mathsf{th}} - P_{\mathsf{ST},t} - P_{\mathsf{HP},t}^{\mathsf{th}} \tag{3.22}$$

$$\mathsf{SoC}_{\mathsf{TESS},t+1} = \mathsf{SoC}_{\mathsf{TESS},t} - \frac{\Delta t}{Q_{\mathsf{TESS}} \cdot 3600} \cdot \eta_{\mathsf{TESS}} \cdot P_{\mathsf{TESS},t} \tag{3.23}$$

3.4. Safe Behaviour

When training agents with RL, the constraints must be learned, which means that safety cannot be ensured during the training process [13]. In this work, the training takes place within a simulated version of the actual MCES, as explained in Section 3.3. As a result, any violation of constraints will only have **real consequences** after deployment.

The virtual environment will be used by the RL agent to learn the constraints of the MCES. However, even when the agent appears to adhere to all boundaries, a *safe projection* mechanism is implemented after deployment to minimise the risk of unsafe states.

The safety approach after deployment relies on the known system dynamics to estimate the effects of the agent's actions on the following timestep. Although more advanced methods are found in the literature, as noted in Section 2.9.6, they proved unnecessary for the current project, since the *safe projection* layer delivered safety performance comparable with the Expert's (presented in Section 3.5). Further details on the comparison of safety performance are provided in Section 6.2.3.

The upcoming sections present a comprehensive explanation of the safety measures applied in this study. Section 3.4.2 addresses training safety, Section 3.4.1 covers the safety projection mechanism, and Section 3.4.3 examines the safety considerations of deployment.

3.4.1. Safety Projection

Before describing the approach taken to address safety during training or after deployment, two relevant terms need to be introduced. The constraint-enforcing projections are categorised into two types: **initial** and **operational**. The initial projection is applied on the actions that are proposed by the agent (which are obtained by sampling the Actor DNN, as detailed in Section 4.4.2) **prior to their implementation within the RL virtual environment**. The operational projections will take place after the transition function has been used, i.e. after advancing to the next timestep. This projection will act upon the state variables of the environment, modifying them if any constraint has been exceeded.

The primary constraints of the MCES originate from the limitations imposed by system components, such as the maximum current output of a battery cell, the HP's power limit or the minimum acceptable SoC of the BESS, TESS, and EV. A list of all the variables that keep track of possible projections can be found in Section 3.3.1.

Initial projection implementation can be accomplished through either a simple or complex method. The straightforward approach involves restricting the agent's decisions to their predetermined limits (e.g. $-12.5kW \leq P_{\mathsf{EV}} \leq 12.5kW$). However, this solution proves unsatisfactory, as it fails to account for the boundaries of system components directly impacted by these decisions. As an illustration, P_{EV} may be within its limits while demanding power from an EV battery currently at its minimum allowed SoC ($SoC_{\mathsf{EV}} = SoC_{\mathsf{EV}}^{\mathsf{min}}$). This approach, referred to as *simple projection*, inevitably leads to a considerable number of operational projections.

The more refined approach to safety is to use an initial projection that aims to minimise (and ideally eliminate) the need for operational projections, which will be henceforth referred to as *safe projection*. With this approach, the **limits on the agent's actions will become dynamic**, changing with every

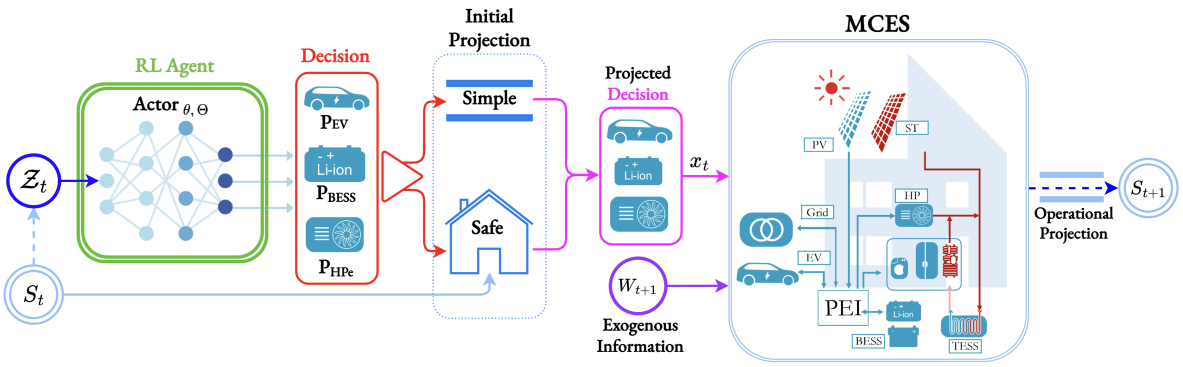

Figure 3.3: This diagram demonstrates the initial and operational projections as part of the sequential decision problem. As shown, the output from the Actor DNN can only be submitted to either the *simple* or the *safe* initial projection. Dashed lines indicate that data transformations occur between the connected nodes.

timestep to accommodate the constraints of system components that will be affected by them.

To be clear, two types of initial projections have been described: *simple* or *safe*. A diagram representing both options, as well as how initial and operational projections are part of the sequential decision problem, can be seen in Figure 3.3.

Safe projection as an optimisation problem

Reducing operational projections through the initial *safe projection* constitutes an optimisation problem, one that aims at achieving the minimal projection necessary of the agent's decisions to satisfy all the physical system constraints.

The optimisation problem needs to be turned into a *model*, which in this case specifically refers to the mathematical programming formulation that abstracts the real physical system into a set of variables, an objective function, and constraints. Therefore, the *model* contains all the safety-relevant equations that are within the transition function of the SDP. A **comprehensive description** of the utilised *model*, including all the equations that will be mentioned in this Section, is provided in Appendix A.2.

The *model*'s objective function focuses on minimising two distinct components: the squared deviation from the reference power setpoints and the constraint violations (through slack variables). These setpoints are selected by the agent at each timestep. The mathematical optimisation is performed using Ipopt (Interior Point OPTimizer) [63], an established open source nonlinear solver accessible via Julia's wrapper interface (Ipopt.jl). The reason behind using a nonlinear solver can be found as well in Appendix A.2.

The *model* faces two principal limitations: the **absence of future knowledge** and the **lack of guaranteed feasibility**. The challenge of operating without future knowledge arises because decisions must be made in real-time, and so their impact on the environment can at most be estimated. Hence, *safe projection* actually produces actions that are only *safe with a high probability*. The impact of this limitation can be lessened in various ways, such as by training an agent that –through its implicit understanding of the future– will produce safe actions, or incorporating explicit estimators of the exogenous information (see Section 3.3.4). The second approach is at the core of Model Predictive Control.

In this thesis, the agents will be trained to avoid operational projections (as will be detailed below), and a safety margin will be used to constrain the *model* more than necessary, therefore incurring a potential loss in the optimality of the solutions, in order to **enhance safety levels**.

Certain MCES components maintain independence from exogenous information (i.e., unknown to the agent at the time of making a decision), such as the BESS SoC, and therefore can be guaranteed to be within bounds for subsequent timesteps. However, variables like grid power exchange (P_{grid}) are partially determined by the exogenous information, which may result in presenting the solver with an unfeasible problem. A practical example occurs when excess thermal energy encounters a fully

charged TESS, the energy will have nowhere to go. This situation could be ameliorated by creating a more realistic transition function that accounts for natural thermal absorption by buildings and air. The second limitation, that of possible infeasibilities, is therefore closely linked to the first. To ensure solution convergence, slack variables are introduced for grid and TESS components, with significant penalties for non-zero deviations in the *model's* objective function.

3.4.2. Safety during training

The agent will be rewarded for its adherence to system constraints. When an action undergoes projection, the *projection distance*–the difference between initial and final vectors–feeds into the reward function, guiding the agent towards safer behaviour. If the *safe projection* (described in Section 3.4.1) is used, the operational projections will be negligible, and so the whole projection distance can be passed at once to the reward function. Alternatively, one might employ *simple projection*, measure the projection distance, and maintain a record of all operational projections.

The *safe projection* approach, while more straightforward, is considerably less efficient, due to its requirement for nonlinear optimisation at each timestep. To provide some reference values, *safe projection* consumes approximately $6.7\ ms$ per timestep, whereas without it, processing time reduces to $0.04\ ms$, which represents a $151x$ speed up. Since both approaches train the agent to avoid constraint violations, the fastest was chosen, as it enables more comprehensive hyperparameter optimisation within the same computational time (see Section 5.3), allowing a wider exploration of the parameter space. Figure 4.1 displays the training of the Actor DNN in some detail, indicating the use of the *simple* initial projection.

3.4.3. Safety after deployment

Once training is over, all the learned patterns and safe behaviours have been encoded in the black box that is the Actor's DNN. All the costly computations have been performed, and now a very simple forward pass over the DNN (i.e. inference) is all it takes to make a decision for the next timestep.

In the system considered for this thesis, each timestep represents 15 minutes of real time, rendering the $6.7\ ms$ inference time with *safe projection* insignificant. More importantly, it is of **utmost importance to minimise operational projections** in a real-world setting, where recovering the system from an unsafe state is not as trivial as in virtual environments. For these reasons, even though the agent may have converged to a safe policy, the ***safe projection* is always used after deployment**, as well as for any test that estimates post-deployment performance. A visualisation of the safe projection in operation can be found in Appendix A.2.6.

Model bias must be recognised as an ever-present possibility, causing actions that have been deemed safe by the *safe projection* to result in real-world constraint violations. Even when beginning with an accurate model, system changes such as battery degradation will introduce bias over time. To address these concerns, clear warning thresholds must be established for potential model bias, and post-deployment validation of the RL agent must be conducted to ensure the MCES operates within acceptable bounds. In system components where high model uncertainty is identified, projections can be adjusted to be more conservative through automated safety margin correction.

3.5. Expert EMS

The Expert EMS operates within a Model Predictive Control framework, a clear example of direct lookahead approximation. This approach is widely recognised for its efficacy in handling complex processes under constraints. The Expert is implemented as a **day-ahead planner**, featuring a 48-hour prediction horizon and 24-hour control steps. This approach differs from more common MPC applications, which usually operate at shorter intervals, such as 15-minute steps, while keeping similar prediction horizons. Although this higher-frequency execution represents a more refined control approach, the Expert's formulation retains the essential characteristics of MPC and will be categorized as such. The

SDP modelled by the Expert aligns directly with the one discussed in Section 3.3.

MPC relies on frequent replanning within a predictive horizon, leveraging the approximate linearity of non-linear systems within narrow ranges [44]. The quality of the solution depends heavily on how accurately the transition function is modelled, though simplifications are often applied to balance computational efficiency with stability and feasibility [5]. The Expert uses a *white-box model*, which is grounded in physical principles to accurately represent system dynamics [41].

Despite their high accuracy and performance, *white-box models* demand extensive **domain expertise and development costs**. Furthermore, their lack of adaptability means that changes in boundary conditions and system periphery cannot be automatically taken into account [57]. The optimisation challenge faced by the Expert is expressed as a Mixed-Integer Quadratically Constrained Program (MIQCP), incorporating a degradation-free battery model as described by Slaifstein et al.[55].

The MIQCP involves 32 variables per 15-minute timestep, including derivatives representing the states of charge of the EV battery, BESS, and TESS. The model is subject to six quadratic or non-linear equality constraints, 22 linear constraints, 29 variable bounds, and four binary variables. With a **predictive horizon spanning two days**, 192 steps of the MIQCP are linked via derivative constraints, transforming the problem into a large-scale optimisation task. For instance, the total variables considered amount to $32 \times 192 = 6144$. When the optimal solution is reached, **the first day's decisions are implemented**, and the predictive window is shifted forward accordingly. Equation 3.24 succinctly formulates the optimisation problem solved by the expert, with $\tilde{J}$ indicating that the objective function relies on approximations (as the future states are predictions).

$$X_t^{\text{exp}}(S_{a,t}) = \arg\min_{P_{a,t}^{\text{exp}}} \tilde{J}^{\text{exp}} \tag{3.24a}$$

$$\text{s.t.} \quad \text{Eqs. 3.13 - 3.23} \tag{3.24b}$$

Given the scale and complexity, solving this model requires advanced computational techniques. The problem is tackled using a heuristic *branch-and-bound* method, which evaluates combinations of binary variables to explore feasible solutions. Continuous variables are optimised for each binary configuration, with convexity ensuring convergence to an optimal solution within these configurations.

3.6. Data: Training and Testing

The information needed to train and test the reinforcement learning agents was obtained from the research by Slaifstein et al.[55], and for the sake of completeness, some of the sources they used will be mentioned in this Section. The provided data covers all the exogenous information (see Section 3.3.4) that affects directly the MCES, including electrical load ($P^{\text{e}}_{\text{load}}$), thermal load ($P^{\text{th}}_{\text{load}}$), power generated by photovoltaic panels (P_{PV}), energy prices (λ_{sell}, λ_{buy}), electric vehicle presence (γ_{EV}), and power used by the electric vehicle when not connected to the MCES ($P^{\text{drive}}_{\text{EV}}$).

- The electrical load was obtained directly from the residential building in the Green Village (of Delft University of Technology) from the year 2021 to 2023. This is the building that is modelled by the MCES.
- The thermal load data was provided by Joel Alpízar-Castillo, who used it to analyse the thermal and electrical performance in Dutch homes of four distinct configurations of PV-thermal, BESS, TESS and heat pump technologies [3].
- The solar energy profiles for the Green Village were constructed using models from the book on Solar Energy by Smets et al. [56].
- The electricity prices for the Netherlands were obtained from the European Power Exchange (EPEX) [16], in particular for the year 2022.

- The ElaadNL Research Centre [14] provides some open source statistics about the arrival times of EV users, as well as the expected duration of their stay. This information was used to construct the models that generate samples for γ_{EV} and $P_{\text{EV}}^{\text{drive}}$.

The data covers one year and was formatted into 15 minute timesteps. To obtain the **test set** (91 days), one day was extracted from every four, leaving behind what will be called the **training set** (274 days). Therefore, the training and test sets are **completely separated**.

3.6.1. Expanding the Data

By making use of the **training set**, which encompasses 274 days, three synthetic datasets have been generated. First will be described the creation of the *expanded training set*, followed by the two validation sets.

It was decided to extend the original training set's duration of 274 days to a full year of training data for the RL agent, therefore 91 days must be synthesized. The methodology is straightforward, and has been thoroughly outlined in Algorithm 1. The main benefit of expanding the training set in such a manner is the cost-effective increase in data diversity. In addition, this expansion allows rare scenarios to become more prevalent, potentially improving the agent's ability to generalise. However, there is the risk of creating data that moves beyond real-world situations, hampering the agent's deployment. To mitigate this possibility, the *test set* is made exclusively of real data.

Algorithm 1 Synthetic Data Generation for Expanded Training Set

Require: Training set of $D = 274$ days, each day containing $T = 96$ timesteps of 15 min, where $X_{d,t}$ is the value at day d and timestep t

1: Initialize $S = 91$ (number of synthetic days to create)
2: **for** $d = 1$ to $D - 4$ with step 3 **do**
3: **for** each timestep $t = 1, \ldots, T$ **do**
4: Calculate mean at timestep t across the 3 days:

$$\mu_{3,t} = \frac{1}{3} \sum_{i=0}^{2} X_{d+i,t}$$

5: Sample synthetic value for timestep t from normal distribution:

$$X_{\text{syn},t} \sim \mathcal{N}\left(\mu_{3,t}, \frac{\mu_{3,t}}{6}\right)$$

6: Restrict $X_{\text{syn},t}$ within the minimum and maximum values of the original dataset:

$$X_{\text{syn},t} \leftarrow \max(X_{\min}, \min(X_{\text{syn},t}, X_{\max}))$$

7: **end for**
8: Store X_{syn} as a new synthetic day
9: **end for**
10: **return** Full training set with $D + S = 365$ days.

Note: $\mathcal{N}(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ.

The *validation set* will be fundamental for optimising the model's hyperparameters. Its creation follows a similar procedure to the *expanded training set*, as detailed in Algorithm 2 (found in Appendix

A.3). The primary modifications include doubling the standard deviation and generating only a 91-day set to match the *test set* duration. As expected, the seed for the random number generator (RNG) used to sample the synthetic values is different from the *expanded training set*.

Synthetic data generation algorithms were not applied to the electric vehicle parameters (γ_{EV} and $P_{\text{EV}}^{\text{drive}}$), because they are produced through sampling from distributions that model their real-world behaviour.

To enhance the generalisation capabilities of the RL agent, the *robust validation set* has been developed. While its creation mirrors the *validation set*, it presents more challenging scenarios for the agent and utilises a unique RNG seed value. For instance, there is considerably less solar radiation and a higher electrical load, as well as a higher power demand of the electric vehicle when outside the MCES. The alterations upon the *validation set* creation methodology are simple and thus not merit a detailed explanation. Here is a list of the main changes:

- Increased the average value of: λ_{buy}, $P_{\text{load}}^{\text{e}}$, $P_{\text{load}}^{\text{th}}$, and $P_{\text{EV}}^{\text{drive}}$.
- Reduced the average value of: λ_{sell}, P_{PV}.
- Increased standard deviation of: λ_{buy}, λ_{sell},$P_{\text{load}}^{\text{e}}$, $P_{\text{load}}^{\text{th}}$, P_{PV}.
- Reduced standard deviation of: $P_{\text{drive, EV}}$

4

RL Agent Construction

4.1. Introduction

This chapter introduces the building blocks of the RL agent designed to manage the MCES. Some of these components, such as rewards, DNN architectures, and feature vector configurations, **are treated in this thesis as hyperparameters**, each with their own range of possible values, that will be systematically optimised in Chapter 5. The reward formulation combines independently designed elements that address the demands of the SDP: minimising grid exchange costs, EV SoC requirements, safety considerations, and boundary margins. These elements are combined to create distinct versions of the final reward.

Various feature vector configurations are developed to expand the space of achievable policies during hyperparameter optimisation. Neural network design is examined through several lenses, with architecture emerging as a crucial hyperparameter, complemented by comments on the dimensionality of the network, output processing, and weight initialisation strategies.

Three policy gradient algorithms are presented in order of increasing complexity: Vanilla Policy Gradient with Critic (VPG-C), Advantage Actor-Critic with generalized advantage estimation (A2CGAE), and PPO, with a clear documentation of implementation details that, while often overlooked in the literature, are crucial for reproducibility. Finally, the data collection technique utilised during the agent's training in the simulated environment is described.

4.2. Reward Functions

The objective function presented in Section 3.3.1 establishes clear minimisation goals, and the performance metric used to discern clearly between good and bad policies is based upon it (Section 5.2. Following all the conclusions presented in Section 2.7.1, any reward function that is created to train the agent must ultimately encode this same objective. In addition, various rewards can be formulated to achieve the same outcome, to later allow hyperparameter optimisation to reveal which one is better at maximising performance.

It is worth noting, for clarity, that all reward functions will present an opposite sign to the costs or penalties they are trying to reduce (as mentioned in Section 2.7.1). The terms *penalty* and *cost* are utilised interchangeably, serving as antonyms to *reward*. Rewards may be negative, and would therefore relate to actions leading to higher costs. Figure 4.1 provides a simplified visual representation of the components that make up the final reward function, which will be detailed below.

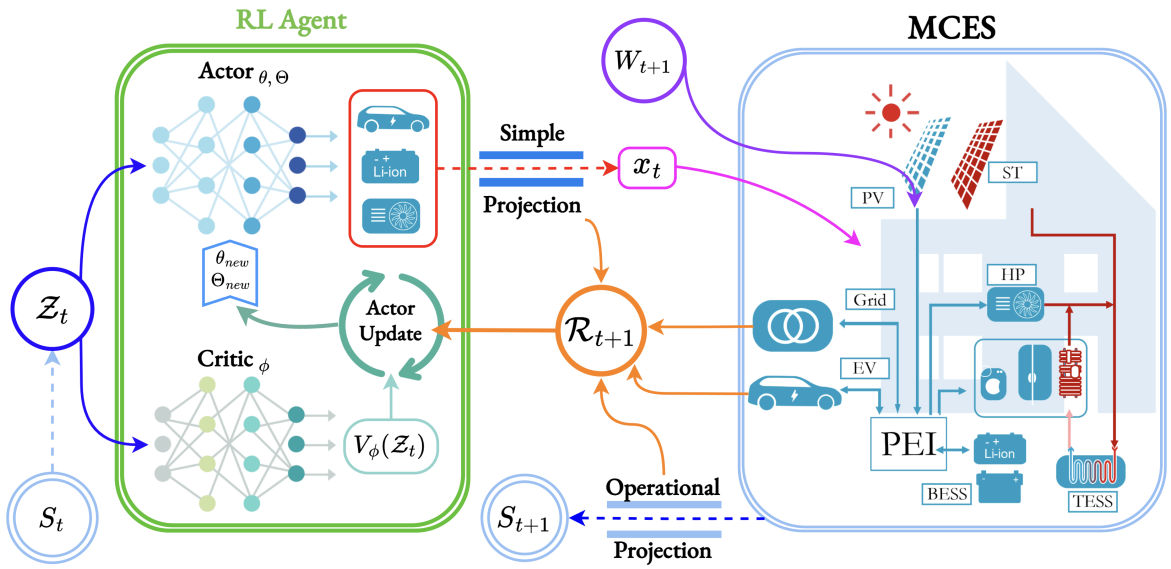

Figure 4.1: The diagram illustrates how the RL agent interacts during training with the MCES, which encapsulates the transition function. The notation aligns with that used in Section 3.3 to formulate the sequential decision problem. Other relevant symbols include $\mathcal{R}$, denoting the reward function (refer to Section 4.2), and $\mathcal{Z}$, which represents the feature vector (refer to Section 4.3). The figure consolidates some of the project's core components, framing their relationships. Dashed lines indicate that data transformations occur between the connected nodes.

4.2.1. Grid Cost

The initial component of the final reward function attempts to reduce the grid cost term C_{grid} within the objective function. The first implementation computed the reward signal at each timestep t as the negative value of energy exchange costs with the grid.

$$r_{\text{grid},t} = -\left(\lambda_{\text{buy,t}} \cdot P^{+}_{\text{grid},t} + \lambda_{\text{sell,t}} \cdot P^{-}_{\text{grid},t}\right) \cdot \Delta t \tag{4.1}$$

This methodology proved unsuccessful in practice. A potential explanation may be found in the long-term nature of grid costs, which accumulate over daily or multi-day periods. Optimal results might require more extended time horizons, since present sacrifices may be needed for superior long-term outcomes. It is likely that the reward discounting mechanism was insufficient to indicate to the updating algorithm which timesteps contained the decisions that led to minimised overall grid costs.

The next approach was designed to encode a long-term objective into the agent by computing the average grid cost over multiple episodes. For any episode e, the reward is non-zero **only at the final timestep**:

$$r^{\Sigma}_{\text{grid},e} = -\frac{1}{x} \sum_{i=e-x+1}^{e} \sum_{t=1}^{T} \left(\lambda^{i}_{\text{buy},t} \cdot P^{i,+}_{\text{grid},t} + \lambda^{i}_{\text{sell},t} \cdot P^{i,-}_{\text{grid},t}\right) \cdot \Delta t \tag{4.2}$$

Where $x = 2$ was chosen to **match the expert's optimisation horizon**. This approach provided better results in the preliminary tests, as it allowed the agent to learn from the cumulative impact of its decisions. The sparse reward proved effective through the discounting, advantage estimation, and normalisation processes applied before the weight updates of the DNN, as explained in Section 2.7.1. The subscript e is used for clarity, to indicate that it is computed only once per episode.

In an effort to create a denser reward function while maintaining long-term perspective, an arbitrage-

based approach was developed. This reward signal is computed as:

$$r^{\text{arb}}_{\text{grid},t} = s_t \cdot \begin{cases} (\bar{\lambda}_{\text{buy}} - \lambda_{\text{buy},t})/\bar{\lambda}_{\text{buy}} & \text{if } P_{\text{grid},t} > 0 \\ (\lambda_{\text{sell},t} - \bar{\lambda}_{\text{sell}})/\bar{\lambda}_{\text{sell}} & \text{if } P_{\text{grid},t} \leq 0 \end{cases} \tag{4.3}$$

Where:

$$s_t = \frac{|P_{\text{grid},t}|}{\max(|\bar{P_{\text{grid}}}|, \epsilon)} : \text{Scaling factor based on relative power magnitude}$$

$\bar{\lambda}_{\text{buy}}$: Mean buying price over a 2-day window

$\bar{\lambda}_{\text{sell}}$: Mean selling price over a 2-day window

$|\bar{P_{\text{grid}}}|$: Mean absolute grid power over a 2-day window

ϵ : Small constant to prevent division by zero (10^{-1})

This reward function provides immediate feedback, encouraging the agent to buy power when prices are below average and sell when they are above. The reward is scaled based on the relative magnitude of the power exchange. Following the same reasoning as for the function $r^{\Sigma}_{\text{grid},e}$, the window is defined to be 2 days.

The arbitrage-based reward (Equation 4.3) would fall under the category of **reward shaping** (as defined in Section 2.7.1). The reward signal will encourage the agent to maximise the price differences between the trading operations and their corresponding 48-hour averages. Although this approach potentially accelerates policy convergence, it may introduce theoretical constraints on reaching global optimality. The ultimate proof of its usefulness will be found in the results of the hyperparameter optimisation.

4.2.2. EV Penalty at Departure

The second reward component addresses the SoC requirements for the electric vehicle at departure times. Two distinct reward formulations were developed. The initial approach utilises:

$$r^{\text{abs}}_{\text{EV},t} = -|\xi_{\text{SoCDep},t}| = -|\text{SoC}_{\text{EV},t} - \text{SoC}^{*}_{\text{dep}}| \tag{4.4}$$

The value of all EV-focused rewards become non-zero exclusively at departure instances $t \in \mathcal{T}_{\text{dep}}$. While the objective function utilises a quadratic penalty term $(\xi_{\text{SoCDep},t})^2$, the initial implementation makes use of an absolute value function. This choice was motivated by the permissive nature of the quadratic function in the lower range of $\xi_{\text{SoCDep},t}$, since there seems to be a low incentive (i.e. a low gradient) to exactly match the desired $\text{SoC}^{*}_{\text{dep}}$.

To follow this line of reasoning, a sigmoid-based reward function was developed to heavily penalise high values of $\xi_{\text{SoCDep},t}$ while providing a steep gradient in the intermediate range. A variety of steepness values and inflection points were considered, trying to balance a steep gradient with too severe penalties for the agent (which might prevent proper learning). The final coefficients reached are displayed below:

$$r^{\sigma}_{\text{EV},t} = \frac{v(|\xi_{\text{SoCDep},t}|) - v(x_{\text{max}})}{v(0) - v(x_{\text{max}})} - 1 \tag{4.5}$$

where $v(x)$ is the sigmoid function:

$$v(x) = \frac{1}{1 + e^{k(x - x_{\text{mid}})}} \tag{4.6}$$

with parameters:

$k = 15$: Steepness of the sigmoid curve

$x_{\text{mid}} = 0.1$: Inflection point.

$x_{\text{max}} = \text{SoC}^{*}_{\text{dep}} - \text{SoC}^{\text{min}}_{\text{EV}}$: Maximum considered $|\xi_{\text{SoCDep},t}|$

A comparative visualisation is presented in Figure 4.2, showing the characteristics and providing insight into the possible impact that each EV-focused reward could present. Their values have been normalised for a fair comparison, as the relative changes across $|\xi_{\text{SoCDep},t}|$ carry more significance than absolute values (which may be modified with specific reward weights). It is worth noting that the performance metric (Section 5.2) uses the same sigmoid function to evaluate the capacity of the agent to satisfy the EV charging requirements (Equation 5.2). The sigmoid is used for the same reason in both cases, to clearly discern between good and bad performances. This similarity could potentially benefit the agents trained with $r^{\sigma}_{\text{EV},t}$ as a reward, nevertheless, the sigmoid might prove too severe as a guide to the optimal policy, preventing convergence.

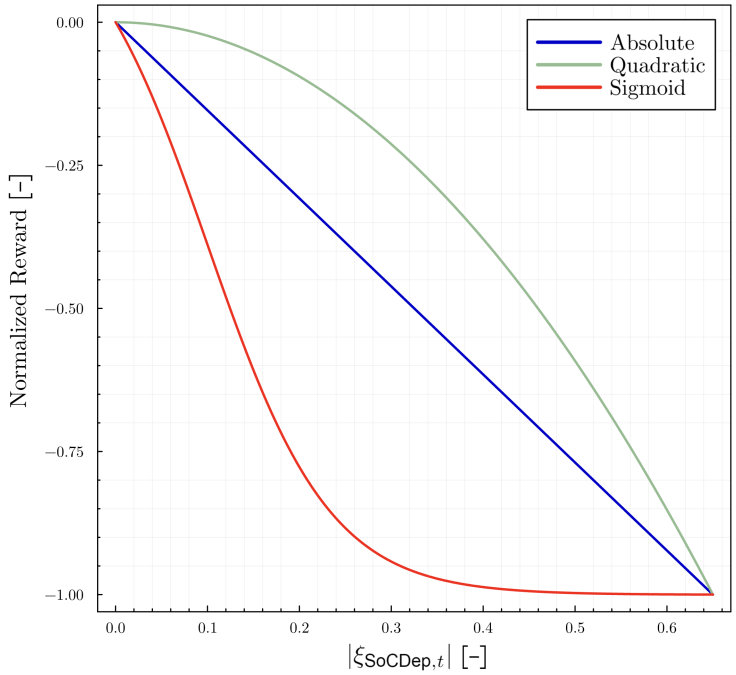

Figure 4.2: Comparison of normalised EV rewards. It becomes clear that the sigmoid is relatively the most severe, focusing on pushing the agent's behaviour to the minimum possible $|\xi_{\text{SoCDep},t}|$. $x_{\text{max}} = 0.65$.

4.2.3. Projection Penalty

The objective function incorporates a penalty term that quantifies necessary projections onto the feasible action space. As elaborated in Section 3.3.1, the projection term of RL will differ from classical control approaches, where physical constraints are explicitly embedded in the mathematical formulation of the program.

As was detailed in Section 3.4.1, two types of projection mechanisms safeguard the MCES environment: initial projections verify the action proposed by the agent is within bounds before implementation, while operational projections adjust state variables post-transition to maintain system constraints.

In Section 3.4.2, the rationale for not utilising *safe projection* during training was presented. As a result, both initial and operational projections will take place during training. and by monitoring their *projection distances*, a reward function may be constructed. The variables mentioned in the following functions can all be found in Section 3.3.1.

The reward function addressing projections is separated into two components, one focusing on initial projections and another on operational projections. In practical terms, this separation simply indicates that distinct weights are applied when all rewards are combined at each timestep (see Section 4.2.5). The mathematical formulation of the first reward function is:

$$r^{\text{init}}_{\text{proj,t}} = -\sqrt{(\xi^{\text{p}}_{\text{EV,t}})^2 + (\xi^{\text{p}}_{\text{BESS,t}})^2 + (\xi^{\text{p}}_{\text{HP,t}})^2} \tag{4.7}$$

Here, the Euclidean norm of the initial *projection distance* provides a measure of how far the agent's proposed actions deviate from the feasible space. The operational projection reward is formulated as:

$$r^{\text{op}}_{\text{proj,t}} = -\frac{1}{5}\sum_{i=1}^{5} \beta_i \cdot \xi_{i,t} \tag{4.8}$$

$$\{\beta_i\} = \{\beta_{\text{BESS}}, \beta_{\text{EV}}, \beta_{\text{TESS}}, \beta_{\text{grid}}, \beta^{\text{p}}_{\text{TESS}}\},$$
$$\{\xi_i\} = \{\xi^{\text{SoC}}_{\text{BESS}}, \xi^{\text{SoC}}_{\text{EV}}, \xi^{\text{SoC}}_{\text{TESS}} \xi^{\text{p}}_{\text{grid}}, \xi^{\text{p}}_{\text{TESS}}\}$$

β_i represents the normalising weight for each projection term, and the negative sign ensures that the minimisation of constraints maximises the reward. The final value is computed as the mean of all terms.

4.2.4. Margin Reward

With the aim of discouraging the agent to approach out-of-bounds states, a new reward was conceived that penalises decisions that move state variables towards their limits. As defined in Section 2.7.1, this is a clear case of reward shaping, since the objective function is not influenced by the proximity of variables to their boundaries. However, it was considered useful to include –as a part of the hyperparameter optimisation– a *margin reward* to **optionally complement** the operational projection reward described in Equation 4.8. Here is the mathematical formulation:

$$r_{\text{margin}} = r^{\text{BESS}}_{\text{margin}} + r^{\text{TESS}}_{\text{margin}} + r^{\text{EV}}_{\text{margin}} + r^{\text{grid}}_{\text{margin}} \tag{4.9}$$

Each term represents the margin reward for a specific MCES system component, with the fundamental building block for these penalties defined by the function r_{border}:

$$r_{\text{border}}(x, x_{\text{safe}}, x_{\text{limit}}, \alpha) = \begin{cases} \alpha \cdot \dfrac{1 - e^{4\cdot\hat{x}}}{e^4 - 1} & \text{if } 0 < d \cdot \hat{x} < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

The normalised state $\hat{x}$ is given by:

$$\hat{x} = \frac{x - x_{\text{safe}}}{|x_{\text{limit}} - x_{\text{safe}}|} \tag{4.11}$$

The direction d indicates is defined as:

$$d = sign(x_{\text{limit}} - x_{\text{safe}}) \tag{4.12}$$

and:

x : State variable being evaluated
x_{limit} : Undesirable boundary or constraint threshold
x_{safe} : Desirable, safe threshold
α : Maximum penalty coefficient

The directional parameter d is assigned $+1$ when x_{limit} constitutes an upper boundary, that is, x would need to increase from the *safe threshold* to reach the *undesirable threshold*. The parameter assumes -1 for lower boundaries. This configuration enables the function r_{border} to be used for the upper and lower bounds.

The function $r_{\text{border}}(x)$ computes a smooth exponential reward as the state variable x approaches an undesirable boundary x_{limit} from a desirable state x_{safe}. This reward diminishes gradually as x nears the unsafe threshold. Notably, when x resides outside the interval $[x_{\text{safe}}, x_{\text{limit}}]$, no penalty is incurred.

The component-specific margin rewards are defined as follows:

$$\begin{aligned} r^{\text{BESS}}_{\text{margin}} = r_{\text{border}}(&\text{SoC}_{\text{BESS}},\ 1.25 \cdot \text{SoC}^{\text{min}}_{\text{BESS}},\ 0.95 \cdot \text{SoC}^{\text{min}}_{\text{BESS}},\ 0.35) + \\ r_{\text{border}}(&\text{SoC}_{\text{BESS}},\ 0.95 \cdot \text{SoC}^{\text{max}}_{\text{BESS}},\ 1.05 \cdot \text{SoC}^{\text{max}}_{\text{BESS}},\ 0.35) \end{aligned} \tag{4.13}$$

$$\begin{aligned} r^{\text{TESS}}_{\text{margin}} = r_{\text{border}}(&\text{SoC}_{\text{TESS}},\ 1.25 \cdot \text{SoC}^{\text{min}}_{\text{TESS}},\ 0.95 \cdot \text{SoC}^{\text{min}}_{\text{TESS}},\ 0.25) + \\ r_{\text{border}}(&\text{SoC}_{\text{TESS}},\ 0.95 \cdot \text{SoC}^{\text{max}}_{\text{TESS}},\ 1.05 \cdot \text{SoC}^{\text{max}}_{\text{TESS}},\ 0.25) + \\ r_{\text{border}}(&|P_{\text{TESS}}|,\ 0.90 \cdot P^{\text{max}}_{\text{TESS}},\ 1.05 \cdot P^{\text{max}}_{\text{TESS}},\ 0.25) \end{aligned} \tag{4.14}$$

$$\begin{aligned} r^{\text{EV}}_{\text{margin}} = \gamma_{\text{EV}} \cdot [&r_{\text{border}}(\text{SoC}_{\text{EV}},\ 1.03 \cdot \text{SoC}_{\text{dep}},\ 0.95 \cdot \text{SoC}^{\text{min}}_{\text{EV}},\ 1.0) + \\ &r_{\text{border}}(\text{SoC}_{\text{EV}},\ 0.97 \cdot \text{SoC}^{\text{max}}_{\text{EV}},\ 1.05 \cdot \text{SoC}^{\text{max}}_{\text{EV}},\ 1.0)] \end{aligned} \tag{4.15}$$

$$r^{\text{grid}}_{\text{margin}} = r_{\text{border}}(|P_{\text{grid}}|,\ 0.90 \cdot P^{\text{max}}_{\text{grid}},\ 1.05 \cdot P^{\text{max}}_{\text{grid}},\ 1.25) \tag{4.16}$$

The maximum penalty coefficients (α) are calibrated to balance respect of the boundary with the danger of over constraining (and thus underutilising) the asset. For example, P_{grid} presents a high margin penalty because it disposes of a considerable range of operation, so margin rewards will be infrequent. However, approaching the boundary is highly discouraged as P_{grid} is dependent on exogenous information and therefore its safety is not guaranteed (see Section 3.4.1 for a deeper explanation).

The EV margin reward is only applied when the vehicle is connected to the MCES, since there is nothing the agent can do otherwise, this is implemented via the γ_{EV} term. As can be seen, the margin is considering $x_{\text{safe}} = 1.03 \cdot \text{SoC}_{\text{dep}}$, that is, the agent will be trained not only to keep the EV battery within limits, but to keep it near SoC_{dep}. This signal works alongside $r^{\sigma}_{\text{EV},t}$ or $r^{\text{abs}}_{\text{EV},t}$ to provide denser feedback.

The BESS and TESS components' SoC penalties were kept low to prevent underutilisation. Figure 4.3 displays the BESS margin reward, demonstrating how the signal extends beyond $\text{SoC}^{\text{max}}_{\text{BESS}}$ and $\text{SoC}^{\text{min}}_{\text{BESS}}$. While evident from Equation 4.13, this design choice deserves an explanation: it offers a smoother gradient for the RL optimisation to follow, and maintains reward consistency across the boundary regions. Furthermore, these margin rewards contribute to the operational projection rewards (see Equation 4.8), which are not *shaped*.

4.2.5. Final Reward

A unified reward is essential for the policy update mechanism (Section 2.7). Therefore, the final reward must be a combination of all the reward functions previously described in Section 4.2. Since various functions have been developed to minimise the same term of the objective function (Section 3.3.1), this will result in diverse variations of the final reward based on different combinations. Here is the mathematical formulation:

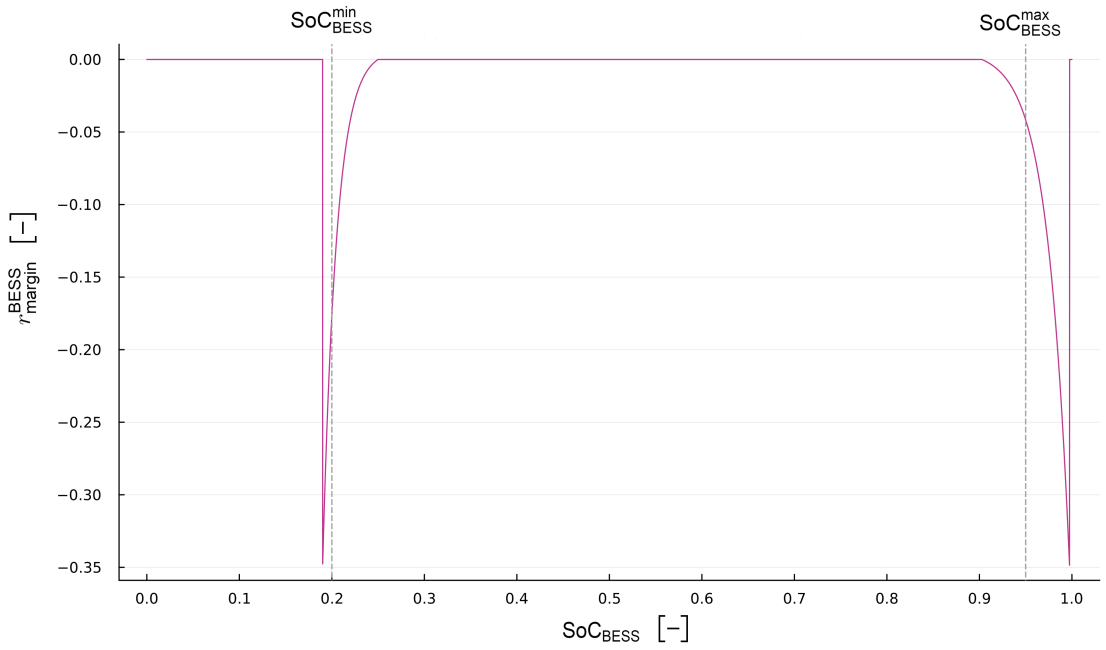

Figure 4.3: Margin reward for the BESS component of the MCES. Visualisation of Equation 4.13.

$$
\begin{aligned}
r_{\text{final},t} = b\Big(& w_{\text{grid}} \cdot (r^{\Sigma}_{\text{grid},e} \text{ or } r^{\text{arb}}_{\text{grid},t}) \\
& + w_{\text{EV}} \cdot (r^{\text{abs}}_{\text{EV},t} \text{ or } r^{\sigma}_{\text{EV},t}) \\
& + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} \\
& + w^{\text{op}}_{\text{proj}} \cdot (r^{\text{op}}_{\text{proj},t} + (r_{\text{margin}} \text{ or } 0)) \Big)
\end{aligned}
$$

where :

$$
b(x) = \begin{cases} -50, & \text{if } x < -50 \\ x, & \text{if } -50 \leq x \leq 50 \\ 50, & \text{if } x > 50 \end{cases} \tag{4.17}
$$

Eight distinct final rewards ($r_{\text{final},t}$) have been formulated, arising from the combination of two options for lowering grid costs, two EV penalty approaches, and the optional inclusion of margin rewards (see Appendix B.2.2 for full list of combinations). Though additional reward functions were conceived during development, computational limitations and the extensive hyperparameter space (including DNN architectures) made testing of 16 or 32 final reward versions impractical. Hence, all rewards functions were carefully examined and those that offered the greatest paradigm shift were retained.

From the eight final rewards, two will present **no reward shaping**, as it was defined in Section 2.7.1:

$$
r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\text{abs}}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t} \Big) \tag{4.18}
$$

$$
r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\sigma}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t} \Big) \tag{4.19}
$$

Boundaries have been established for all final rewards to avoid extreme values from destabilising the updating procedure, encouraging thus more consistent policy updates. Weights have been introduced to adjust the relative importance of each reward, with various value sets being tested during

hyperparameter tuning (Appendix B.2.3). These weights are different from those utilised in the expert's objective function (Section 3.3.1). Additional studies investigating the impact of steepness in Equation 4.5, maximum penalties in margin rewards, or nonlinear versions of Equation 4.8 could prove valuable, if the computational resources allow to further expand the hyperparameter search space.

4.3. Feature vector

The inputs to the DNNs that constitute the agent are referred to as **features**. It is likely that these features will have some overlap with the state variable described in the problem formulation (Section 3.3.2), but this is not a requirement. As long as the agent's decision is nearly optimal, the features may be any information considered useful to arrive at that decision. Therefore, selecting the feature vector introduces significant uncertainty for researchers, and trial and error is a common approach to the problem. As part of Sub-research Question 2 2.11, this section will explore the creation of feature vector configurations with access to longer temporal patterns, i.e., state information further in the past.

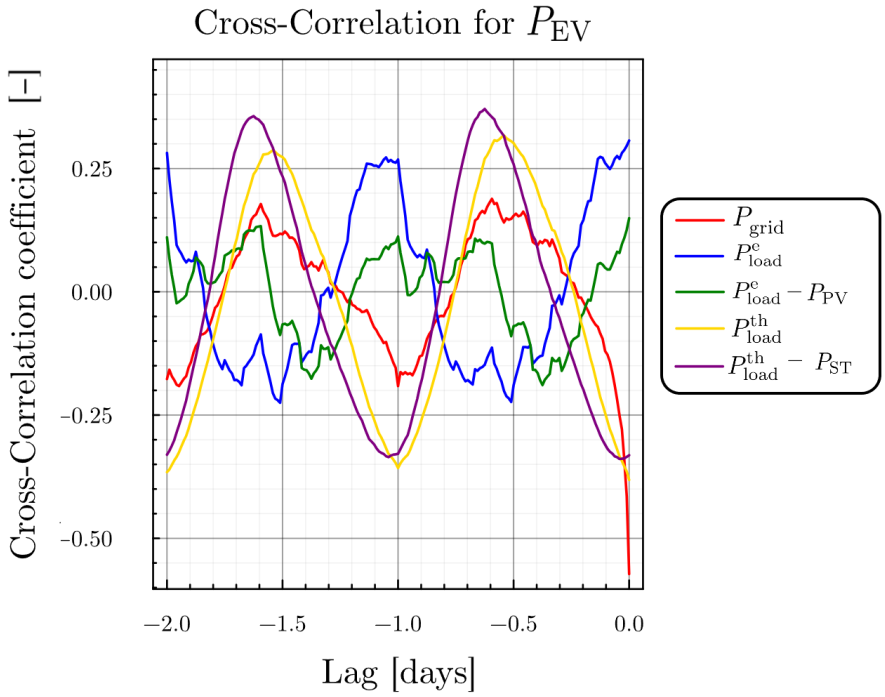

Figure 4.4: Cross-correlation coefficients for the P_{EV} with the variables shown in the legend.

It seems reasonable to provide the agent with the variables that human experts would use to solve the problem in a classical manner, or by providing features that are directly related to the objective function. These decisions may succeed, but it is not guaranteed. Providing only one version of the feature vector to the DNNs will constrain the space of policies that can be achieved. Although this may not be the primary limiting factor preventing the agent from learning desirable behaviour, the impact of a particular feature vector is a priori unknown. **To reduce the dependence on trial an error**, and attempting to enhance the basic optimisation results (Section 5.3.2), the approach that was followed **expanding the feature vector configurations available to the hyperparameter optimisation algorithm.** Computational power will be used in Section 5.3 to evaluate hundreds of samples and observe which feature vectors are most effective in allowing the DNNs to abstract useful patterns from the data.

From all the state variable components, 14 are decided to be of possible usefulness to the agent, which are displayed in Table 4.1. For each of these variables, different time lags are available. Given that there are 96 timesteps in a day (with 15-minute intervals), the number of possible combinations is remarkably high, particularly if more than one day of delay is made available. In an attempt to identify

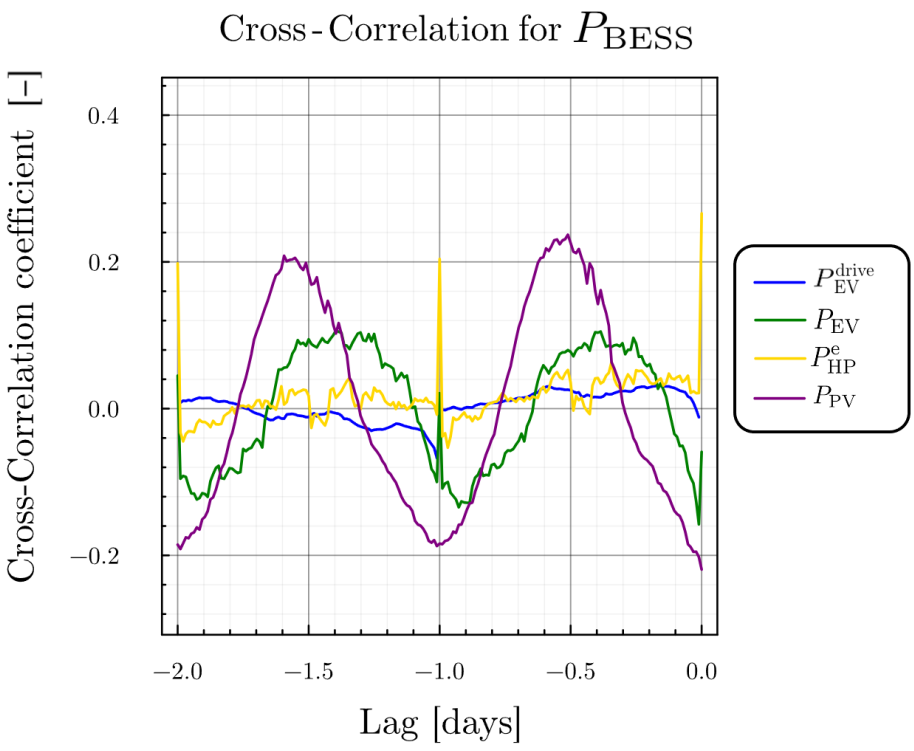

Figure 4.5: Cross-correlation coefficients for the P_{BESS} with the variables shown in the legend.

the most useful time delays, it was decided to conduct a **correlation study on the Expert** (Section 3.5. This study involved testing the cross-correlation of all 14 variables with the three decisions made by the expert at each timestep, as well as the auto-correlations of the decisions themselves.

Figures 4.4 and 4.5 are examples of the results obtained from the analysis. It can be noticed that certain variables exhibit significantly higher correlations than others, with a distinct daily pattern emerging. The variable $P_{\mathrm{EV}}^{\mathrm{drive}}$, which represents the power demanded by the electric vehicle when outside the EMS, shows almost no correlation, which is not surprising since the agent's decisions when $P_{\mathrm{EV}}^{\mathrm{drive}}$ is in effect cannot alter the behaviour of the electric vehicle. The correlation patterns demonstrate periodic recurrence at approximately 24-hour intervals, even when examining lags exceeding two days. In order to simplify the feature vector, a maximum lag of one day has been deemed sufficient for the agent. The combinations of variables $P_{\mathrm{load}}^{\mathrm{th}} - P_{\mathrm{ST}}$ and $P_{\mathrm{load}}^{\mathrm{e}} - P_{\mathrm{PV}}$ have shown very similar correlations to $P_{\mathrm{load}}^{\mathrm{th}}$ and $P_{\mathrm{load}}^{\mathrm{e}}$, respectively, and will thus be omitted. The variable λ_{sell} shows the same cross-correlations as λ_{buy} because it was synthesised from it, by adding noise and a constant down-scaling.

Table 4.1 shows the time lags exhibiting the highest cross-correlation coefficients. The variable t_{ep} represents the timestep within the episode, normalised to the range $0-1$. All variables will be passed to the DNNs after the application of *z-score normalisation*, as recommended by Andrychowicz et al.[4]. To avoid confusion with the state variable (S_t), the feature vector will be denoted as Z_t.

Configuration 2, as shown in Table B.1, represents a feature vector that incorporates all 14 variables at the final timestep. This simple feature vector is used for the basic hyperparameter optimisation described in Section 5.3.2. A reduced form of this vector, where the agent's final decisions and P_{grid} are omitted, is designated as Configuration 1.

Since the feature vector will be treated as an additional hyperparameter –in the Extended optimisation–, a diverse array of feature vectors has been constructed, displayed in Appendix B.2.1, primarily by utilising the time lags derived from the analysis. Some options omit the previous decisions to mitigate the risk of trapping the agent in a feedback loop, where the past influences the present too strongly. Other options simply reduce the amount of time lags offered. To assess the relative effectiveness of

Variable	Time Lags (days)
P^{e}_{load}	0.0, 0.5, 1.0
P^{th}_{load}	0.0, 0.5, 1.0
P_{PV}	0.0, 0.5, 0.65, 1.0
λ_{buy}	0.0, 0.1, 0.45, 0.65, 1.0
λ_{sell}	0.0, 0.1, 0.45, 0.65, 1.0
γ_{EV}	0.0, 1.0
P_{grid}	0.0, 0.5, 1.0
P_{BESS}	0.0, 1.0
P_{EV}	0.0, 0.5, 1.0
P^{e}_{HP}	0.0, 0.5, 1.0
SoC_{BESS}	0.0, 0.15, 0.4, 0.8, 1.0
SoC_{EV}	0.0, 0.25, 0.75, 1.0
SoC_{TESS}	0.0, 0.25, 0.5, 0.75
t_{ep}	0.0, 0.3, 0.9

Table 4.1: Most relevant variables for the Expert and the time lags with highest cross-correlation coefficients. Corresponds to the configuration 3 as described in Appendix B.2.1.

using high cross-correlation time lags, a feature vector with randomised time lags has been included in the set, described as Configuration 6 in Table B.2. Additionally, a feature vector with *periodic* time lags, representing the last timestep, hourly patterns, 6-hour patterns, and a complete day of delay, is also considered and described as Configuration 7 in Table B.2.

It is worth noting that the higher the number of elements in the feature vector, the wider the first layer of the DNN will be, and so the comparison between the set of possible feature vectors is not strictly fair. Nevertheless, the impact is minor and conducting a fair comparison is not the primary aim of this thesis; finding the best set of hyperparameters is considerably more relevant towards the goal.

4.4. Neural Network Design

4.4.1. Dimensions of the Deep Neural Network

It can be argued that the representational power of DNNs stems from their **depth**, from their ability to nest a series of linear and nonlinear transformations. Each layer increases exponentially the complexity of the representable functions [42]. All these layers are trained together, allowing highly abstract concepts to be learned, usually in a hierarchical fashion, where each layer perceives properties of the data that grow in complexity with the forward pass of the input. When testing the optimal DNN depths on RL benchmark environments, such as those provided by OpenAI Gym [10], two hidden layers offered the best performance for Actor and Critic networks [4], and so this becomes an appropriate reference from which to start testing architectures.

The **width** of the network, i.e. the number of neurons on each layer, will increase the information that can be preserved during the forward pass [35]. Wider networks can handle multiple features and functions in parallel. In more practical terms, Andrychowicz and colleagues [4] discovered that the Actor's DNN width is much more sensitive to each environment than the Critic's. They propose that the optimal width of the policy – the Actor's DNN – depends on the complexity of the problem faced

by the agent, and thus should be tuned for each case. In regards to the value function approximation done by the Critic, they find no downside to wider networks, in some cases even making them wider than the policy. However, narrower networks sometimes outperform wider ones, while also requiring significantly fewer computational resources to train.

4.4.2. Policy Output

The policy, represented by the Actor, can be thought to reproduce a Gaussian distribution density function, and in this case it is implemented using two distinct DNNs. The primary network will be destined for the inference of the **mean** of the action distribution, while the secondary network generates the **standard deviations**. The primary network operates via a conventional feed-forward mechanism: it ingests the feature vector (detailed in Section 4.3) and processes it through multiple layers, performing matrix multiplications and applying nonlinear activation functions.

The secondary network employs a less intuitive approach. Research by Andrychowicz et al. [4] suggests that there is no significant performance difference between a standard deviation that is dependent on the feature vector of each timestep and one that is independently obtained. In the interest of implementation simplicity and computational efficiency, a thin network is utilised to produce a global standard deviation. Although the parameters of this network are updated along with the Actor, the standard deviation obtained after training will be **constant for all states**.

As per the recommendation of Andrychowicz et al. [4], the initial standard deviation will be a **tuneable hyperparameter** (see Section 5.3.1), and the output of the standard deviation network will be passed through the *softplus* function (see Equation 4.20). A transformation of the output is necessary since the standard deviation must be above zero. Usually, exponentiation is used, but in their research it is found to perform slightly worse. To guarantee a minimum value for the standard deviation, 0.01 will be added to the output of the softplus function. 0.01 showed better performance than higher values in the research of Andrychowicz et al. [4], although the authors affirm that the exact value is not relevant, unless it is too high.

$$softplus(x) = \log(e^x + 1) \tag{4.20}$$

4.4.3. Weight Initialisation

Andrychowicz et al. [4] most surprising finding was the relevance of the initialization scheme on the final performance, particularly because it is rarely mentioned in RL literature. They suggest that to improve performance, the action distribution at the start of training should have zero mean and be as independent as possible from the observations. To achieve this, the last layer should be initialised with considerably smaller weights (e.g. 100x reduction). Additionally, the action standard deviation should start low, and be tuned for optimal performance. Altering the last layer's weights is much less relevant for the Critic DNN.

It is important to note that the environments used by Andrychowicz et al. [4] experiments expected actions in the range -1 to 1. By initialising the network in this manner, they avoided biasing the agent before training commenced. This scenario is comparable to the environment used in this thesis, where the range of decisions for the power delivered by BESS (P_{BESS}) or the EV (P_{EV}) has zero mean, whereas the power range delivered by the heat pump (P^{e}_{HP}) is positive. Since using the heat pump will always increase costs, initialising its value close to zero does not introduce problematic bias.

The researchers also found that the specific initialisation algorithm was less significant. They tested various schemes including *Glorot normal, Glorot uniform, He normal, He uniform, LeCun normal, LeCun uniform, Orthogonal, Orthogonal(gain=1.41)*. Their results indicated that only the "He" algorithms performed notably worse. Given all the other algorithms as choices with equal performance, it is decided to select the algorithm following the recommendations of Philipp et al. [42] to avoid exploding gradients. They advocate initialising the network as a sequence of **orthogonal** transformations. This approach should considerably reduce the growth of gradients, as orthogonal matrices keep the norm of input vectors.

4.4.4. Architecture

As an initial approach to DNN architecture, it has been proposed by Andrychowicz et al. [4] that a simple Multi-Layer Perceptron (MLP) be utilised for the Actor, and another for the Critic. Each will have 2 hidden layers of uniform width. The input layer is designed to accommodate the dimensions of the feature vector (Z_t), while the output layer corresponds to those of the decision vector (x_t). This configuration will be henceforth referred to as Constant Width (CW). As described in Section 2.8, nonlinearity will be introduced through the implementation of the Tanh function. The weight initialisation will be performed, following the reasoning of Section 4.4.3, by the *orthogonal algorithm*, with the weights of the final layer in the Actor's network being scaled down by a factor of 100. These specifications will be the standard for all other architectures, unless otherwise specified.

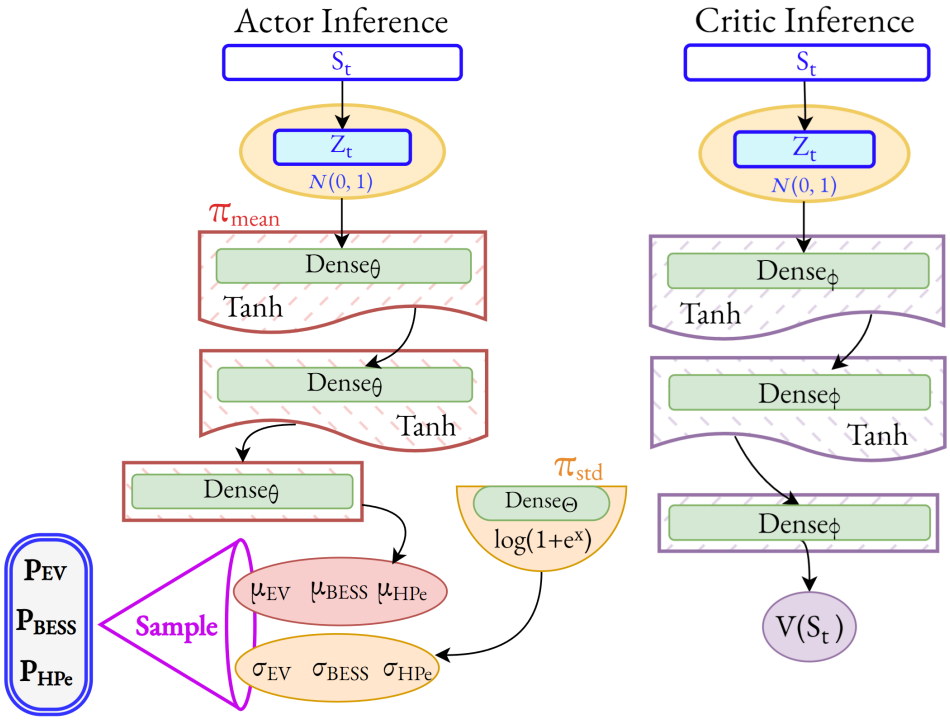

Figure 4.6: Diagrams of the Actor (left) and Critic (right) DNNs with *Constant Width* architecture. The Actor consists of two distinct DNNs, with the standard deviations produced independently from the feature vector's Z_t input. The Critic is utilised for value estimation, with one output per feature vector input.

A diagram representing the main features of the CW architecture can be seen in the Actor's π_{mean} network, shown in Figure 4.6. The state of the environment S_t is transformed into a feature vector Z_t (see Section 4.3), which is then normalised with the *z-score* function (see Equation 4.21). As detailed in the Section 4.4.2, the DNN π_{mean} is given a substantially greater capacity for abstraction in comparison to π_{std}, which is characterised by its independence from the feature vector and its single-layer depth.

$$z = \frac{x - \mu}{\sigma} \tag{4.21}$$

x : Original value
μ : Mean of the original values
σ : Standard deviation of the original values
z : Normalised value

More relevant information can be seen in the diagram (Figure 4.6), for example, the layers that constitute both networks are *dense*, i.e., each neuron in the layer is connected to every neuron in the previous layer. In addition, the last layer will not have a Tanh activation, so as not to bound the output. The means and standard deviations for each of the three possible decisions (as described in Section 3.3.3) define three Gaussian distributions, from which samples are drawn to determine the decision for the respective timestep.

The schematic representation of the inference process within the Critic network, specifically for the *Constant Width* architecture, is illustrated in Figure 4.6. This network is similarly subjected to the Tanh activation function throughout its layers, with the exception of the final layer. It should be noted that, in contrast to the Actor's network, the initial weights of the last layer in the Critic's network are not scaled down. The output will be a scalar for each timestep, representing the estimated value of the current state, which will be crucial when updating the Actor's DNN parameters.

Architecture as a hyperparameter

For the extended hyperparameter optimisation (Section 5.3.3), to increase the ability of the DNN to abstract patterns from the data, various new configurations, or architectures, were introduced. Andrychowicz et al.[4] tested the impact of different widths and depths of the Actor and Critic networks (as well as sharing an MLP for both), and their recommendations were followed for the basic hyperparameter optimisation (see Section 5.3.2). However, new configurations may prove useful and better the results obtained with the *Constant Width* architecture.

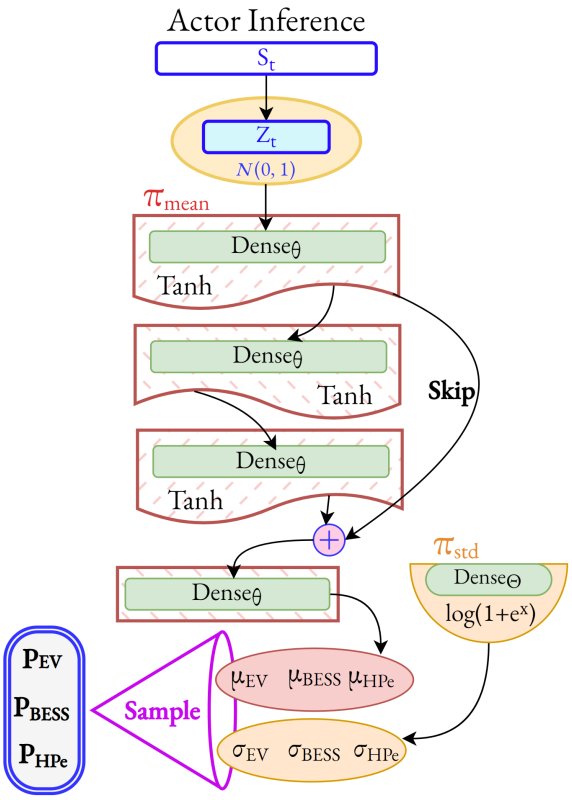

Figure 4.7: Diagram of the Actor DNN with *Residual* architecture. The output of the first dense layer goes through and **around** the residual block, which is made up of two dense layers. Both signals are added before entering the last layer of the network.

The architectures were designed with a focus on simplicity, trying to leverage some common designs to ascertain whether a notable increase or variation in performance could be achieved. Tables B.5 and B.6 show **compact symbolic representations of all the DNN architectures** that were used (found in Appendix B.2.4). The following paragraphs will elucidate the reasoning behind the inclusion of the

new configurations.

The *Bottleneck* design consists on considerably reducing the width of the middle hidden layers. This reduction in available neurons forces the network to compress the relevant information for decision-making into a lower-dimensional representation. This style of architecture is commonly used to compress data and extract its relevant features, potentially aiding in noise reduction and enhancing the agent's generalisation capabilities. However, various risks are associated with restricting the network's representation capacity. For instance, important information might be lost, complex relationships within the data could be overlooked (such as daily patterns), or the model might be prone to underfitting (i.e., not having enough capacity to model the data).

Residual networks, first introduced by He et al. [21], proved to be an effective tool at easing the training of the increasingly deep NNs used for image recognition. These networks are built from residual blocks, which divide the flow of input information, so that it travels two distinct paths. One path traverses dense layers (or convolutional layers), while the other bypasses all layers within the block. Both paths converge at the end of the residual block, where they are commonly concatenated or summed.

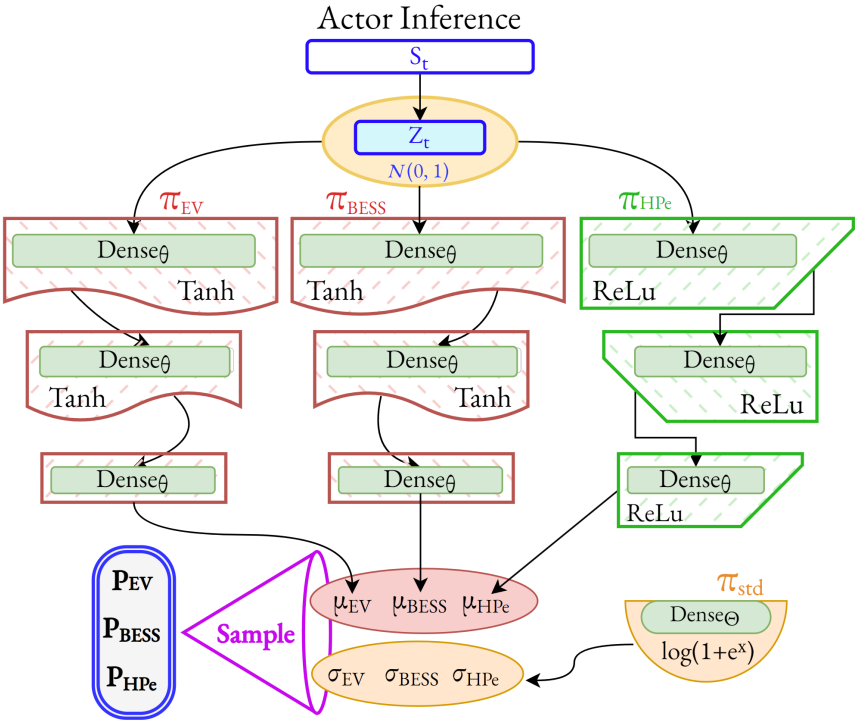

Figure 4.8: A diagram of the Actor DNN with *3 Pyramid Branches* architecture is presented. This configuration is also available to the Critic during hyperparameter optimisation, in which case an extra final layer will be added to combine the output of all branches into just one value.

As could be expected, most of the benefits of this architecture are found when training very deep NNs, which is not particularly relevant to this thesis. Nonetheless, it was selected as an option due to the high correlation between input and decisions (described in Section 4.3), as residual networks afford a more direct route for the feature vector to reach the final layers. Furthermore, residual blocks enable the network to augment the data rather than completely transform it, which may prove beneficial during the initial stages of training. Figure 4.7 illustrates the simple architecture designed to implement the concept of a residual block. Two additional residual architectures were implemented: *Deep Residual* utilises two consecutive residual blocks, while *Residual + Bottleneck* employs the bottleneck configuration on a single residual block.

Another straightforward modification to *Constant Width* was to create two deeper architectures with a uniform width, referred to as *Mid Constant Width* and *Deep Constant Width*. The aim of this alteration

is to increase the available representational capacity of the DNN used for the basic hyperparameter optimisation (Section 5.3.2). As detailed in Section 4.4.2, the research recommendations of Andrychowicz et al. [4] have been followed by implementing a state-independent NN to determine the standard deviation of the agent's decisions. However, in the interest of comprehensive hyperparameter optimisation, the *CW + Std* architecture will connect the feature vector back to the std network (π_{std}) and provide it with as many neurons as the mean network (π_{mean}).

The pyramid structure is also employed for various architectures; in this case, the layer width progressively decreases from input to output, not as abruptly as in *Bottleneck*. This gradual decrease in available neurons may guide the network towards reducing signal noise by naturally compressing the feature vector information. However, choosing the wrong size for intermediate layers could lead to underfitting or overfitting (i.e., too much capacity in the early layers to model data). As with *Bottleneck*, there is also the potential for the loss of useful information.

The final architecture that will be detailed makes use of parallel branches, in particular, one branch for each of the three decisions the agent has to make. The objective behind this structure is to force the Actor to specialise in each of the decisions independently. In an attempt to simplify slightly the agent's training, the third layer, associated with the power allocated to the heat pump (P^{e}_{HP}), will employ ReLu activation functions, including the final layer, since this is the only decision that cannot be negative. It is important to note that this architectural choice is highly focused on the problem at hand, and it will likely impact the agent's ability to generalise if the type or number of decisions were altered.

The parallel architecture might result in an inefficient use of resources, due to redundant patterns being learned in each branch. However, as previously mentioned, its aim is to expand the parameter space and allow the hyperparameter tuning algorithm to find the most optimal configuration (see Section 5.3). With a less constrained RL agent, the final design will be determined solely by the empirical results. Two versions of this architecture were designed, depending on the internal structure of their branches, that will be referred to as 3 Constant Width Branches (3CW) and 3 Pyramid Branches (3PYR) (represented in Figure 4.8).

Architecture Scaling

Many of the new architectures developed for the extended hyperparameter optimisation (Section 5.3.3) are considerably different to CW and therefore the amount of parameters available for the same input width varies considerably. This situation brings forth a possible issue, if a new architecture outperforms CW it might be attributed solely to an increase in parameter count. In an effort **to avoid conflating the size of the network with the architecture itself**, careful adjustments have been implemented to ensure that the relation between the input width (w) and the number of parameters remains highly consistent across all architectures, taking *Constant Width* as the reference.

The size of each DNN has been made solely dependent on the input width (w), the number of input features (ns), and the quantity of output values (na). The value of ns is determined by the chosen feature vector (described in Section 4.3), and na will always be 3 for the Actor and 1 for the Critic. As will be seen, the impact of small variations of ns or na on the total number of parameters is negligible, and these values shall be fixed at $ns = 30$ and $na = 3$ for the remainder of this Section.

As illustrated in Tables B.5 and B.6, the value w' represents the adjusted width relative to w. It should be clarified that the input width (w) and the actual width of the network will only coincide for the CW architecture, the rest will present an adjusted width w' that guarantees a very similar number of parameters to that of CW with width w. The input width w will take values between 16 and 512 during hyperparameter optimisation, defining the relevant range within which the DNN architectures should maintain similar sizes.

Figure 4.9 displays the relation between w and the number of parameters of each network architecture. As can be seen, all networks have been altered to follow the growth of CW. They are not perfect matches, but still sufficiently close so as to rule out DNN size as the culprit for any observed variation in performance. To compare the different growths with respect to w, all curves were fitted to a second-degree polynomial function of the form $f(w) = aw^2 + bw + c$, where a, b, and c are constants.

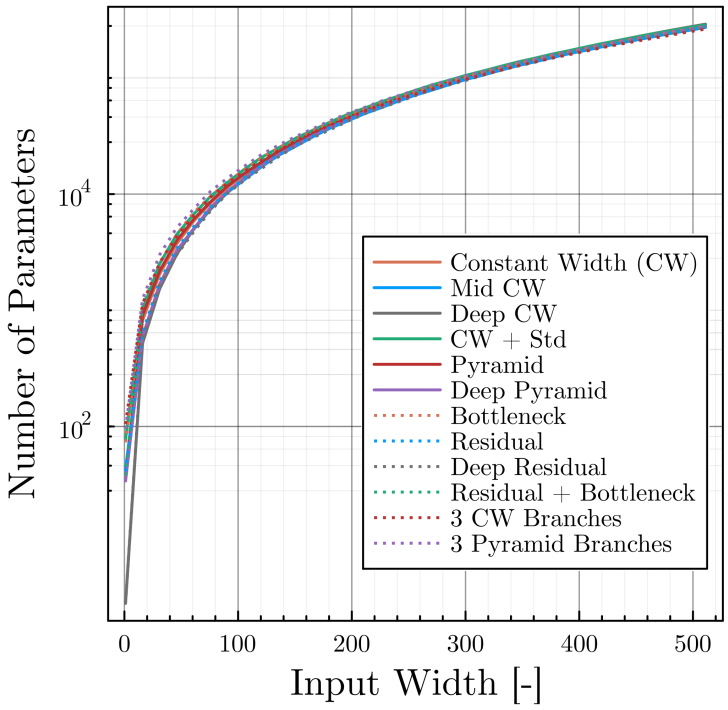

Figure 4.9: Parametric scaling characteristics of tested DNN architectures as a function of input width (w). The vertical axis employs a logarithmic scale (base 10) to accommodate the wide range of parameter values. Note the convergence across architectures for the displayed w range

This quadratic model was found to provide the best fit for the observed data, capturing the nonlinear relationship between input width (w) and parameter count. This is an intuitive conclusion, because in a fully connected (dense) layer, the number of parameters is approximately proportional to the product of the input and output dimensions, and both scale linearly with w. The layer connecting the feature vector (size ns) with the next layer (of size w) will therefore be represented in the first degree term of the approximation, since ns is unrelated to w.

The resulting polynomial approximations and their associated coefficients of determination (R^2) for each DNN architecture are presented in Appendix B.2.4, allowing for a quantitative comparison of the parameter growth rates.

4.5. Implementation of Policy Gradient Algorithms

reinforcement learning algorithms are often identified by short acronyms referring to specific loss functions or data collection methods, yet these represent only two relevant parts of the implementation. Unfortunately, the hidden low-level implementation choices may be crucial to the algorithm's good performance [4].

To investigate such factors, Andrychowicz et al. [4] conducted extensive evaluations of over 50 decisions taken during implementation. Their findings differentiate case-specific decisions from more stable choices. When implementing the algorithms, the former will be fine-tuned whenever possible (see Section 5.3), while the latter will be selected based on their recommended values.

4.5.1. Vanilla Policy Gradient with Critic (VPG-C)

The Vanilla Policy Gradient (VPG) algorithm, while often conflated with the REINFORCE algorithm [65] discussed in Section 2.7.2, represents a broader category of methods with diverse implementations,

yet all usually under the same name. VPG emerged as a practical enhancement to REINFORCE, addressing the high variance of policy gradient methods, improving learning stability. The particular approach taken to variance reduction depends on each implementation, but all approaches maintain the core concept of direct policy optimisation (by following the gradient of expected return).

For this version of Vanilla Policy Gradient (VPG), the **most significant addition is the Critic**, a DNN tasked with estimating the value function. The inclusion of the Critic allows the loss function to utilise an *N-step advantage estimator* (see Section 2.7.1), assigning weights to guide policy updates rather than relying on the *Reward-to-go* of REINFORCE [65]. For the sake of clarity, the current implementation will be henceforth referred to as **VPG with Critic (VPG-C)**.

Thanks to the theory presented in the literature review (Section 2.7), the equations used to implement reward processing (Section 2.7.1) have already been discussed. The basis for Actor and Critic losses has also been presented (Sections 2.7.2 and 2.7.3). Unless mentioned otherwise, the optimal decisions described in those sections have been implemented (e.g., the Critic loss is always evaluated using the MSE).

This classic algorithm has been improved by normalising advantages per update batch, which provides more stability and robustness, although it seems to improve performance only slightly [4]. After the advantages are estimated for a whole episode, the samples are shuffled and divided into batches. The data collection process itself, which is independent of VPG-C, can be found in Section 4.6. The loss function for a batch (size B) is computed as follows:

$$L_\pi(\theta) = -\frac{1}{B}\sum_{i=1}^{B}\left[\hat{A}_i^{N-step}\log\pi_\theta(x_i|S_i)\right] \tag{4.22}$$

It is worth noting that the logarithm of probabilities is always upper-bounded by zero. Therefore, when using SGD (see Section 2.7.4) to **minimise the loss function**, the sign of the log probabilities must be inverted (as shown in Equation 4.22). This will result in positive advantages being moved towards zero, by increasing the probability of the actions that led to them. Conversely, the loss associated with negative advantages can only be minimised if pushed towards $-\infty$, making the corresponding actions become less likely.

There are dozens of implementation details left, but they are not specific to VPG-C, decisions like the number of layers (Section 4.4.1), the type of optimiser, or the minimum limit for the standard deviation, can be altered without interfering with the VPG-C algorithm. Most of these hyperparameters, tunable or not, can be found in Section 5.3.1.

The implementation is written in the Julia programming language [8], which offers high performance computing without sacrificing productivity, and the initial structure of the algorithm comes from the RL Package in Julia [61]. The repository of SpinningUp [1] has also proved useful to connect theory and code, although the final implementation has suffered some modifications. The complete source code employed to address the research objectives of this thesis is accessible at: https://github.com/Victor-Andres-RdeTrio/RL4MCES

4.5.2. Advantage Actor Critic with Generalized Advantage Estimation (A2CGAE)

The research paper by Mnih et al. [39] presented the Asynchronous Advantage Actor-Critic (A3C), which was later simplified in a synchronous version and termed A2C. The fundamental structure of A2C bears many similarities to VPG-C, but it incorporates several improvements that merit a distinction between the two, such as the use of a Generalized Advantage Estimator, gradient clipping, or the addition of an entropy term to the loss function.

The implementation adopts a Generalized Advantage Estimator instead of the N-step estimator from the VPG-C (see Section 2.7.1), and given that Mnih et al. [39] only briefly mentioned GAE as a potential improvement in their original A3C paper, the algorithm needs to be referred to by a different name: A2CGAE. The decision to implement GAE aligns with the goal of maintaining computational

resource efficiency, since GAE values can be computed in linear time for all states within an episode [4].

Another major addition from A2CGAE when compared to VPG-C is the incorporation of an **entropy** term in the loss function to encourage exploration, which prevents premature convergence to suboptimal very deterministic policies. The approach is known as entropy regularisation.

$$L_\pi(\theta) = -\frac{1}{B}\sum_{i=1}^{B}\left[\hat{A}_i^{GAE}\log\pi_\theta(x_i|S_i) + w_{\text{entropy}}\cdot H(\pi_\theta(S_i))\right] \tag{4.23}$$

Equation 4.23 represents the loss function that will be used to estimate the gradients for the Actor update; the advantage will also be normalised (see Section 4.5.1). $H(\pi_\theta(S_i))$ is the entropy of the probability distribution provided by the policy π_θ for every state S_i, which should be reminded, will be **replaced in any practical implementation by the feature vector** Z_i (see Section 4.3). w_{entropy} is a hyperparameter controlling the strength of entropy regularisation. As can be seen, the sign of the entropy is also inverted, since the minimisation of the loss should be aided by the increase in entropy.

Entropy quantifies the expected information content of a distribution, and its basic formula for the continuous case is $H(X) = \mathbb{E}_x[I(x)] = -\int p(x)\cdot\log p(x)dx$. Since the output of the policy is a Gaussian distribution, the operation performed to obtain the entropy is:

$$H(X) = \frac{1}{2}\log(2\pi e\sigma^2) \tag{4.24}$$

The Critic of A2CGAE is trained to minimise the loss detailed in Equation 2.14, which applies to all algorithms that use GAE. Unlike in many A2C implementations [19, 64, 23] the Critic update is done independently from the Actor, this slightly complicates the implementation to improve readability and modularity in the code, since two loss functions are used. It should also be noted that this is an *episodic implementation* of A2C, where all the updating takes place between episodes.

After obtaining the gradients for both the Actor and the Critic, they will be **clipped** according to the chosen *maximum norm* allowed for the gradients. This is another improvement upon VPG-C.

Comments on parallelisation

Apart from the use of GAE, this particular implementation of A2CGAE diverges from the standard A2C in another significant aspect: it does not utilise parallel environments for agent training. This deviation could challenge the classification of the algorithm as a true A2C. However, the decision to retain the name of A2C comes from the considerable resemblance of the code to common A2C implementations [23, 19].

It was decided not to train with multiple environments for various reasons, although the most crucial of them was the **high consumption of computational resources**. Results from Andrychowicz and colleagues [4] point to the fact that parallelisation leads usually to a marked decrease in performance across some environments. They attribute this phenomenon to two main factors: firstly, the reduction in the size of experience chunks used for agent updates, and secondly, the introduction of early value bootstrapping, which likely adds considerable bias to the learning process.

Andrychowicz et al. affirm that parallel environments accelerate wall clock time at the cost of sample efficiency [4]. However, parallelisation is a powerful tool and its use should be considered when resources allow it, since it increases considerably the number of transition samples, providing robustness.

4.5.3. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO), introduced by Schulman et al. [51], represents a significant advancement in policy gradient methods. PPO builds upon the foundations laid by the TRPO algorithm

[52], with the aim of simplifying implementation, improving performance, and reducing computational cost [1]. In particular, the PPO-Clip algorithm is chosen, as it is simpler than PPO-Penalty and much more widely used.

As in A2CGAE, the advantage will be estimated with GAE and then normalised per update batch. Entropy regularisation will also play a role in the loss function.

$$r_i(\theta, \theta_{old}) = \frac{\pi_\theta(x_i|S_i)}{\pi_{\theta_{old}}(x_i|S_i)} \tag{4.25}$$

Let $r_i = r_i(\theta, \theta_{old})$ and $\text{clip}_i = \text{clip}(r_i, 1-\epsilon, 1+\epsilon)$:

$$L_\pi(\theta) = \frac{1}{B}\sum_{i=1}^{B}\left[\min\left(r_i \cdot \hat{A}_i^{GAE}, \text{clip}_i \cdot \hat{A}_i^{GAE}\right) - \beta H(\pi\theta(S_i))\right] \tag{4.26}$$

The ratio r_i represents the first-order attempt from PPO to approximate the Kullback-Leibler (KL) divergence between the new and old policies, which technically would be the expected value of the log likelihood ratio. A higher r_i implies that the action x_i has been made more likely when the agent is faced with S_i. The use of a ratio introduces the idea of a relative update, absent in the previous algorithms. Now, if an action is desirable –produces high positive advantage– its probability will be compared against the behavioural policy, not in absolute terms. To prevent this idea from creating an ever diverging policy, the clipping threshold (defined by ϵ) is used, regularising the NN. For optimal performance, it is recommended by Andrychowicz et al.[4] to tune the clipping threshold, with $\epsilon = 0.25$ being a good starting value.

In developing a Julia implementation of PPO (for the reasons stated in Appendix B.5), the focus was on incorporating only components with documented performance benefits, avoiding unnecessary complexity. This approach led to the exclusion of certain PPO features: the value loss clip (i.e. clipping of the Critic's loss function) was excluded, as [4] showed it hurt performance for any clipping threshold, while learning rate decay showed insufficient evidence of consistent benefits.

Other common features, like orthogonal weight initialisation, or *mini-batch* (subsets of episodic experience) policy updates, were already included in the previous algorithms (see Section 4.4.3). PPO can also be used in multiple environments in a synchronous manner, but this part of the implementation was avoided for the reasons exposed in Section 4.5.2.

As shown in Equation 4.26, the key innovation of PPO lies in its **elegant solution to a crucial challenge** in policy gradient methods: *what is the correct step size?* While earlier approaches made us of techniques such as adaptive learning rates and gradient clipping to control the magnitude of the update –techniques that PPO retains– Schulman et al. [51] thought to *clip* the loss function itself, limiting it to a *trust region*.

With this *trust region* in place, updates cannot move the new policy too far from the behavioural one (which collected the experience). TRPO [52] addressed this issue using second-order methods, since it not only calculates the gradient, but an approximation of the Hessian as well, taking curvature into account. This technique is computationally very costly, so PPO was developed to achieve similar constraints with first-order methods.

For the interested reader, the current implementation of PPO benefited from Huang and colleagues' detailed analysis [24], which reveals some implementation decisions omitted from the original PPO paper [51].

4.6. Data Collection

Each optimisation loop of the policy takes place over a set number of timesteps, which is defined by the episode length. The duration of each timestep is set before training begins. As noted in [4], the episode

length can have a quite significant impact on performance, so it can also be adjusted before the start of training (see Section 5.3.1). Going through the entire episode multiple times is very beneficial for **performance and sample complexity**, while requiring little extra memory. Each of these passes is called an **epoch**.

There are several ways to divide and process the data, but according to Andrychowicz et al. [4], the best performing method is "*Shuffle transitions (recompute advantage)*". In each epoch, the advantages will be recomputed, **since the Critic is being updated** along with the Actor. Then, each individual transition is randomly assigned to a subdivision of the episode data called a **batch**. When the optimisation process goes through all these batches, one epoch is completed. This approach is implemented **across all algorithms** in this study.

5

Hyperparameter Optimisation and Agent Evaluation

5.1. Introduction

The development of effective RL agents for energy management relies on three interconnected components: **performance metrics, hyperparameter optimisation, and agent selection**. The chapter first introduces comprehensive performance metrics that encompass grid operational costs, EV charging demands, and safety constraints. These metrics, designed to capture the SDP's objectives, are crucial for agent comparison.

The hyperparameter optimisation phase employs these metrics to systematically explore the hyperparameter space using the BOHB [17] algorithm. This exploration encompasses both basic and extended hyperparameter sets: the initial phase examines fundamental algorithm parameters across three stochastic policy gradient implementations (VPG-C, A2CGAE, and PPO), yielding distinct **hyperparameter profiles**, i.e. highest performing configurations of hyperparameters (κ). The following phase explores an extended parameter space including neural network architectures, reward function formulations, and feature vector configurations, resulting in additional profiles (Ω).

The final stage centres on obtaining optimal policies through structured validation. For each policy gradient algorithm, multiple agents are trained with their respective hyperparameter profiles and distinct random seeds. The top-performing agents are then evaluated on validation sets that test their generalisation capabilities. This pipeline is represented as a diagram in Figure 5.1. Once the pipeline has been followed for all policy gradient algorithms, six best RL agents emerge, the final step is to select the highest performing one (Section 5.4.2), which is defined as the *top RL agent (TRLA)*.

5.2. Performance Metrics

Before exploring hyperparameter optimisation in Section 5.3, it is essential to define the *performance metric*, which evaluates and guides the hyperparameter selection process. Algorithms like BOHB [17] (see Appendix B.1) will use this metric to efficiently allocate computational resources to promising hyperparameter configurations. As outlined in Section 2.7.1, the performance metric must accurately capture the core elements of the objective function (see Section 3.3.1) to ensure alignment with the SDPs goals.

The objective function of the RL agent is made up of three distinct components (see Equation 3.5), which will be referred to as the *grid cost, EV penalty* and *projection penalty*. Although combining these terms through simple addition might appear straightforward, this approach still requires careful

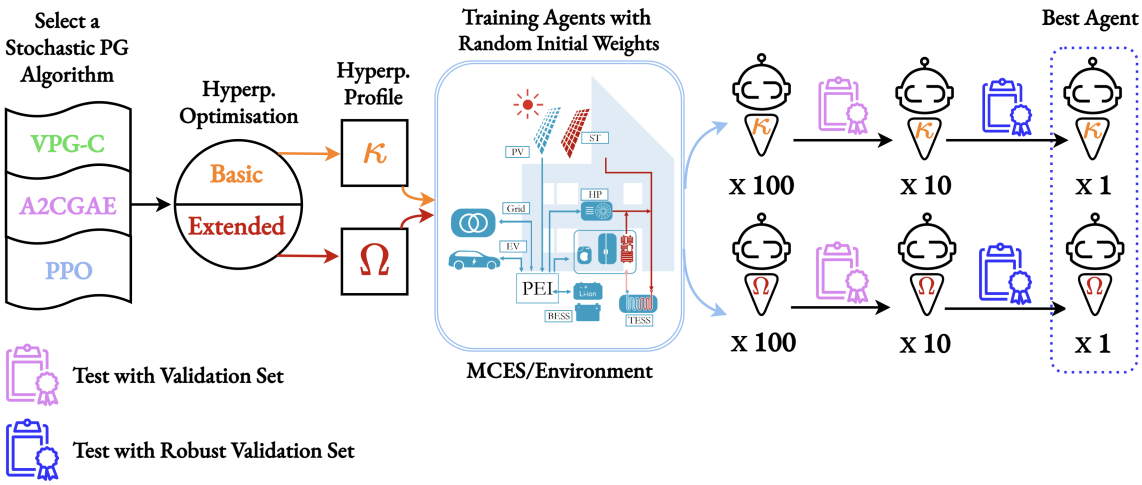

Figure 5.1: Simplified diagram representing the pipeline established by Section 5.3.4 and 5.4 to select the best performing agents for all policy gradient algorithms.

consideration of their relative weights. A significant problem arises from the ambiguous nature of this metric, as the same value could be achieved through different means. For instance, an agent might artificially reduce the grid cost by simply avoiding EV charging altogether.

For this reason, the final evaluations of performance will compare the objective function terms individually (as in Sections 5.4.2 and 6.2). However, for the purpose of hyperparameter optimisation, the performance metric must provide one value to each result.

As will be shown in this Section, all the performance metrics come together in the final metric M_{perf} utilises nonlinear criteria for each of the objective function components. This design choice improves discrimination between agent performances. In other words, a slight improvement in one of the terms causes a relatively bigger –nonlinear– impact on the final performance.

5.2.1. Grid Cost Metric

The average daily cost of exchanging energy with the grid ($C_{\text{grid, day}}$) will be compared with the daily average of the Expert. The grid cost performance metric (M_{grid}) uses a sigmoid function to normalise the metric between 0 and 1, where higher values indicate better performance through reduced grid costs.

$$M_{\text{grid}} = 1 - \frac{1}{1 + e^{-s(C_{\text{grid, day}} - m)}} \tag{5.1}$$

with parameters:

$$s = 1 : \text{slope}$$
$$m = 14.5 : \text{inflection point}$$

The midpoint was chosen at 14.5 because the expert performs with $C_{\text{grid, day}} \approx 14$, which would provide a value of $M_{\text{grid}} = 0.6225$. The expert's performance does not correspond to $M_{\text{grid}} = 1$ because better results of $C_{\text{grid, day}}$ may be obtained, although likely at the cost of worse performance in the other components of the objective function.

5.2.2. EV Penalty Metric

The electric vehicle performance metric is based on the deviation from the desired state of charge ($\text{SoC}^*_{\text{dep}}$) at departure times ($t_{\text{dep}}$). The EV penalty metric (M_{EV}) uses a normalized logistic function to evaluate these deviations.

$$M_{\text{EV}} = \frac{1}{\mathcal{D}} \sum_{t \in \mathcal{T}_{\text{dep}}} v(|\xi_{\text{SoCDep},t}|) \tag{5.2}$$

Where $\mathcal{T}_{\text{dep}}$ is the set of all departure times, $\mathcal{D}$ denotes the total number of days, and $\xi_{\text{SoCDep},t}$ represents the deviation from the desired SoC at departure time t. The function $v(x)$, defined in Equation 4.6, evaluates each departure event independently. A value of M_{EV} close to 1 indicates consistent achievement of desired charging levels, while values approaching 0 are associated with significant failures to meet $\text{SoC}^*_{\text{dep}}$. A perfect score will be obtained when no penalties are incurred, that is, when $\xi_{\text{SoCDep},t}$ is always zero.

5.2.3. Projection Penalty Metric

The performance metric is always used on agent's whose output is filtered by the *safe projection* (defined in Section 3.4.1), as this will be the setup after deployment. Since the aim is to find the RL agent that optimises **real-world performance**, the *projection distance* applied by the safety layer has been excluded from the performance metric. In other words, *projection distance* is only relevant during training (Section 3.4.2), as it helps to reach a final safer policy.

The only concern with regard to projections in real-world performance is that state variables may exceed their bounds even after the *safe projection* on the agent's decisions. Such situations, as explained in Section 3.4.1, may arise for variables dependent on unknown future (exogenous) information. Therefore, $\xi^{\text{p}}_{\text{BESS}}$, $\xi^{\text{p}}_{\text{EV}}$, $\xi^{\text{p}}_{\text{HP}}$ and $\xi^{\text{SoC}}_{\text{BESS}}$ will always remain null if the *safe projection* is being used. However, $\xi^{\text{SoC}}_{\text{EV}}$ may be non-zero due to its dependence on $P^{\text{drive}}_{\text{EV}}$, as the EV's power consumption outside the MCES might push its SoC out of bounds. The average magnitude of $\xi^{\text{SoC}}_{\text{EV}}$ will be related to M_{EV}.

To evaluate the severity of the possible operational projections, the metric M_{proj} is defined:

$$M_{\text{proj}} = \min_{i \in \mathcal{I}} \left(e^{-\Omega \cdot \xi_{i,\text{day}}} \right) \tag{5.3}$$

$$\mathcal{I} = \{\xi^{\text{SoC}}_{\text{EV}},\ \xi^{\text{SoC}}_{\text{TESS}},\ \xi^{\text{p}}_{\text{grid}},\ \xi^{\text{p}}_{\text{TESS}}\}$$

Where $\Omega = 10$ is the penalty coefficient, and $\xi_{i,\text{day}}$ represents the daily average of the constraint violations. The meaning of each of the components of $\mathcal{I}$ can be found in Section 3.3.1.

The exponential form of each component ensures that larger violations are penalized more severely. By taking the minimum value among all the terms, the metric identifies the most critical violation in the system.

5.2.4. Final Metric

The three metrics described previously will be combined into a comprehensive final metric M_{perf}. Safety considerations, represented by the projection metric M_{proj}, have been established as the first priority. The motivation is simple, if even after the *safe projection*, there are significant operational projections, the agent cannot be deployed in a real-world scenario. The second priority is the satisfaction of the desired state of charge (SoC^*_{dep}) for the electric vehicle, as grid cost reductions become irrelevant if the agent fails to charge the EV to accomplish them.

The first step in obtaining M_total is multiplying M_{proj} with M_{EV}, which will force them both to increase together to obtain a reasonably high product. The base component of the final metric is calculated as:

$$M_{\text{base}} = M_{\text{proj}} \cdot M_{\text{EV}} \tag{5.4}$$

Figure 5.2 shows a simplified version of M_{base}, where the values of $\xi_{i,\text{day}}$ and $\xi_{\text{SoCDep},t}$ are transformed by the underlying functions that define M_{proj} and M_{EV}. Then, added to M_{base} will be a variable dependent on grid cost performance M_{grid}, which will only contribute if minimum requirements of the other metrics are achieved.

$$\text{If } M_{\text{EV}} \geq 0.65 \wedge M_{\text{proj}} \geq 0.6:$$

$$M_{\text{bonus}} = (M_{\text{EV}} - 0.65)(1 + M_{\text{grid}}) + (M_{\text{proj}} - 0.60)(0.25 + M_{\text{grid}})$$

Otherwise:

$$M_{\text{bonus}} = 0$$

The final metric is then computed as:

$$M_{\text{perf}} = M_{\text{base}} + M_{\text{bonus}} \tag{5.5}$$

The bonus term is defined to reward incremental improvements above the chosen thresholds, with improvements in M_{EV} weighted more heavily than those of M_{proj}. The relation amongst the weights emerged from empirical observations in preliminary tests, where it proved considerably easier to increase M_{proj} than M_{EV}.

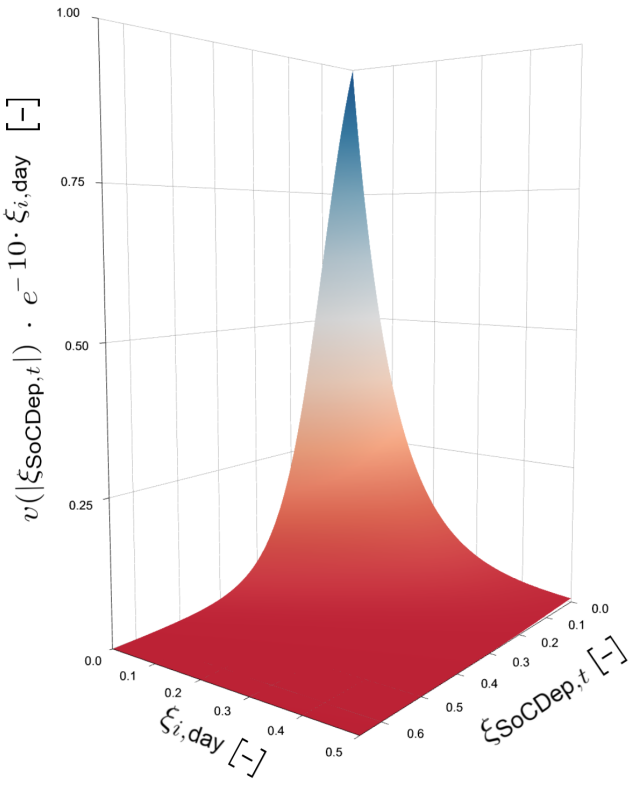

Figure 5.2: Simplified representation of M_{base} computation, incorporating a single generic projection component ($\xi_{i,\text{day}}$) as well as one daily EV penalty term ($\xi_{\text{SoCDep},t}$).

5.3. Hyperparameter Optimisation

The effectiveness of deep learning models is heavily dependent upon their internal hyperparameters. This dependency has been increasing over time, as the growth in computational capabilities has enabled the creation of larger models with more adjustable properties and resulted in higher training

expenses [17]. Even though the training of agents in this thesis is not expensive and can be done with a simple laptop, the difference in performance that is achieved by a correct selection of hyperparameters is more than remarkable. This fact becomes even more evident when one considers that it is not only the internal hyperparameters of the updating algorithm that can be adjusted, but also the **DNN architectures**, **reward functions**, and **feature vectors**.

To navigate this vast space of possibilities, novel hyperparameter optimisation algorithms needed to be developed. These algorithms are required to be scalable, robust, highly parallelisable, efficient, and offer superior performance. A particularly noteworthy example, especially for its simplicity, is the BOHB algorithm, developed by Falkner et al. [17]. This algorithm was used to find the best combinations of hyperparameters for all the algorithms used in this thesis, and therefore it will be briefly explained in Appendix B.1.

In a rather influential paper by Henderson et al. [22], a section is devoted to discussing the influence of hyperparameters. The authors express concern that their impact, along with other sources of uncertainty such as random seeds or code implementations, is so significant that it may obscure or artificially inflate progress in the development of RL algorithms. One of their recommendations, for the sake of reproducibility, is that researchers should ”report all hyperparameters, implementation details, experimental setup, and evaluation methods” [22]. This thesis is not trying to push the boundaries of the RL landscape, but this suggestion will be followed, so that, in combination with the code base, it becomes perfectly clear how the results were obtained.

In the following subsections, the available hyperparameters will be first described, then they will be accompanied by the ranges of values allowed during optimisation. Finally, the methodology followed to reach the best-performing configuration of hyperparameters will be presented in Subsection 5.3.4.

5.3.1. Hyperparameter Set

Hyperparameters are defined prior to training and will influence how the training itself takes place. From a wealth of possibilities, only a few hyperparameters can be optimised. Even with the use of advanced algorithms like BOHB [17], the incorporation of additional hyperparameters expands the dimensionality of the search space, leading to increased demands on computational resources.

A collection of hyperparameters will be referred to as a **hyperparameter set**. Each element within this set will have an associated **hyperparameter range**–a discrete array of possible values that the parameter may assume. The aggregation of all hyperparameter ranges for a given hyperparameter set forms the **hyperparameter search space**. This search space encompasses all possible configurations of hyperparameters and serves as the domain over which optimisation algorithms operate to identify an optimal combination of parameter values, which will be referred to as a **hyperparameter profile**. In this study, two major types of hyperparameter optimisations are carried out: the **basic** optimisation and the **extended** optimisation. Each optimisation will utilise a distinct hyperparameter set.

Based on the comprehensive large-scale analysis conducted by Andrychowicz and the *Google Brain Team* [4] two hyperparameter sets were designed: a basic and an extended one. The extended set supersedes the basic, exploring new hyperparameters or broadening existing ranges. The relevant information regarding each hyperparameter can be found in their related sections, which will be referenced below. The *extended* hyperparameter set elements are listed below, accompanied by brief descriptions (their ranges will be presented afterwards).

- **Discount Factor (γ)**: A crucial variable that governs the relative importance of future rewards in the decision-making process. Discussed in Section 2.7.1.

- **GAE** λ: Controls the trade-off between bias and variance in the Generalized Advantage Estimation. Discussed in detail in Section 2.7.1. This parameter is not used in VPG-C.

- **Initial Standard Deviation (σ_{init})**: The initial value of the standard deviation for the policy output, which can have a significant impact on exploration and convergence. See Section 4.4.2

- **Entropy Loss Weight (w_{entropy})**: Balances the exploration-exploitation trade-off by encouraging the policy to increase or decrease the entropy of its output. The implementation of entropy regularisation was described in Section 4.5.2. Not used in VPG-C. Andrychowicz et al. [4] found limited benefits from entropy regularisation in their research, suggesting that PPO might make it unnecessary. However, preliminary tests in this work revealed interesting results - both increasing and decreasing entropy could enhance performance. Therefore, the entropy loss weight range includes positive, negative and zero values, to explore all possibilities.

- **Actor and Critic Learning Rates (α_A, α_C)**: The step sizes for the parameter updates of the Actor and Critic networks when using the AdaBelief optimisation algorithm. Discussed in Section 2.7.4.

- **Clipping Threshold (PPO_ϵ)**: Limits the magnitude of policy updates during training, providing stability. Discussed in detail in Section 4.5.3. Exclusive to the PPO Algorithm.

- **Years of training (n_y)**: The number of times that the agent is trained on to the same year of data (i.e., the *expanded training set*).

- **Update Frequency (υ)**: The number of policy updates that will take place within each episode, with higher values resulting in reduced batch sizes per update. Episode subdivision methodology was outlined in Section 4.6.

- **Actor and Critic Architectures (ψ_A, ψ_C)**: Defines what DNN architecture is used for learning the policy. All the available architectures are defined in Tables B.5 and B.6, with the context provided by Section 4.4.4.

- **Activation Functions (ς_A, ς_C)**: The non-linear transformations applied to the network's hidden layers. Considering the analysis presented in Section 2.8, **Tanh was selected as the activation function for the Basic Range** (see Table 5.1), as it has shown to perform well [4]. However, to further explore potential performance improvements, **Mish will be introduced** as an additional activation function for the Extended Range (see Table 5.2).

- **Actor and Critic DNN Width ($width_A$, $width_C$)**: Determines the width of the DNN as defined in Section 4.4.4.

- **Feature Vector ($\mathcal{Z}$)**: Selects from the range of feature vector configurations. They are described in Section 4.3, with a comprehensive enumeration in Appendix B.2.1.

- **Reward Function ($\mathcal{R}$)**: The range of possible combinations of the available rewards. The context for reward design can be found in Section 4.2, with the combinations formalised in Equation 4.17.

- **Reward Weights ($\mathcal{R}_w$)**: The weights assigned to each of the final reward components. The list of options can be found in Appendix B.2.3.

Following the methodology employed by Andrychowicz et al. [4], the hyperparameter ranges in this study have been defined as discrete sets rather than continuous intervals. Limited computational resources also motivated this approach.

Several hyperparameters maintain fixed values throughout the optimisation process, mostly those found to be uncorrelated with performance improvements [4]. The **minimum standard deviation** is maintained at 0.01, as explained in Section 4.4.2. A **timestep** duration of 15 minutes was selected to match the highest resolution of real-world data (Section 3.6. Adjustment of **episode length** was initially considered significant (Section 4.6), but preliminary experiments showed its variation had minimal

impact on results despite significantly increasing the hyperparameter search space. Thus, it remains set at 96 timesteps, representing 1 day in the MCES environment.

The **gradient clipping threshold** (discussed in Section 2.7.4) is established at 0.5. Andrychowicz et al. [4] affirm that limiting the gradient norm improves performance, regardless of the specific threshold used. The authors also recommend going over the collected experience more than once, which has been implemented by **performing 3 epochs over each training episode**.

Hyperp.	Range
γ	0.6, 0.8, 0.9, 0.95, 0.99, 0.999
λ	0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 0.99
σ_{init}	0.1, 0.5, 1.0, 2.0
w_{entropy}	-0.01, 0.0, 0.01
α_A	3×10^{-5}, 1×10^{-4}, 3×10^{-4}, 1×10^{-3}, 3×10^{-3}
α_C	α_A
υ	1, 2, 3
PPO_ϵ	0.1, 0.2, 0.3, 0.5
$width_A$	16, 32, 64, 128, 256, 512
$width_C$	128
ψ_A, ψ_C	Constant Width
ς_A, ς_C	Tanh
n_y	3, 6
$\mathcal{Z}$	2 (Table B.1)
$\mathcal{R}$	Equation 4.18
$\mathcal{R}_w$	1 (Table B.4)

Table 5.1: Basic hyperparameter set. See Section 5.3.1 to find context and explanations on all the presented hyperparameters.

5.3.2. Basic Range

Table 5.1 displays the range for all the elements of the *basic hyperparameter set*. The range for hyperparameters that will only be adjustable in the *extended hyperparameter set* is also provided, with the intention of delineating the search space permitted to the optimisation algorithm (BOHB [17]). Notably, the Critic learning rate (α_C) will be the same as that of the Actor, as they are not distinguished by Andrychowicz et al. [4].

5.3.3. Extended Range

The basic hyperparameter set was expanded for several reasons. Firstly, augmenting the number and range of hyperparameters allows the model to represent a wider scope of functions or more complex relationships within the data, mitigating the risk of underfitting. Additionally, a larger search space that explores DNN architectures, advanced feature engineering, and alternative reward functions may enable the **discovery of higher-performing model configurations**. Furthermore, the increased flexibility provided by the extended set can help balance the model's complexity, such as the number of parameters or required input features, and its generalisation capabilities, thereby providing a path for simpler models to achieve competitive performance without overfitting to the *validation set*.

Table 5.2 outlines the available ranges for each hyperparameter in the expanded set. As can be seen, the Critic learning rate (α_C) has been made independent of the Actor's. It should be pointed out that the *CW + STD* neural architecture (see Section 4.4.4) is specific to the Actor, as it is meant to test the effects of connecting the standard deviation network to the feature vector, which is only relevant for decision-making.

Hyperp.	Range
γ	0.6, 0.8, 0.9, 0.95, 0.99, 0.999
λ	0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 0.99
σ_{init}	0.1, 0.5, 1.0, 2.0
w_{entropy}	-0.01, 0.0, 0.01
α_A	3×10^{-5}, 1×10^{-4}, 3×10^{-4}, 1×10^{-3}
α_C	3×10^{-5}, 1×10^{-4}, 3×10^{-4}, 1×10^{-3}, 3×10^{-3}
υ	1, 2, 3
PPO_ϵ	0.1, 0.2, 0.3, 0.5
$width_A$	16, 32, 64, 128, 256, 512
$width_C$	16, 32, 64, 128, 256, 512
ψ_A	All architectures (Tables B.5, B.6)
ψ_C	All architectures except CW + STD (Tables B.5, B.6)
ς_A	Tanh, Mish
ς_C	Tanh, Mish
n_y	3, 6
$\mathcal{Z}$	1 - 9 (Tables B.1, B.2, B.3)
$\mathcal{R}$	Equations 4.18,4.19, B.2 - B.7
$\mathcal{R}_w$	1 - 3 (Table B.4)

Table 5.2: Extended hyperparameter set. See Section 5.3.1 to find context and explanations on all the presented hyperparameters.

5.3.4. Procedure

The hyperparameter optimisation will be conducted in two phases. First, the basic hyperparameter set (see Section 5.3.2) will be explored with each of the policy gradient algorithms implemented in Section 4.5 (VPG-C, A2CGAE, and PPO), yielding three distinct **hyperparameter profiles** (Section 5.3.1), denoted by κ. Subsequently, the extended hyperparameter set (see Section 5.3.3) will be used as the search space, resulting in an additional three hyperparameter profiles, which will be denoted as Ω.

The process to determine the hyperparameter profile for each policy gradient algorithm will follow the following procedure, which can be visualised in Figure 5.3:

1. The hyperparameter set (Section 5.3.1) is defined and serves as the input for the BOHB algorithm [17] (see Appendix B.1 for more information).

2. BOHB will sample the hyperparameter search space.

3. The sampled values for the hyperparameters are used to train three policies, each with weights

initialised based on different random seeds.

4. The policies will be trained using the *expanded training set* (Section 3.6.1) and the *simple projection* (as explained in Section 3.4.2).

5. Once the training is completed, all three agents are tested independently with the *validation set* (Section 3.6.1). For these tests, the *safe projection* is used.

6. The validation runs are evaluated according to a specified performance metric (Section 5.2).

7. The average performance of the three policies is stored and used to guide BOHB in its following hyperparameter sample.

This iterative process of sampling and evaluation will be repeated until a sufficient number of samples have been tested, afterwards the best performing sample becomes the **hyperparameter profile** for that algorithm. To account for the stochasticity in model training that emerges from random weight initialisation, the policies were evaluated using three distinct random seeds values. Although a more comprehensive evaluation would ideally employ a **larger number of seeds**, computational limitations prevented the analysis of more than three seeds.

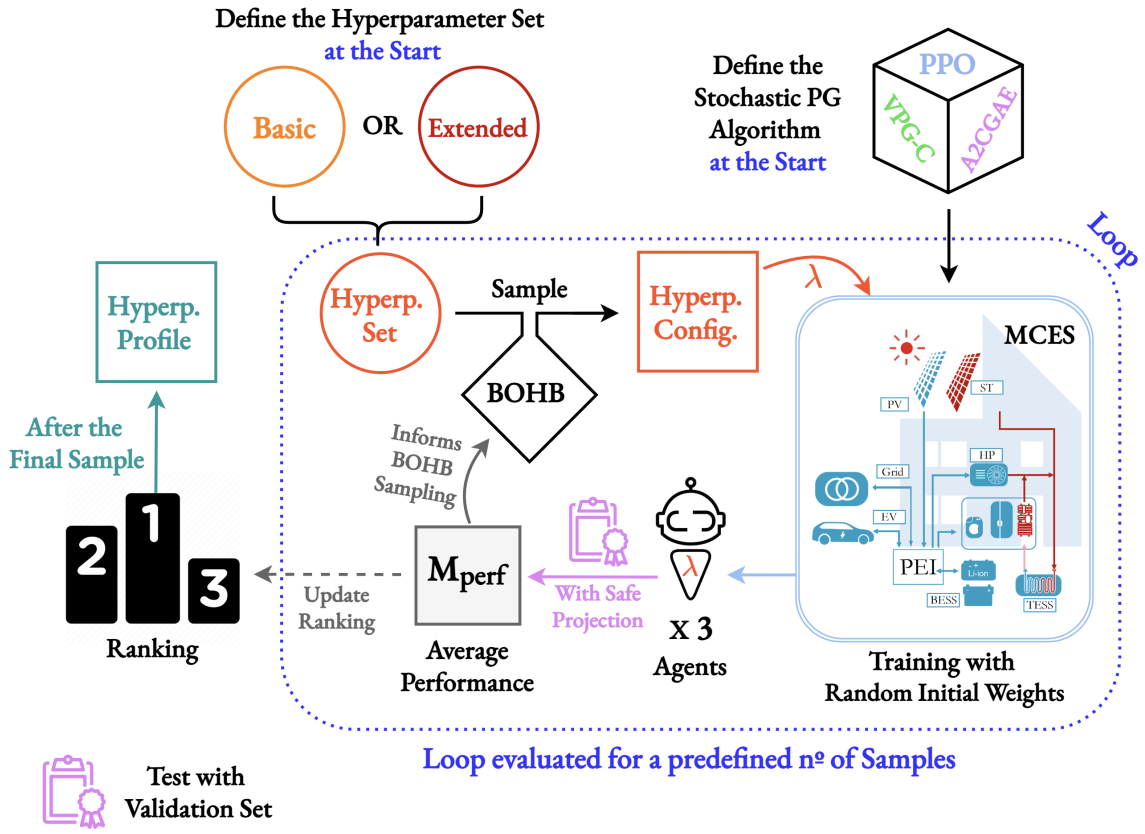

Figure 5.3: Visual representation of the hyperparameter optimisation process described in Section 5.3.4. Each sample of the hyperparameter set produces a hyperparameter configuration denoted in the diagram with λ. The sampling will take place in parallel threads until the predefined number of samples is reached, information is shared amongst threads. As shown, the performance metric used is M_{perf} (Section 5.2). The hyperparameter configuration at the top of the ranking when the optimisation is finished will be defined as the *hyperparameter profile*.

The selection of the **hyperparameter profile** is based on the average policy performance, which is assumed to serve as a reliable indicator of the hyperparameter set's capabilities. This heuristic **prioritises robustness**, focusing on consistently high-performing hyperparameters. As will be detailed

in Section 5.4, the best-performing policy is ultimately identified at the extremes of the performance distribution, implying that the optimal policy may be an outlier. While robust hyperparameter profiles will reduce the spread of performance among the resulting agents, an intriguing **avenue for future research** would be to explore the impact of prioritising hyperparameter profiles with the most distant performance outliers. Given the limited computational resources available, the decision to prioritise robustness was a prudent one.

5.3.5. Results

The optimisation procedure laid out in Section 5.3.4 was applied across all the implemented policy gradient algorithms (Section 4.5). The **basic hyperparameter profiles**, designated by the subscript κ in Table 5.3, were derived with the BOHB algorithm from 500 samples of possible configurations. The **extended hyperparameter profiles** were found after testing 600 samples by BOHB, and they are indicated by the subscript Ω, as can be seen in Table 5.3.

Hyperp.	**VPG-C$_\kappa$**	**A2CGAE$_\kappa$**	**PPO$_\kappa$**	**VPG-C$_\Omega$**	**A2CGAE$_\Omega$**	**PPO$_\Omega$**
γ	0.6	0.99	0.999	0.8	0.99	0.6
λ	-	0.9	0.95	-	0.9	0.99
σ_{init}	1.0	0.5	2.0	2.0	2.0	1.0
w_{entropy}	-	-0.01	0.0	-	0.01	0.0
α_A	1×10^{-3}	3×10^{-5}	1×10^{-4}	1×10^{-4}	3×10^{-4}	1×10^{-3}
α_C	1×10^{-3}	3×10^{-5}	1×10^{-4}	1×10^{-4}	1×10^{-4}	3×10^{-4}
υ	3	2	3	1	2	3
PPO_ϵ	-	-	0.3	-	-	0.1
$width_A$	128	512	256	64	16	64
$width_C$	128	128	128	64	512	32
ψ_A	CW	CW	CW	3PYR	3CW	3CW
ψ_C	CW	CW	CW	RES	DCW	BOT
ς_A	Tanh	Tanh	Tanh	Mish	Mish	Mish
ς_C	Tanh	Tanh	Tanh	Mish	Tanh	Tanh
n_y	6	6	6	6	6	6
$\mathcal{Z}$	2	2	2	2	5	1
$\mathcal{R}$	Eq. 4.18	Eq. 4.18	Eq. 4.18	Eq. B.7	Eq. B.6	Eq. B.3
$\mathcal{R}_w$	1	1	1	3	1	3

Table 5.3: Hyperparameter profiles for the policy gradient algorithms VPG-C, A2CGAE, and PPO. The subscripts κ and Ω indicate the result of an optimisation performed with the Basic (Table 5.1) or the Extended (Table 5.2) Hyperparameter Set, respectively.

Although the identified *hyperparameter profiles* represent local optima in the search space, they offer only a glimpse into the performance landscape across all the available configurations. The following will be an analysis of the broader patterns that emerged during the search for the ***extended** hyperparameter profiles*, as the *extended set* supersedes the *basic* (Sections 5.3.2 and 5.3.3).

As was detailed in Section 5.3.4, each BOHB algorithm sample represents a unique hyperparameter configuration, which will be evaluated according to the average performance metric of the agents trained with that configuration. The analysis must be **approached with caution**, due to the limited dataset size

of 600 samples per algorithm (1800 total) and the highly interconnected nature of the samples, since each hyperparameter value will have an effect on the evaluation. Therefore, no definitive conclusions can be drawn; instead, some of the more clearly observed patterns will be pointed out.

The following datasets, presented through box plots, demonstrate pronounced skewness, evidenced by off-centre median positions. This asymmetric behaviour is likely due to the **rigorous nature of performance metric** (M_{perf}), outlined in Section 5.2. As a result, all agents with minor safety concerns, or those that inadequately address the desired EV SoC, are assigned $M_{\text{perf}} \approx 0$. Only those agents that surpass $M_{\text{perf}} > 0.05$ are deemed relevant, and will therefore be represented in Figures (such as 5.4 or 5.5) and analysed. Despite the implementation of this threshold, performance metrics predominantly cluster toward the lower bounds of M_{perf}.

Algorithm Comparison

Before commenting on the patterns observed in Figure 5.4 it is worth noting that the displayed performance is measured on the *validation set*, as the test set cannot be used for selecting the hyperparameter profile. The Figure includes only **successful agents**, defined by a performance above the threshold of 0.05. Given the stochastic nature of the training process, the count of successful agents can be modelled using a **binomial distribution**. The distribution –displayed in the legend– captures the total count of discrete outcomes (success or failure), represented by n, and the underlying probability of success for each agent (p).

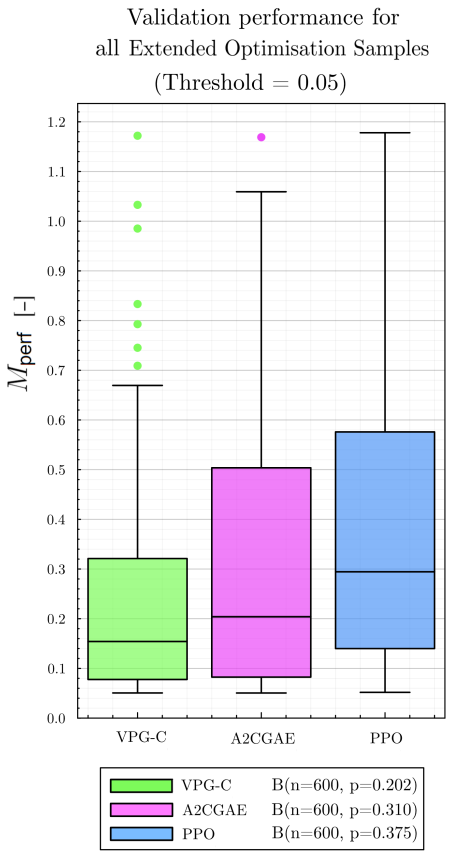

Figure 5.4: Box plot for the performance of the VPG-C, A2CGAE and PPO algorithms when considering all the successful samples taken during the *extended* hyperparameter optimisation. Whiskers extend to 1.5 times the interquartile range and dots represent outliers. The legend includes the binomial distribution parameters B(n,p), indicating the probability of producing agents above the threshold for each sampled hyperparameter configuration.

Several distinct performance patterns can be seen in Figure 5.4, where a box plot analysis goes across all policy gradient algorithms. PPO demonstrates superior median performance and success

probability (37.5%), which may be viewed as a measure of algorithm robustness. The interquartile range (IQR) is wider for A2CGAE and PPO, although this higher dispersion can be partly attributed to the non-linear nature of M_{perf} (Section 5.2), which amplifies performance differences at higher values. The best performance of all the algorithms is very similar, but A2CGAE, and most notably VPG-C, achieve their peak performances through outlier configurations, i.e. outside of the $1.5 \cdot IQR$ range represented by the whiskers.

Comparing Reward Functions

The extension of the hyperparameter space through various reward functions and their combinations (making up different versions of the final reward), presented in Section 4.2, will alter the final policy landscape. Some components of the reward will present shaped alternatives. Observing the results of the *extended* optimisation, a few clear insights can be found regarding the effectiveness of the reward functions.

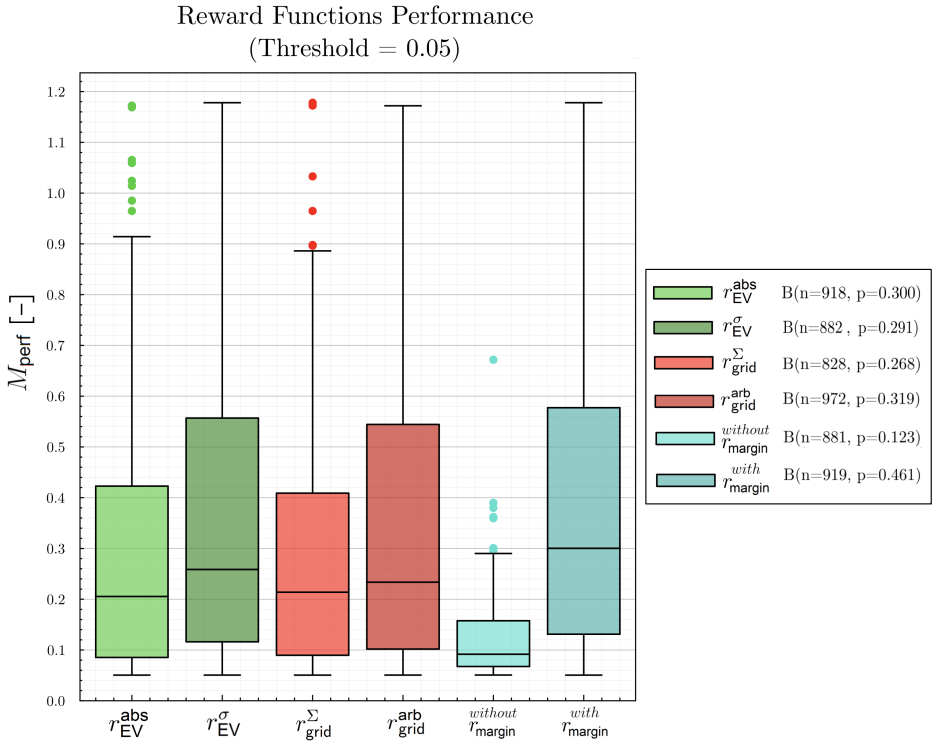

Figure 5.5: Box plot for the performance of the components that constitute the final reward function (Equation 4.17), when considering all the successful samples taken during the *extended* hyperparameter optimisation. Whiskers extend to 1.5 times the interquartile range and dots represent outliers. The legend includes the binomial distribution parameters B(n,p), indicating the probability of producing agents above the threshold when trained using the corresponding reward component.

The final reward function (Equation 4.17) is made up of constant and varying components; therefore, it is only useful to compare the performance of the latter, which address EV SoC requirements (r_{EV}), grid power exchange costs (r_{grid}), and the respect for boundaries for all MCES components (r_{margin}). This comparison is presented in Figure 5.5, where it is important to keep in mind that the performance values are obtained from training agents with a combination of rewards, and therefore the **results for each are interconnected**. For instance, if a reward component had an incredibly low success rate, none of the other components' impacts on performance could be evaluated.

Even when considering the interdependence of all the displayed datasets, a significant performance disparity exists between agents trained with and without the **margin reward** (r_{margin}). The inclusion of r_{margin} yields a $3.75x$ **higher success rate** and **enhanced performance** metrics for the median, dispersion, and peak performance. The dual role of r_{margin} in promoting both safety (i.e., lack of constraints

violations) and charging EV batteries is probably responsible for the considerable impact on performance.

The EV and grid cost reward components show subtle variations in performance patterns. While $r^{\text{arb}}_{\text{grid}}$ and r^{g}_{EV} demonstrate marginally superior medians, they share similar peak performances. However, it may be said that reward shaping (represented by $r^{\text{arb}}_{\text{grid}}$ or r_{margin}, as described in Section 2.7.1) demonstrates beneficial effects on average performance and training success without compromising the upper bound of performance.

The basic hyperparameter set (Table 5.1) utilises a final reward function without the margin reward component (Equation 4.18). The results observed in Figure 5.5, suggest that this omission hampered agent training for the basic optimisation, potentially explaining the considerable performance gap between basic and extended hyperparameter profiles (see Section 5.4.1).

Other Relevant Hyperparameters

The significant impact of policy gradient algorithms and reward functions on the *extended hyperparameter optimisation* was demonstrated. This Section will shift focus to other hyperparameters, aiming to identify their potential contribution to the observed improvements over the *basic hyperparameter optimisation*. To avoid extending the main body of the thesis more than necessary, the findings will be summarised, and those considered of possible interest to the reader will be expanded upon in Appendix B.3.

- **Actor Architecture (ψ_A)**: The introduction of alternative architectures to CW appears to have yielded **performance improvements**, with an 18.7% increase in maximum performance observed using 3CW. Both 3PYR and Deep Constant Width (DCW) showed superior performance as well, while *DPYR* significantly underperformed. High-performing outliers suggest that specific hyperparameter configurations may be able to leverage the more complex architectures. The analysis is expanded upon in Appendix B.3.1.

- **Critic Architecture (ψ_C)**: The analysis reveals performance patterns clearly distinct from the Actor's, with significant improvements observed with *Bottleneck* and *Residual* architectures. The findings are detailed further in Appendix B.3.1.

- **Feature Vector ($\mathcal{Z}$)**: No substantial impact was observed on performance. *Configuration 2*, used in basic hyperparameter profiles, shows great and robust results. Surprisingly, more complex feature vectors often underperformed, likely due to overfitting. The findings are examined further in Appendix B.3.2.

- **Actor Activation Function (ς_A)**: The adoption of the *Mish* activation function in Actor networks consistently achieved higher-end performance and improved training robustness.

- **Critic Activation Function (ς_C)**: Adding *Mish* as a possible activation function had no discernible impact on performance.

- **Critic DNN Width ($width_C$)**: Increasing the width of Critic networks beyond 128 improved the maximum performance and its consistency. Wider networks demonstrated superior robustness, validating observations by Andrychowicz et al. [4].

- **Reward Weights ($\mathcal{R}_w$)**: The variation in reward weights did not affect maximum performance, but *configuration 3* shows improved robustness, achieving higher-end performances more consistently, alongside increased training success rates.

5.4. Finding a Near-Optimal Policy

Once the **hyperparameter profile** has been obtained for each policy gradient algorithm (see Section 5.3.5) the process of finding a nearly optimal policy begins. As presented in Section 5.3 the stochastic nature of RL asks for a rigorous approach to evaluating and selecting policies. As Henderson et al. [22] demonstrate, the **inherent randomness** in RL – brought upon, for instance, by weight initialisation, environment dynamics, or code implementation– will significantly impact policy performance.

The hyperparameter profile is determined by the average policy performance, as elaborated in Section 5.3.4. However, due to the inherent randomness, multiple training iterations with different random seeds are essential to identify the policy closest in performance to the expert. Ideally, with sufficient epochs of training, all policies would converge to the global optimum; regrettably, this is not the observed outcome. Even with the same hyperparameter profile, different initialisations are sufficient to trap agents within local optima, with further training not improving the situation.

Following the presented reasoning, and with the sole objective of obtaining the best performing RL agent, the process followed to select one final policy will be now described, and can also be found as part of Figure 5.1:

1. An algorithm (Section 4.5) and the highest-performing configuration of hyperparameters (i.e. the hyperparameter profile) were selected.

2. 100 agents were trained with distinct random seeds.

3. The agents were evaluated on the *validation set* (see Section 3.6.1) using the *safe projection*.

4. The validation runs were assessed according to a specified performance metric (Section 5.2).

5. The results were analysed, and the best 10 agents were identified.

6. To favour the selection of an agent with improved capacity to generalise (and to avoid the risk of overfitting to the *validation set*), the top 10 agents were tested with the *robust validation set* (Section 3.6.1).

7. The results were analysed, and the optimal agent was selected.

5.4.1. Hyperparameter Profile Performance Analysis

In this Section, further analysis is provided of the validation results after training 100 agents with each of the available policy gradient algorithms (VPG-C, A2CGAE, and PPO) and their corresponding **hyperparameter profile**. The training process yields agents with a range of behaviours, those that achieve a *validation set* performance exceeding 0.05, will be defined as *successful* agents. This classification was presented in Section 5.3.5, where the binomial distribution used to model the likelihood of success is also detailed.

Each algorithm is presented twice, first with the hyperparameter profile obtained from the *basic optimisation* (represented by κ, see Section 5.3.2) and then with the resulting hyperparameters from the *extended optimisation*, indicated by Ω (see Section 5.3.3). When observing Figure 5.6 the great difference between the *basic* and *extended* hyperparameter optimisations becomes apparent. This considerable divergence in performance medians suggests that the expansion of the hyperparameter search space was a good decision. The enhanced consistency in performance also indicates that more robust hyperparameter configurations have been made available.

The *basic optimisation* results are the only ones exhibiting failing agents, as evident from the binomial approximations presented in the legend. Notably, VPG-C_κ is highly unlikely to produce a successful agent, while more advanced algorithms present a considerably higher success rate under similar conditions. Surprisingly, VPG-C_Ω presents the greatest median and maximum, showing that even a

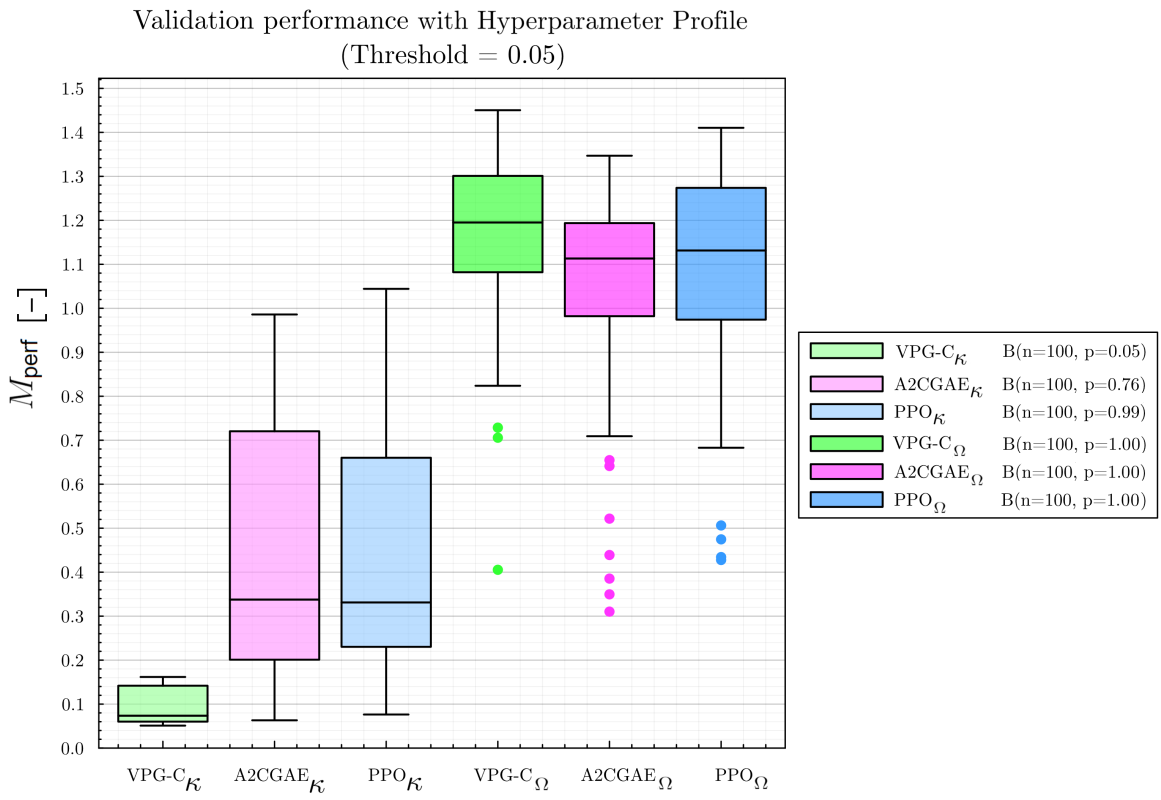

Figure 5.6: Comparison of the performance in the *validation set* (Section 3.6.1) for VPG-C, A2CGAE, and PPO algorithms when trained with **basic (κ) and extended (Ω) hyperparameter profiles** (see Section 5.3.1). Box plots show the distribution of performance metrics for 100 agents, only considering those above the success threshold of 0.05. Whiskers extend to 1.5 times the interquartile range and dots represent outliers. The legend includes the binomial distribution parameters B(n,p), indicating the probability of producing agents above the threshold for each hyperparameter profile.

very unsophisticated algorithm can outperform the rest with the proper configuration of hyperparameters. Furthermore, Figure 5.6 demonstrates a strong correlation between the mean and the best achieved performance, as anticipated within the hyperparameter optimisation framework (see Section 5.3.4). However, it is worth noting that the hyperparameters obtained through such a procedure may inherently bias the results towards performances grouped closer to the mean.

5.4.2. Comparison of Best Agents

To obtain the six best agents, each policy gradient algorithm is used following the procedure in Section 5.4, utilising the basic (κ) and extended (Ω) hyperparameter profiles. Six RL agents are obtained, each of which achieves the best *validation* performance out of 100 trained agents. Further training might yield higher-performing agents, but this could also lead to overfitting specific to the *validation* and *robust validation sets*. However, the selected agents are regarded as good representatives of their respective algorithm's potential under suitable hyperparameter configurations.

The six RL agents will now be compared using results from the *robust validation set* (Section 3.6.1). The final evaluation against the Expert relies on specific metrics derived from the terms of its objective function (Section 3.3.1), termed *Objective Metrics* (O_M). Therefore, these metrics will be used to select the *top RL agent (TRLA)* in this final stage. These metrics are assessed independently; thus the weights used by the Expert's objective function are not needed.

To obtain the **average values** for the O_M (with confidence intervals) displayed in Table 5.4, a series of 100 simulations were performed for each of the six agents, utilising the *robust validation set*. Distinct random seed values were used for the simulations, altering the policy's decision sampling (see Section

	C_{grid} [€]	p_{SoCDep} [-]	Aux_{TESS} [s]
VPG-C$_{\mathcal{K}}$	2363.7 $\pm$ 0.0	8.429 $\pm$ 0.0	0.0 $\pm$ 0.0
A2CGAE$_{\mathcal{K}}$	2253.1 $\pm$ 0.6	0.404 $\pm$ 0.003	0.0 $\pm$ 0.0
PPO$_{\mathcal{K}}$	2491.1 $\pm$ 2.3	0.542 $\pm$ 0.017	0.0 $\pm$ 0.0
VPG-C$_{\Omega}$	**2226.2** $\pm$ 1.1	0.184 $\pm$ 0.004	0.0 $\pm$ 0.0
A2CGAE$_{\Omega}$	2297.9 $\pm$ 1.3	0.297 $\pm$ 0.003	0.0 $\pm$ 0.0
PPO$_{\Omega}$	2406.7 $\pm$ 0.9	**0.068** $\pm$ 0.002	0.0 $\pm$ 0.0

Table 5.4: Comparison for the best six RL agents obtained by using VPG-C, A2CGAE, and PPO with basic (κ) and extended (Ω) hyperparameter profiles. Results are obtained from the *robust validation set*, using the Expert's objective function components as the evaluation criteria, termed the *Objective Metrics* (O_M). Note: The *robust validation set* reduced solar radiation to heighten challenges in grid cost minimisation and EV SoC requirements, making TESS SoC upper bounds unlikely to be reached, resulting in no Aux_{TESS}.

4.4.2). The 95% confidence interval was obtained through a non-parametric bootstrapping procedure with 10,000 resamples of the original dataset.

As could be expected from the results displayed in Section 5.4.1, the extended hyperparameter profiles have produced the most promising RL agents. In particular, VPG-C$_{\Omega}$ achieves a notable reduction in C_{grid}, whereas PPO$_{\Omega}$ excels in minimising the EV SoC satisfaction penalty (p_{SoCDep}), as summarised in Table 5.4. After observing the results, **the *TRLA* is deemed to be that produced by PPO$_{\Omega}$**. Although it incurs a higher C_{grid}, its substantial reduction in p_{SoCDep} justifies the trade-off, particularly given the role of EV charging demands in elevating C_{grid}.

6

RL EMS Validation and Conclusions

6.1. Introduction

This chapter focuses on validating the policy of the *TRLA* –obtained in Chapter 5– (*TRLA*) through a detailed comparison with the Expert. The discussion begins with an evaluation of key performance metrics derived from the Expert's objective function, followed by an in-depth analysis of the differences found in the decision-making of the agents. Particular attention is given to their impact on grid energy exchange and the SoC of the storage assets.

Safety considerations also occupy a relevant section in this chapter, measuring the agents' compliance with the MCES's established constraints. In addition, a performance comparison synthesis is made, supporting the validation of *TRLA*. The chapter concludes by addressing the research questions, posed at the start of this work (Section 2.11), and provides commentary on some opportunities for further exploration.

6.2. Comparison with Expert

After successfully training a RL agent to manage the MCES (presented in Section 3.2), the crucial next step is to validate its performance against an Expert, described in Section 3.5. The RL agent deemed most successful in Section 5.4.2 will be referred to as the *top RL agent* (*TRLA*).

This comparison will be conducted using the *TRLA* and the Expert under identical simulations with the *test set* (Section 3.6), where the reliability, efficiency and robustness of the *TRLA* will be assessed. As indicated in Section 3.4.3, the *safe projection* will always be in place when testing an RL agent. The initial conditions and the exogenous information will be the same, however, it is relevant to note what specific information is available to each system for decision-making.

The *TRLA* generates decisions by processing past states, represented within the feature vector (Section 4.3). In contrast, the Expert is provided with **future exogenous information** of the next two days of simulation. Since the future is essentially unknown, an imperfect prediction module is generally used to estimate what the future states of the system are likely to be. For this comparison, however, a *perfect* prediction module was implemented to evaluate the RL agent against the **most optimal version of the MPC adversary**.

Unlike the predictable and steady performance of the Expert, which relies on classical optimisation, the *TRLA* bases its decisions on sampling distributions. While this difference is evident at individual timesteps, their aggregate performance may converge when extended over time.

6.2.1. Objective Function Metrics

The performance of the *TRLA* will initially be validated over the long term by analysing results from the entire test. As specified in Section 3.3.1, the comparison will focus on metrics derived from the Expert's objective function (Equation 3.1). These metrics, identified as *Objective Metrics* in Section 5.4.2, will be examined for each component individually.

It is important to recall that the *test set* includes one out of every four days from an entire year, ensuring all seasons are represented equally (see Section 3.6). This arrangement places the summer solstice near day 43.

	C_{grid} [€]	p_{SoCDep} [-]	Aux_{TESS} [s]
Top RL Agent	1308.2 $\pm$ 0.5	**0.098 $\pm$ 0.002**	**0.0 $\pm$ 0.0**
Expert	**1258.0**	0.489	711.8

Table 6.1: Comparison of the best-performing RL agent (*TRLA*) and the Expert, evaluated using the the *test set* with *Objective Metrics* (O_M). The results for the RL agent are derived from 100 test runs with distinct random seed values. Note: Computed using Equation 3.4, the temporal dimension of Aux_{TESS} serves to quantify the cumulative duration and severity of TESS overcharge events.

The experimental outcomes for the *top RL agent*, as displayed in Table 6.1, were obtained following the same simulation and statistical analysis procedure described in Section 5.4.2, but utilising the *test set* instead of the *robust validation set*. The Expert's decision-making approach (Section 3.5) produces consistent outcomes with minimal random seed variance, making a single simulation's O_M adequate for comparison.

The results of Table 6.1 provide some interesting insights. The long-term behaviour of the *TRLA* proves to be considerably robust to the stochasticity of its decision-making (Section 4.4.2). While the Expert achieves superior grid exchange cost optimisation, this advantage comes with a significant trade-off in p_{SoCDep}. *The top RL agent*, trained using r_{margin} reward (Section 4.2.4), successfully maintains appropriate TESS SoC boundaries, avoiding any overcharging scenarios. Though these average metrics –with their confidence intervals– provide a useful summary of performance, they offer limited visibility into the underlying causes.

Figure 6.1 expands the interpretation of O_M from Table 6.1 by providing a temporal dimension, enabling further analysis. The Expert successfully anticipates charging needs and charges the EV precisely to the target SoC (SoC^*_{dep}) before departure. However, in a few occasions, the Expert's objective function can be minimised further by accepting the p_{SoCDep} penalty to lower C_{grid}. This is a clear example of the compromises involved in adjusting the weights of the objective function. While *TRLA* performs consistently, it lacks the precision of the Expert, leading to minor daily penalties.

Overcharging of the TESS by the Expert is confined to the warmer months, as depicted in Figure 6.1. This behaviour can be attributed to the brief two-day planning horizon coupled with periods of negative energy prices. Such circumstances result in short-term decision-making that increases immediate heat production without accounting for upcoming periods of abundant solar thermal generation and reduced heat demand.

The graph of daily grid exchange costs illustrates the *TRLA*'s delivery of **consistent outcomes** across the complete *test set*, closely aligning with the Expert's performance. To provide another quantitative index for daily cost comparison, the root mean squared error (RMSE) was calculated, resulting in a value of 3.03 €. This suggests the existence of a variety of near-optimal solutions, making the total grid cost at the end of the test set (Table 6.1) a more relevant metric.

6.2.2. Agent Behaviour

After observing the impact of the agents' decisions on the *Objective Metrics* (Section 6.2.1), the following analysis delves into the decisions themselves and their influence on key state variables.

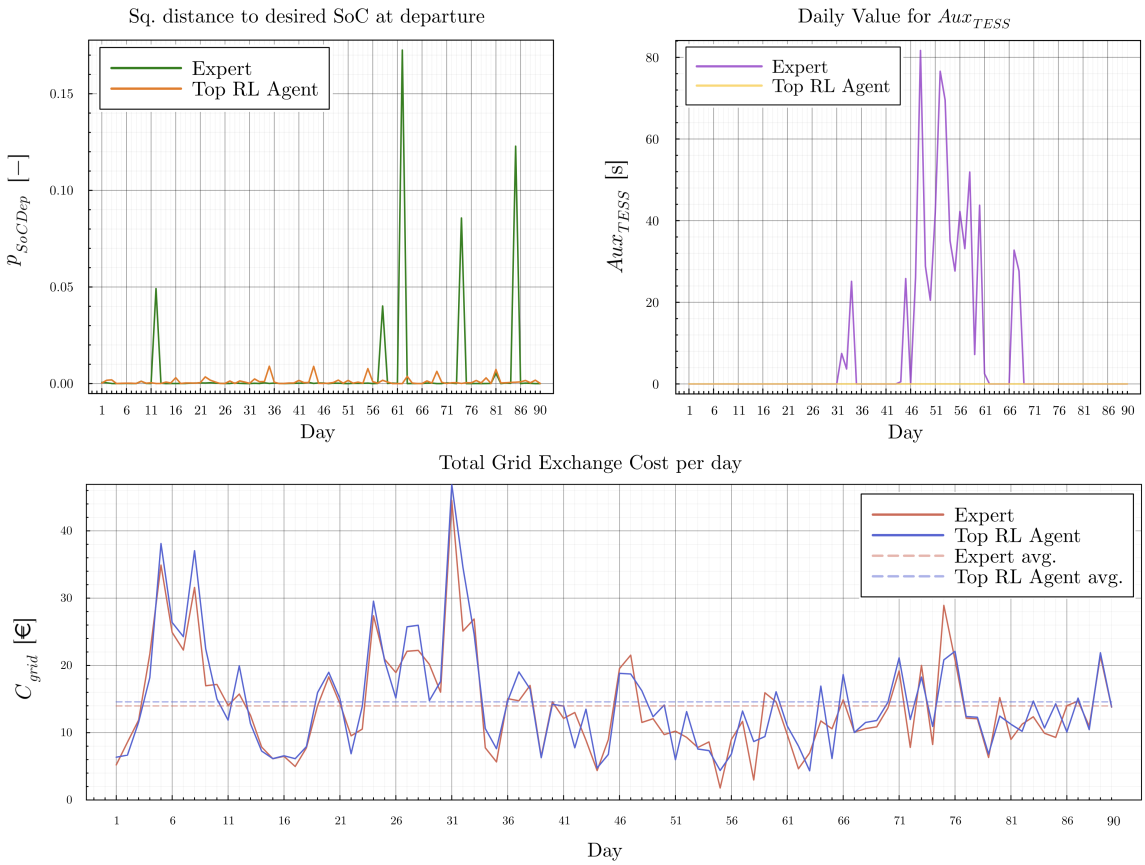

Figure 6.1: The *Objective Metrics* for the Expert and the top RL agent are displayed for the *test set* simulations. One representative sample is used from the *TRLA*. The top-left panel depicts the squared distance to the desired SoC at departure (p_{SoCDep}), with occasional deviations for the Expert and low, stable values for *TRLA*. The top-right panel presents daily Aux_{TESS} values, peaking between days 40 and 65. The bottom panel represents total daily grid exchange costs (C_{grid}), including average costs for both policies.

It is worth reminding that the *TRLA's* architecture is 3CW (Section 4.4.4), and therefore the standard deviation used for sampling will be constant, as it was optimised during the training phase. The optimal values were determined to be $\sigma_{\text{EV}} = 1.791$, $\sigma_{\text{BESS}} = 0.394$, and $\sigma^{\text{e}}_{\text{HP}} = 0.037$.

Electrical Power Balance

The agent decisions are all integrated within the electrical subsystem (Section 3.3.5), but this Section will focus on the power management strategies for the EV battery (P_{EV}) and BESS (P_{BESS}). The seasonal variations are shown in Figure 6.2, through data from day 1 (winter) and day 45 (summer) of simulation.

Day 1 is of particular interest because both agents start with identical state variables ($\text{SoC}_{\text{BESS}} = \text{SoC}_{\text{EV}} = 50\%$). Under these conditions, there is a similar strategic approach, as both prioritise EV battery charging to achieve the desired SoC ($SoC^{*}_{\text{dep}} \rightarrow \text{SoC}_{\text{EV}} = 85\%$) before potential departure. However, even with similar intentions, the specific implemented **decisions are quite distinct**. These variations propagate through time, creating divergent state variables and rendering timestep-resolution analysis ineffective. Even in this situation, the **similarity in long-term performance** metrics (Section 6.2.1) suggests the existence of multiple near optimal policies.

A closer analysis of Figure 6.2 reveals that the *TRLA* charges the EV battery when its SoC deviates significantly from $\text{SoC}_{\text{EV}} = 85\%$. The behaviour reveals a preference for avoiding p_{SoCDep} penalties over minimising grid exchange costs. On day 45, this method proves more cost-effective than on day

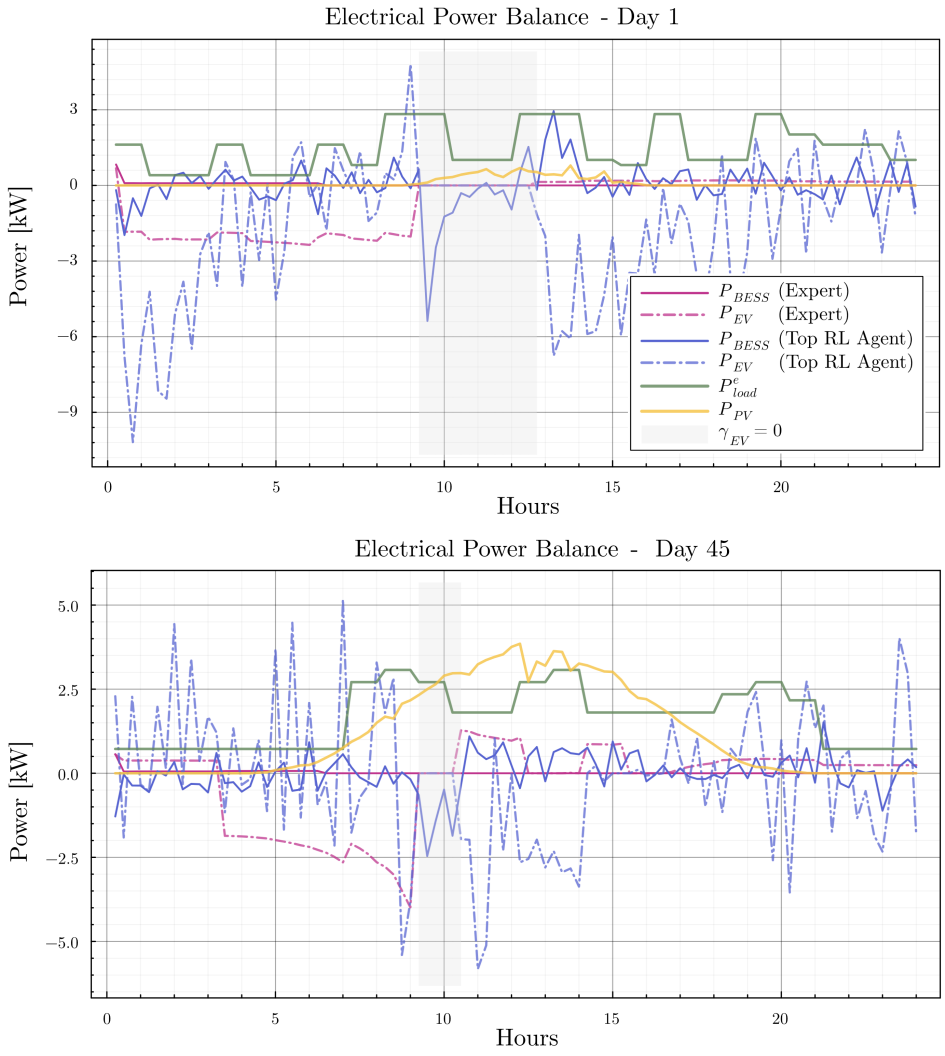

Figure 6.2: The electrical power balance is shown for day 1 and day 45 in the top and bottom graphs, respectively. The displayed components are those related more closely to the EV battery and BESS. Both the Expert and the *TRLA* are represented, with the grey-shaded regions denoting periods when the EV is absent.

1, as charging coincides with periods of high solar radiation. Once the desired SoC has been reached, the *TRLA* operates around that state, balancing the electrical load (P^{e}_{load}) or demanding power.

During the morning of day 45, the behaviour of the *TRLA* enables net energy sales to the grid, as can be seen in Figure 6.3. Further analysis of the image reveals that the Expert has a more conservative approach to grid exchange, tending to purchase less energy rather than selling accumulated reserves. The *TRLA's* output is oscillating, occasionally achieving a reduced cumulative daily cost but eventually matching or exceeding the Expert's cost due to poorly timed energy demands from the grid. This occurs, for example, in the final hours of day 46, when both cumulative costs rapidly converge. Such oscillations, characterised by pronounced power peaks in either direction, are particularly susceptible to sharp hourly electricity price changes, which lead to an overall higher cumulative cost in the long term, as evidenced in Table 6.1.

The Expert charges the EV to exactly $\text{SoC}_{\text{EV}} = 85\%$, considering both days 1 and 2 during its planning. In contrast to the more short-sighted approach by the *TRLA*, the Expert delays charging the EV after its return, slightly depleting its battery instead. The Expert avoids using the BESS, whereas the *TRLA* charges the BESS at night or during EV absence, using it to supply energy after the EV's

arrival and interact more significantly with the grid.

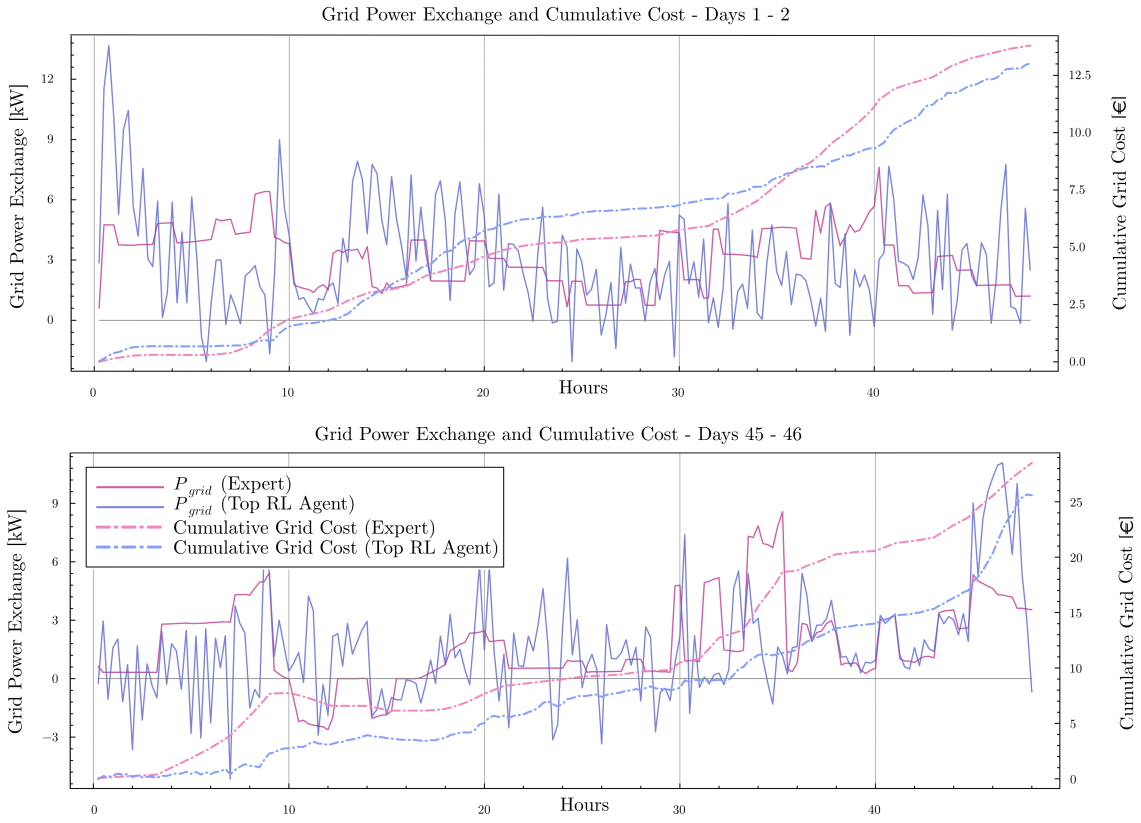

Figure 6.3: The figure presents the Grid Power Exchange and Cumulative Cost for the Expert and *TRLA* over 2 consecutive days. The top graph shows the data for the winter period (days 1-2), while the bottom graph covers the summer period (days 45-46). Negative P_{grid} values correspond to selling energy to the grid.

Thermal Power Balance

The heat pump (HP) serves as the primary link between the electrical and thermal subsystems. Its electrical input (P^{e}_{HP}), which translates directly to thermal power (P^{th}_{HP}), is the remaining decision to analyse. Figure 6.4 illustrates the entire thermal balance (described in Section 3.3.5) across two representative days-—a winter day (day 1) and a summer day (day 45).

The simulation on day 1 begins with $SoC_{\text{TESS}} = 40\%$ for both the Expert and the *TRLA*, yet it becomes evident their strategies diverge. The Expert prioritises filling the TESS, delivering a P^{th}_{HP} that exceeds the thermal load demand (P^{th}_{load}). In contrast, the *TRLA* opts to deplete the TESS, activating the heat pump only when forced to do so.

Day 45 begins with the *TRLA* and the Expert at different states of charge for the TESS: 20% and 95%, respectively. Despite the high SoC, the Expert engages the heat pump to recover any lost energy, finishing the day at 95%. By contrast, the *TRLA* relies on available solar radiation and barely activates the heat pump. While this approach successfully avoids Aux_{TESS} penalties, it incurs the risk of encountering high electricity prices with a depleted TESS.

States of Charge

The state of charge (SoC) of the EV battery, BESS, and TESS (see Section 3.2) must be examined across the entire simulation to better understand how the agent's decisions influence the system. Figure 6.5 will guide this discussion, as it illustrates the performance of the agents during the 90-day simulation, which is representative of the dynamics of a complete year.

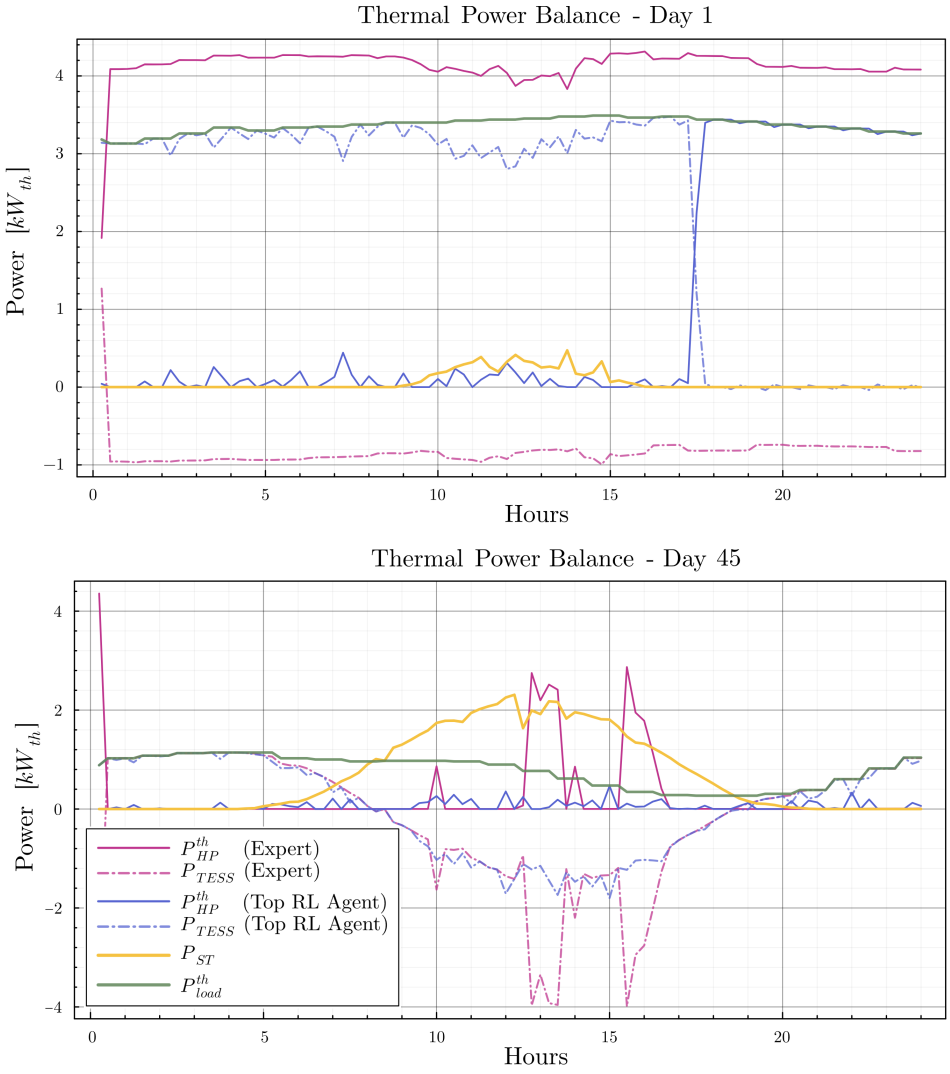

Figure 6.4: The thermal power balance is shown for day 1 and day 45 in the top and bottom graphs, respectively. The graphs illustrate the distinct strategies employed by the Expert and the *top RL agent* in managing the thermal energy subsystem, since they can determine P^{th}_{HP}.

The patterns observed in Figure 6.5 concerning the EV battery (SoC_{EV}) are similar for both the Expert and the *TRLA*. However, the Expert primarily charges the battery to the required SoC, while the *TRLA* actively adjusts around this value, significantly engaging in the MCES's electrical power balance.

The Expert's strategy for the BESS is characterised by abrupt charging spikes during low electricity price periods, followed by quick discharges or extended inactivity. This approach makes use of a significant fraction of the total energy capacity. In contrast, the *TRLA* uses only about 20% of the total capacity on average, following shorter cycles. The BESS is generally charged by the *TRLA* during the EV's absence, aligning with high solar radiation hours, and is slowly depleted during the remaining hours.

The TESS displays the most notable differences in behaviour, as evident from Figure 6.4. The *TRLA* relies heavily on the TESS to meet thermal demands, which keeps it near to its minimum SoC during most of the simulation, only reaching half capacity on warmer days when thermal loads are low, and solar energy is abundant. The Expert, however, adopts a more conservative strategy, maintaining the TESS close to full capacity. This approach minimises the need for the heat pump during high electricity price periods but occasionally leads to overcharging events.

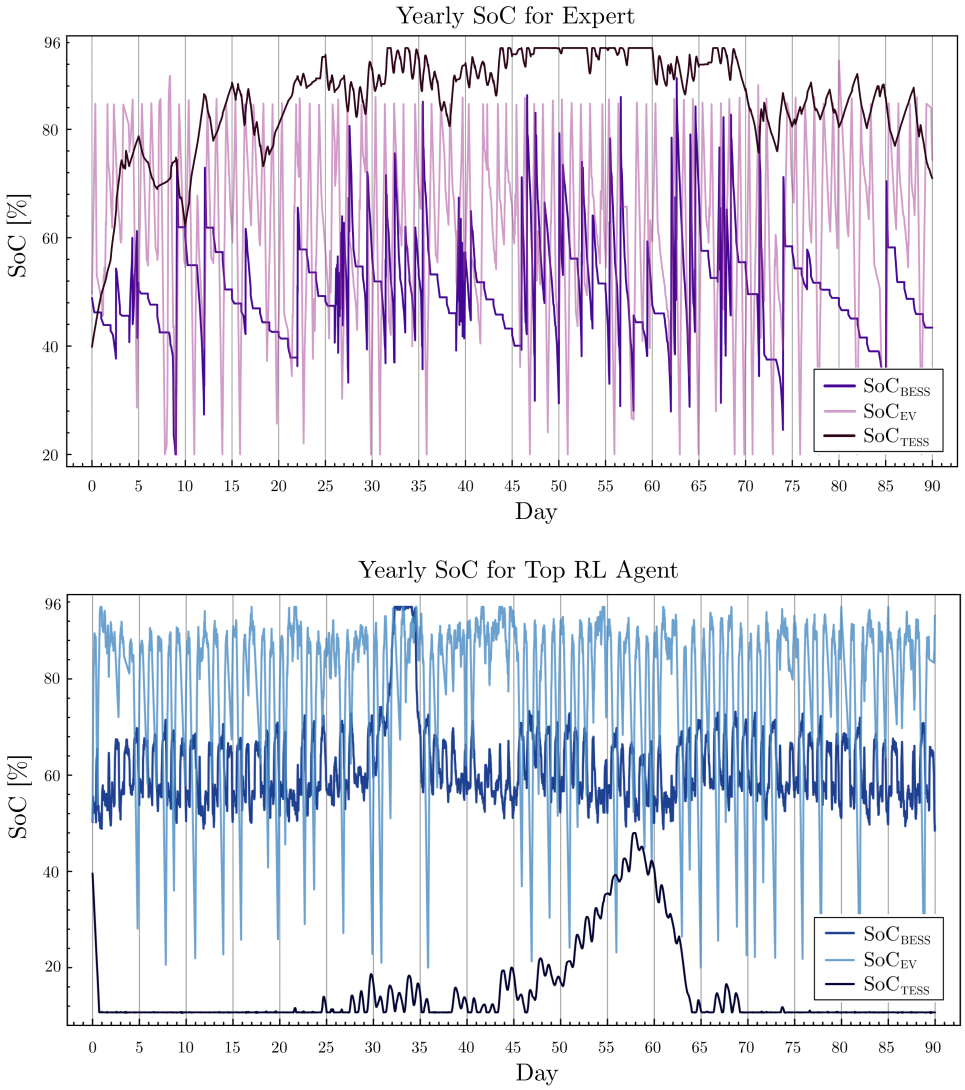

Figure 6.5: Visualisation of the states of charge for the BESS, EV battery and TESS over a 90-day simulation (*test set*, see Section 3.6). The Expert's behaviour is depicted in the upper panel, while the *TRLA* is shown below. Seasonal variability across the year is represented. Clear distinctions may be observed in SoC_{TESS}, as well as the narrower charge range utilised by the *TRLA's* BESS.

6.2.3. Safety Performance

Adhering to safety constraints is essential for decision-making agents. Section 3.4.1 provided a comprehensive discussion of the safety concerns and the measures designed to address them. A key fact was that *initial projections* do not pose a threat to the real MCES, whereas high *operational projections* could result in significant issues –if deeper protective mechanisms are absent from system components. To address this, the *safe projection* was introduced. Together with the learned safe behaviours, the *safe projection* enhances the adherence of RL agents to safety requirements.

The results presented in Table 6.2 present the sum of all operational projections observed during the evaluation on the *test set*. The experimental outcomes for the *TRLA* were derived from 100 simulations employing *safe projection*, only changing the random seed value. The 95% confidence intervals were computed using the statistical methods outlined in Section 5.4.2.

The Expert incurs more operational projections than *TRLA*, but considering the *test set* is 90 days long, the values for both agents are insignificant. For instance, the Expert misallocated $0.356\ kW_{th}$,

	$\xi^{\text{SoC}}_{\text{BESS}}$ $[-]$	$\xi^{\text{SoC}}_{\text{EV}}$ $[-]$	$\xi^{\text{SoC}}_{\text{TESS}}$ $[-]$	$\xi^{\text{p}}_{\text{grid}}$ $[kW]$	$\xi^{\text{p}}_{\text{TESS}}$ $[kW_{th}]$
Top RL Agent	**0.0** $\pm$ **0.0**	**0.155** $\pm$ **0.011**	**0.0** $\pm$ **0.0**	0.0 $\pm$ 0.0	**0.0** $\pm$ **0.0**
Expert	0.040	0.193	0.791	0.0	0.356

Table 6.2: Comparison of the cumulative operational projections incurred by the *top RL agent* and the Expert, when evaluated using the *test set*. The *TRLA* employed *safe projection* (Section 3.4.1). The results for the RL agent are derived from 100 simulations with distinct random seed values. Definitions of operational projection terms are detailed in Section 3.3.1.

which amounts to $4\ W_{th}$ on average per day, for a thermal energy storage capacity of $200\ kWh$. This minor discrepancy can be attributed to the feasibility tolerance of the solver, since the **solver minimises the norm** (typically the Euclidean norm) of constraint violations across all system constraints. This is a necessary measure for computational efficiency and feasibility, as perfectly satisfying every constraint in complex optimisation problems is often unattainable within the given time and computational resources.

The *TRLA* only incurred operational projections of the SoC_{EV}, which are closely linked to the penalty p_{SoCDep} (Equation 3.3). The energy needed by the electric vehicle outside the MCES is unknown to the agent and changes daily. Such a projection arises only when the EV leaves the MCES without achieving its desired SoC (e.g. $SoC_{\text{EV}} = 85\%$), and the required energy for driving reduces it below the minimum allowed threshold (e.g. $SoC^{min}_{\text{EV}} = 20\%$). From the results of *TRLA*, it is evident that these projections are minor, averaging a daily infraction of the SoC^{min}_{EV} constraint by 0.17%.

If the reader is interested in the *initial* projections (performed by the safety layer) required to ensure the *TRLA's* negligible *operational* projections, see Figure A.2.

6.2.4. Validation

The results analysed in Section 6.2 demonstrate that the *TRLA* is **safe to use** and its performance has been **validated across multiple dimensions**.

Table 6.1 demonstrates the *TRLA's* capacity to balance conflicting long-term objectives. The temporal trends in Figure 6.1 further highlight its effectiveness in minimising grid exchange costs and maintaining near-optimal EV SoC.

The *TRLA's* performance in managing the electrical subsystem shows a close alignment with the Expert's behaviour. Both agents prioritise achieving the desired SoC for the EV battery before departure, and balance the electrical load effectively across different seasons. Despite some differences in execution, such as the timing and sources of charging, the *TRLA* achieves comparable outcomes to the Expert in terms of energy balance and grid interaction.

In the thermal subsystem, the *TRLA* adopts strategies that differ considerably from the Expert's. Nevertheless, the *TRLA* manages the thermal resources successfully, respects the TESS constraints, and avoids significant increases in grid energy costs.

Thanks to the *safe projection* mechanism, the *TRLA* exhibits negligible *operational projections* across all critical components (Table 6.2). Furthermore, these minor infractions are well within tolerable bounds, preserving the overall integrity of the MCES.

6.3. Conclusions

In this final section, the answer to the central research question is first presented, and then developed through the findings of three complementary sub-questions, which were chosen to investigate the research gaps identified in the literature. Section 6.2.4 can be considered a relevant part of this conclusion, as it validates the RL agent's performance.

Main Research Question

In the management of multicarrier urban energy systems, to what extent can practitioners without expertise in optimal control theory and physics-based modelling leverage reinforcement learning to create a computationally efficient alternative to model predictive control, while maintaining comparable safety and performance?

Research has shown that RL can be used effectively by practitioners who lack extensive knowledge of optimal control theory, offering a viable alternative to MPC. Furthermore, the development of successful and safe agents can be made more accessible through existing building and system dynamics simulators, which simplify the physical modelling process. Additionally, a structured approach to hyperparameter optimisation can be effectively replicated across different MCESs, reducing implementation costs.

Although training RL agents requires significant computational resources, their inference during deployment has a negligible cost, resulting in a significantly more efficient approach than traditional MPC, even with suboptimal implementations.

Sub-Research Question 1

How do progressively advanced policy update algorithms (from basic advantage estimation to GAE to trust region constraints) contribute to achieving near optimal performance?

The findings demonstrate that more sophisticated algorithms enhance robustness in the optimisation process, increasing the likelihood that diverse hyperparameter configurations will converge towards superior policy performance.

While algorithms such as PPO need additional computational resources (e.g., for storing the previous version of the policy) and require more processing time per hyperparameter configuration, their inherent training stability significantly reduces the number of samples required to reach a near-optimal policy. This stability-sample trade-off, when implemented efficiently, typically results in reduced overall computational demands.

This conclusion is particularly evident when contrasted with simpler approaches such as VPG-C. As illustrated in Figure 5.4, whilst VPG-C achieved comparable peak performance, this outcome represented an extreme outlier configuration, despite the use of sophisticated hyperparameter optimisation methods such as Bayesian Optimisation and Hyperband (BOHB) to navigate the search space efficiently. These findings underscore the value of advanced policy update algorithms in establishing more consistent pathways to optimal performance.

Sub-Research Question 2

What is the impact of DNN architecture sophistication, temporal feature engineering, and reward component shaping for RL algorithms in achieving performance that is close to optimal?

Impact of DNN Architecture Sophistication:

The considerable variance in performance across hyperparameter configurations, amplified by random weight initialisation, highlighted the critical importance of searching for more robust DNN architectures.

The addition of more sophisticated DNN architectures, such as Residual or Three-Branched structures, to the hyperparameter search space demonstrated significant performance and robustness improvement with respect to the CW design. However, the testing of 12 distinct architectures led to a substantial increase in search space dimensions, revealing a crucial trade-off.

This trade-off is centred on balancing two key aspects: the provision of better and more abundant local optima through new paths for information abstraction (offered by distinct DNN designs), against

both the expansion of the hyperparameter search space and the cost of implementing new architectures. The DCW configuration demonstrated both robustness and performance that matched top-performing architectures, showing that effective results could be achieved by focusing on varying the network depth rather than altering the structure more substantially. These findings suggest a practical approach: begin with depth exploration, and only gradually introduce distinct DNN architectures if performance improvements plateau.

Impact of Temporal Feature Engineering:

The investigation revealed that simpler feature representations outperform complex ones. Feature vector configurations using only the previous timestep's state information achieved optimal performance, while the expansion of the feature space, guided by cross-correlation analysis between state variables and Expert decisions, decreased performance.

However, this addition of past data (e.g., 4-hour-old or 1-day-old states) kept robustness at a similar level, suggesting that the DNN could not effectively utilise these extended temporal patterns. The simpler approach, focusing on recent temporal information, reduced overfitting and improved generalisation, as shown by better validation set performance.

Impact of Reward Component Shaping:

Shaping reward components has shown significant benefits in achieving near-optimal performance. This approach reduced dependence on trial-and-error methods and minimised reward overfitting risks.

By treating the final reward as a hyperparameter and aligning its components with the agent's multiple objectives, the testing of eight distinct reward versions led to notable improvements in both performance and robustness. Within the shaped components, the *margin reward* proved most effective. By constraining the set of possible policies into a safer subset, it guided the optimisation of a primary objective –the respect of safety bounds– and increased robustness.

Sub-Research Question 3

How do the RL and MPC approaches compare in terms of operational safety, energy storage use, and economic performance?

The *top RL agent* (*TRLA*) has been shown to operate safely through a *safe projection* mechanism, which resulted in negligible operational projections across all critical components and even outperformed the Expert in maintaining certain constraints, particularly the SoC upper bound of the TESS.

With respect to energy storage utilisation, significant differences have been observed amongst the behaviour of the agents. While both manage the EV battery similarly, the *TRLA* takes a more active role in the MCES's power balance as it makes dynamic adjustments around the required SoC. The BESS management strategies differ notably, with the *TRLA* operating within a narrower range (about 20% of total capacity) and employing shorter cycles compared to the Expert's approach of deep charging and discharging. The most pronounced distinction lies in the TESS operation, where the *TRLA* maintains lower storage levels to avoid any overcharge, contrasting with the Expert's conservative approach of keeping the storage near full capacity.

The economic performance analysis shows that both agents achieved comparable results, with the *TRLA*'s daily grid exchange costs being only 4% higher than the Expert's. Notably, the *TRLA* maintained consistent performance throughout the test set.

Other Relevant Findings:

The development of a comprehensive performance metric helped identify meaningful improvements among highly variable results. This metric's ability to identify outliers in the performance distribution was particularly useful, as these agents often presented the most promising behaviours. When examining the TESS SoC throughout the year, the *TRLA* demonstrated superior long-term planning capabilities

through its implicit understanding of the future, whereas the Expert could only optimise within its 2-day planning horizon.

Though incurring an implementation cost, the synthetic expansion of training data enabled to create a comprehensive validation set that identified well-generalising agents. The practice of training 100 agents with identical hyperparameter configurations added robustness to the selection process, and helped increase maximum performance.

6.3.1. Further Work

The research gaps that were not addressed during this study, as discussed in Section 2.10, highlight promising opportunities for future work. This section, however, focuses on smaller steps forward that align closely with the contributions of the thesis.

As RL has proven to be an efficient alternative to an MPC framework, various interesting paths exist regarding the MCES in the Green Village, which was central to this thesis. One such option is the incorporation of a data-driven approach to support the Expert MPC, which was already considered within the research of Darío et al. [55].

The developed RL agent could become a backup policy to be activated when the Expert struggles to solve the nonconvex optimisation problem it is faced with. Another possibility is that the RL agent could support the Expert solver by supplying a warm start, while requiring negligible additional computational resources. Another option is to retrain the RL agent –with the same pipeline described in this work– to operate at the minute or second level, optimising performance at resolutions where MPC becomes computationally impractical.

Imitation learning could be employed to initialise the RL agents before training them in the MCES virtual environment. This approach might produce agents whose behaviour is closer to that of the Expert, creating a more predictable backup policy. However, imitation learning without an additional RL training phase is not encouraged, as there are no guarantees of good performance in situations not addressed by the imitation data [28].

To better utilise extended temporal patterns in the feature vector, alternative deep learning architectures could be tested, taking into account the relationship between their number of parameters and their effectiveness.

An important area for further investigation involves improving the **safety guarantees** of the RL agent. This is particularly critical in scenarios where weather conditions deviate significantly from those observed in the training data, or when the real-world MCES is expected to behave differently from the simulated environment.

One possible approach to safety is the method proposed by Wabersich and Zeilinger [62], where an RL agent's actions are deemed safe only if a trajectory can be identified that avoids violating constraints and guides the MCES toward a safe state. If no such feasible solution exists, a backup safe solution is deployed instead. This approach extends the *safe projection* employed by this thesis, using knowledge of the system's dynamics to predict the long-term effects of the agent's decisions, rather than limiting the analysis to a single timestep.

An alternative pathway to safety, as mentioned in Section 2.9.6, is the method recently developed by Ceusters et al. [11]. This approach integrates hard constraint satisfaction with policy optimisation and, in principle, can be applied to any RL formulation.

References

[1] Joshua Achiam. “Spinning Up in Deep Reinforcement Learning”. In: *OpenAI* (2018).

[2] Kari Alanne and Seppo Sierla. “An overview of machine learning applications for smart buildings”. In: *Sustainable Cities and Society* 76 (Jan. 2022). ISSN: 22106707. DOI: 10.1016/j.scs.2021.103445.

[3] Joel Alpízar-Castillo, Laura M. Ramírez-Elizondo, and Pavol Bauer. “Modelling and evaluating different multi-carrier energy system configurations for a Dutch house”. In: *Applied Energy* 364 (2024), p. 123197. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2024.123197. URL: https://www.sciencedirect.com/science/article/pii/S0306261924005804.

[4] Marcin Andrychowicz et al. “What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study”. In: *CoRR* abs/2006.05990 (2020). arXiv: 2006.05990. URL: https://arxiv.org/abs/2006.05990.

[5] Javier Arroyo et al. “Reinforced model predictive control (RL-MPC) for building energy management”. In: *Applied Energy* 309 (Mar. 2022). ISSN: 03062619. DOI: 10.1016/j.apenergy.2021.118346.

[6] Fredrik Bagge Carlson. “Hyperopt. jl: Hyperparameter optimization in Julia.” In: (2018). URL: https://lup.lub.lu.se/search/publication/6ec19989-9b30-448c-be5e-bae4c4257c7b.

[7] Chayan Banerjee et al. “A Survey on Physics Informed Reinforcement Learning: Review and Open Problems”. In: *arXiv preprint arXiv:2309.01909* (Sept. 2023).

[8] Jeff Bezanson et al. “Julia: A fresh approach to numerical computing”. In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671.

[9] Serena Booth et al. “The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 5. 2023, pp. 5920–5929.

[10] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG]. URL: https://arxiv.org/abs/1606.01540.

[11] Glenn Ceusters et al. “An adaptive safety layer with hard constraints for safe reinforcement learning in multi-energy management systems”. In: *Sustainable Energy, Grids and Networks* 36 (Dec. 2023), p. 101202. ISSN: 23524677. DOI: 10.1016/j.segan.2023.101202.

[12] Glenn Ceusters et al. “Safe reinforcement learning for multi-energy management systems with known constraint functions”. In: *Energy and AI* 12 (2023), pp. 2666–5468. DOI: 10.1016/j.egyai.2022.100227. URL: https://doi.org/10.1016/j.egyai.2022.100227.

[13] Bingqing Chen, Zicheng Cai, and Mario Bergés. “Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy”. In: *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 2019, pp. 316–325.

[14] ElaadNL. *Home - Elaad NL*. 2022. URL: https://platform.elaad.io/ (visited on 2024).

[15] Lasse Espeholt et al. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*. 2018. arXiv: 1802.01561 [cs.LG]. URL: https://arxiv.org/abs/1802.01561.

[16] European Power Exchange (EPEX). *EPEX Spot*. 2022. URL: https://www.epexspot.com/en/ (visited on 2024).

[17] Stefan Falkner, Aaron Klein, and Frank Hutter. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*. 2018. arXiv: 1807.01774 [cs.LG]. URL: https://arxiv.org/abs/1807.01774.

[18] Qiming Fu et al. “Applications of reinforcement learning for building energy efficiency control: A review”. In: *Journal of Building Engineering* 50 (2022), p. 104165. ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2022.104165. URL: https://www.sciencedirect.com/science/article/pii/S2352710222001784.

[19] Satchel Grant. *PyTorch-A2C: General implementation of Advantage Actor Critic using Pytorch*. https://github.com/grantsrb/PyTorch-A2C. 2020.

[20] Murilo Gustineli. *A survey on recently proposed activation functions for Deep Learning*. 2022. arXiv: 2204.02921 [cs.LG]. URL: https://arxiv.org/abs/2204.02921.

[21] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[22] Peter Henderson et al. *Deep Reinforcement Learning that Matters*. 2019. arXiv: 1709.06560 [cs.LG]. URL: https://arxiv.org/abs/1709.06560.

[23] hermesdt. *Reinforcement Learning: A2C Implementation*. https://github.com/hermesdt/reinforcement-learning/tree/eb69484bb6d5a415633eb6e1fc07e12aa193cbb0/a2c. 2019.

[24] Shengyi Huang et al. “The 37 Implementation Details of Proximal Policy Optimization”. In: *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. 2022. URL: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[25] Michael Innes. *Don’t Unroll Adjoint: Differentiating SSA-Form Programs*. 2019. arXiv: 1810.07951 [cs.PL]. URL: https://arxiv.org/abs/1810.07951.

[26] AI ML – Artificial Intelligence and Machine Learning. *Most Used Activation Functions in Neural Networks*. https://ai-artificial-intelligence.webyes.com.br/most-used-activation-functions-in-neural-networks/. Accessed: 2024-07-17.

[27] Hyuna Kang et al. “Reinforcement learning-based optimal scheduling model of battery energy storage system at the building level”. In: *Renewable and Sustainable Energy Reviews* 190 (2024), p. 114054.

[28] Benjamin Karg and Sergio Lucia. “Reinforced approximate robust nonlinear model predictive control”. In: Institute of Electrical and Electronics Engineers Inc., June 2021, pp. 149–156. ISBN: 9781665403306. DOI: 10.1109/PC52310.2021.9447448.

[29] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[30] W. Bradley Knox et al. “Reward (Mis)design for autonomous driving”. In: *Artificial Intelligence* 316 (2023), p. 103829. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2022.103829. URL: https://www.sciencedirect.com/science/article/pii/S0004370222001692.

[31] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[32] Aristotelis Lazaridis. “Deep Reinforcement Learning: A State-of-the-Art Walkthrough”. In: *Journal of Artificial Intelligence Research* 69 (2020), pp. 1421–1471. DOI: 10.1613/jair.1.12412.

[33] Matthias Lehmann. *The Definitive Guide to Policy Gradients in Deep Reinforcement Learning: Theory, Algorithms and Implementations*. 2024. arXiv: 2401.13662 [cs.LG]. URL: https://arxiv.org/abs/2401.13662.

[34] Tianyu Liu. “Coulombic Efficiency, Energy Efficiency and Effective Capacitance”. In: (2019). URL: https://www.researchgate.net/publication/330238104_Coulombic_Efficiency_Energy_Efficiency_and_Effective_Capacitance.

[35] Zhou Lu et al. *The Expressive Power of Neural Networks: A View from the Width*. 2017. arXiv: 1709.02540 [cs.LG]. URL: https://arxiv.org/abs/1709.02540.

[36] Miles Lubin et al. “JuMP 1.0: Recent improvements to a modeling language for mathematical optimization”. In: *Mathematical Programming Computation* (2023). DOI: 10.1007/s12532-023-00239-3.

[37] Aditya Krishna Menon et al. “Can gradient clipping mitigate label noise?” In: *International Conference on Learning Representations*. 2020.

[38] Diganta Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. 2020. arXiv: 1908.08681 [cs.LG]. URL: https://arxiv.org/abs/1908.08681.

[39] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG]. URL: https://arxiv.org/abs/1602.01783.

[40] Zoltan Nagy et al. “Ten questions concerning reinforcement learning for building energy management”. In: *Building and Environment* 241 (2023), p. 110435.

[41] ATD Perera and Parameswaran Kamalaruban. “Applications of reinforcement learning in energy systems”. In: *Renewable and Sustainable Energy Reviews* 137 (2021), p. 110618.

[42] George Philipp, Dawn Song, and Jaime G. Carbonell. *The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions*. 2018. arXiv: 1712.05577 [cs.LG]. URL: https://arxiv.org/abs/1712.05577.

[43] Watcharakorn Pinthurat, Tossaporn Surinkaew, and Branislav Hredzak. “An overview of reinforcement learning-based approaches for smart home energy management systems with energy storages”. In: *Renewable and Sustainable Energy Reviews* 202 (2024), p. 114648.

[44] Aske Plaat, Walter Kosters, and Mike Preuss. “High-accuracy model-based reinforcement learning, a survey”. In: *Artificial Intelligence Review* 56 (9 Sept. 2023), pp. 9541–9573. ISSN: 15737462. DOI: 10.1007/s10462-022-10335-w.

[45] Warren B Powell. *Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions*. John Wiley & Sons, 2022.

[46] Warren B Powell. *What is Reinforcement Learning*. 2022. URL: https://castle.princeton.edu/what-is-rl/ (visited on 01/14/2024).

[47] Zhaoming Qin et al. “Does Explicit Prediction Matter in Deep Reinforcement Learning-Based Energy Management?” In: *2021 IEEE International Conference on Energy Internet (ICEI)*. IEEE. 2021, pp. 13–19.

[48] Christopher Rackauckas. *Engineering Trade-Offs in Automatic Differentiation: from TensorFlow and PyTorch to Jax and Julia*. https://www.stochasticlifestyle.com/engineering-trade-offs-in-automatic-differentiation-from-tensorflow-and-pytorch-to-jax-and-julia/. Accessed: 2024-06-14. 2021.

[49] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE]. URL: https://arxiv.org/abs/1710.05941.

[50] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).

[51] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[52] John Schulman et al. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477 [cs.LG]. URL: https://arxiv.org/abs/1502.05477.

[53] Ayas Shaqour and Aya Hagishima. “Systematic Review on Deep Reinforcement Learning-Based Energy Management for Different Building Types”. In: *Energies* 15 (22 Nov. 2022), p. 8663. ISSN: 1996-1073. DOI: 10.3390/en15228663.

[54] Jingnan Shi. *Automatic Differentiation: Forward and Reverse*. https://jingnanshi.com/blog/autodiff.html. Accessed: 2024-08-14. 2023.

[55] Darío Slaifstein et al. “Aging-aware Energy Management for Residential Multi-Carrier Energy”. In: (2024).

[56] Arno Smets et al. *Solar Energy: The physics and engineering of photovoltaic conversion, technologies and systems*. Bloomsbury Publishing, 2016.

[57] Phillip Stoffel et al. “Evaluation of advanced control strategies for building energy systems”. In: *Energy and Buildings* 280 (2023), p. 112709.

[58] Tong Su et al. “A review of safe reinforcement learning methods for modern power systems”. In: *arXiv preprint arXiv:2407.00304* (2024).

[59] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. ISBN: 0262352702.

[60] Richard S. Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS’99. Denver, CO: MIT Press, 1999, pp. 1057–1063.

[61] Jun Tian and other contributors. *ReinforcementLearning.jl: A Reinforcement Learning Package for the Julia Programming Language*. 2020. URL: https://github.com/JuliaReinforcementLearning/ReinforcementLearning.jl.

[62] Kim P Wabersich and Melanie N Zeilinger. “Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning”. In: *arXiv preprint arXiv:1812.05506* (2018).

[63] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106 (2006), pp. 25–57.

[64] Jiayi Weng et al. “Tianshou: A Highly Modularized Deep Reinforcement Learning Library”. In: *Journal of Machine Learning Research* 23.267 (2022), pp. 1–6. URL: http://jmlr.org/papers/v23/21-1127.html.

[65] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8 (1992), pp. 229–256.

[66] Hao Yu, Vivian WY Tam, and Xiaoxiao Xu. “A systematic review of reinforcement learning application in building energy-related occupant behavior simulation”. In: *Energy and Buildings* (2024), p. 114189.

[67] Jingzhao Zhang et al. *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. 2020. arXiv: 1905.11881 [math.OC]. URL: https://arxiv.org/abs/1905.11881.

[68] Juntang Zhuang et al. “AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 18795–18806. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/d9d4f495e875a2e075a1a4a6e1b9770f-Paper.pdf.

Code Repository

The complete source code employed to address the research objectives is accessible at:
https://github.com/Victor-Andres-RdeTrio/RL4MCES

A

Safety, Data Augmentation and Exploding Gradients

A.1. Exploding Gradients

Typically, the problem of exploding gradients is understood as an exponential growth in Jacobians through backpropagation, which complicates training by causing the step size to be excessively large for some parameters and too small for others. However, they offer a more general definition called the ”*gradient scale coefficient*” [42]. Although an exploration of the definition will not be provided here, as it is not directly related to this thesis, further reading is recommended for those interested.

The parts of the research by Philipp et al.[42] that are of interest are their comments on how to prevent gradients from exploding or simply how to prevent training from being disturbed by the explosion. To address the different step sizes that would be ideal for each layer (if the gradient scale coefficient grows with depth), **optimisation algorithms** such as Adam, RMSprop or their successors are recommended. Normalisation layers enables a compromise between the representational power of the network and the increase in gradient magnitude [42]. The use of an orthogonal initial state to the network's weights can be very useful (see Section 4.4.3), although there are other effective initialisation approaches that also claim to reduce gradient growth.

On the path to prevent the explosion of gradients one can find the idea of large entropy reduction, this may cause a *collapsing domain* [42], where the latent space in which data points can be represented is severely reduced. The collapse of the domain may bring about pseudo-linearity, a scenario where the nonlinearities separating linear layers can be approximated by a linear function. A representative example of this situation is shown by a ReLU activation that mostly gets fed inputs between 0 and 0.5, which could be easily approximated by a line with slope 1, as can be seen on the left of Figure A.1. As it is known, any DNN without nonlinearities has the same capacity to represent data as a network of a single layer [42], which defeats the purpose of using a deep neural network in the first place. **Avoiding pseudo-linearity is thus crucial**, the previously mentioned idea of normalisation layers can be useful, as well as relying on activation functions that are harder to approximate linearly.

A.2. Safety Projection Model

This section presents a comprehensive description of the safety projection optimisation problem. To ensure good decision projection capabilities, the model incorporates most of the environmental state space and transition functions employed during the RL training phase. Therefore, this section is useful for completing the understanding of the system presented in Chapter 3.

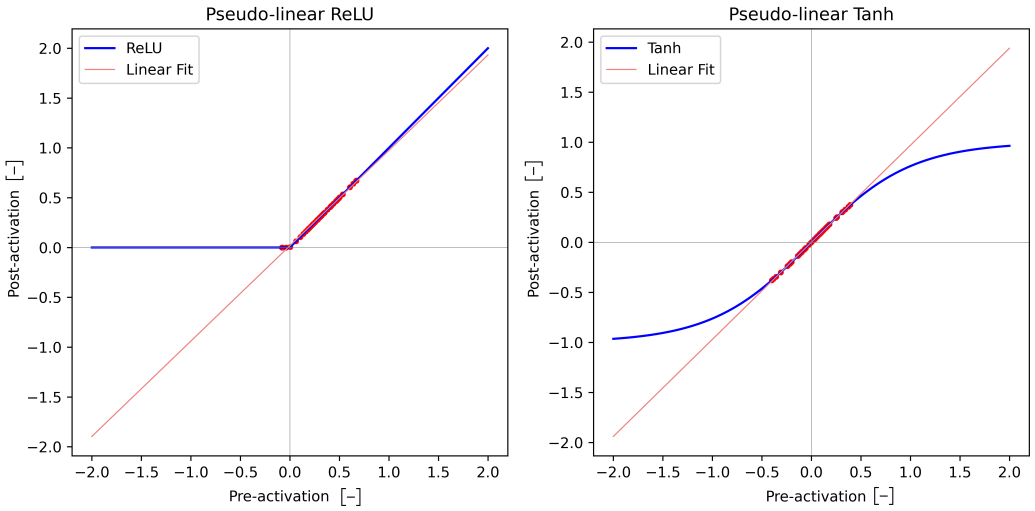

Figure A.1: Example of pseudo-linearity in ReLU and Tanh activation functions. The pre-activation values are sampled from a Gaussian with $\mu_{ReLU} = 0.3$ and $\mu_{Tanh} = 0$ and both with $\sigma = 0.2$. The figures have been recreated from examples in the paper by Philipp et al. [42].

The problem can be described more precisely as follows. A nonlinear program with 15 variables, 10 parameters (externally defined variables), minimising a quadratic objective and subject to 7 equality constraints (4 nonlinear, 3 linear) and 20 bound constraints.

A.2.1. Objective Function

$$\begin{aligned} \min \quad & (P_{\text{EV}} - \varrho P_{\text{EV}})^2 + (P_{\text{BESS}} - \varrho P_{\text{BESS}})^2 + (P^{\text{e}}_{\text{HP}} - \varrho P^{\text{e}}_{\text{HP}})^2 + \\ & 10^7 \cdot (\epsilon^2_{\text{grid}} + \epsilon^2_{\text{TESS}} + \epsilon^2_{\text{SoC}_{\text{TESS}}}) \end{aligned}$$

A.2.2. Equality constraints:

Grid Power Balance:

If $P_{\text{grid}} \geq 0.0$:

$$(P_{\text{grid}} \cdot \eta_{\text{grid}} + \epsilon_{\text{grid}}) - (P^{\text{e}}_{\text{HP}} - \gamma_{\text{EV}} \cdot P_{\text{EV}} - P_{\text{BESS}} + P^{\text{e}}_{\text{load}} - P_{\text{PV}}) = 0$$

Else:

$$\left(P_{\text{grid}} \cdot \frac{1}{\eta_{\text{grid}}} + \epsilon_{\text{grid}}\right) - (P^{\text{e}}_{\text{HP}} - \gamma_{\text{EV}} \cdot P_{\text{EV}} - P_{\text{BESS}} + P^{\text{e}}_{\text{load}} - P_{\text{PV}}) = 0$$

The use of the conditional *ifelse* statements makes the constraints nonlinear.

BESS Power Balance:

If $i_{\text{BESS}} \geq 0.0$:

$$P_{\text{BESS}} - \left(N_{\text{s, BESS}} \cdot N_{\text{p, BESS}} \cdot v_{\text{BESS}} \cdot i_{\text{BESS}} \cdot \eta_{\text{c,BESS}} \cdot \frac{1}{1000}\right) = 0$$

Else:

$$P_{\text{BESS}} - \left(N_{\text{s, BESS}} \cdot N_{\text{p, BESS}} \cdot v_{\text{BESS}} \cdot i_{\text{BESS}} \cdot \frac{1}{\eta_{\text{c,BESS}}} \cdot \frac{1}{1000}\right) = 0$$

EV Power Balance:

If $i_{\text{EV}} \geq 0.0$:

$$P_{\text{EV}} - \left(N_{\text{s, EV}} \cdot N_{\text{p, EV}} \cdot v_{\text{EV}} \cdot i_{\text{EV}} \cdot \eta_{\text{c,EV}} \cdot \frac{1}{1000} \right) = 0$$

Else:

$$P_{\text{EV}} - \left(N_{\text{s, EV}} \cdot N_{\text{p, EV}} \cdot v_{\text{EV}} \cdot i_{\text{EV}} \cdot \frac{1}{\eta_{\text{c,EV}}} \cdot \frac{1}{1000} \right) = 0$$

Thermal Storage Balance:

$$P_{\text{TESS}} + \epsilon_{\text{TESS}} - P_{\text{load}}^{\text{th}} + P_{\text{ST}} + \eta_{\text{HP}} \cdot P_{\text{HP}}^{\text{e}} = 0$$

If $P_{\text{TESS}} \geq 0.0$:

$$(\text{SoC}_{\text{TESS},0} - \text{SoC}_{\text{TESS}} + \epsilon_{\text{SoC}_{\text{TESS}}}) - \left(P_{\text{TESS}} \cdot \frac{\Delta t}{Q_{\text{TESS}} \cdot 3600} \cdot \frac{1}{\eta_{\text{TESS}}} \right) = 0$$

Else:

$$(\text{SoC}_{\text{TESS},0} - \text{SoC}_{\text{TESS}} + \epsilon_{\text{SoC}_{\text{TESS}}}) - \left(P_{\text{TESS}} \cdot \frac{\Delta t}{Q_{\text{TESS}} \cdot 3600} \cdot \eta_{\text{TESS}} \right) = 0$$

State of Charge Dynamics:

$$\text{SoC}_{\text{BESS}} + \frac{\Delta t}{3600 \cdot Q_{\text{BESS}}} \cdot i_{\text{BESS}} - \text{SoC}_{\text{BESS},0} = 0$$

$$\text{SoC}_{\text{EV}} + \frac{\Delta t}{3600 \cdot Q_{\text{EV}}} \cdot i_{\text{EV}} - \text{SoC}_{\text{EV},0} = 0$$

A.2.3. Variables and Bound Constraints

Power Bounds [kW]:

$$-P_{\text{grid}}^{\text{max}}/sf \leq P_{\text{grid}} \leq P_{\text{grid}}^{\text{max}}/sf$$

$$0 \leq P_{\text{HP}}^{\text{e}} \leq P_{\text{HP}}^{\text{e,max}}$$

$$-P_{\text{EV}}^{\text{max}} \leq P_{\text{EV}} \leq P_{\text{EV}}^{\text{max}}$$

$$-P_{\text{BESS}}^{\text{max}} \leq P_{\text{BESS}} \leq P_{\text{BESS}}^{\text{max}}$$

$$-P_{\text{TESS}}^{\text{max}}/sf \leq P_{\text{TESS}} \leq P_{\text{TESS}}^{\text{max}}/sf$$

State of Charge Bounds [-]:

$$\text{SoC}_{\text{EV}}^{\text{min}} \leq \text{SoC}_{\text{EV}} \leq \text{SoC}_{\text{EV}}^{\text{max}}$$

$$\text{SoC}_{\text{BESS}}^{\text{min}} \leq \text{SoC}_{\text{BESS}} \leq \text{SoC}_{\text{BESS}}^{\text{max}}$$

$$\text{SoC}_{\text{TESS}}^{\text{min}} \cdot sf \leq \text{SoC}_{\text{TESS}} \leq \text{SoC}_{\text{TESS}}^{\text{max}}/sf$$

Current Bounds [A]:

$$-i_{\text{BESS}}^{\text{max}} \leq i_{\text{BESS}} \leq i_{\text{BESS}}^{\text{max}}$$

$$-i_{\text{EV}}^{\text{max}} \leq i_{\text{EV}} \leq i_{\text{EV}}^{\text{max}}$$

Slack Variables:

ϵ_{grid} : Electric balance slack

ϵ_{TESS} : Thermal balance slack

$\epsilon_{\text{SoC}_{\text{TESS}}}$: TESS SoC slack

A.2.4. Constants

General Constants:

$$sf = 1.05 \quad \text{[-]}$$
$$\Delta t = 900 \quad \text{[s]}$$

Grid Constants:

$$\eta_{\text{grid}} = 0.90 \quad \text{[-]}$$
$$P_{\text{grid}}^{\text{max}} = 17.0 \quad \text{[kW]}$$

BESS Constants:

$$P_{\text{BESS}}^{\text{max}} = 17.0 \quad \text{[kW]}$$
$$\text{SoC}_{\text{BESS}}^{\text{min}} = 0.20 \quad \text{[-]}$$
$$\text{SoC}_{\text{BESS}}^{\text{max}} = 0.95 \quad \text{[-]}$$
$$i_{\text{BESS}}^{\text{max}} = 7.80 \quad \text{[A]}$$
$$\eta_{\text{c,BESS}} = 0.99 \quad \text{[-]}$$
$$Q_{\text{BESS}} = 5.20 \quad \text{[Ah/cell]}$$
$$N_{\text{p,BESS}} = 10 \quad \text{[-]}$$
$$N_{\text{s,BESS}} = 100 \quad \text{[-]}$$

EV Constants:

$$P_{\text{EV}}^{\text{max}} = 12.5 \quad \text{[kW]}$$
$$\text{SoC}_{\text{EV}}^{\text{min}} = 0.20 \quad \text{[-]}$$
$$\text{SoC}_{\text{EV}}^{\text{max}} = 0.95 \quad \text{[-]}$$
$$i_{\text{EV}}^{\text{max}} = 7.80 \quad \text{[A]}$$
$$\eta_{\text{c,EV}} = 0.99 \quad \text{[-]}$$
$$\gamma_{\text{EV}} = 1.00 \quad \text{[-]}$$
$$Q_{\text{EV}} = 5.20 \quad \text{[Ah/cell]}$$
$$N_{\text{p,EV}} = 25 \quad \text{[-]}$$
$$N_{\text{s,EV}} = 100 \quad \text{[-]}$$

Heat Pump Constants:

$$P_{\text{HP}}^{\text{e,max}} = 4.00 \quad \text{[kW]}$$
$$\eta_{\text{HP}} = 4.50 \quad \text{[-]}$$

TESS Constants:

$$P_{\text{TESS}}^{\text{max}} = 5.00 \quad \text{[kW}_{\text{th}}\text{]}$$
$$\text{SoC}_{\text{TESS}}^{\text{min}} = 0.10 \quad \text{[-]}$$
$$\text{SoC}_{\text{TESS}}^{\text{max}} = 0.95 \quad \text{[-]}$$
$$\eta_{\text{TESS}} = 0.95 \quad \text{[-]}$$

Parameters

Some variables are defined by JuMP [36] as *Parameters*, which means that they are constants in the model optimised for frequent updates. In this case, they are updated at each timestep. Here is the list of *Parameters*:

$\varrho P_{\text{EV}}, \varrho P_{\text{BESS}}, \varrho P_{\text{HP}}^{\text{e}}$: Decisions proposed by the RL agent [kW] (possibly unsafe)

$P_{\text{load}}^{\text{e}}, P_{\text{load}}^{\text{th}}$: Electrical and thermal loads [kW]

$P_{\text{PV}}, P_{\text{ST}}$: PV generation and solar thermal generation [kW]

$\text{SoC}_{\text{BESS},0}, \text{SoC}_{\text{EV},0}, \text{SoC}_{\text{TESS},0}$: Initial state of charge of storage systems [-]

$v_{\text{BESS}}, v_{\text{EV}}$: BESS and EV (Battery) Voltage [V]

A.2.5. Safety Projection Details

Some implementation details of the safety projection mechanism, which were previously omitted in Section 3.4.1 for conciseness, will now be elaborated upon. As previously established, the safety projection intercepts the agent's proposed actions (denoted by ϱ) before they are introduced into the environment. Therefore, the model does not yet contain exogenous information from the next timestep. This lack of perfect knowledge of the future is a necessary limitation.

One of the consequences of this limitation, when considering safety constraints, is that the model cannot know for certain whether the EV will depart or arrive in the next timestep. Not knowing the future does not compromise the safety of BESS or EV battery dynamics, as these depend solely on agent decisions. However, P_{grid} may go off limits if the next timestep changes considerably past assumptions.

Given the current dataset, the combined electrical load and heat pump requirements cannot exceed grid limitations, thus leaving $P_{\text{EV}}, P_{\text{BESS}}$ (the projected decisions) as responsible for maintaining grid safety. Consequently, the value of γ_{EV} has been maintained at 1 (or $true$) throughout all timesteps within the model. Assuming the EV is always present will limit the power range of the BESS, although this is undesirable, it presents fewer complications than an unexpected EV arrival with substantial power requirements. While this approach merely addresses worst-case scenarios, more refined methods could be developed, such as utilising continuous values for γ_{EV} (to show uncertainty) or incorporating a basic prediction model for EV arrival and departure times.

A.2.6. Safety Projection in Operation

An example of how the *safe projection* (Section 3.4.1) constrains a Reinforcement Learning agent will be displayed in this Section. The focus is placed on the *Top RL Agent* (obtained in Section 5.4.2) and its performance on the *test set*. Figure A.2 presents the three components of the *projection distance*, corresponding to the decisions required of the agent.

Figure A.2: Visualisation of the safe projection distance affecting the decisions of the *Top RL Agent* (Section 5.4.2) applied to the *test set*. The *test set* encompasses 90 days, but serves as a representation of an entire year.

A.3. Synthetic Expansion of Training Data

This Section contains the algorithm used for creating the *validation set* (Algorithm 2), crucial for the hyperparameter optimisation, as it allows to evaluate the agents –produced by each combination of hyperparameters– on new data without making use of the *test set*. All the relevant information regarding the origin, division and synthetic expansion of the used data can be found in Section 3.6.

Algorithm 2 Synthetic Data Generation for Validation Set

Require: Training set of $D = 274$ days, each day containing $T = 96$ timesteps of 15 min, where $X_{d,t}$ is the value at day d and timestep t

Initialize $S = 91$ (number of synthetic validation days to create)
for $d = 1$ to $D - 4$ with step 3 **do**
  for each timestep $t = 1, \ldots, T$ **do**
    Calculate mean at timestep t across the 3 days:

$$\mu_{3,t} = \frac{1}{3} \sum_{i=0}^{2} X_{d+i,t}$$

    Sample synthetic value for timestep t from normal distribution:

$$V_{\text{syn},t} \sim \mathcal{N}\left(\mu_{3,t}, \frac{\mu_{3,t}}{3}\right)$$

    Restrict $V_{\text{syn},t}$ within the minimum and maximum values of the original dataset:

$$V_{\text{syn},t} \leftarrow \text{max}(X_{\text{min}}, \text{min}(V_{\text{syn},t}, X_{\text{max}}))$$

  end for
  Store V_{syn} as a new synthetic validation day
end for
return Validation set of $S = 91$ synthetic days.

Note: $\mathcal{N}(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ.

B

Hyperparameter Optimisation: Extra Information

B.1. Finding the Best Hyperparameters: BOHB Algorithm

BOHB is an acronym that stands for **Bayesian optimisation and Hyperband**, two methods that, although decent on their own, present an even greater synergy. It was developed by Falkner, Klein, and Hutter [17], who managed to craft a method that is able to surpass both Bayesian optimisation and Hyperband in numerous problems, including the **training of deep reinforcement learning models**.

Its effectiveness in optimising a wide range of problems is due to its robustness, a particularly relevant feature in the noisy world of RL [17]. BOHB is versatile; it can handle categorical, binary, integer, and continuous variables. The implementation of BOHB in Julia, provided by Fredrik Bagge Carlson's Hyperopt.jl package [6], provides an efficient framework for hyperparameter optimisation. Quite crucial for this thesis is the fact that Hyperopt.jl allows for an easy parallelisation of computational resources.

Bayesian optimisation works by modelling the objective function to be minimised using available data points. Traditionally, this has been achieved using Gaussian processes (GPs) to create a distribution over possible objective functions [17]. GPs naturally provide predictions (from their mean functions) and uncertainties (from the covariance functions) for any input. However, they face scalability issues in higher dimensions due to their cubic training complexity relative to the number of data points. In addition, they lack robustness, relying on good initialisation. For these reasons, Falkner et al. implemented a Bayesian optimisation method using the Tree Parzen Estimator (TPE) that models the densities [17]:

$$l(x) = p(y < \alpha | x, D), \quad g(x) = p(y > \alpha | x, D) \tag{B.1}$$

Where:

- x : Input configuration (hyperparameters being optimized)
- y : Observed value of the objective function
- α : Threshold value separating good and bad performance. E.g. current best performance.
- D : Set of observed data points (configurations and their performance)
- $l(x)$: Probability density of configurations performing better than α
- $g(x)$: Probability density of configurations performing worse than α

This approach differs from traditional methods that directly model the objective function. To choose a new candidate x, the ratio between the two probability densities ($l(x)/g(x)$) is maximised. Notably,

TPE's computational complexity grows **linearly** with the dataset size [17], offering the scalability that GPs cannot.

The other pillar supporting BOHB is Hyperband, an algorithm that samples random hyperparameter configurations and distributes the available resources to the most promising ones [17]. It operates by halting the worst performing evaluations, typically half or a third, and redirecting the freed resources to explore new possibilities. It is a fast, flexible, and well-performing algorithm, scaling easily to higher-dimensional inputs. However, Hyperband is limited by its reliance on random sampling, as there is no model of the objective function, no direction to the search [17]. These limitations become more pronounced in vast configuration spaces, as the combinations increase, the sampler will need exponentially more resources to find quickly the most promising hyperparameters.

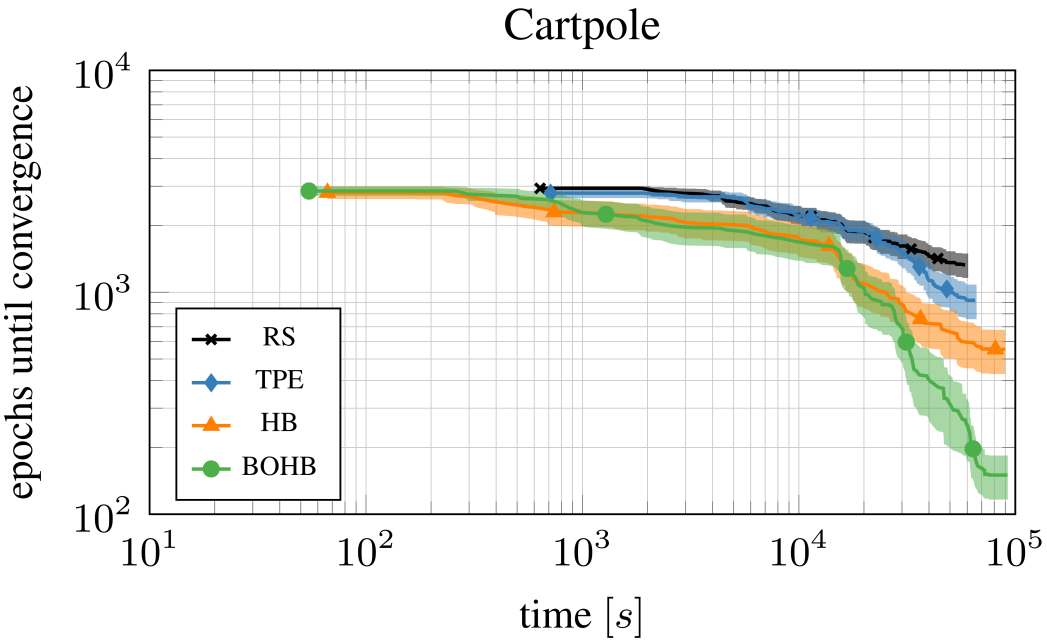

Figure B.1: Comparison of 4 methods for hyperparameter optimisation on the Cartpole benchmark with PPO. 8 hyperparameters are being tuned. RS stands for Random Sampler. Hyperband (HB) and BOHB perform similarly at the beginning, but BOHB manages to converge to a considerably better configuration. Source: [17]

The approach taken by Falkner and colleagues involves using Hyperband's methodology for resource management and budget selection. Additionally, they incorporate a mechanism for storing hyperparameter configurations and their corresponding performances [17]. This stored information is then used to build a model using TPE, allowing for informed decisions on the next batch of hyperparameters. As can be seen in Figure B.1, the balance of exploration and exploitation of this approach is adequate, allowing it to achieve better final performance than Hyperband or Bayesian optimization within the same runtime.

B.2. Auxiliary Hyperparameter Ranges

Due to space constraints, the most extensive hyperparameter ranges are presented in this appendix, with the main body of the thesis focusing on their development and impacts.

B.2.1. Feature Vectors

See Section 4.3 for a detailed explanation of the feature vector's role and definition. The Tables (B.1, B.2, B.3) represent the possible configurations of the feature vector, created for the purpose of expanding the hyperparameter search space (see Section 5.3).

Variable	Time Lags (days)			
	1	2	3	4
P^{e}_{load}	0.0	0.0	0.0, 0.5, 1.0	0.0, 0.5, 1.0
P^{th}_{load}	0.0	0.0	0.0, 0.5, 1.0	0.0, 0.5, 1.0
P_{PV}	0.0	0.0	0.0, 0.5, 0.65, 1.0	0.0, 0.5, 0.65, 1.0
λ_{buy}	0.0	0.0	0.0, 0.1, 0.45, 0.65, 1.0	0.0, 0.1, 0.45, 0.65, 1.0
λ_{sell}	0.0	0.0	0.0, 0.1, 0.45, 0.65, 1.0	-
γ_{EV}	0.0	0.0	0.0, 1.0	0.0, 1.0
P_{grid}	-	0.0	0.0, 0.5, 1.0	0.0, 0.5, 1.0
P_{BESS}	-	0.0	0.0, 1.0	-
P_{EV}	-	0.0	0.0, 0.5, 1.0	-
P^{e}_{HP}	-	0.0	0.0, 0.5, 1.0	-
SoC_{BESS}	0.0	0.0	0.0, 0.15, 0.4, 0.8, 1.0	0.0, 0.15, 0.4, 0.8, 1.0
SoC_{EV}	0.0	0.0	0.0, 0.25, 0.75, 1.0	0.0, 0.25, 0.75, 1.0
SoC_{TESS}	0.0	0.0	0.0, 0.25, 0.5, 0.75	0.0, 0.25, 0.5, 0.75
t_{ep}	0.0	0.0	0.0, 0.3, 0.9	0.0, 0.3, 0.9

Table B.1: Time lag configurations for all variables across 1-4 feature vector configurations.

Variable	Time Lags (days)		
	5	6	7
P^{e}_{load}	0.0, 0.5, 1.0	0.17, 0.58, 0.92	0.0, 0.04, 0.25, 1.0
P^{th}_{load}	0.0, 0.5, 1.0	0.03, 0.41, 0.79	0.0, 0.04, 0.25, 1.0
P_{PV}	0.0, 0.25, 0.65	0.22, 0.55, 0.88	0.0, 0.04, 0.25, 1.0
λ_{buy}	0.1, 0.45, 1.0	0.09, 0.36, 0.71	0.0, 0.04, 0.25, 1.0
λ_{sell}	-	0.14, 0.47, 0.83	-
γ_{EV}	0.0, 1.0	0.06, 0.39, 0.75	0.0, 0.25, 1.0
P_{grid}	0.0, 0.5, 1.0	0.28, 0.61, 0.95	0.0, 0.04, 0.25, 1.0
P_{BESS}	0.0, 1.0	0.11, 0.44, 0.80	0.0, 0.04, 0.25, 1.0
P_{EV}	0.0, 1.0	0.19, 0.52, 0.87	0.0, 0.04, 0.25, 1.0
P^{e}_{HP}	0.0, 1.0	0.08, 0.33, 0.69	0.0, 0.04, 0.25, 1.0
SoC_{BESS}	0.1, 0.8	0.25, 0.58, 0.91	0.0, 0.04, 0.25, 1.0
SoC_{EV}	0.0, 0.25, 0.75	0.05, 0.38, 0.72	0.0, 0.25, 1.0
SoC_{TESS}	0.0, 0.25, 0.8	0.31, 0.64, 0.97	0.0, 0.04, 0.25, 1.0
t_{ep}	0.0	0.13, 0.45, 0.78	0.0, 0.25

Table B.2: Time lag configurations for all variables across 5-7 feature vector configurations.

Variable	Time Lags (days)	
	8	9
P^{e}_{load}	0.0, 0.1, 0.5, 0.65, 1.0	0.0, 0.1, 0.5, 0.9
P^{th}_{load}	0.0, 0.4, 0.5, 0.75, 1.0	0.0, 0.4, 0.75
P_{PV}	0.0, 0.1, 0.25, 0.5, 0.65, 1.0	0.0, 0.25, 0.5, 0.75
λ_{buy}	0.0, 0.04, 0.1, 0.2, 0.45, 0.65, 1.0	0.0, 0.1, 0.4, 0.65
λ_{sell}	0.0	0.0
γ_{EV}	0.0, 0.25, 1.0	0.0, 1.0
P_{grid}	0.0, 0.04, 0.25, 0.5, 1.0	0.0, 0.5, 1.0
P_{BESS}	0.0, 0.04, 0.65, 1.0	0.0, 1.0
P_{EV}	0.0, 0.5, 0.7, 1.0	0.0, 0.5, 1.0
P^{e}_{HP}	0.0, 0.5, 0.95	0.0, 0.5, 1.0
SoC_{BESS}	0.0, 0.15, 0.25, 0.4, 0.8, 1.0	0.0, 0.1, 0.25, 0.5, 0.8
SoC_{EV}	0.0, 0.04, 0.25, 0.75, 1.0	0.0, 0.25, 0.8
SoC_{TESS}	0.0, 0.25, 0.4, 0.65, 0.8	0.0, 0.2, 0.8
t_{ep}	0.0, 0.25, 0.4, 0.9	0.0, 0.25, 0.4

Table B.3: Time lag configurations for all variables across 8-9 feature vector configurations.

B.2.2. Reward Functions

In this Section the reader will find all the available reward functions for the extended hyperparameter set (see Section 5.3.3), which stem from the possible combinations of reward functions represented concisely in Equation 4.17. The first two combinations were already described in Section 4.2.5, as Equations 4.18 and 4.19, and therefore have not been assigned a label below.

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\text{abs}}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t}\Big)$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\sigma}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t}\Big)$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\text{abs}}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot (r^{\text{op}}_{\text{proj},t} + r_{\text{margin}})\Big) \tag{B.2}$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\Sigma}_{\text{grid},e} + w_{\text{EV}} \cdot r^{\sigma}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot (r^{\text{op}}_{\text{proj},t} + r_{\text{margin}})\Big) \tag{B.3}$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\text{arb}}_{\text{grid},t} + w_{\text{EV}} \cdot r^{\text{abs}}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t}\Big) \tag{B.4}$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\text{arb}}_{\text{grid},t} + w_{\text{EV}} \cdot r^{\sigma}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot r^{\text{op}}_{\text{proj},t}\Big) \tag{B.5}$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\text{arb}}_{\text{grid},t} + w_{\text{EV}} \cdot r^{\text{abs}}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot (r^{\text{op}}_{\text{proj},t} + r_{\text{margin}})\Big) \tag{B.6}$$

$$r_{\text{final},t} = b\Big(w_{\text{grid}} \cdot r^{\text{arb}}_{\text{grid},t} + w_{\text{EV}} \cdot r^{\sigma}_{\text{EV},t} + w^{\text{init}}_{\text{proj}} \cdot r^{\text{init}}_{\text{proj},t} + w^{\text{op}}_{\text{proj}} \cdot (r^{\text{op}}_{\text{proj},t} + r_{\text{margin}})\Big) \tag{B.7}$$

B.2.3. Reward Weights

The Table B.4 represents three distinct arrangements for weighting the final reward (Equation 4.17). Each of the rewards described in Section 4.2 has a different magnitude. Although normalisation has been applied to most reward functions, the relative importance assigned to each component is ultimately a decision made prior to optimising, and as such it could prevent certain reward combinations (see Section 4.2.5) from guiding the agent towards an optimal policy.

Weight	Config 1	Config 2	Config 3
w_{grid}	1.0	1.5	0.75
w_{EV}	3.0	4.0	2.5
$w^{\text{init}}_{\text{proj}}$	1.0	2.0	0.75
$w^{\text{op}}_{\text{proj}}$	3.0	4.0	5.0

Table B.4: Weight configurations for the final reward, as defined in Equation 4.17.

B.2.4. Neural Network Architectures

This Section contains Tables B.5 and B.6, which provide compact symbolic representations of all the DNN architectures used for the extended hyperparameter optimisation (Section 5.3.3). To aid interpretation, a legend accompanies the Tables, explaining the notation and structure of the symbolic representations.

Legend for Tables B.5 and B.6:

ns : number of input features
w : width parameter
na : number of output values
$\xrightarrow{\text{act}}$: dense layer with activation function 'act'
act: {*Tanh*,*Mish*}
id : identity function
br : branch
res : residual connection
par : parallel branches
bot : bottleneck branch
+ : element-wise addition
contract : contraction layer.
If $na < 3$, a dense layer bridges the output to na width
out_i : output of branch i

Name	Symbolic Representation
Constant Width (CW)	$ns \xrightarrow{\text{act}} w \xrightarrow{\text{act}} w \xrightarrow{\text{id}} na$
Mid CW (MCW)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot 0.58$
Deep CW (DCW)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot 0.46$
CW with Standard Dev. (CW + STD)	$\pi_{\text{mean}} : ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} na$ $\pi_{\text{std}} : ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot \sqrt{1/2}$
Pyramid (PYR)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w'/2 \xrightarrow{\text{act}} w'/4 \xrightarrow{\text{id}} na$ where $w' = w \cdot 5/4$
Deep Pyramid (DPYR)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} 5w'/6 \xrightarrow{\text{act}} 2w'/3 \xrightarrow{\text{act}} w'/2 \xrightarrow{\text{act}} w'/3 \xrightarrow{\text{act}} w'/4 \xrightarrow{\text{act}} w'/6 \xrightarrow{\text{id}} na$ where $w' = w \cdot \sqrt{1/2}$

Table B.5: Symbolic Representation of Constant Width and Pyramid Architectures.

Name	Symbolic Representation
Bottleneck (BOT)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w'/5 \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot 1.55$
Residual (RES)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{res}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot \sqrt{1/2}$ and res $=$ id $+ (\xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w')$
Deep Residual (DRES)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{res}} w' \xrightarrow{\text{res}} w' \xrightarrow{\text{id}} na$ where $w' = w/2$ and res $=$ id $+ (\xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w')$
Residual + Bottleneck (RES + BOT)	$ns \xrightarrow{\text{act}} w' \xrightarrow{\text{res+bot}} w' \xrightarrow{\text{id}} na$ where $w' = w \cdot 1.48$ and res+bot $=$ id $+ (\xrightarrow{\text{act}} w'/5 \xrightarrow{\text{act}} w'/5 \xrightarrow{\text{act}} w')$
3 CW Branches (3CW)	$ns \xrightarrow{\text{id}}$ par $\xrightarrow{\text{contract}} na$ where par $= (br_1 + br_2 + br_3)$ $br_{1,2} : ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w' \xrightarrow{\text{id}} out_{1,2}$ $br_3 : ns \xrightarrow{\text{relu}} w' \xrightarrow{\text{relu}} w' \xrightarrow{\text{relu}} out_3$ where $w' = w \cdot 0.55$
3 Pyramid Branches (3PYR)	$ns \xrightarrow{\text{id}}$ par $\xrightarrow{\text{contract}} na$ where par $= (br_1 + br_2 + br_3)$ $br_{1,2} : ns \xrightarrow{\text{act}} w' \xrightarrow{\text{act}} w'/2 \xrightarrow{\text{act}} w'/4 \xrightarrow{\text{id}} out_{1,2}$ $br_3 : ns \xrightarrow{\text{relu}} w' \xrightarrow{\text{relu}} w'/2 \xrightarrow{\text{relu}} w'/4 \xrightarrow{\text{relu}} out_3$ where $w' = w \cdot \sqrt{1/2}$

Table B.6: Symbolic Representation of Bottleneck, Residual, and Multi-Branch Architectures

Quadratic Approximations of Parameter Count

Section 4.4.4 examines the alignment of parameter growth across all tested DNN architectures with the growth observed in the *CW* architecture. To enable a quantitative comparison, quadratic approximations were employed to model the relationship between input width (w) and the number of parameters, capturing the observed nonlinear trends. Table B.7 summarises the fitted polynomial coefficients and the corresponding coefficients of determination (R^2), where values approaching 1 reflect a strong agreement between the model and parameter count.

B.3. Supplementary Hyperparameter Optimisation Results

The most relevant patterns found when investigating the samples taken during *extended hyperparameter optimisation* were presented in Section 5.3.5. However, it was considered that additional performance patterns observed across other significant hyperparameters merited examination, especially to ascertain if they contributed to the performance improvement observed in comparison to the basic optimisation. This Section serves as an extension of Section 5.3.5.

Name	Quadratic Approximation	R^2
Constant Width (CW)	$1.0w^2 + 35.0w + 3.0$	1.000000
Mid CW	$0.9985w^2 + 21.94w - 27.8581$	0.999986
Deep CW	$1.0723w^2 + 17.7601w + 18.5515$	0.999978
CW + Std	$1.0002w^2 + 49.4402w + 6.7132$	0.999991
Pyramid	$0.9766w^2 + 40.1114w + 2.1263$	0.999994
Deep Pyramid	$1.0154w^2 + 22.8946w + 7.1809$	0.999985
Bottleneck	$0.9602w^2 + 53.7561w - 16.0052$	0.999988
Residual	$1.0002w^2 + 25.3991w + 3.9274$	0.999991
Deep Residual	$1.0006w^2 + 18.7598w + 18.5517$	0.999972
Residual + Bottleneck	$0.9630w^2 + 51.1769w - 19.0971$	0.999985
3 CW Branches	$0.8999w^2 + 54.3184w - 4.9992$	0.999985
3 Pyramid Branches	$0.9378w^2 + 66.7416w + 3.4118$	0.999982

Table B.7: Quadratic Approximations for Network Architectures

B.3.1. Architectures Comparison

Actor

The extended hyperparameter set encompasses various DNN architectures for the Actor network, expanding beyond the basic *CW* configuration. As was mentioned in various occasions along Section 5.3.5, dataset size and sample interdependence do not allow for strong conclusions to be drawn. The methodology focuses on identifying observable tendencies across the architectures, which were detailed in Section 4.4.4.

Performance metrics exhibit consistency across Actor architectures, with a few notable exceptions. *Deep Pyramid* (*DPYR*) demonstrates the lowest median values with reduced variance, as well as the worst training success rate ($p = 0.201$). This last result is specially surprising when considering that the shallower version of the pyramid architecture (*PYR*) has the second highest probability of successfully training agents ($p = 0.383$).

The three-branched configurations display slightly higher median values, but the most notable result is the high outliers obtained with these two architectures, which led to their selection for extended hyperparameter profiles (see Section 5.3.5). The presence of high-performing outliers suggests the **existence of specific hyperparameter configurations that can leverage the more complex architecture**. It is worth noting that the results from Andrychowicz et al. [4] have been experimentally confirmed (see Section 4.4.2), as there is no significant performance difference between the *CW* and *CW + STD* architectures.

The introduction of alternative architectures to *CW* appears to have yielded **performance improvements**, since an 18.7% increase in maximum performance was observed when using *3CW* ($M_{\text{perf}} = 1.178$). Apart from the three-branched outliers, the deeper variant *DCW* consistently achieves superior M_{perf} values, demonstrating as well the highest training success rate at $p = 0.387$.

Critic

Critic architectures display clearly different patterns compared to their Actor counterparts. The previously underperforming *DPYR* appears as a top performer when used as a Critic. *Bottleneck* and *Residual* configurations, which did not stand out with the Actor, demonstrate consistent high perfor-

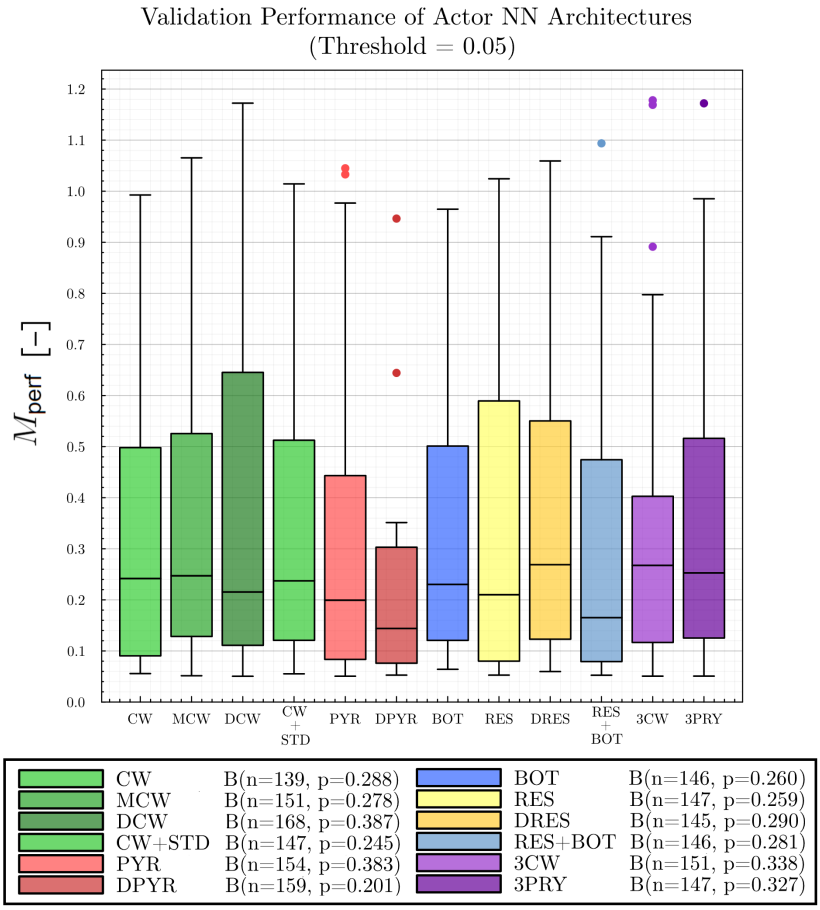

Figure B.2: Validation set performance (M_{perf}) distribution across DNN architectures for the Actor network. Box plots illustrate performance metrics for twelve distinct architectures, color-coded by architectural family (*green: CW variants, red: pyramid variants, blue/yellow: residual, bottleneck and their combination, purple: branched architectures*). A success threshold of 0.05 was applied. Whiskers extend to 1.5 times the interquartile range, with dots representing outliers. The legend presents binomial distribution parameters B(n,p), where n indicates sample size (139-168) and p denotes the probability of producing agents above the threshold for each architecture.

mance, placing them in extended hyperparameter profiles (see Section 5.3.5). The variation in peak performance between the Actor and the Critic architectures, coupled with the mediocre results obtained by *CW*, validate the implementation of diverse architectural configurations.

B.3.2. Feature Vectors Comparison

The feature vector configurations, introduced in Section 4.3, demonstrate in Figure B.4 varying impacts on agent performance, with *configurations 1, 2 and 5* exhibiting significantly higher medians. The randomised time lags in *configuration 6* yield the lowest success rate ($p = 0.186$), which could be expected. *Configuration 7*, which stems from the assumption that patterns of agent behaviour should be correlated with the last 0,1,6 and 24 hours, presents similar outcomes to configuration 6, albeit with a higher likelihood of success.

The performance analysis of *configurations 3, 4, 5, 8,* and *9*, which were constructed based on the expert's state-action cross-correlation values yields unexpected results. There appears to be an inverse relationship between information quantity and performance, which suggests potential overfitting when too many time lags are included. Except for *configuration 5* –the simplest– these feature vectors underperform relative to the straightforward last-timestep approach of *configurations 1* and *2*.

Configuration 2, employed in the basic optimisation, demonstrates remarkable robustness with a

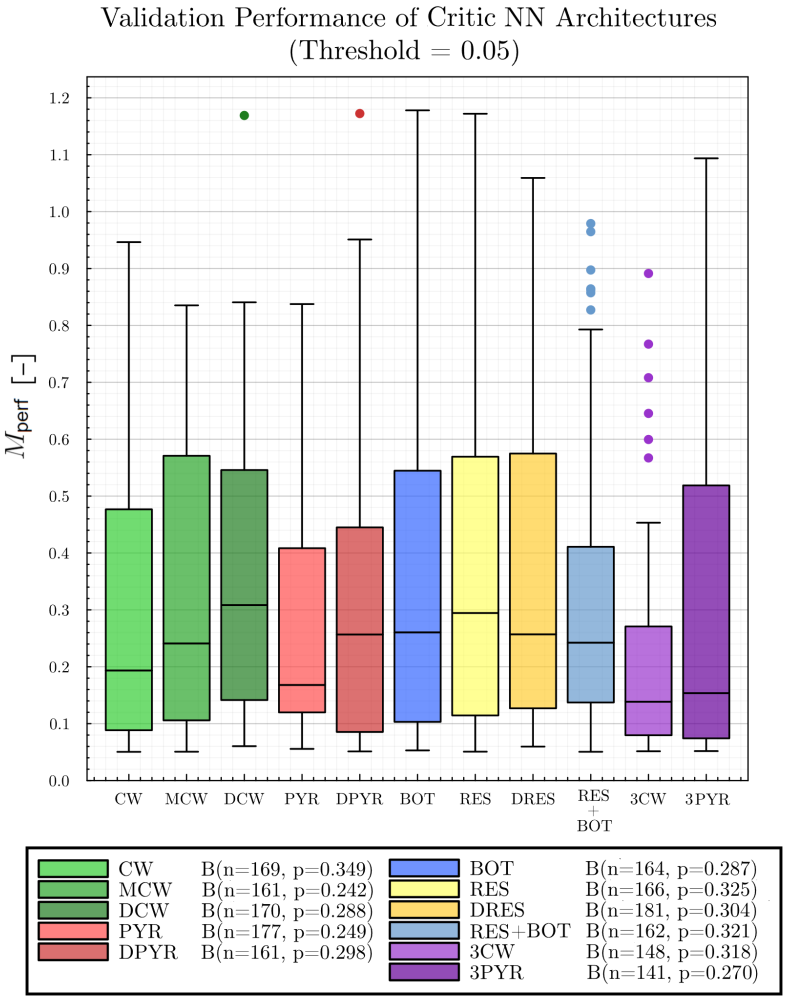

Figure B.3: Validation set performance (M_{perf}) distribution across DNN architectures for the Critic network. Box plots illustrate performance metrics for eleven distinct architectures, color-coded by architectural family (*green: CW variants, red: pyramid variants, blue/yellow: residual, bottleneck and their combination, purple: branched architectures*). A success threshold of 0.05 was applied. Whiskers extend to 1.5 times the interquartile range, with dots representing outliers. The legend presents binomial distribution parameters B(n,p), where n indicates sample size (141-181) and p denotes the probability of producing agents above the threshold for each architecture.

success probability of $p = 0.346$, and a peak result of $M_{\text{perf}} = 1.172$. The limited impact of feature vector range expansion on the extended hyperparameter profiles indicates that performance improvements stem from other factors.

B.4. System Specifications for Training RL Agents

The following list outlines the hardware and software resources utilised throughout this thesis for training RL agents and conducting hyperparameter optimisations:

- **Model:** Asus ProArt StudioBook.
- **Processor:** 12th Gen Intel(R) Core(TM) i7-12700H, operating at 2300 MHz, featuring 14 cores and 20 logical processors.
- **Memory (RAM):** 32.0 GB of physical memory.
- **Graphics Card:** NVIDIA RTX A3000 Laptop GPU, with 12 GB VRAM.
- **Operating System:** Windows 11 Pro.

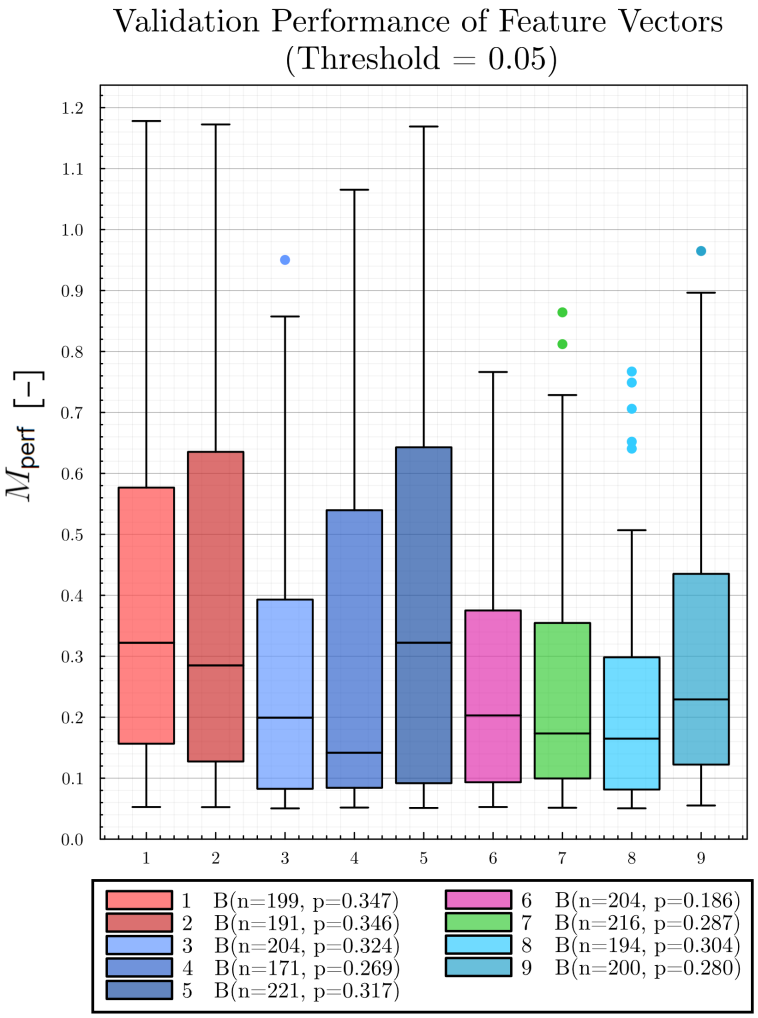

Figure B.4: Validation set performance (M_{perf}) distribution across feature vector configurations. Box plots illustrate performance metrics for nine distinct feature combinations, color-coded by feature family (*red: last timestep, blue: based upon expert's state-action cross-correlation, pink: randomised time lags, green: 0,1,6 and 24 hours of lag*). A success threshold of 0.05 was applied. Whiskers extend to 1.5 times the interquartile range, with dots representing outliers. The legend presents binomial distribution parameters B(n,p), where n indicates sample size (171-221) and p denotes the probability of producing agents above the threshold for each configuration.

B.5. Working with Julia's RL Package

The RL package in Julia, *ReinforcementLearning.jl* [61], has proven useful for this thesis by establishing a basic structure on which to build. It creates abstractions such as agents, environments, policies, and stages in the run. However, it has been undergoing a refactoring process for a considerable time, resulting in stable versions that **lack many features**.

The decision not to give up the incredible performance and efficiency of Julia [8] meant implementing or adapting considerable parts of the policy gradient algorithms described in Section 4.5. In this circumstances, the decisions were made primarily by adhering to the principle of *adding those changes that have proven to improve performance reliably and substantially*.

B.5.1. Code Repository

The complete source code employed to address the research objectives is accessible at: https://github.com/Victor-Andres-RdeTrio/RL4MCES

The author may be reached by email: vandres.trio@proton.me