Universiteit Leiden

Master Computer Science

Multi Dynamics- and Q-Learning with Shared State Representations for Battery Dispatch Optimisation

Name:	Jerry Schonenberg
Student ID:	2041022
Date:	21/07/2023
Specialisation:	Artificial Intelligence
1st supervisor:	Dr. Thomas Moerland
2nd supervisor:	Prof. Dr. Aske Plaat
Ext. supervisor:	Dr. Christian Michler (Shell)
Ext. supervisor:	Kapil Mathur (Shell)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgements

I would like to express my gratitude to Thomas Moerland for supervising me throughout the research project, and for providing useful insights, feedback and the many discussions we have had. Also many thanks to Aske Plaat for being my second supervisor.

Moreover, I would also like to thank Christian Michler, Kapil Mathur, Carlos Ros Perez and Richie Lo for guiding me through the project, the weekly discussions, and helping me shape the research project along the way.

This research has been carried out in collaboration with Shell Global Solutions International B.V. The data used in this work is owned by Shell. The author would like to acknowledge the compute resources provided by Shell.

Abstract

With the ongoing energy transition, there is a push for the electrification of various appliances and the use of Behind-the-Meter (BtM) smart batteries. In combination with solar panels, the carbon footprint and household expenses can be reduced by intelligently managing the power flow around the battery. This can be formulated as a sequential decision making process, where a Home Energy Management System (HEMS) can decide at each hour whether to charge or discharge the battery. We introduce an environment where a HEMS has to manage the power flow around a BtM battery of a residential household, which is equipped with a heatpump and solar panels. Moreover, the environment makes use of historical data with regards of weather (forecasts) and grid tariffs, and with as consequence that it is strictly speaking not an online setting.

In this work, we have conducted an extensive comparison between various reinforcement learning algorithms, Mixed-Integer Linear Programming (MILP) with perfect foresight and a Heuristic-Based System. In addition, we propose a novel model-based reinforcement learning architecture aimed at environments where epistemic uncertainty plays a significant role, Multi Dynamics- and Q-Learning (MDQL), which consists of sharing state representations between two ensembles of dynamics models and DDQNs. Moreover, we introduce a novel planning strategy, Hindsight-Weighed Planning, where the accuracy of the dynamics models at the previous planning iteration is used to weight the rollouts of the current iteration.

By means of experimentation, we show that MDQL without planning outperforms all other benchmarks, with the exception of MILP. With planning, the overall performance deteriorates slightly, which is attributed to the somewhat inaccurate next state and reward predictions. However, MDQL plus planning over the ground truth shows to outperform MILP by a significant margin, thus being a promising solution to the battery dispatch optimisation setting.

Executive Summary

Battery dispatch management for a residential household can be optimised through the use of an algorithmic approach, such as linear programming or reinforcement learning (RL). In this work, we have developed a virtual environment which simulates Shell's EcoGenie house at The Hague. The environment simulates a smart Behind-the-Meter (BtM) battery, photovoltaic panels, access to the power grid, variable grid tariffs and a heatpump, and makes use of historical weather (forecasts) and day-ahead market pricing.

We demonstrate that linear programming and reinforcement learning can save up to 9 euros in consumption costs each month, thus providing more affordable and more renewable power. However, incorporating battery-operation related costs (i.e., the deterioration of the battery's state-of-health) results in neither linear programming nor reinforcement learning making use of the battery at all. It suggests that a BtM battery is not cost effective as of 2023, and has as its only benefit the reduction of the carbon footprint. Incorporating more realistic battery simulation dynamics is required to confirm this.

In addition, we highlight some potential issues that may arise when deploying reinforcement learning, caused by the fact that the real world is not stationary in terms of weather. At some point in time, the model might lose its robustness against more frequent extreme weather occurences. Similarly, a reinforcement learning algorithm might have to be trained for each region separately in case of different weather/climate characteristics.

Lastly, we argue that in the current problem setting, linear programming seems to be the preferred choice due to its performance and the lack of a high-dimensional action space in the environment. But in case of more complex problem settings, we believe that RL would be able to outperform linear programming, when given an equal amount of computational resources.

Table of Contents

List of Figures

List of Tables

1 Introduction

In order to reach IEA's climate target of Net Zero Emissions (NZE) by 2050 and limit the rise of the global temperature to 1.5 °C, we have to increase the electricity generation from renewable sources (e.g., solar, hydro and wind) from 156 TWh in 2020 to 821 TWh in 2030 (IEA, 2021). Subsequently, a yearly linear increase of 66.5 TWh would be required to reach the target.

However, an increase in electricity generation poses as a major challenge for the power grid. Overloading the grid will occur more frequently if no further investments are made into the infrastructure. By investing in smart Behind-the-Meter batteries, each building will be able to locally store its surplus of self-generated power through photovoltaic (PV) panels, instead of netting it back into the power grid. Consequently, the load on the power grid will be decreased, and the building reduces its carbon footprint by maximising its utilisation of PV-power. An additional benefit is the fact that less electricity will be drawn from the power grid, thus decreasing the consumption costs. Alternatively, this can be extended to a larger scale, over a cluster of buildings. This is referred to as micro-grids, which are significantly smaller compared to the current infrastructure. In this work, we consider one residential building equipped with PV-panels and directly connected to the power grid.

For the energy management, we must take the unreliable generation of electricity through PV-panels into account; it is only generated during sun-hours, of which the duration and intensity can differ significantly per week, day and hour. Moreover, the battery might get depleted in the case of a prolonged period with no sunlight (e.g., winter season) or intensive consumption profiles. Thus, solely relying on PV-power is infeasible in order to meet the power demands at all times. The shortage in power will instead be drawn from the grid.

This poses an optimisation problem, where we want to maximise the utilisation of PV-power, and minimise the costs of drawing power from the grid by exploiting low tariffs. Moreover, in case of a tariff spike, we may opt for netting the surplus of power in exchange for a small profit. However, the primary objective is to maintain an acceptable user-comfort, such that all power-related demands are met.

This problem can be formulated as a sequential decision making problem, where a Home Energy Management System (HEMS) (Zafar et al., 2020; Zhou et al., 2016) makes, at a constant interval, a decision with regards to the battery: charge, discharge or remain idle. Various scheduling optimisation techniques for HEMS are already proposed, such as Mixed-Integer Linear Programming (MILP) (Lokeshgupta and Sivasubramani, 2019), genetic programming (Hu and Xiao, 2018), and reinforcement learning (RL) (Mason and Grijalva, 2019; Vázquez-Canteli and Nagy, 2019; Yu et al., 2020a).

In this work, we investigate how reinforcement learning (Sutton and Barto, 1998) can take on the role as HEMS for one household, specifically Shell's EcoGenie house in The Hague. This is a residential house that is kitted out with various devices and sensors, and provides the opportunity to experiment with state-of-the-art techniques, related to renewable energies, in the real world without any major risks. More specifically, the HEMS will manage the power flow around the smart battery and heatpump, making this a Heating Ventilation and Air Conditioning (HVAC) management problem.

Datasets on weather and grid-tariffs, along with a set of rule-based dynamics, serve as training data for the reinforcement learning method. Subsequently, no traditional simulator/environment is available, i.e.: the agent is not able to sample new transitions from the environment continuously in order to lower its epistemic uncertainty. This poses an issue for online RL methods; when the agent finds itself in an unsupported region of the search space, it would essentially perform random actions. This is because the actions are based on the knowledge acquired from the supported regions, which may not be applicable in out-of-distribution states. Thus, there is some degree of uncertainty when the agent interacts with the real environment. To mitigate this, we can utilise offline RL (Lange et al., 2012; Levine et al., 2020), which learns a policy from a static offline dataset and takes the epistemic uncertainty into consideration (e.g., introducing uncertainty penalties or forming explicit trust-regions in the state-action space).

Moreover, model-based reinforcement learning (MBRL) (Moerland et al., 2020; Plaat et al., 2021) is often deployed in offline settings. By adding an internal model that learns the dynamics of the environment based on the static offline dataset, we can mitigate the absence of a simulator and replace it with the internal model. MBRL also offers the possibility of using the model for planning at testing time. However, learning an internal model adds more complexity to the RL-method, on top of the fact that the dataset itself still does not cover the whole search space. Subsequently, the internal model will introduce a degree of bias in the unsupported subregions. This should not be a major issue, since the environment is based upon the real-world, which in itself is systematic (e.g., it follows the laws of physics/mathematics/nature/etc.), and therefore the model should be able to accurately infer the dynamics in the unknown subregions based upon the dataset. In short, internal models will probably generalise better than value-/policy-approximators in offline settings.

As one of the main contributions of the thesis, we introduce a novel algorithm, Multi Dynamics- and Q-Learning (MDQL), where two ensembles are utilised to robustly solve the environment. The members of the first ensemble are trained to estimate the q-values, while the other members learn the dynamics of the environment. Due to the high-dimensional state space, neural networks are used as function approximators. The novelty of the algorithm comes from the network architecture: both ensembles share the first few hidden layers. Consequently, the shared layers will act as feature extractor, enabling the use of smaller networks for the ensemble members, along with providing more rich latent states to both ensembles.

In short, the contributions of the thesis are three-fold:

(i) Providing a novel take on the HEMS problem setting, by developing a new environment with as aim to accurately model a residential household located in The Hague. In addition, we evaluate a varied set of benchmarks (MILP, RL-algorithms and a Heuristic-Based System) on the environment, and empirically demonstrate how the approaches fare against each other.

(ii) We propose a novel network architecture in the reinforcement learning domain, MDQL, where a feature extractor is shared between the ensemble of dynamics models and q-value estimators.

(iii) The use of two ensembles enables a novel strategy of planning, Hindsight-Weighted Planning (HWP), where we evaluate the accuracy of the dynamics models after we have executed the best action based upon the previous planning iteration. This evaluation then influences the weight per head for the current planning iteration.

By means of experimentation, we show that MDQL is able to outperform all RL baselines and a Heuristic-Based System. However, MILP with perfect foresight is slightly better. In addition, planning has resulted in a slight deterioration of the overall performance, which can be

contributed to the somewhat inaccurate next state and reward estimations. Given the ground truths, MDQL plus planning outperforms MILP with perfect foresight by a significant margin, showing the potential of the proposed algorithm. Moreover, a set of ablations has shown that each component of MDQL (e.g., ensembles, sharing weights between all approximators, learning the dynamics) contributes positively to the final performance of the model.

Overall, the behaviour of MDQL on the EcoGenie environment can be summarised as follows: the agent tries to maximise its PV-utilisation by storing it to the smart battery, and discharges this power to the heatpump during hours without any sunlight. In addition, the agent sometimes opts for a quick charge of the battery via the grid at midnight, since the battery is almost depleted at that point in time. The main shortcomings can be found at the time at which the discharging occurs, since the price spikes are often not fully exploited by the agent. Now, the agent is sometimes too early or late with its discharge-operation, and misses the peak entirely. On the other hand, MILP receives the exact pricing for the upcoming hours, and is able to fully exploit the outliers in the grid tariff.

The remainder of the thesis is structured as follows: Section 2 contains related work; Section 3 discusses the required preliminaries; Section 4 describes the problem setting and datasets; Section 5 elaborates upon the proposed algorithm, Multi Dynamics- and Q-Learning; Section 6 and Section 7 contain the experimental setup and results, respectively; and lastly, Section 8 discusses the main takeaways from this work along with future work suggestions.

2 Related Work

This section contains an overview of work that we share one or more domains with. In short, we discuss research on Home Energy Management Systems, methods from the reinforcement learning (RL) paradigm, specifically model-based and offline RL. Lastly, RL-environments aimed at Heating Ventilation and Air Conditioning (HVAC) are discussed.

2.1 Home Energy Management Systems

Home Energy Management Systems (HEMS) have become an increasingly relevant research topic due to the energy transition. The main task is to efficiently manage the energy flow within a house/building, for instance through scheduling the charge and discharge cycles of a smart battery, scheduling household devices and/or predicting future power demands based on historical data.

For instance, (Mixed-Integer) Linear Programming (LP) (Yu et al., 2013; Lokeshgupta and Sivasubramani, 2019; Souza Dutra et al., 2019) solvers have been proposed that optimise the scheduling of appliances with a set of defined constraints (e.g., an appliance must run within the next 12 hours, the smart battery can not exceed a State of Charge of 13 kWh, etc.). However, these solvers scale exponentially with the horizon, which makes the method computationally expensive and quickly intractable. In addition, a (near-)perfect foresight or accurate forecasts over the horizon is a prerequisite, which makes it less suitable for deployment in the real world. Instead, LP can quantify an upper-bound on the performance and function as a baseline for other HEMS.

Other work proposes the use of evolutionary algorithms to optimise the scheduling. For instance, Hu and Xiao (2018) make use of genetic programming to deal with demand response. A population of candidates can be altered by repeatedly applying mutation, cross-over and selection to the population. With every generation, the population converges closer to an optimum in the search space. The selection after each generation is based on a hand-crafted fitness-rule (i.e., objective function) that incorporates the user-comfort and costs. Similarly, particle swarm optimisation has also been applied to this domain, for instance by Lugo-Cordero et al. (2011).

Lastly, machine learning techniques have also been proposed as HEMS. For instance, deep neural networks, which are optimised by a genetic algorithm, have been applied to managing the power-flow (Matallanas et al., 2012), appliance scheduling (Yuce et al., 2016) or lighting control (Hernandez et al., 2010; Ahmed et al., 2016) in a residential house.

Most similar to our work, reinforcement learning has also been applied to energy management within a household. Various papers have been dedicated to HVAC-management optimisation (Wang et al., 2017; Chen et al., 2018), appliance scheduling (Kim et al., 2016; Bahrami et al., 2018) and power generation optimisation (Mbuwir et al., 2017; Tan et al., 2018). A survey by Vázquez-Canteli and Nagy (2019) provides an elaborate overview on the work that has been done in the HEMS-domain with reinforcement learning. The authors make the observation that it is difficult to compare the algorithms, due to the varying nature of the problems that are being solved over the papers. There is no mainstream benchmark available to evaluate a RL-agent against.

2.2 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 1998) has had exciting breakthroughs in recent years. For instance, the Deep Q-Network (Mnih et al., 2013) is able to exceed expert-level performances in seven Atari 2600 games by utilising Q-learning, convolutional layers and experience replay. Next, DeepMind's AlphaGo (Silver et al., 2016) defeated the world champion of Go in 2016, and a more general version of the model has been published in the form of AlphaZero (Silver et al., 2017). Both algorithms make use of self-play, however, AlphaGo requires expert games and the rules of the game, while AlphaZero only needing the latter. Lastly, OpenAI developed OpenAI Five (Berner et al., 2019), a multi-agent RL method able to perform at a very high level in the game Dota 2, where cooperation and coordination between team-members are prerequisites in order to win. In the context of our research, we now discuss approaches that relate to model-based and/or offline reinforcement learning.

2.2.1 Model-Based Reinforcement Learning

One of the first architectures to be proposed in the subdomain model-based RL (MBRL) is Dyna (Sutton, 1991), which provides a framework for tabular Q-planning. It utilises the transitions experienced in the environment for learning policy-/value-functions and learning the internal model. The model itself can then also be used to further learn the policy-/value-function.

In addition to a table as the internal model, papers have also proposed graphs (Eysenbach et al., 2019; Huang et al., 2019; Zhang et al., 2021), Gaussian Processes (Deisenroth and Rasmussen, 2011), and, most notably, deep neural networks (Buckman et al., 2018; Matsushima et al., 2021), to represent the internal dynamics model.

Neural networks are frequently selected for function approximation due to its ability to learn complex (non-)linear functions. For instance, Chua et al. (2018) proposed PETS, which makes use of a bootstrapped ensemble of probabilistic neural networks in order to separately model the aleatoric and epistemic uncertainty. Each network parameterises a Gaussian, and is trained on a subset of the available experience. The ensemble is used during the planning phase with Model Predictive Control (MPC) (Camacho and Alba, 2013), and each ensemble head propagates a set of particles with Trajectory Sampling. The actions are selected with CEM (Botev et al., 2013), which selects actions in proportion to the historical received rewards.

Janner et al. (2019) took inspiration from PETS, and developed MBPO. The main difference is that a Soft Actor-Critic (Haarnoja et al., 2018) agent is trained, instead of directly using the model for planning. The internal model is fitted to the experienced transitions in the real environment, while the policy is trained on model rollouts. Each model rollout starts at a state sampled from the set of known environment-states, after which the policy and model are used to perform the rollout.

Meta-Learning has also proven to be useful in the MBRL domain. MB-MPO (Clavera et al., 2018) interprets each member of the ensemble as a different task, and then trains a meta-policy on all tasks to find the meta-objective (i.e., finding the set of parameters such that the optimal set of parameters for one task can be obtained with few gradient updates). For this, MB-MPO makes use of MAML (Finn et al., 2017) to update the policy-parameters appropriately. The authors argue that the meta-policy is forced to find the commonalities between the tasks, thus minimising the effect of model bias and model exploitation.

One of the main common divisors between the discussed papers is the use of an ensemble of deep neural networks, aimed at quantifying the epistemic uncertainty. Another approach to uncertainty quantification are Bayesian neural networks. However, it has been demonstrated

by Lakshminarayanan et al. (2017) that deep ensembles match or exceed the quality of the uncertainty estimation, along with overall performance, of Bayesian neural networks on a varied set of regression and classification tasks.

In contrast, Luo et al. (2019) proposed SLBO, a MBRL method that does not use an ensemble of dynamics models nor explicit uncertainty estimations. SLBO is based on a theoretical framework the authors provided, which guarantees a monotonic improvement based on the discrepancy between the real environment and internal model. As for the practical implementation, the model and policy are trained alternatively. The model is trained on a dataset with traces collected by a data-collection policy in the real environment. The policy itself is modelled by TRPO (Schulman et al., 2015), and is trained on a separate dataset with traces generated by the model. The authors argue that this can be seen as implicitly making use of an ensemble, since the dataset of the policy contains traces from different instances of the internal model.

2.2.2 Offline Reinforcement Learning

The main challenge in offline reinforcement learning is that a policy has to be learned from a finite static dataset, with the main issue being that the dataset can not cover the whole state-action space sufficiently. As a consequence, there is a degree of epistemic uncertainty that arises in the unsupported regions, where the agent will perform worse, or even random, when its actions are naively inferred from the dataset. This has been empirically demonstrated by Fujimoto et al. (2019b), where an off-policy method performs significantly worse than the behaviour policy, even though both policies are trained according to the same algorithm and share the same dataset.

Various approaches have been proposed, such as regularising the policy in order to limit the distribution shift. For instance, Fujimoto et al. (2019b) propose an offline RL method, BCQ, where the actions it considers are restricted by the state-action distribution of the dataset. The set of to-be-considered actions are generated by a Variational Auto-Encoder (Sohn et al., 2015), after which two Q-networks are used for value estimation. With this, the authors aim to minimise the extrapolation error that arises due to poor generalisation to out-of-distribution state-action pairs.

Model-Based RL is also frequently used in the offline RL setting. The internal model functions as the environment, mitigating the issue of the finite dataset. Similarly to various work in the online MBRL space, MOReL (Kidambi et al., 2020) and MOPO (Yu et al., 2020b) also make use of a deep ensemble of internal models for uncertainty estimation. MOReL partitions the state-action space into two sets; the 'known' and 'unknown' space. The known space is supported by the offline dataset, while the unknown space is not. Whether a state-action pair is supported, is determined by the maximum pair-wise discrepancy of the ensemble. When it exceeds a preset threshold, the state-action pair is categorised to be unknown. In case of unknown, the agent receives a negative reward and the episode terminates.

As for MOPO, it enhances the rewards in the static offline dataset with an uncertainty penalty. It is based on the maximum standard deviation of the deep ensemble. This is then multiplied with a scalar, and subtracted from the reward given in the transition.

Moreover, deep ensembles are also used for planning. For instance, MBOP (Argenson and Dulac-Arnold, 2021) combines the ensemble with MPC (Camacho and Alba, 2013). In addition, bootstrap ensembles are also learned for a behaviour-cloned (BC) policy and fixed-horizon value function. Given the three ensembles, the algorithm performs rollouts where the action is selected based on the BC-policy and the previous rollout. The cumulative return of each rollout is used to select the next action based on the return-weighted average trajectory.

Chen et al. (2021) took a different approach with the deep ensembles. The authors proposed MAPLE, a RL-algorithm that learns a context-aware adaptable policy. In short, the policy takes as input the state along with the context. This context is computed by an extractor, that receives as input the previous context, along with the transition to the current state. Subsequently, the context extractor is modelled by a recurrent neural network.

Unlike the earlier discussed offline MBRL work, the algorithm COMBO (Yu et al., 2021) augments the dataset with one probabilistic dynamics model, thus not requiring explicit uncertainty estimation. The dynamics model is trained on the offline dataset, after which short rollouts are executed starting from a state uniformly sampled from the dataset. Transitions from both the offline and generated datasets are used in the policy improvement step, with the generated transitions being slightly lowered in importance to the objective function. The agent is slightly conservative on updating the policy with the generated data, due to the possible model bias that is present in the dynamics model.

2.3 HVAC Management Environments

The setting of energy management for buildings has been an active field to apply RL algorithms on. Environments need to accurately simulate the complex processes that occur in managing the Heating Ventilation and Air Conditioning (HVAC) for buildings. This has been simplified by the development of open source simulators EnergyPlus (Crawley et al., 2000) and Modelica (Mattsson and Elmqvist, 1997), which accurately simulate the main components in building management (e.g., energy consumption, heat loss, HVAC-devices).

The Gym-Eplus (Zhang and Lam, 2018) and Testbed for EnergyPlus (Moriyama et al., 2018) provide a wrapper for the EnergyPlus simulator. More specifically, Gym-Eplus provides an environment tailored to their workplace. The simulator is build according to the properties of the workplace, after which the system is calibrated using historical data from the workplace.

Other environments aimed at building management and HVAC simulation make use of the OpenAI Gym framework (Brockman et al., 2016). Sinergym (Raboso et al., 2021) extends Gym-Eplus, and provides an energy building simulation and control framework. It offers a set of EnergyPlus models (5-zone buildings, datacenters and workplaces) along with the ability to customise various aspects of the Markov Decision Process (MDP). For instance, the state space, action space and reward function (CO_2 emissions, thermal comfort and energy consumption) can be altered. In addition, the weather variability can be tuned with three presets, each representing a type of climate (hot dry, mixed humid and cool marine). The weather can also be simulated, following the Ornstein-Uhlenbeck process (Benth and Šaltytė-Benth, 2005).

BOPTEST (Arroyo et al., 2021; Blum et al., 2021) is another environment build on top of the Gym framework. It is based on the Modelica building models, and offers the ability to customise various aspects of the MDP, such as the sampling interval, forecasting period, state space, and action space. The default reward function consists of the energy consumption and thermal comfort.

Lastly, Energym (Scharnhorst et al., 2021) provides an environment that combines models from both EnergyPlus and Modelica. Specifically, eleven buildings are included (e.g., apartments, offices, houses) and can be easily extended to new types of buildings. It also includes the possibility to add the forecast of the weather, electric-vehicle usage, and power consumption to the state space.

Overall, the three main environments (BOPTEST, Energym and Sinergym) are also included in the Python package BeoBench (Findeis et al., 2022). It provides a standard for evaluating RL algorithms in the Energy Management Systems setting.

3 Preliminaries

In this section, we cover the preliminaries regarding formal definitions of the reinforcement learning paradigm, along with a brief overview of offline and model-based reinforcement learning plus the main challenges and benefits that come with the two subdomains.

The machine learning domain can be subdivided into three main types of learning techniques: (i) supervised learning; (ii) unsupervised learning; and (iii) reinforcement learning. With supervised learning, a dataset with ground truth data is available, such as labels for a classification task, or numerical values for regression tasks. Given the dataset, we can fit a model/algorithm to it, such as linear regression models, Support Vector Machines and (deep) neural networks. On the other end of the spectrum, unsupervised learning problems do not have ground truth data. In this case, we can use clustering methods (e.g., *K*-means clustering) to classify each datapoint into one or more groups based on their attributes. The algorithm has to find trends or key features based upon which datapoints are subdivided into categories. For instance, datapoints with similar combination of attributes will be categorised into the same group. In contrast, reinforcement learning (RL) (Sutton and Barto, 1998) methods do not require a dataset, and instead create their own training samples. Moreover, reinforcement learning methods receive partial feedback (e.g., only the immediate reward for the selected action), while supervised learning receives complete feedback (i.e., the ground truths).

3.1 Reinforcement Learning

The RL paradigm consists of an environment and an agent. At timestep t, the agent receives a state s_t from the environment, which describes features of the current state of the environment (e.g., velocity of a car, outdoor temperature, position of a chess piece). Based upon this, the agent performs an action a_t in the environment, after which the environment responds with a new state s_{t+1} and a reward r_t. This forms a cycle in which the agent and environment interact with each other, which can be used to model sequential decision making problems. Figure 3.1 visualises the cycle, which can continue indefinitely, or end when a terminal state is reached. When a task has naturally defined terminal states (i.e., checkmate or stalemate in a chess game), it is defined as an episodic task. Following this, an episode is defined as one complete session of interactions between the agent and environment; from s_0 to a terminal state. The objective of the agent is to maximise its cumulative reward received throughout an episode.

3.1.1 Markov Decision Processes

A sequential decision making problem can be formalised into a Markov Decision Process (MDP) (Puterman, 1994) $\mathcal{M}$, where $\mathcal{S}$ defines the state space ($s \in \mathcal{S}$); $\mathcal{A}$ is the set of actions ($a \in \mathcal{A}$); $\mathcal{T}$ is the transition function which provides a mapping from state-action pairs to states, i.e.: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto p(\mathcal{S})$; $\mathcal{R}$ is the reward function, mapping a transition to a numerical value: $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$; $p(s_0)$ is the probability distribution of the initial state of an episode; and γ is the discount factor.

$$\mathcal{M} \doteq \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_0), \gamma\} \tag{3.1}$$

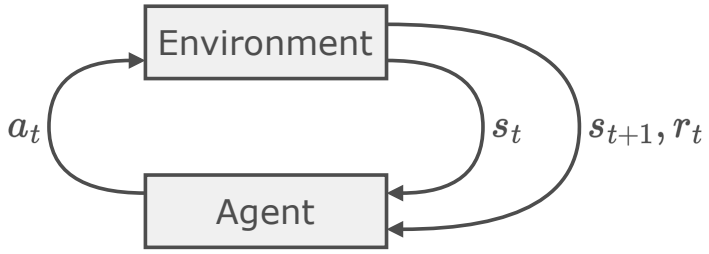

FIGURE 3.1: The interaction between the agent and environment. The agent performs action a_t at timestep t based on the state s_t. The environment then processes the action, and returns the updated features in the form of s_{t+1}, and the reward r_t for performing a_t at s_t. Note that this loop continues indefinitely or until a terminal state is reached.

Given the MDP, the objective of the agent is to maximise its cumulative reward, and doing so will result in solving the problem that the environment represents. However, actions influence not only the immediate reward, but also future rewards. In case of non-episodic tasks (also referred to as continuing tasks), the cumulative reward would approach or become infinite. To prevent this, the discount factor $\gamma \in [0, 1]$ is introduced, with which we can scale the importance of future rewards. For instance, a reward received n steps in the future would be scaled with γ^n. Subsequently, the cumulative reward has a finite value, given that we adhere to some constraints (Sutton and Barto, 1998, p. 55). With this in mind, we can define the discounted cumulative return from timestep t to $t + k$ on-wards as $G_{t:t+k}$:

$$G_{t:t+k} \doteq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^k r_{t+k} = \sum_{n=0}^{k} \gamma^n \cdot r_{t+n}. \tag{3.2}$$

The main idea is that the agent will estimate the discounted cumulative return for a given state, or state-action pair. These estimators are referred to as value functions; the state-value function v and action-value function q. Based upon this estimation, the agent can select the action with the maximal estimated return. This is referred to as the policy π of the agent, which can be either stochastic or deterministic. The optimal policy is denoted by π_*, of which there exists at least one for each MDP (Sutton and Barto, 1998, p. 62). In addition, the optimal value functions are denoted by v_* and q_*.

The state-value function $v_\pi(s)$ is the expectation of G when starting at s and following policy π. Equation 3.3 denotes the state-value function, along with its recursive form (i.e., the Bellman equation).

$$v_\pi(s) \doteq \mathbb{E}_{\pi,\mathcal{T}}\big[G \,|\, s\big] = \underbrace{\mathbb{E}_{a\sim\pi(\cdot\,|\,s),\, s'\sim\mathcal{T}(\cdot\,|\,s,a)}\Big[\mathcal{R}(s,a,s') + \gamma v_\pi(s')\Big]}_{\text{Bellman equation}} \tag{3.3}$$

Action-value function $q_\pi(s, a)$ estimates G given that we perform action a in state s, after which we follow the policy. In similar fashion to the state-value function, $q_\pi(s, a)$ is denoted in Equation 3.4 along with its Bellman equation.

$$q_\pi(s,a) \doteq \mathbb{E}_{\pi,\mathcal{T}}\big[G \,|\, s,a\big] = \underbrace{\mathbb{E}_{s'\sim\mathcal{T}(s,a)}\Big[\mathcal{R}(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi(\cdot\,|\,s')}\Big[q_\pi(s',a')\Big]\Big]}_{\text{Bellman equation}} \tag{3.4}$$

The value-functions can be learned from traces, which are generated by the agent using its policy. A trace consists of the following sequence: $\{s_0, a_0, r_0, s_1, a_1, r_1, s_2, ...\}$. These traces describe the transitions experienced by the agent from its interactions with the environment. A set of traces can be seen as our dataset, similarly to supervised and unsupervised learning.

Given this, the objective of the agent is to obtain the optimal policy through the q- or v-value functions according to the first or second line of Equation 3.5, respectively.

$$\begin{aligned} \pi_* &\doteq \underset{\pi}{\operatorname{argmax}}\, q_\pi(s, a) \\ \pi_* &\doteq \underset{\pi}{\operatorname{argmax}}\, v_\pi(s) \end{aligned} \tag{3.5}$$

3.1.2 Foundations of Value Estimation

In its simplest form, value estimation can be accomplished with tabular methods. Tables are formed to represent, for instance, the entire state-action space, where each cell contains a numerical value corresponding to the q-value of one state-action pair. However, these methods are only feasible for low dimensional spaces. This is due to the fact that the number of states often grows exponentially with the state space, which is referred to as the curse of dimensionality. Additionally, tabular methods are only suitable for discrete state- and action-spaces. In case of a high dimensional state space and/or a continuous search space, approximate methods are preferred.

The foundational methods for value learning can be subdivided into three main approaches: (i) dynamic programming (Bellman, 1966); (ii) Monte-Carlo methods; and, (iii) Temporal Difference learning. We have to note that there is also a fourth approach, exhaustive search, where each state-action pair is fully expanded up to a terminal state. However, this is often not feasible due to the exponential nature of expanding each state with all possible actions. In the remainder of this section, we discuss the three main approaches in its tabular form.

Dynamic Programming. Dynamic programming (DP) requires a perfect model of the environment; the transition probabilities need to be known. With this requirement, we can compute the expectation of the state-value function, for instance. Then, the update rule to approach v_*, which is based on the recursive form of Equation 3.3, is as follows:

$$v_\pi(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p\big(s', r \,\big|\, s, \pi(s)\big) \big[r + \gamma v_\pi(s')\big]. \tag{3.6}$$

Then, using policy evaluation, we iterate over the whole state-space and update the state-value function for every state until convergence. Note that the update rule is partially based on the estimation of v_π itself, which is referred to as bootstrapping.

This update scheme is part of the policy iteration method, where we also use policy improvement after policy evaluation, in order to greedily update the policy based on the updated value-function. The method alternates between policy improvement and evaluation, and guarantees a strict improvement after each iteration (given $v_\pi \neq v_*$).

One of the drawbacks of DP is the fact that it requires a lot of computational efforts (i.e., we have to update every state, even though not all states will be part of the relevant sub-region of the search space). In addition, the transition model is for most environments not known, thus making DP only suitable for specific use-cases.

Monte-Carlo methods. Monte-Carlo (MC) methods make use of traces that are generated by the policy. For each state in a trace, we compute the exact cumulative discounted return when following our policy. This means that, in contrary to DP, Monte-Carlo methods do not bootstrap its values and, subsequently, that the return for every state can be computed only after an episode has terminated.

As a first approach, we can update the value function in an on-policy manner, by generating traces with the target policy (which will approximate π_*) plus ϵ-greedy. Next, after a set of traces have been generated, the action-value or state-value is computed by taking the mean reward over all traces according to first-visit MC or every-visit MC. In case of first-visit, we update the approximator with the mean discounted return starting from the first occurence of a state or state-action pair, up to the terminal state. For every-visit, we divide the cumulative discounted return of all traces by the number of times that a state or state-action pair occurred in said traces.

The second approach, where we use a separate policy to interact with the environment, is called off-policy learning. Moreover, the additional policy is referred to as the behaviour policy π_b. However, the main issue that arises is the fact that a different distribution is used to generate traces, compared to the target policy. This is taken into account by scaling the return for each transition according to Importance Sampling (IS). The scalar $\rho_{t:k-1}$ is computed per transition:

$$\rho_{t:k-1} \doteq \frac{\prod_{l=t}^{k-1} \pi(a_l \mid s_l)\mathcal{T}(s_{l+1} \mid s_l, a_l)}{\prod_{l=t}^{k-1} \pi_b(a_l \mid s_l)\mathcal{T}(s_{l+1} \mid s_l, a_l)} = \prod_{l=t}^{k-1} \frac{\pi(a_l \mid s_l)}{\pi_b(a_l \mid s_l)}. \tag{3.7}$$

Scaling the return is only required for multi-step back-ups, since the distributions between π and π_b are different, with π_b covering π (i.e., if $\pi(a \mid s) > 0$ then $\pi_b(a \mid s) > 0$). For instance, to obtain $v_\pi(s_t)$ we scale the cumulative discounted reward from s_t on-wards with $\rho_{t:k-1}$.

Temporal Difference. Lastly, Temporal Difference (TD) learning also makes use of previously experienced transitions. In its simplest form, TD(0), the learning strategy updates value-estimates over one transition at a time:

$$v(s) \leftarrow v(s) + \alpha \underbrace{\left[r + \gamma v(s') - v(s)\right]}_{\text{TD-error}, \delta}. \tag{3.8}$$

In Equation 3.8, α is the step-size (also known as the learning rate) with which we update the value-estimator. The learning scheme can also be generalised to TD(n), unrolling the discounted reward term to multiple transitions. The TD-error would then become:

$$\delta \doteq G_{t:t+n} - v(s_t) \;\text{ with }\; G_{t:t+n} \doteq r_t + ... + \gamma^{n-1} r_{t+n} + \gamma^n v(s_{t+n}). \tag{3.9}$$

Note that, similarly to Monte-Carlo, we can not immediately update the value estimates. Now, we can update v after the next n transitions have occurred. In addition, also note that TD(∞) is identical to Monte-Carlo.

The two main ways of approximating π_* with the target policy are SARSA and Q-learning, which are on-policy and off-policy methods, respectively.

3.2 Deep Reinforcement Learning

As touched upon in Section 3.1.2, tabular methods are only suitable for discrete state-action spaces and MDP's with a relatively low state-space dimensionality. For high dimensional problems, we have to use function approximators as value function and/or policy. The subdomain deep reinforcement learning (Arulkumaran et al., 2017) provides algorithms which use deep

neural networks as function approximators. An adaptation on TD learning can be used to compute the gradients in order to update the weights.

Variations upon neural networks have been proposed in recent years, such as convolutional layers and recurrent layers. With convolutional layers, RL algorithms are able to directly learn from images by essentially abstracting them into denser representations with lower dimensionality, thus lowering the computational efforts and reducing the effects of the curse of dimensionality. Recurrent layers have proven to be useful when the dynamics of the MDP follow some recurring pattern, such as power grid tariffs that are lower during night-time, or the amount of PV-radiation (zero during nighttime, high around noon).

3.3 Model-Based Reinforcement Learning

Model-based reinforcement learning (MBRL) (Moerland et al., 2020; Plaat et al., 2021) methods learn (or receive) an internal model[1] that models the dynamics of the MDP. For instance, it can be represented as a deep neural network, that receives as input a state-action pair, and returns the reward and next state. The main benefits of the internal model are three-fold:

1. **MBRL is more sample efficient than model-free RL algorithms.** With model-free RL, experience is directly used to train the policy and/or value-functions, after which it is thrown away at the end of the episode (or eventually replaced in the replay buffer). In contrast, MBRL methods train the internal model on this experience, such that the transitions are preserved in the model. Overall, the MBRL methods will have a higher sample efficiency, which is especially beneficial when sampling from the environment is expensive.

2. **MBRL enables the use of planning**: decision-time planning can be used to map out the possible next actions over a longer horizon than the traditional one-step planning. For instance, a search tree can be formed where each path from the root (the current state in the episode) to a leaf-node represents an imaginary trace of actions and states. Then, we can assign the expected return to each leaf node using rollouts and back up the value to the corresponding action below the root. Moreover, a variation on this is the Monte-Carlo Tree Search (Chaslot, 2010), where we have a dynamic depth and breadth of the search tree, and expand nodes that look promising.

   Lastly, we can also use Model Predictive Control (MPC) (Camacho and Alba, 2013) instead of building a search tree. With MPC, we continuously plan over a constant short horizon, select the first action of the most promising trajectory based upon some metric, and then start planning again over the short horizon.

3. **Deploying two-phase exploration**: differing exploration schemes can be used during background and decision-time planning. For instance, when we are training the internal model, more exploration during data gathering might be beneficial such that a more diverse subset of the state-action space will be seen by the dynamics model. However, more exploitation could be beneficial during decision-time planning, since we are only interested in expanding actions that look promising.

The main drawback of MBRL is the fact that it adds another layer of complexity on top of learning a policy/value function, in case the internal model is not given. The model may not be accurate over the whole search space and may introduce bias in the sub-regions with little to no datapoints. This can lead to model exploitation by the policy, where the internal model gives higher rewards for transitions in comparison to the environment itself. In other words,

[1] Also referred to as the dynamics model

during training with the imagined traces, it will seem that the policy is performing well and receives high rewards, however this performance does not transfer to the real environment.

3.4 Offline Reinforcement Learning

Offline reinforcement learning (Lange et al., 2012; Levine et al., 2020) deals with problem settings where there is no traditional environment/simulator available to generate traces with. Instead, we receive a static finite dataset $\mathcal{D} = \left\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\right\}_{i=1}^N$ containing a set of traces throughout the search space. These traces are generated with a data-collection policy π_b, such as a handcrafted heuristic. In short, offline RL deals with methods that have a deployment efficiency of one, while its counterpart, online RL, may require thousands up to tens of millions deployments.

The main objective of offline RL is to infer what would happen if the policy diverges from the traces and traverses to unsupported regions of the search space. This is required, since we would otherwise train the policy to resemble the data-collection policy, with as consequence that the policy would not be able to outperform π_b. In addition, the more the policy is trained, the more it will deviate from the traces as it will become increasingly different from π_b. This is referred to as the distributional shift, and poses the main challenge of offline RL.

Without taking the distributional shift into account, the agent would become unstable in the unsupported regions. In these regions, the actions are based upon the knowledge acquired from the static dataset, and may not apply to the out-of-distribution states. Consequently, the agent would essentially perform random actions. Instead, we have to incorporate the distributional shift into the RL method. For instance, the target policy can be constrained to π_b (e.g., forming trust regions), or the epistemic uncertainty can be estimated and taken into consideration during the decision making process.

An additional issue is the evaluation of the policy. Since we are not able to deploy the policy into the environment, and with it producing differing traces from the static dataset, we do not have explicit rewards. One option is to use the offline traces, in combination with IS, to evaluate a candidate policy. Here, we multiply the discounted cumulative reward from the offline traces with the ratio ρ, computed according to Equation 3.7. There exist variations upon IS, such as weighted, per-decision and marginalised IS. In addition, the IS-ratio can also be directly incorporated into the policy gradient estimation. However, since the behaviour and candidate policy will become increasingly different, the importance weights will have too much variance to be useful for high dimensional state-action spaces and over long horizons (Levine et al., 2020). Thus, in case of utilising IS, the candidate policies have to be constrained by π_b to some degree in order to limit the variance.

Model-based RL has often been applied to the offline paradigm. The internal model augments the static dataset such that the RL-agent is less constrained by the dataset. Inferring the unsupported regions can be more accurately done by learning the environment dynamics, due to the fact that environments are often based on the real world. Thus, the transitions follow some form of system (e.g., laws of physics, mathematics, etc.). Consequently, the model should be able to infer unknown sub-regions accurately from the provided datapoints.

4 Problem Setting

As discussed in Section 1, Behind-the-Meter (BtM) smart batteries play an essential role in the energy transition. In the upcoming years, more (residential) buildings will be equipped with photovoltaic (PV) panels, consistently generating electricity. However, in case of a surplus of PV-generated power and an absence of a smart battery, the surplus can often not be netted back to the power supplier due to the grid already operating at its maximum capacity. Subsequently, the PV-panels will instead be turned off, resulting in wasting PV-power. By deploying a smart battery along with a Home Energy Management System (HEMS), the utilisation of PV-power can be maximised by managing the power flow around the battery.

This section contains the components of the EcoGenie environment, along with its definition in MDP-form. Lastly, we cover the datasets used by the environment.

4.1 EcoGenie Environment

The EcoGenie environment models a residential household equipped with a smart battery and heatpump (HP). It specifically models the properties and circumstances of Shell's EcoGenie house in The Hague. The HEMS controls the power flow around the battery, and is able to select the operation of the battery on an hourly basis. In addition, the heatpump will be continuously demanding power in order to maintain a comfortable indoor temperature which is specified by the User Profile (UP). Given this, the main objective of the HEMS is to minimise the costs made by consuming power from the grid, which can be achieved by maximising the PV-utilisation and drawing power from the grid at low tariffs. However, to prevent the HEMS from unnecessarily using the battery, prolonging the battery lifespan has been added as a secondary objective.

Action Space. The action space $\mathcal{A}$ is discrete, consisting out of five actions, where each action corresponds to an operation of the smart battery. Table 4.1 denotes the five possible actions $a \in \mathcal{A}$ can take.

TABLE 4.1: Discrete action space $\mathcal{A}$ of the EcoGenie environment.

Action	Description
`IDLE`	Do not charge nor discharge the battery.
`CHARGE_GRID`	Charge the battery with power from the grid.
`CHARGE_PV`	Charge the battery with power generated by the PV-panels.
`DISCHARGE`	Supply power from the battery to the heatpump.
`NETTING`	Sell power stored in the battery back to the supplier.

Note that concurrently charging and discharging (i.e., float charging) is not supported, since float charging severly deteriorates the battery's state of health (Wang et al., 2011). In case `DISCHARGE` is not selected, the HP power requirements will be met by drawing power from the grid. This is to ensure that the user comfort is maintained at all times. In addition, the (dis)charge rates are maximised, and constrained by the SoC and maximum (dis)charge rates of the battery itself. Important to note is that the heatpump can only receive power from one

source at a time, which means that if the action `DISCHARGE` is selected despite it having insufficient power, the power will instead be entirely drawn from the grid.

For the `NETTING` action, we have to adhere to a regulation: when the amount of netted kWh's exceeds a threshold, we receive a 'reasonable' compensation per kWh instead of the grid tariff. The threshold is set to a fraction of the current amount of power consumed from the grid. For example, if we net 5 kWh while being 3 kWh below the threshold, then we receive for 3 kWh the current tariff, and for the remaining 2 kWh the reasonable compensation. In The Netherlands, netting for the actual tariff is expected to be phased out from 2025 to 2030 (Wilt, 2022), after which a household can only receive the reasonable compensation. In our problem setting, the fraction will be set to 1.0.

State Space. The state space $\mathcal{S}$ consists out of 17 features, which are denoted in Table 4.2. The environment is partially based on historical data, namely the weather and pricing features are sampled from a dataset. For these features holds that we have one trace of the weather and pricing features available, and subsequently an entry of the dataset will always be followed by the same next entry. The remaining features are computed according to a set of dynamics rules.

The EcoGenie environment is not a traditional online environment, since it contains an offline component in the form of the weather and tariff features. For these features, we have a finite number of datapoints, and they are independent on the actions. Thus, not the whole state space is covered by the environment, given that the dataset is finite. We define this as a pseudo-environment, since we are not able to continuously sample from the environment in order to lower the epistemic uncertainty of an algorithm. However, the exploration-exploitation trade-off is still something that has to be considered, unlike traditional offline settings, since an agent still has to gather its transitions.

TABLE 4.2: State space $\mathcal{S}$ of the EcoGenie environment. The '†' denotes features that are sampled or inferred from a historical dataset. For the outdoor temperature and PV production, we plug in the values of the previous timestep, since those of the current can not be known by the HEMS.

Feature	Symbol	Unit	Description
Battery State of Charge	SoC_t	kWh	The amount of charge in the smart battery.
Battery State of Health	SoH_t	%	Estimation of the health, based on discharge cycles.
Indoor temperature	T_t^{in}	°C	Temperature of residential house.
Outdoor temperature†	T_{t-1}^{out}	°C	Temperature outside the residential house.
Forecasted outdoor temperature†	T_t^f	°C	Mean forecasted outdoor temperature over the next 24 hours.
Heatpump input power	$\text{HP}_{t-1}^{\text{in}}$	kWh	Number of kWh inserted into the heatpump.
Heatpump output power	$\text{HP}_{t-1}^{\text{out}}$	kWh	Heat produced by the heatpump.
PV production†	PV_{t-1}	kWh	Available power generated by the solar panels.
Forecasted PV radiation†	PV_t^f	kW	Forecasted cumulative PV radiation over the next 24 hours.
Grid tariff†	p_t	€/kWh	Tariff of drawing/netting 1 kWh from/to the power grid.
MA grid tariff†	p_t^{MA}	€/kWh	Moving average (MA) of grid tariff over the current episode.
Mean grid tariff day-ahead†	p_t^{μ}	€/kWh	Mean grid tariff for day-ahead market.
Min. grid tariff day-ahead†	p_t^{min}	€/kWh	Minimum grid tariff for day-ahead market.
Max. grid tariff day-ahead†	p_t^{max}	€/kWh	Maximum grid tariff for day-ahead market.
Netting	η_t	-	Boolean indicating whether we can net for the full tariff.
Workday	-	-	Boolean indicating whether it is currently a workday.
Hour	-	-	Denoting hour of the day.

Reward Function. The reward function $\mathcal{R}$ consists out of two components: (i) the costs c made over the previous timestep; and, (ii) the deterioration of the battery SoH as a result of performing the selected action. The reward function is denoted in Equation 4.1, where λ_{SoH} is an importance scalar for the SoH, κ the cost per percent of SoH, and $\lambda_{w,t}$ a scalar for the overall reward. The cost per percent of SoH is computed by dividing the rated number of discharge

cycles with the cost of purchase.

$$r_t \doteq \big(-c_t - \lambda_{\text{SoH}} \cdot \kappa \cdot \text{SoH}_\Delta \big) \cdot \lambda_{w,t} \;\text{ with }\; \text{SoH}_\Delta \doteq \text{SoH}_{t-1} - \text{SoH}_t \tag{4.1}$$

The scalar $\lambda_{w,t}$ is added to the reward function to penalise the agent when an action was selected that did not change the battery SoC, thus wasting an action. Its value at a given timestep t is determined based on Equation 4.2.

$$\lambda_{w,t} = \begin{cases} m, & \text{if } a_t \in \mathcal{A} \setminus \{\texttt{IDLE}\} \text{ and } \text{SoC}_\Delta = 0 \\ 1, & \text{otherwise} \end{cases} \tag{4.2}$$

Overall, the EcoGenie environment consists out of both stochastic and deterministic elements, e.g.: the weather and battery dynamics, respectively. However, the environment has no terminal states (i.e., continuing task), and instead episodes will be ended after a predefined number of timesteps. An overview of the environment dynamics is shown in Figure 4.1.

Two of the main components of the environment, the battery and user profile, are discussed in further detail in the next subsections. In addition, Appendix A denotes the values for the constants that are part of the environment (e.g., battery capacity, battery purchase price, maximum PV-panel generation, etc.). The dynamics of the heatpump are described in Appendix B, and are based on an environment developed internally at Shell.

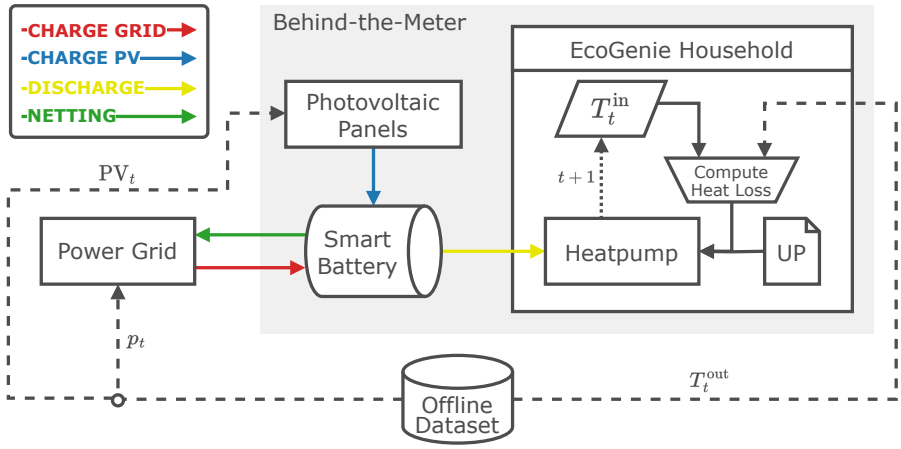

FIGURE 4.1: Overview of the EcoGenie environment, which models the battery dispatch problem setting on a smaller scale. The heatpump demands power in order to meet the comfort requirements set by the User Profile (UP), and the HEMS should efficiently manage the power flow such that the consumption costs are minimised and the battery lifespan is maximised. As shown, a subset of the environment is based on an offline dataset, which is highlighted with the dashed arrows. Note that the environment consists of five discrete actions; the one not shown in the figure is `IDLE`.

4.1.1 Smart BtM Battery

The sonnenBatterie10 [1] is used as reference for the smart battery dynamics in the environment, which consists out of $LiFePO_4$ energy cells. While simulating the battery, we take the limits on charge/discharge rates, battery capacity and the effect of float charging into account. The

[1] Specifications of the sonnenBatterie10 can be found at `https://sonnenbatterie.co.uk/products/sonnenbatterie-10/`

maximum charge and discharge rates are constants, and are conservatively approximated due to the potential losses that can occur during the charge- and discharge-operations. In order to preserve the battery health, it is not possible to fully charge the battery nor let it get fully depleted.

Lastly, the SoH of the battery is estimated by the number of discharge cycles it has endured, compared to the number of discharge cycles it is rated for. One discharge cycle is defined as discharging the battery for its maximum capacity. This does not have to be uninterrupted nor from maximum to minimum capacity.

We have to note that this is a simplistic approach to model a battery, and there are numerous complex reactions/processes occurring in a battery that influence the mechanics. For instance, operating and ambient temperature, capacity fade, and depth of discharge all influence the performance and lifespan of the battery (Wang et al., 2011; Zhang, 2011), but have not been implemented in the environment.

4.1.2 User Profile

The comfortable indoor temperature is dynamic and changes based on whether it is day- or nighttime. A crossover point from day to night to day is sampled for each day from a Gaussian distribution with a variance of 0.5 and mean $\tilde{t}^{\mu}$. Night starts at $\tilde{t}_{\text{night}}$ and ends at $\tilde{t}_{\text{day}}$. At nighttime, the comfortable temperature is lowered with a constant offset (see Equation 4.3[2], where $\tilde{t}$ refers to the actual time, instead of a timestep). With this, we simulate how households would manually operate the thermostat; lower temperature before going asleep, and normal during daytime. However, we have to note that this approach does not take into account that a residential house may be unoccupied at certain moments in time, such that during daytime the thermostat is also set to a lower temperature.

$$T_{i,t}^{\text{comfort}} \doteq T_i^{\text{comfort}} - T^{\text{offset}} \cdot \mathbb{1}\{\tilde{t}_{\text{night}} \leq \tilde{t}_t < \tilde{t}_{\text{day}}\},\ i \in \{\text{low}, \text{high}\} \tag{4.3}$$

4.2 Data

As Table 4.2 denotes, some features of the state space are sampled from a static offline dataset. The outdoor temperature and PV production are sampled from the National Solar Radiation Database (NSRDB) of the National Renewable Energy Laboratory (NREL)[3], which is part of the U.S. Department of Energy. As for the grid tariffs, the day-ahead market prices, which are being reported by ANWB Energie[4], have been used. In addition, weather forecasts for temperature and PV production are sampled from the ERA5-Land dataset (Sabater et al., 2021), which can be obtained from the Climate Data Store (C3S, 2022). For all datasets holds that we retrieved the data that is applicable to the EcoGenie house over the years 2017, 2018 and 2019. They have identical granularity, i.e.: a datapoint every hour. Thus, no further transformations were required to the data.

In the next subsections, we discuss some noteworthy signals and patterns that were found during data exploration. These insights indicate that the use of certain techniques or features may result in performance boosts of a HEMS.

[2]In this context, the semantic meaning of '$<$' and '$\leq$' is changed from *smaller than* to *is before*.

[3]The data viewer of NSRDB can be accessed here: `https://nsrdb.nrel.gov/data-viewer`

[4]The day-ahead market prices can be found here: `https://energie.anwb.nl/actuele-tarieven`. However, we have to note that the historical prices have been scraped from the API that is provided by EnergyZero (`https://api.energyzero.nl/v1/`).

4.2.1 NSRDB Weather

The relevant attributes that have been sampled from the NSRDB dataset are the outdoor temperature in degrees Celcius and Global Horizontal Irradiance (GHI) in kWh/m^2. GHI denotes the total amount of terrestial irradiance received by a surface horizontal to the ground, and includes solar radiation received directly (Direct Normal Irradiance) and indirectly (Diffuse Horizontal Irradiance) from the sun. We argue that GHI is a reasonable, but conservative, value to take as the potential PV-production, taking into consideration that the efficiency of commercial PV panels is around 15% to 20% and that there is around 10 m^2 of PV-panels installed at the EcoGenie house.

Figure 4.2 shows the boxplots for the hourly temperature and GHI attributes over the three years. The plots highlight how there is a clear pattern occurring for both attributes; the GHI peaks around noon, while the temperature is higher during daytime, unsurprisingly. The trends highlight that a RL method might benefit from recurrent layers, in order to make more accurate predictions based on the last n timesteps. In addition, adding the hour-feature into the state space should make it easier to predict when the sun will rise and set, and combine it with the GHI and temperature features to identify whether it is a sunny or cloudy day.

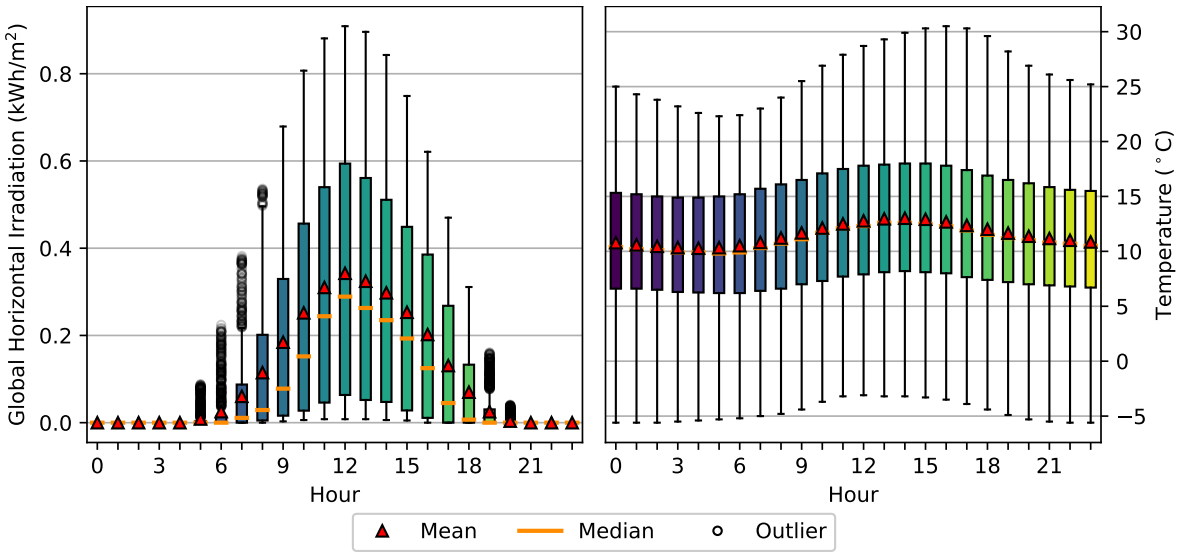

FIGURE 4.2: The hourly Global Horizontal Irradiation (GHI) (left) and temperature (right) over 2017, 2018 and 2019. Noteworthy is the number of outliers in the GHI boxplot, which can be explained by switching between daylight saving time and standard time. The darkness of an outlier denotes the frequency of it occurring.

4.2.2 ERA5-Land Weather Forecasts

Next to the true weather data, we also provide weather forecasts to a HEMS. The forecasts are made by a numerical model (i.e., Numerical Weather Predictions, NWP), and in case of the ERA5-Land dataset (Sabater et al., 2021) are based on a reanalysis of historical weather data. From this, we have taken the outdoor temperature and GHI at The Hague. Table 4.3 denotes four error metrics on the accuracy of the forecasts, in comparison to the ground truth weather data.

The forecasted temperature mostly follows the true temperature, as indicated by a root mean squared error (RMSE) of 1.35. However, the peaks of the forecasted GHI are often more

extreme, given that the maximum recorded error over one hour is 716.32 kWh/m^2, compared to the true GHI. Fortunately, the maximum mean error over 24 hours is significantly lower. Given these statistics, we argue that the features regarding weather forecasts, T_t^f and PV_t^f, are reasonable approximations of the true weather for the upcoming day.

TABLE 4.3: Four metrics on the weather forecast data. The maximum error over one hour denotes the maximum absolute difference of one datapoint. In addition, the max. and min. mean error over 24 hours is denoted in the rows below.

Metric	Temperature (°C)	GHI (kWh)
RMSE	1.35	110.59
Max. error (1 hour)	7.33	716.32
Max. mean error (24 hours)	3.68	170.35
Min. mean error (24 hours)	0.01	0.00

4.2.3 Day-Ahead Market Tariff

Lastly, the tariff data is based on the Day-Ahead (DA) Market taken from ANWB Energie. In this market, the hourly tariffs of the next day are made public one day beforehand at 15:00 CET.

Figure 4.3 shows two boxplots with the hourly pricing data binned per hour and weekday. Most notably is the fact that the power grid pricing is quite volatile. For instance, the outliers can be up to five times as high as the mean found for an hour or weekday. Similarly, the pricing can also be significantly lower, or even negative. In case of negative pricing, a residential household would get paid to consume power from the grid.

In addition, there are clear trends visible in the boxplots. Grid tariffs are on average higher during the hours 6 to 10 and 15 to 18, which is when the demand on power is at its highest, and the generation of PV-power relatively low. The valley in between the peaks is the result of the peak in PV-radiation, during which there is often a surplus of power. As for weekdays, the tariffs are slightly lower during weekends, and with less extreme outliers.

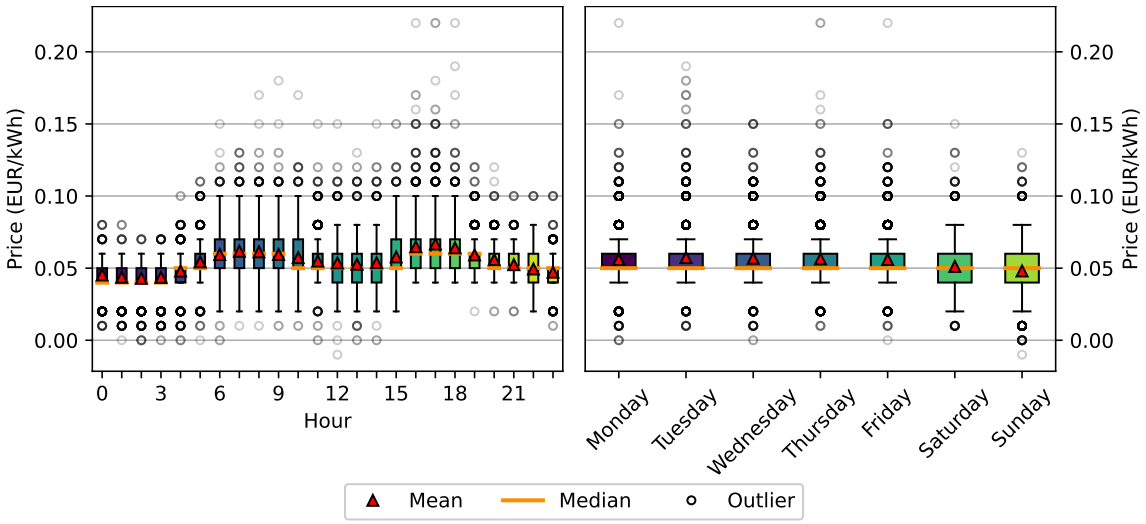

FIGURE 4.3: Boxplots for the hourly pricing (left) and daily pricing (right) over 2017, 2018 and 2019. The darkness of the outliers denotes its frequency. Note how the range over the hours 6 to 10 and 15 to 18 are significantly wider, and also with more extreme outliers. As for the datapoints binned per weekday, the weekend has slightly lower tariffs compared to workdays.

5 Multi Dynamics- and Q-Learning

We now turn our attention to a novel reinforcement learning approach, that has been applied to the EcoGenie environment introduced in the previous section. It is, next to benchmarking reinforcement learning to energy management, one of the main contributions of this work.

Sharing state representations between multiple value- and/or policy-approximators has shown promising results in the reinforcement learning (RL) paradigm. As the main benefit, the state representations are generalised over the subsequent function-approximators, providing rich latent states. In addition, fewer weights are required by the overall model.

Research has been dedicated to network architectures where state predictors/discriminators are used as a way of regularising the shared weights. For instance, Leibfried and Vrancx (2018) empirically demonstrate that next state predictions offer useful signals to a q-value approximator in environments with sparse rewards. Similarly, Moon et al. (2022) show that adding an auxiliary task to the objective function, in the form of discriminating true and false next states, can improve generalisation when training an agent on multiple environments concurrently.

On the other hand, Oh et al. (2017) propose the Value Prediction Network, where state abstractions are learned. The reward, discount factor next abstract state (given an action), plus its state-value, are predicted based upon the current abstract state. Subsequently, it enables the model to unroll a state multiple steps, thus being able to deploy a planning strategy, such as Monte-Carlo Tree Search (Chaslot, 2010).

Lastly, it has been shown that ensembles are able to accurately quantify the epistemic uncertainty that arises in offline RL settings (Kidambi et al., 2020; Yu et al., 2020b). However, sharing the weights between the ensemble members can theoretically reduce the effect of bootstrapping over the members, thus reducing its capabilities to detect sparse sub-regions in the state-action space. However, Osband et al. (2016) empirically show with Bootstrapped DQN that this effect is neglegible. In addition, sharing weights over an ensemble of q-value approximators (Smit et al., 2021) has also shown to yield performance gains in the offline RL setting.

In this work, we take it a step further by proposing to share the state representations between two ensembles; ensembles of q-value and dynamics approximators. Subsequently, the feature extractor (i.e., the layers that are shared between the ensembles) is forced to generalise over the multiple beliefs of the ensemble members, while also providing rich latent states which can be linearly mapped to the next state and immediate reward. We refer to this model as Multi Dynamics- and Q-Learning (MDQL).

In addition, the novel network architecture enables the use of a novel planning strategy, Hindsight-Weighted Planning (HWP). The strategy consists out of weighted rollouts by pairs of q-value approximators and dynamics models.

5.1 Architecture

MDQL consists out of two ensembles along with a shared feature extractor $\mathcal{F}$, with each ensemble consisting out of n members. In addition, there is also a separate reward function approximator that is shared by all ensemble members. The feature extractor converts a state

s_t to a latent state, which is fed into a function approximator. As for the ensembles, the first one consists out of q-value approximators, which are all modelled as a Double DQN (DDQN) (Hasselt et al., 2016). Each DDQN receives the latent state, with which it predicts the q-value for each action.

The second ensemble $\hat{D}$ consists out of dynamics models, which receive the latent state concatenated with an action in its one-hot encoding form. Each member predicts the state difference s_Δ, given that we perform action a_t. Next, there is a shared reward function approximator $\hat{R}$, which predicts the immediate reward when action a_t is executed at state s_t.

In addition, we also propose a recurrent variation upon MDQL, namely rMDQL. In this variation, we replace the fully connected layers in $\mathcal{F}$ with LSTM layers (Sak et al., 2014). Figure 5.1 visualises the network architecture of MDQL. Each fully connected layer (bar the final layer) is followed by the ReLU activation function. In addition, Dropout-layers (Hinton et al., 2012) have been added to the dynamics models and reward approximator, in order to minimise overfitting on the training transitions.

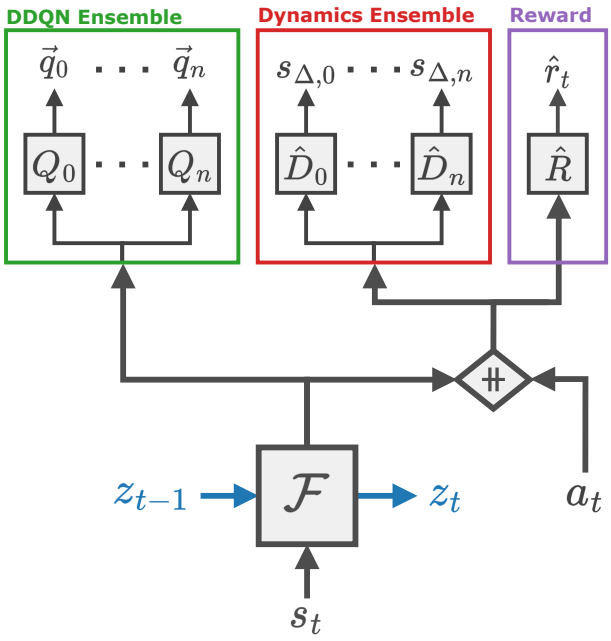

FIGURE 5.1: Network architecture of Multi Dynamics- and Q-Learning. The algorithm makes use of shared state representations, which are generated by feature extractor $\mathcal{F}$. The ensemble highlighted in **green** consists out of heads Q_i that estimate the q-values of a state. The other ensemble, highlighted in **red**, contains the heads $\hat{D}_i$ that predict the state difference s_Δ. Lastly, there is a shared reward function approximator $\hat{R}$, highlighted in **purple**. The latent state and action are concatenated, which is denoted with '$\#$'. In addition, we also propose the recurrent variant of MDQL, namely **rMDQL**, with LSTM layers in $\mathcal{F}$.

All networks are parameterised by θ. In addition, a set of target networks have been created for the Q-heads and $\mathcal{F}$, which are parameterised by θ'. The predictions that MDQL, and its recurrent counterpart, can make are summarised in Equation 5.1. The next state can then be obtained as follows: $s_{t+1} = s_t + s_\Delta$.

$$\begin{aligned} Q_{\theta,i}\big(\mathcal{F}_\theta(s_t, z_{t-1})\big) &\mapsto \vec{q}_i, z_t \\ \hat{D}_{\theta,i}\big(\mathcal{F}_\theta(s_t, z_{t-1}), a_t\big) &\mapsto s_{\Delta,i}, z_t \\ \hat{R}_\theta\big(\mathcal{F}_\theta(s_t, z_{t-1}), a_t\big) &\mapsto \hat{r}_t, z_t \end{aligned} \tag{5.1}$$

For all upcoming equations and notations, we have omitted the hidden states z unless it is specifically aimed at rMDQL.

Next, we form pairs between the two ensembles that share the same replay buffer, for instance, pair $(Q_0, \hat{D}_0)$ share replay buffer $\mathcal{D}_0$. Subsequently, we will have n separate replay buffers for experience replay. For bootstrapping, we divide the transitions over the n buffers, where each transition is represented as a tuple $(s_t, a_t, r_t, d_t, s_{t+1})$, with d_t being the terminal flag.

However, in case of rMDQL, we create transitions of l steps. This results in tuples of the form $(s_t, a_t, r_t, d_t, s_{t+1}, a_{t+1}, r_{t+1}, d_{t+1}, s_{t+2}, ..., s_{t+l})$. This is to ensure that there is no overlap between the replay buffers, and that rMDQL is trained on traces of at most l steps.

At testing time, the next action is selected based on the aggregation of the Q-heads. As aggregation strategy, we take the Confidence Lower Bound (CLB) over the estimated q-values, which we refer to with $\vec{q}_{\text{CLB}}$. Given this vector, we take the action with the maximum CLB value. Consequently, we penalise actions with high disagreement between the Q-heads, which should be higher in state-action regions with sparse datapoints. Thus, we penalise actions that have high epistemic uncertainty.

The CLB q-values are computed according to Equation 5.2, where $\vec{q}_\mu$ and $\vec{q}_\sigma$ are the mean and standard deviation over the q-values of the Q-ensemble, respectively. The penalty term has been taken from the work conducted by Smit et al. (2021).

$$Q_{\text{CLB}}\big(\mathcal{F}_\theta(s)\big) \doteq \vec{q}_{\text{CLB}} = \vec{q}_\mu - \vec{q}_\sigma \tag{5.2}$$

Intuitively, state-action pairs with a high epistemic uncertainty will more likely have outliers for the q-values. The standard deviation captures this, and thus can act as a way to quantify this uncertainty.

5.2 Learning

In this section, we cover the components related to optimizing the MDQL; the loss-functions, exploration strategy, maintaining high diversity in the replay buffers and overall training process.

Loss Functions. The ensembles of MDQL are optimised through two separate loss functions; $\mathcal{L}_Q$ and $\mathcal{L}_{\hat{D}}$ for the Q- and $\hat{D}$-heads, respectively. The Q-loss function largely follows the default DDQN loss function, with some slight adaptations to accommodate for $\mathcal{F}$.

In addition, the reward approximator is optimised on its own loss function $\mathcal{L}_{\hat{R}}$, and is trained to predict the normalised reward. We normalise the reward through min-max feature scaling, where the minimum and maximum are based upon multiple runs conducted in the environment (denoted in Appendix A). All three loss functions are denoted in Equation 5.3.

$$\begin{aligned}
\mathcal{L}_{Q_i} &\doteq \text{smooth}_{\ell_1}\Big(r + d \cdot \gamma \cdot Q_{\theta',i}\Big(\mathcal{F}_{\theta'}(s'), \underset{a'}{\text{argmax}}\, Q_{\theta,i}\big(\mathcal{F}_\theta(s'), a'\big)\Big) - Q_{\theta,i}\big(\mathcal{F}_\theta(s), a\big)\Big) \\
\mathcal{L}_{\hat{D}_i} &\doteq \text{smooth}_{\ell_1}\Big(\hat{D}_i\big(\mathcal{F}_\theta(s), a\big) + s - s'\Big) \\
\mathcal{L}_{\hat{R}} &\doteq \text{smooth}_{\ell_1}\Big(\hat{R}\big(\mathcal{F}_\theta(s), a\big) - r\Big)
\end{aligned} \tag{5.3}$$

Note that at this point in time, the dynamics models are mainly used for representation learning by the feature extractor. Thus, there is no background planning occurring during training. Moreover, we have opted for the smooth_{ℓ_1} loss function since it is less sensitive to outliers than

more traditional regression losses, such as the MSE loss. Consequently, this property results in less severe (if at all) exploding gradients (Girshick, 2015). Given the separate losses, the feature extractor $\mathcal{F}$ is optimised over the summation, see Equation 5.4.

$$\mathcal{L}_{\mathcal{F}} \doteq \sum_{i=0}^{n-1} \left[\mathcal{L}_{Q_i} + \mathcal{L}_{\hat{D}_i} + \mathcal{L}_{\hat{R}} \right] \tag{5.4}$$

We also considered taking as target q-value an aggregation strategy over the whole Q-ensemble, similar to PEBL (Smit et al., 2021). However, this results in less distinctions between the Q-heads, thus less accurate quantifications of the epistemic uncertainty. We provide empirical evidence on this in Appendix E.

Lastly, the loss functions of rMDQL require a slight adaptation. Given the l-steps that occur in one sample of the replay buffer, we still compute the losses between one step at a time. However, we take the average loss over the l steps in order to reduce the gradient updates. Following this, the loss function is the mean over all 1-step losses.

Exploration. Since we are dealing with a pseudo-environment (see definition in Section 4.1), the algorithm still has to collect its transitions according to a strategy in order to optimise the ensembles efficiently. Thus, the exploration-exploitation trade-off is still present in the EcoGenie environment. As exploration strategy, we opted for ϵ-greedy with exponential decay. The training samples are gathered from the environment by one Q-head at a time, while in addition we alternate between Q-head after every interaction. Consequently, there is also implicit exploration occurring, since the state Q_i receives is dependent on the actions taken by the preceding Q-heads. Then, when head Q_i has interacted with the environment, the resulting transition is stored in replay buffer $\mathcal{D}_i$.

Maintaining Data Diversity. The network architecture restricts us in how the weights can be optimised. The traditional approach is to first train the dynamics models on the available data, after which the dynamics models remain frozen while the policy- and/or value-approximator is optimised. In our case, both ensembles have to be optimised concurrently, since it is sharing a set of weights. This makes the overall training process more complex, since the Q-ensemble will converge to a set of optimal trajectories through the environment. Subsequently, the replay buffers will be filled with more similar transitions over time, reducing the data diversity for the dynamics models, with the risk of forgetting the suboptimal transition dynamics.

We propose to split up the training procedure into two phases; Q-Ensemble Training (QET) and Diverse Dynamics Training (DDT). During Q-Ensemble Training, we primarily focus on obtaining a high cumulative return through the Q-ensemble. All components are trained on transitions generated by the Q-ensemble plus ϵ-greedy. During this phase, we keep track of the mean dynamics model loss over the last ten evaluations on the validation environment. When the loss did not improve over this period, we can say that the Q-ensemble has somewhat converged. This can be inferred from the fact that once the dynamics models do not improve in terms of loss on the validation environment, it means that the Q-ensembles produced near-identical traces during the evaluations. Once the convergence point has been reached, the algorithm switches to the second phase.

With Diverse Dynamics Training, MDQL creates a new replay buffer for each pair of the ensembles, which is substantially larger than the buffers from the first phase. These buffers are immediately filled with new transitions. Every 10% of the buffer is filled with traces from the training environment with a unique $\epsilon \in \{0.1, 0.2, ..., 1.0\}$. Subsequently, each replay buffer will

contain a diverse set of transitions. Next, each pair (along with the reward approximator) is then trained on its new replay buffer for the remainder of the training process.

To ensure that the Q-ensemble does not deteriorate, the target update coefficient is set to zero, such that each DDQN is still able to finetune its one-step estimations, but will not be influenced by the suboptimal traces. As an additional benefit, it prevents the Q-heads from overestimating its state-action values. Experiments have shown that Q-learning is prone to diverging in this environment. More details on this are discussed in Appendix E.

Training Process. With the environment being based on three years of historical data, we opted to not use the dynamics models in order to generate new samples and possibly improve the sample efficiency. The overall training process is described in Algorithm 1, where the **blue** highlighted parts are only applicable to rMDQL. In this context, T is the total number of timesteps MDQL is trained on, and I and E are the frequency of updating the target networks with τ and evaluating the model on the validation environment, respectively.

Algorithm 1: Recurrent Multi Dynamics- and Q-Learning

Input : Empty buffers $\mathcal{D}^{\text{QET}} = \{\mathcal{D}_0^{\text{QET}}, ..., \mathcal{D}_{n-1}^{\text{QET}}\}$, randomly initialised θ
Output : θ^*

Partially fill $\mathcal{D}_0^{\text{QET}}, ..., \mathcal{D}_{n-1}^{\text{QET}}$ with random transitions
for $t = 0, ..., T$ **do**
  if phase is QET **then** ▷ Q-Ensemble Training (QET)
    $i \leftarrow t \bmod n$ ▷ Select next head to interact with the environment
    Reinitialise z with previous $l - 1$ states
    Gather trace with $\mathcal{F}_\theta, Q_{\theta,i}$ and ϵ-greedy
    $\mathcal{D}_i^{\text{QET}} \leftarrow (s_t, a_t, r_t, d_t, s_{t+1}, ..., s_{t+u})$
    $\mathcal{D} \leftarrow \mathcal{D}^{\text{QET}}$
  end
  else ▷ Diverse Dynamics Training (DDT)
    $\mathcal{D} \leftarrow \mathcal{D}^{\text{DDT}}$
  end
  if $t \bmod n = 0$ **then** ▷ Update Online Networks
    $\mathcal{L}_\mathcal{F} \leftarrow 0$
    forall $(Q_i, \hat{D}_i)$ pairs **do**
      $B_i \sim \mathcal{D}_i$ ▷ Sample Batch B_i
      Compute $\mathcal{L}_{Q_i}, \mathcal{L}_{\hat{D}_i}, \mathcal{L}_{\hat{R}}$ with B_i, via Eq. 5.3
      Add $\mathcal{L}_{Q_i} + \mathcal{L}_{\hat{D}_i} + \mathcal{L}_{\hat{R}}$ to $\mathcal{L}_\mathcal{F}$
    end
    $\mathcal{L}_\mathcal{F} \leftarrow \mathcal{L}_\mathcal{F} / l$ ▷ Compute mean loss over l steps
    Update θ via backpropagation on $\mathcal{L}_\mathcal{F}$
  end
  if $t \bmod I = 0$ **then** ▷ Update Target Networks
    $\theta' \leftarrow \theta' \cdot (1 - \tau) + \theta \cdot \tau$
  end
  if $t \bmod E = 0$ **then** ▷ Evaluate MDQL
    score $\leftarrow$ evaluate(θ)
    phase, τ, $\mathcal{D}^{\text{DDT}} \leftarrow$ is_converged(score) ▷ Check whether Q has converged. If so: switch to DDT, set τ to zero and create DDT replay buffer
  end
end

5.3 Hindsight-Weighted Planning

The dynamics models enable the agent to plan its actions ahead at testing time, as an alternative to selecting the action with the highest q_{CLB}-value. However, the accuracy of one dynamics model may fluctuate significantly during evaluation. This can occur when the distribution of datapoints occurred such that its density in certain parts of the state-action space are low. In these regions, it is most likely less accurate than another dynamics model which has a high density in that region. We argue that during planning, the agent should rely more on the dynamics models that are accurate, and not treat all heads equally.

We propose a variation upon Monte-Carlo rollouts, namely Hindsight-Weighted Planning (HWP). For each action, we perform n paired rollouts of length u, thus one rollout per pair $(Q_i, \hat{D}_i)$. At the end of u steps, we take the maximum value of Q_i as the state-value. Figure 5.2 visualises the backup diagram of Hindsight-Weighted Planning, where $s_{t+1}^{(i)}$ is the predicted next state by dynamics model $\hat{D}_i$.

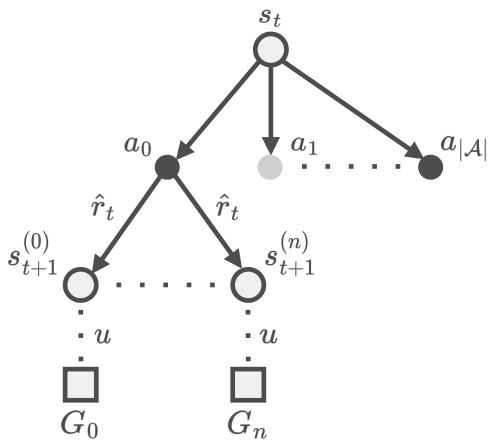

FIGURE 5.2: Backup diagram of Hindsight-Weighted Planning. For each action, the model generates n paired rollouts, with each pair $(Q_i, \hat{D}_i)$, of length u. The weighted cumulative reward over all rollouts is then assigned to the corresponding action. In the backup diagram, $s_{t+1}^{(i)}$ represents the prediction made by $\hat{D}_i$. In addition, some actions at the root node may be pruned (e.g., a_1 in this example), which is based on $Q_{\text{CLB}}\big(\mathcal{F}_\theta(s_t)\big)$.

For each rollout, the states are generated by $\hat{D}_i$, while the actions are greedily selected by the corresponding Q_i-head. Given this, the cumulative return $G_i^{(a)}$, when selecting a as the first action, is computed according to Equation 5.5. Note that the terminal flags are omitted, since the EcoGenie environment is a continuing task.

$$\begin{aligned} G_i^{(a)} \doteq \hat{R}_\theta\big(\mathcal{F}_\theta(s_t, a)\big) + \sum_{j=t+1}^{t+u-1} \Big[\gamma^{j-t} \cdot \hat{R}_\theta\Big(\mathcal{F}_\theta\big(s_j^{(i)}\big), \operatorname{argmax} Q_{\theta,i}\big(\mathcal{F}_\theta(s_j^{(i)})\big)\Big)\Big] \\ + \gamma^u \cdot \max Q_{\theta,i}\Big(\mathcal{F}_\theta\big(s_{t+u}^{(i)}\big)\Big) \end{aligned} \tag{5.5}$$

Due to bootstrapping, each dynamics model member will have a different confidence for a sub-region in the state space. We argue that during the paired rollouts, the dynamics models that are more accurate at the current location in the state space should be more predominant than others. The confidence of a dynamics model $\hat{D}_i$ is determined by comparing the predicted next state $s_{t+1}^{(i)}$ at the previous planning iteration, with the true next state s_{t+1} we received

from the environment at the current iteration. The confidence is quantified by $\mathcal{L}_{\hat{D}_i}$ between the prediction and true next state, and is computed at the start of the planning iteration.

In addition, we argue that if a dynamics model is inaccurate for the selected transition at the previous planning step, then it will probably also be inaccurate for the next step. Thus, we have to reduce its importance during this planning step.

Given the loss for each dynamics head, we compute the hindsight-weights W. By dividing the minimum loss seen at the current planning iteration with the loss of a dynamics head, we scale the weights based on the best dynamics head at that point in time. If the difference is small between the heads, then all hindsight weights will be close to 1. Next, the score $G^{(a)}$ for action a at the root can be computed by taking the weighted sum over the paired rollouts according to Equation 5.6. In this equation, ω is a small number to prevent division by zero.

$$G^{(a)} \doteq \sum_{i=0}^{n-1} \Big[G_i^{(a)} \cdot \underbrace{\frac{\min_{j \in \{0,\dots,n-1\}} \mathcal{L}_{\hat{D}_j} + \omega}{\mathcal{L}_{\hat{D}_i} + \omega}}_{\text{Hindsight-weight } W_i} \Big] \tag{5.6}$$

When all scores $G^{(a)}$ have been computed, the action with the highest value is selected and the remainder of the tree is thrown away. Note that at the start of an episode, all hindsight-weights are set to 1. Figure 5.3 illustrates an example of how the weights shift per planning iteration.

In addition, it is also possible to prune a set of actions at the root of the backup diagram based on Q_{CLB} (see the gray node of a_1 in Figure 5.2). Consequently, we do not perform rollouts on actions that do not seem promising. In addition, actions with high disagreement are filtered, which is an additional benefit since the rollouts would be inaccurate for some (if not all) $(Q_i, \hat{D}_i)$-pairs. We define k as the top-k actions that are *not* pruned at the root.

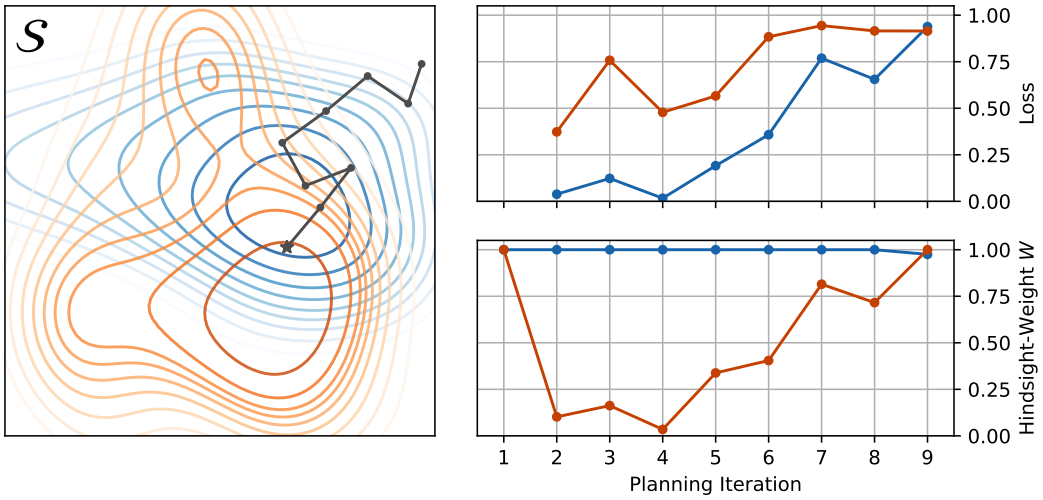

FIGURE 5.3: Example of how the hindsight-weights for an ensemble of two members (**blue** and **orange**) would theoretically proceed. The left plot shows the density plot of how the datapoints in the two replay buffers are divided. In this example, we assume that the density of transitions in the state space directly correlates to the loss. Next, the planning starts at the star, after which each step is represented by a dot in the state space. The trajectory in $\mathcal{S}$ shows that we first remain in the region with high and low density for **blue** and **orange**, respectively. Here, the hindsight-weight for **blue** is significantly higher than **orange**. However, in the end, when both are near-identical in terms of accuracy, **orange** and **blue** have similar weights. Important to note that all the data in the example is randomly generated.

6 Experimental Setup

By means of experimentation, we evaluate Multi Dynamics- and Q-Learning (MDQL) against a set of benchmarks from different domains: a Heuristic-Based System (HBS), Mixed-Integer Linear Programming with perfect foresight and a few model-free reinforcement learning (RL) algorithms. In addition, the most important environment configuration settings are discussed along with the strategy behind the hyperparameter optimisation.

All experiments are conducted on a shared compute cluster, where we can specify what (and how many) CPU's and GPU's we require. For this work, we requested a compute unit with the following specifications:

- CPU: 1 Intel Xeon Gold 6248 core @ 2.50 GHz
- GPU: 1 NVIDIA Tesla V100 (32 GB HBM2 VRAM)

6.1 Implementation Details

This section contains a brief overview of some of the implementation details of MDQL and the EcoGenie environment.

MDQL. The neural network architecture is implemented in PyTorch (v1.13.1) (Paszke et al., 2019), with the parameters being optimised according to the AdamW optimiser (Loshchilov and Hutter, 2019) with AMSGrad enabled. In addition, the gradients have been clipped by value (100) and norm (1.0).

For experience replay, we make use of Prioritized Experience Replay (Schaul et al., 2016) in order to speed up the training process. To ensure a fair comparison against other RL-methods, each method (if applicable) is able to save N transitions in its replay buffer. However, this does not apply to MDQL due to its training process being split up into two phases. For the implementation, we opted for the CPPRB library (Yamada, 2019), which offers an easy to use interface while the computations are sped up by utilising Cython. Important to note is that all losses of MDQL — $\mathcal{L}_Q$, $\mathcal{L}_{\hat{D}}$ and $\mathcal{L}_{\hat{R}}$ — are taken into account when computing the sample probabilities.

EcoGenie Environment. The environment is implemented as a Gym environment (Brockman et al., 2016), with the heatpump and heat loss dynamics being based on an environment that has been developed internally at Shell.

6.2 Environment Configuration

The reward function is configured such that we incentivise the agent to use the battery primarily for the household. In order to accomplish this, we enable the battery SoH component of the reward function only when the `NETTING` action has been chosen. In addition, its scalar λ_{SoH} is set to 1.0. Consequently, netting power back to the grid is only profitable during high spikes of the grid tariff, since otherwise the SoH component would nullify the profits made.

Moreover, since we are dealing with a pseudo-environment, we have to approach the evaluation of an algorithm differently to traditional environments. In our case, we opted to split the dataset up into a train, validation and test set, with a split of 26:5:5. The training set contains the first 26 months of the three years of data, i.e.: from January 2017 up to, and including, February 2019. The remaining months are divided over the validation and test set by alternatively assigning a month to the test and validation set. This results in the following split:

- **Train set**: 2017, 2018, {January, February} of 2019.
- **Validation set**: {April, June, August, October, December} of 2019.
- **Test set**: {March, May, July, September, November} of 2019.

The reasoning behind alternatively assigning a month to validation/test is to ensure that each set contains roughly the same number of spring, summer, fall and winter months. This is since winter months will have significantly different weather to summer months, thus the optimal behaviour is dependent on the time of year. During validation and testing time, we evaluate the method on *all* five months, but with the initial state of the battery, heatpump input/output power, User Profile and household temperature randomised within a predefined interval. We refer the reader to Appendix A for the exact settings of the environment.

A training episode takes 1,420 hours (i.e., $\sim$2 months), irrespective of the actions taken by an agent. This means that the agent is able to face the long-term consequences of actions taken at an earlier stage in the episode. The initial state is initialised similarly to the validation and test environment, with the only difference being that the initial point in the 26 months is uniformly sampled (with the constraint that at least 1,419 datapoints follow it).

Lastly, the features of the state space are normalised by min-max feature scaling. The minimum and maximum have been obtained from the whole dataset. Moreover, features expressed in the same unit, such as the (forecasted) outdoor and indoor temperature, have been normalised with the same minimum and maximum values.

6.3 Baselines

Three types of methods have been evaluated on the EcoGenie environment, and serve as benchmarks for MDQL. In addition, we also quantify a lower-bound on the performance in the form of the 'Idle'-baseline. With this baseline, only the action `IDLE` will be selected, irrespective of the state. It demonstrates what the power consumption and costs would have been if the household would not have had access to a smart BtM battery nor PV-panels.

For all algorithms that contain a stochastic component holds that we ran it for three replications with differing random seeds. Over the replications, we report on the mean and standard deviation. Next, we save the weights/configuration that obtain the best score on the validation environment, which will then be evaluated on the test environment. This also applies to MDQL, but with the additional constraint that the dynamics models must first be converged (i.e., we select the best weights from the set of weights generated during the Diverse Dynamics Training phase). We consider the models to be converged when the mean loss $\mathcal{L}_D$ on the validation environment did not improve over the last ten evaluations, with each evaluation occurring every twenty episodes.

In addition, learn curves and losses are smoothed with the Savitzky-Golay filter, taken from SciPy (Virtanen et al., 2020). The window length is set to 21 and the order of the polynomial fitted to the samples is set to 4.

Lastly, the RL-methods are trained for 100 epochs, where each epoch consists of 20 training episodes. Subsequently, each model is trained on 2.8M timesteps. With the machine specifications reported above, the training time is around 3.5 hours per replication for MDQL.

6.3.1 Heuristic-Based System

As the first baseline, we introduce a Heuristic-Based System (HBS), which is modelled as a binary decision tree. It consists of a sequence of six handcrafted if-statements, with all but one containing parameters from the set of parameters $\Phi = \{\phi_0, ..., \phi_7\}$. Algorithm 2 contains the pseudocode of the HBS. Note that the HBS also interacts with the EcoGenie environment implemented in Gym, i.e.: it receives normalised features.

Algorithm 2: Heuristic-Based System

Input : SoC_t, PV_{t-1}, p_t, p_t^{MA}, η_t
Output : a_t

if $\text{PV}_{t-1} \geq \phi_0$ **and** $\text{SoC}_t \leq \phi_1$ **then** ▷ Statement 1
  return `CHARGE_PV`
end
if $p_t \leq 0.0$ **and** $\text{SoC}_t < \phi_1$ **then** ▷ Statement 2
  return `CHARGE_GRID`
end
if $\text{SoC}_t \geq \phi_2$ **and** $p_t \geq p_t^{\text{MA}} + \phi_3$ **then** ▷ Statement 3
  return `DISCHARGE`
end
if $\text{SoC}_t \geq \phi_4$ **and** $p_t > p_t^{\text{MA}} + \phi_5$ **and** η_t **then** ▷ Statement 4
  return `NETTING`
end
if $\text{SoC}_t \leq \phi_6$ **and** $p_t \leq p_t^{\text{MA}} + \phi_7$ **then** ▷ Statement 5
  return `CHARGE_GRID`
end
if $\text{PV}_{t-1} > 0.0$ **and** $\text{SoC}_t + \text{PV}_t \leq 1.0$ **then** ▷ Statement 6
  return `CHARGE_PV`
end
return `IDLE`

A brief explanation of each statement is given below:

1. Charge the battery if the PV-production of the previous timestep exceeds threshold ϕ_0, while the battery SoC is below threshold ϕ_1.
2. In case the current grid tariff is zero or negative, and the SoC is below threshold ϕ_1: charge the battery from the grid. With this special case we try to maximise the use of free power.
3. When the SoC is above threshold ϕ_2 and the current tariff is relatively high: discharge to the household. This way, we prevent that the heatpump uses expensive power.
4. If the SoC is above threshold ϕ_4, the current tariff is relatively high and we are able to net for the full tariff (i.e., η_t): net power back to the grid.
5. If the SoC is below threshold ϕ_6 and the current tariff is relatively low: charge the battery from the grid.
6. If there is some PV available and the battery is not fully charged: charge from PV to prevent the waste of PV-power.

The intuition behind the ordering of the six statements is that the HBS should first consider charging the battery with PV-power or free power from the grid, after which it should consider discharging its power into the heatpump, such that the power consumption of the household will be minimised. The next two statements are aimed at minimising the costs of the household, while the final statement is to prevent the waste of PV-power.

The parameters $\{\phi_0, \phi_1, \phi_2, \phi_4, \phi_6\}$ are optimised in the range $(0, 1)$, while $\{\phi_3, \phi_5, \phi_7\}$ are optimised in the range $(-0.02,\ 0.02)$. Moreover, Φ is optimised over the whole training *and* validation environment. As optimisation approach, we used Bayesian Optimisation with 200 random initial points and then another 800 iterations generated by the optimiser. The implementation of the optimiser is taken from GitHub, and is written by Nogueira (2014). The exact values of Φ after optimisation, plus some insights into the behaviour of HBS, are discussed in Appendix C.

We have to note that there are many features in the state space that remain unused by the HBS, such as forecasts, time and heatpump status, but we were unable to logically incorporate those into the statements.

6.3.2 Mixed-Integer Linear Programming

The next type of baseline is Mixed-Integer Linear Programming (MILP) with perfect foresight, for which we have developed a solver specifically aimed at the EcoGenie environment. Given a horizon of the next h timesteps in the environment, the program computes the optimal action sequence based on its objective function. For this, we provide perfect information to the MILP, i.e.: the exact PV-production, grid-tariff, and heatpump power requirements. This is not feasible in practice, and this baseline is instead aimed at quantifying an upper-bound on the performance that can be achieved. The package GEKKO (Beal et al., 2018) is used to write the program.

The solver requires as input variables the following: SoC_t, $\text{PV}_{t:t+h}$, $\text{HP}^{\text{in}}_{t:t+h}$ and $p_{t:t+h}$. Important to note is that the input variables are not normalised, unlike the HBS and RL-methods. Furthermore, the decision variables of the program are represented by a Boolean matrix X of size $h \times |\mathcal{A}|$. The intermediate variables, constraints, and objective function are described next. In addition, we use i as an offset to the current timestep, where each definition applies to all $i \in \{0, ..., h-1\}$.

Intermediate Variables. Four arrays of intermediate variables are computed while finding the sequence of actions that solves the horizon. First, we compute the maximum possible (dis)charge rates ψ for the actions in $\mathcal{A} \setminus \{\texttt{IDLE}, \texttt{DISCHARGE}\}$ with Equation 6.1. For `DISCHARGE`, there is only one rate possible, namely equal to the required power by the heatpump. In this case, it is sufficient to check that the SoC remains positive (see Equation 6.5) in order to verify that it is possible to execute `DISCHARGE`.

$$\begin{aligned}
\psi_i^{\texttt{CHARGE_GRID}} &\doteq \min\left(\psi^{\max,c},\ \text{SoC}^{\max} - \text{SoC}_{t+i}\right) \\
\psi_i^{\texttt{CHARGE_PV}} &\doteq \min\left(\text{PV}_{t+i},\ \text{SoC}^{\max} - \text{SoC}_{t+i}\right) \\
\psi_i^{\texttt{NETTING}} &\doteq \min\left(\psi^{\max,d},\ \text{SoC}_{t+i} - \text{SoC}^{\min}\right)
\end{aligned} \tag{6.1}$$

In these equations, $\psi^{\max,c}$ and $\psi^{\max,d}$ are the maximum charge and discharge rate of the battery, respectively. Furthermore, $\text{SoC}^{\min}$ and $\text{SoC}^{\max}$ are the minimum and maximum capacity of the battery. The values for the four battery constants can be found in Appendix A.

Given the (dis)charge rates, we can compute the battery SoC after each timestep with Equation 6.2.

$$\begin{aligned}
\text{SoC}_{t+i+1} \doteq \text{SoC}_{t+i} &+ X_{i,\texttt{CHARGE_GRID}} \cdot \psi_i^{\texttt{CHARGE_GRID}} \\
&+ X_{i,\texttt{CHARGE_PV}} \cdot \psi_i^{\texttt{CHARGE_PV}} \\
&- X_{i,\texttt{NETTING}} \cdot \psi_i^{\texttt{NETTING}} \\
&- X_{i,\texttt{DISCHARGE}} \cdot \text{HP}^{\text{in}}_{t+i}
\end{aligned} \tag{6.2}$$

Constraints. With the input, decision and intermediate variables in place, we define a set of constraints to ensure that the program returns a feasible solution. First, constraints are made such that the program has to select exactly one action per timestep.

$$\sum_{a \in \mathcal{A}} X_{i,a} = 1 \tag{6.3}$$

Next, we define constraints in Equation 6.4 for the battery SoC, such that it is not possible to exceed the upper and lower bound at any timestep.

$$\begin{aligned} \text{SoC}_{t+i} &\leq \text{SoC}^{\max} \\ \text{SoC}_{t+i} &\geq \text{SoC}^{\min} \end{aligned} \tag{6.4}$$

Lastly, we want to prevent that an action (with the exception of `IDLE`) is selected that does not have any effect on the battery SoC. To achieve this, the maximum possible rate must exceed a constant threshold $\Psi = 0.2$ in case the corresponding action at that timestep is selected. As for `DISCHARGE` (i.e., the final constraint in Equation 6.5), we only have to verify that the discharge rate can be met by the battery, explaining why Ψ is replaced with 0.

$$\begin{aligned} \left(\psi_i^{\texttt{CHARGE\_GRID}} \cdot X_{i,\texttt{CHARGE\_GRID}}\right) + \left(1 - X_{i,\texttt{CHARGE\_GRID}}\right) &\geq \Psi \\ \left(\psi_i^{\texttt{CHARGE\_PV}} \cdot X_{i,\texttt{CHARGE\_PV}}\right) + \left(1 - X_{i,\texttt{CHARGE\_PV}}\right) &\geq \Psi \\ \left(\psi_i^{\texttt{NETTING}} \cdot X_{i,\texttt{NETTING}}\right) + \left(1 - X_{i,\texttt{NETTING}}\right) &\geq \Psi \\ \text{SoC}_{t+i} - \text{SoC}^{\min} - \text{HP}_{t+i}^{\text{in}} &\geq 0 \end{aligned} \tag{6.5}$$

Objective Function. The objective of the program is to minimise the costs made over the horizon by finding the optimal sequence of actions. Thus, the program has to solve the following minimisation task:

$$\begin{aligned} f(X) = \sum_{i=0}^{h-1} \Big[& X_{i,\texttt{IDLE}} \cdot \left(p_{t+i} \cdot \text{HP}_{t+i}^{\text{in}}\right) \\ & + X_{i,\texttt{CHARGE\_GRID}} \cdot \left(p_{t+i} \cdot \left(\text{HP}_{t+i}^{\text{in}} + \psi_i^{\texttt{CHARGE\_GRID}}\right)\right) \\ & + X_{i,\texttt{CHARGE\_PV}} \cdot \left(p_{t+i} \cdot \text{HP}_{t+i}^{\text{in}}\right) \\ & + X_{i,\texttt{DISCHARGE}} \cdot 0 \\ & + X_{i,\texttt{NETTING}} \cdot \left(p_{t+i} \cdot \left(\text{HP}_{t+i}^{\text{in}} - \psi_i^{\texttt{NETTING}}\right)\right) \\ & + X_{i,\texttt{NETTING}} \cdot \psi_i^{\texttt{NETTING}} \cdot \lambda_{\text{SoH}} \cdot \kappa \Big] \end{aligned} \tag{6.6}$$

Note that the objective function is currently written for the environment configuration as described in Subsection 6.2, i.e.: the SoH-penalty only applies to the `NETTING` action. In case it should also apply to `DISCHARGE`, or dismissed altogether, then some minor changes will need to be made to $f(X)$.

One issue that arises with this is the fact that the program does not take the future into consideration when optimising its horizon. Due to this, we introduce the execution horizon e. With this, the program still solves the h timesteps, but only the first e actions are executed in the environment. Thus, the first $h - e$ states of the next horizon will be equal to the last $h - e$ states of the previous horizon. This way, we implicitly let the program take the near future into account.

Due to the nature of the program, where the decision variables exponentially increase with horizon H, it is infeasible to solve horizons of multiple weeks or months at once. In order to speed up the process and maximise the horizons, we initialise the decision variables by solving the problem with the BPOPT solver of GEKKO. Thus, we first relax the constraints on the decision variables (i.e., they can be continuous), and the result of this is used as initialisation for APOPT.

Moreover, we have also set a time constraint of two minutes within which the solver has to come up with a solution. If the solver was not able to find a feasible solution, it throws an exception. When this occurs, we execute the `IDLE` action in the environment once, after which we try again with the solver. Thus, we have to find the optimal values h and e. Figure 6.1 shows the validation and test cumulative returns for a set of h and e values. In the figure, we plotted the different h values, with $e = h/2$. Indeed, the number of exceptions shows to exponentially grow with h. For instance, in ∼23% of the timesteps an exception occurred at $h = 24$. Thus, with the current constraints in place, the optimal h and e according to the validation curve is $h = 12$ and $e = 6$. Consequently, the program will not quantify the absolute upper bound on the performance, and instead one that is tractable given the current constraints.

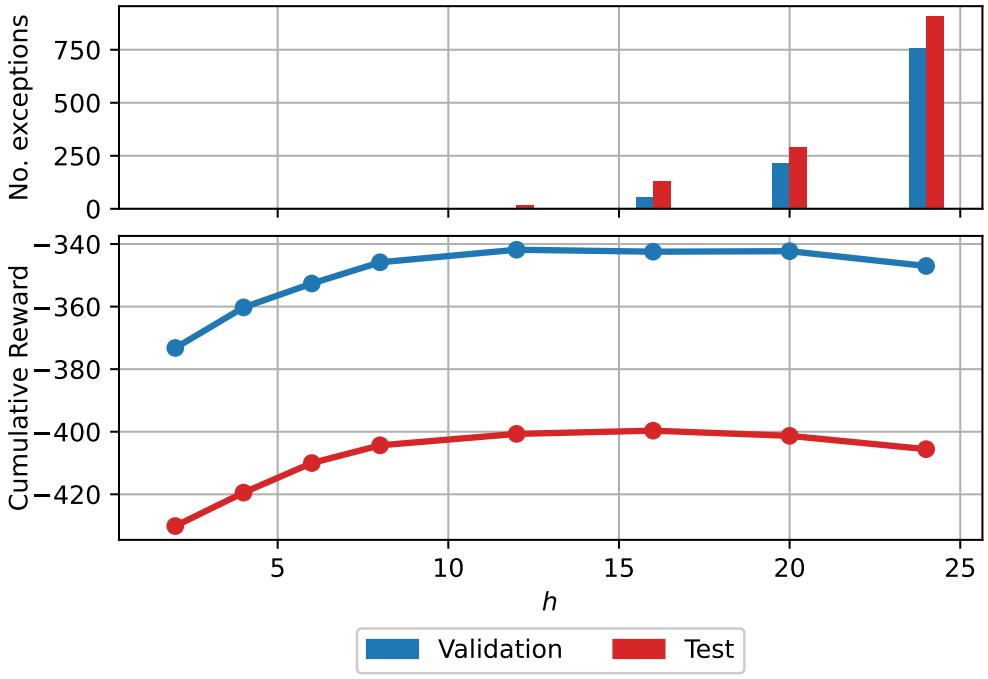

FIGURE 6.1: The performance of MILP with varying h and e values. For all points holds that $e = h/2$. In addition, the number of exceptions occurred while solving the validation/test environment is plotted at the top. The curves show that we are dealing with diminishing returns. Even when only considering $h = 2$ to $h = 12$, where there are few exceptions, the curve tapers off. With larger h, the number of exceptions significantly increases, with as consequence that it harms the overall performance (in all those cases the `IDLE` action is executed).

6.3.3 Reinforcement Learning

Lastly, we compare MDQL to a set of reinforcement learning algorithms, for which we selected algorithms from different subdomains within RL. The exact hyperparameter configuration for each algorithm is denoted in Appendix D.

- **DQN** (Mnih et al., 2013): This algorithm is an online model-free RL method and makes use of Q-learning. In addition, it is one of the building blocks of MDQL. The implementation is taken from Stable-Baselines3 (Raffin et al., 2021).
- **PPO**, **rPPO** (Schulman et al., 2017): An on-policy optimisation algorithm, which is also model-free and online. In addition, its recurrent variant, rPPO, is also taken as a baseline. The implementations are taken from Stable-Baselines3 (Raffin et al., 2021) and Stable-Baselines3 Contrib, respectively.
- **BCQ** (Fujimoto et al., 2019b; Fujimoto et al., 2019a): An offline model-free RL algorithm, which restricts the policy to the batches in the static dataset. Since we are dealing with a discrete action space, we used the author's implementation of Discrete BCQ, which can be found at `https://github.com/sfujim/BCQ`. However, since we are dealing with a pseudo-environment, where the agent still has to collect its transitions, we have made some small changes to the algorithm. We added a replay buffer and ϵ-greedy as exploration strategy. For the value of ϵ, we utilised a decay scheme that exponentially decays ϵ from 1.0 to 0.05 in the first 10% of the training timesteps.

6.4 Hyperparameter Optimisation

The hyperparameters of each RL algorithm have been optimised through one or more grid searches. In addition, we ran each hyperparameter configuration for three replications.

The best configuration is primarily decided upon the peak performance obtained by the algorithm. If there is no significant difference between the peak performance of two configurations, then we consider the overall stability and shape of the learning curve, where a monotonic learn curve is the most desireable. The hyperparameter optimisation process is denoted below.

- **DQN**: First, we ran a grid search over the learning rate $\alpha \in \{0.001, 0.0001, 0.00001\}$ and ϵ reduction period in $\{0.1, 0.3, 0.5\}$ (i.e., over what period in the training process we schedule ϵ from its initial value to its minimum value). Given the best configuration, we ran a second grid search over the network architecture, for which we evaluated two, three and four hidden layers, and with 32 to 256 nodes per layer (note: not all possible combinations were evaluated). Lastly, we also looked at the strategy of updating the target network; hard updates every 10,000 timesteps and soft updates with $\tau = 0.001$ every timestep.
- **PPO**, **rPPO**: First, we optimised the network architecture through a grid search of five architectures. Given the best architecture, we optimised the learning rate and entropy coefficient over $\{0.001, 0.0001, 0.00001\}$ and $\{0.01, 0.001, 0.0001\}$, respectively. Lastly, we evaluated three values for the Generalized Advantage Estimator, namely $\{0.90, 0.95, 1.00\}$.
- **BCQ**: Here, we have taken the epsilon reduction period from DQN. Given this, we optimised the learning rate and its threshold, Φ_{BCQ}, regarding how much the policy is allowed to deviate from the dataset. The settings are optimised over $\{0.001, 0.0001, 0.00001\}$ and $\{0.1, 0.3, 0.5\}$, respectively.
- **MDQL**: Various hyperparameter settings, such as network architecture, ϵ reduction period, and target network update strategy, have been taken from the DQN grid searches. Given this, we optimised the learning rate over $\{0.001, 0.0001, 0.00001, \text{adaptive}\}$. For the adaptive learning rate, we use the `ReduceLROnPlateau`-scheduler from PyTorch. It halves the learning rate if no improvements, in terms of validation performance, have been made over the last ten evaluations. The starting learning rate is set to 0.001. Its threshold for improvements and minimum learning rate are set to the default values of PyTorch. Lastly, the settings k and u for planning are optimised over $\{3, 4, 5\}$ and $\{4, 6, 8, 10\}$, respectively.

7 Experimental Results

Given that all algorithms are optimised on the validation environment according to the experimental setup, we conducted a set of experiments that have been subdivided into the following three categories:

(7.1) **Benchmarking MDQL.** We empirically compare the optimised MDQL against all optimised benchmarks. We consider the learning curves on the validation environment, along with how the (trained) methods score on the test environment based on a set of metrics. These metrics include the cumulative reward, PV-utilisation, battery usage, and grid consumption, for instance. Lastly, we identify how each approach differs in its optimal behaviour on the test environment based on the action distribution.

(7.2) **MDQL Ablation and Sensitivity Study.** We have conducted an ablation and hyperparameter sensitivity study, considering the main components of the algorithm. For instance, we empirically measure the benefit of Hindsight-Weighted Planning, shared state representation learning, and the use of LSTM layers in the feature extractor. As for the hyperparameter sensitivity study, the sensitivity of the rollout depth and pruning during planning, along with the ensemble size are investigated. Lastly, we also investigate to what extent the amount of training data attributes to the performance of the Q- and $\hat{D}$-ensemble.

(7.3) **Insights into the EcoGenie Environment.** Lastly, we take the best performing RL-model, and train it on three different configurations of the reward function. The first configuration is the default, as described in Section 6.2. As the second configuration, the battery SoH penalty is applied to both `NETTING` and `DISCHARGE`. Thirdly, we dismiss the battery SoH penalty altogether. In addition, we observe how the agent behaves on an hour-to-hour basis over a period of a few weeks, and provide a more in depth understanding of the decision making process of the agent.

In addition, we provide additional results with regards to the Heuristic-Based System behaviour/optimisation in Appendix C, and regarding Q-learning and the dynamics models of (r)MDQL in Appendices E and F, respectively.

7.1 Benchmarking MDQL

In this experiment, we empirically determine how MDQL and rMDQL compare against other approaches on the EcoGenie environment.

Learning Curves on the Validation Environment. First, we plot the learn curves on the validation environment in Figure 7.1. Note that this excludes (r)MDQL with planning, due to the fact that this would increase the training time significantly.

The figure shows how MDQL, and to a lesser extent rMDQL, outperform all benchmarks, with the exception of MILP. MDQL has an asymptotic learn curve, while rMDQL slightly drops off after its initial peak at 500k timesteps. Noteworthy is the learning curve of DQN, one of the building blocks of MDQL, which initially shows a similar curve. However, it then drops off

significantly to the level of the 'Idle'-benchmark. This is the result of diverging q-values due to bootstrapping. It is caused by the fact that the EcoGenie environment returns a negative reward for almost all actions. We provide more results on this in Appendix E, but it is something that rMDQL also seems to suffer from, but to a lesser extent in comparison to DQN.

In addition, rMDQL obtains a lower peak performance than MDQL. We believe that this is due to the fact that the state space already provides features that indicate the state of the environment in the near future (e.g., weather forecasts, minimum/maximum grid tariff over next few hours). Thus, adding recurrent layers to be able to predict how the environment will change seems to be less beneficial, and may instead introduce instability, as is demonstrated by recurrent PPO.

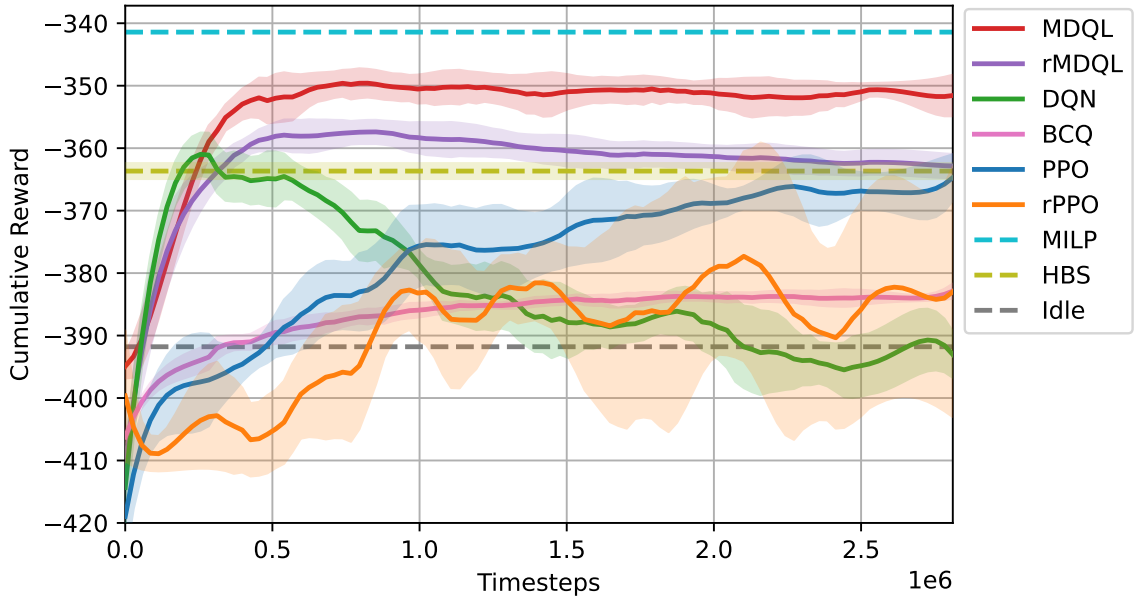

FIGURE 7.1: Smoothed learn curves on the validation environment. Benchmarks without a learn curve are denoted with the dashed lines. Note that the parameters of the HBS are optimised on both the training and validation environment. Moreover, MDQL and rMDQL do *not* make use of planning here. As the figure shows, MDQL performs the best out of all benchmarks, bar MILP with perfect foresight (which mainly serves to quantify an upper bound). Somewhat surprisingly, adding recurrent layers to the feature extractor results in a measurable deterioration of the overall performance.

Moreover, the difference between DQN and BCQ is noteworthy. It shows that the offline aspect of the environment plays a minimal role in obtaining a high cumulative reward on the validation/test environment. Instead, constraining the updates to the data distribution resulted in a conservative policy, since BCQ only barely outperforms the 'Idle'-baseline.

Lastly, PPO shows a stable learning curve and obtains a performance similar to the Heuristic-Based System (note that HBS is trained on both the training *and* validation environment). Notably, PPO seems to not have settled yet, and more training time might result in approaching or exceeding MDQL. With the current amount of training time, PPO is currently outperformed by DQN and some replications of recurrent PPO, in terms of peak performance. Recurrent PPO proves to be rather unstable, with the largest standard deviation out of all algorithms. Perhaps a different hyperparameter configuration would be able to resolve the instabilility.

Test Environment Performance. Next, we take the best weights of each algorithm and evaluate them on the test environment. Table 7.1 denotes the metrics on the test environment. It

includes PV utilisation, consumption/netting profile, mean monthly spendings and cumulative reward.

Noteworthy is that no method makes extensive use of the `NETTING` action. The action is selected in rare occasions, mainly when the grid tariff is relatively high, as is denoted by the mean grid tariff metric for netting. It shows that only with high tariffs it is possible to overcome the battery SoH-penalty. So instead, the power is used by the residential household itself. Compared to the 'Idle'-baseline, all other methods consume less power from the grid. Over the five months in the test set, the methods are able to consume up to 0.5 MWh's less than 'Idle'. Consequently, this saves the household up to nine euros each month.

TABLE 7.1: Metrics of each method on the test environment. For each metric, we report on the mean and (if applicable) standard deviation, which are top and bottom per metric, respectively. The units reported in Table 4.2 also apply here, unless specified otherwise. In addition, we highlight the overall best metrics in *italic*, while we also underline the best metrics excluding MILP. As for MDQL, the non-recurrent and recurrent versions are denoted with 'D' and 'R', respectively, and the versions with planning enabled are denoted by the 'P'. Lastly, for netting, we only consider the replications that have netted some power.

Metric								**MDQL**			
	Idle	**MILP**	**HBS**	**DQN**	**BCQ**	**PPO**	**rPPO**	**D**	**D+P**	**R**	**R+P**
PV util. (%)	0.00	64.00	52.63	72.32	50.27	74.73	75.69	76.54	*76.77*	74.01	68.03
			4.40	13.96	1.74	9.44	12.45	2.67	0.01	0.03	1.35
Montly costs	89.07	*79.87*	83.13	82.35	86.49	82.99	82.72	80.78	81.19	81.92	81.71
			0.29	0.52	0.09	0.53	1.17	0.30	0.20	0.57	0.26
Cum. reward	-445.34	*-400.69*	-417.89	-414.35	-436.94	-418.24	-421.55	-405.76	-407.83	-413.84	-410.86
			1.29	1.98	0.61	1.69	7.93	1.06	1.09	3.64	2.31
Consumption											
No. MWh	8.73	8.38	8.45	8.37	8.48	8.35	8.35	*8.35*	*8.35*	8.36	8.40
			0.02	0.07	0.01	0.04	0.05	0.01	0.01	0.01	0.01
Costs	445.34	*399.59*	415.65	411.77	433.51	414.96	413.62	403.97	405.99	409.75	408.56
			0.01	2.56	0.31	2.66	5.84	1.48	1.00	2.91	1.31
Mean tariff	0.051	*0.048*	0.049	0.049	0.051	0.050	0.050	*0.048*	0.049	0.049	0.049
			0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000
Netting											
No. kWh	0.00	3.96	0.00	0.91	*16.66*	0.42	0.00	0.86	0.30	3.01	0.48
			0.00	1.28	6.13	0.59	0.00	0.67	0.33	1.40	0.22
Profits	0.00	0.24	0.00	0.04	*1.06*	0.02	0.00	0.05	0.01	0.16	0.02
			0.00	0.05	0.39	0.03	0.00	0.04	0.02	0.08	0.01
Mean tariff	—	0.061	—	0.050	*0.064*	0.042	—	0.057	0.048	0.053	0.053
				—	0.001	—		0.000	0.002	0.002	0.014

Interestingly, all methods have a PV-utilisation below 80%. This can be explained by the fact that the battery is only able to perform one action at a time, so achieving a PV-utilisation of 100% *and* having a high cumulative reward is extremely difficult, if not impossible. It would mean that in order to achieve 100% PV-utilisation, the battery must charge continuously during daytime, and is only able to provide power to the heatpump at night. This in itself is not cost-effective, since the general trend of grid tariffs shows that they are relatively low during night time (see Figure 4.3). So instead, the algorithms show that it is more cost-effective to sometimes 'waste' PV-power.

For instance, MILP has one of the lowest PV-utilisation at 64%, and instead makes up for this by exploiting lower tariffs and providing power from the battery to the heatpump at high tariff periods. This can be derived from the grid consumption costs and mean tariff, where it has the lowest (or joint lowest) values, while its grid consumption in terms of MWh's is slightly higher than other methods.

Final insight is the difference that Hindsight-Weighted Planning (HWP) makes in terms of cumulative reward for MDQL and rMDQL. As is shown, MDQL with HWP deteriorates slightly, while rMDQL with planning improves slightly. This indicates that perhaps the error accumulation during the rollouts is quite significant for the non-recurrent MDQL, and that LSTM layers seem to help to minimise the error. We discuss this aspect in more detail in Section 7.2 and Appendix F.

Lastly, Figure 7.2 visualises the action distribution for one replication and for each algorithm. The figure confirms some of the insights described above, such as that BCQ takes a more conservative strategy, and selects the `IDLE` action relatively frequently. Moreover, it is also the only method that makes use of `NETTING` action. Overall, all methods seem to converge to similar strategies, after which the efficiency of managing the battery is the main difference between MILP and the other approaches (e.g., MILP has relatively few `IDLE` and many `DISCHARGE` actions).

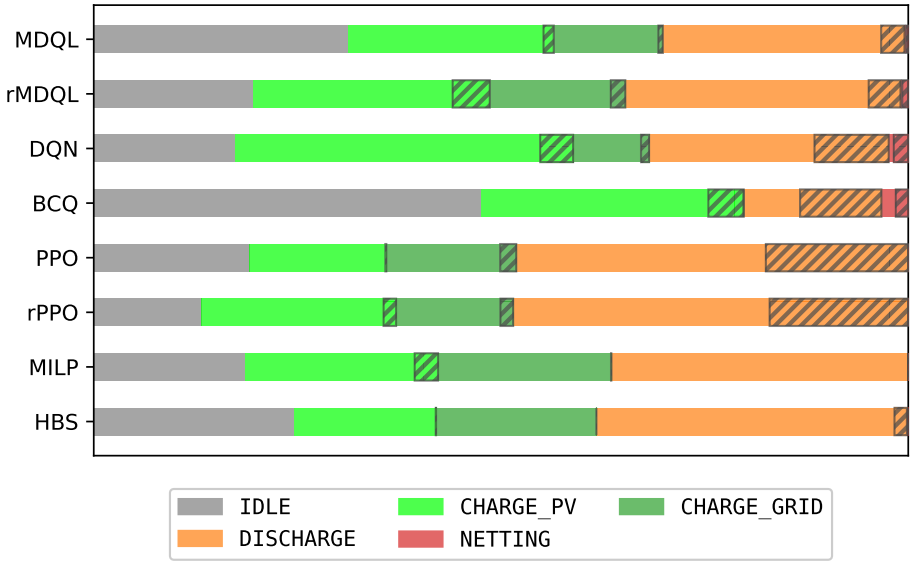

FIGURE 7.2: Action distribution per algorithm of one replication. (r)MDQL with planning are omitted in these distributions, due to them being similar in performance and distribution itself. The hatched parts highlight when an action has been wasted, e.g.: selecting `CHARGE_PV` while the battery is already fully charged. Recall that in these cases the penalty multiplier m is applied to the reward function. Overall, we can observe that most algorithm converge to similar action distributions, where the `NETTING` action is rarely selected. Despite the penalty multiplier, there is still a significant portion of actions that are being wasted by the RL-agents. Perhaps a more severe penalty would solve this. Lastly, BCQ does not follow the other algorithms, and takes a more conservative approach by selecting `IDLE` relatively often.

7.2 MDQL Ablation and Sensitivity Study

For the second set of experiments, we study the impact certain components and hyperparameters have on the overall performance of MDQL, namely the ensemble size (Section 7.2.1), planning hyperparameters (Section 7.2.2), and ablation study (Section 7.2.3). In addition, we also look at the effect of supplying less training data to the algorithm (Section 7.2.4).

7.2.1 Ensemble Size Sensitivity

In this section, we evaluate how MDQL performs with different number of heads in the Q- and dynamics-ensembles. Figure 7.3 shows the validation performance and mean Q-head disagreement for ensemble sizes of $\{1, 2, 4, 8\}$.

Firstly, we observe that there are significant differences in validation performance between the four ensemble sizes. In the first 250k timesteps, all configurations show a similar linear increase of validation performance. However, the more heads there are in an ensemble, the less steep this increase is. This can be explained by the fact that the data is divided over more heads, and therefore each head receives fewer transitions when the ensemble size is high and will require more timesteps in order to converge. After the initial peak, the ensemble size of 1 shows a steep decline in performance and settles to a significantly lower cumulative reward compared to the other configurations.

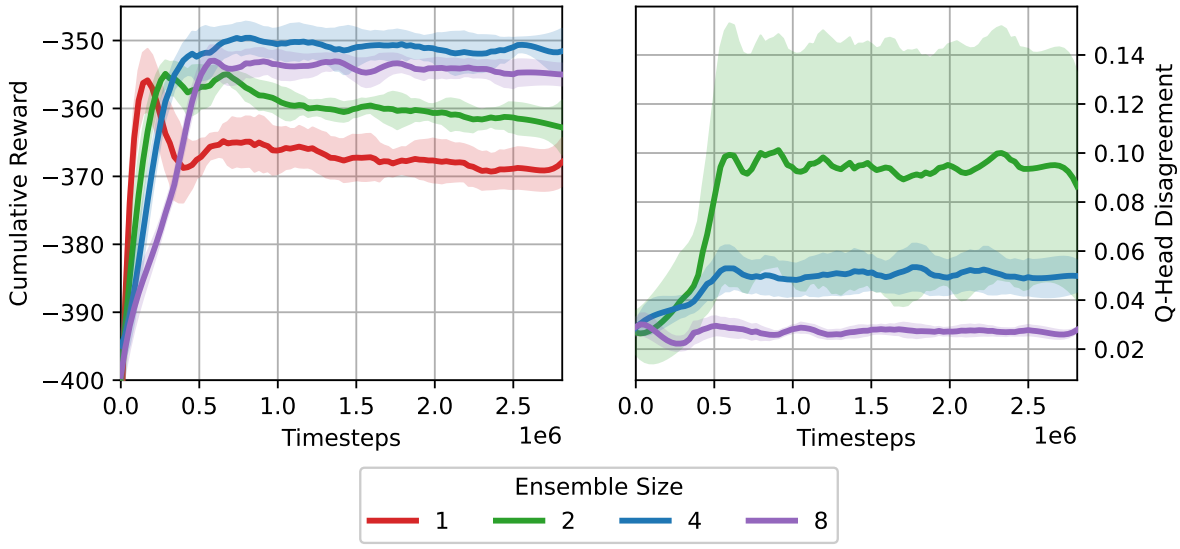

FIGURE 7.3: Validation performance (left) and Q-head disagreement (right) for differing ensemble sizes. The Q-head disagreement is computed by taking the action-wise standard deviation over the whole ensemble, after which the mean over all the standard deviations is taken as the Q-head disagreement for that evaluation episode. Based upon this, we report on the mean and standard deviation over the three replications. Note, the sudden stop in the increase of Q-head disagreement is caused by setting τ to zero when the dynamics models have been converged on the validation traces. The convergence point occurs at around 500k timesteps.

Increasing the ensemble size results in measurable performance improvements up to an ensemble size of 4, from which the performance deteriorates slightly. A possible explanation for this is the fact that Q-head disagreement decreases when increasing the ensemble size (see right plot of Figure 7.3).

We hypothesise that there are few optima in the search space, with as consequence that some heads can converge to the same point. With larger ensembles, this probability increases, and with as consequence that similar heads in an ensemble will result in underestimations of the epistemic uncertainty. This is also supported by the standard deviation of the Q-head disagreement, which is relatively large for an ensemble size of 2, and significantly smaller for 4 and 8.

7.2.2 Planning Hyperparameter Sensitivity

In order to identify any sensitivity in the two main hyperparameters of Hindsight-Weighted Planning (HWP), we conducted a grid search over $k \in \{2,3,4,5\}$ and $u \in \{2,4,6,8,10\}$. Recall that k is the top-k actions that are not pruned at the root of the planning tree, while u is the rollout length. Figure 7.4 shows two heatmaps for HWP and unweighted planning.

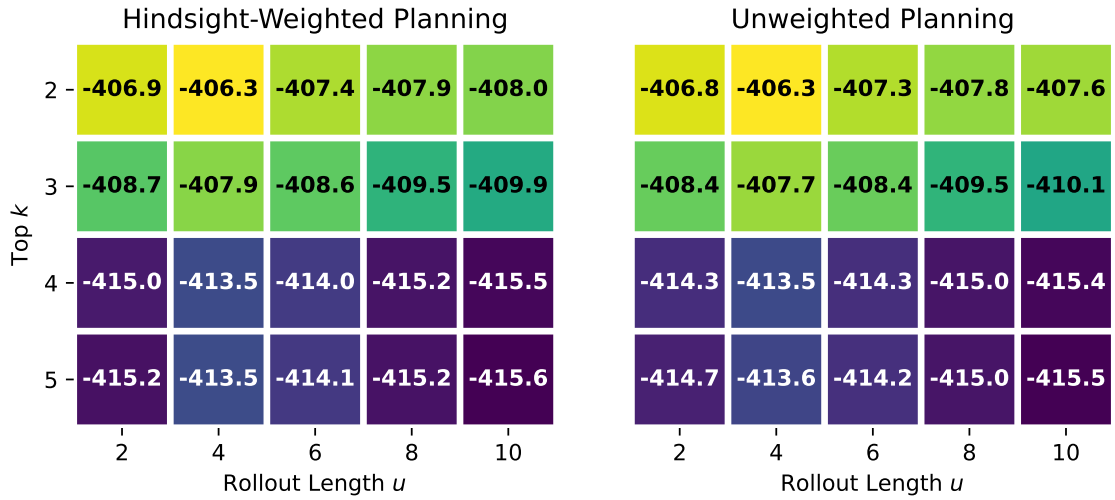

FIGURE 7.4: Heatmap of the cumulative reward received on the test environment for different hyperparameter configurations. At the left, we denote the scores obtained with MDQL and Hindsight-Weighted Planning, while on the right we deployed unweighted planning (i.e., the HWP rollout-strategy without weighting the estimated returns). We report on the mean over three replications for each (k,u)-combination. As context, MDQL obtained a cumulative reward of **-405.8** without planning.

The main takeaway is the fact that HWP does not seem to influence the final performance at all. Its scores are near-identical to unweighted planning. We hypothesise that the loss of the dynamics heads were all similar on this environment, resulting in similar hindsight weights. Moreover, the offline aspect of the EcoGenie environment seems to have little influence, as is also discussed in Section 7.1. HWP might be better suited for true offline environments.

When we compare the planning performance with MDQL without planning (-405.8), we observe that planning results in a slight to substantial deterioration of the test performance. And most notably, the best score is obtained with the lowest possible k, thus with the most restrictions on the planning strategy. Moreover, increasing k results in a significant drop off. This also holds for the rollout length u, but to a lesser degree. Based upon this, we believe that the dynamics models are still inaccurate for suboptimal (according to the Q-ensemble) actions, despite the measures we have taken to mitigate this (i.e., high minimum epsilon of 0.4, and Diverse Dynamics Training). In addition, accumulation of errors by increasing the rollout length is also present, while the best obtained scores for each k are obtained with $u = 4$.

Planning with rMDQL follows the same trends as shown in Figure 7.4, see Figure 7.5. Notably, it is able to improve upon the non-planning performance, albe-it by a small margin. For instance, if we take $k = 3$ and $u = 4$, then rMDQL with HWP improves on average three points compared rMDQL without planning. Overall, setting k to 2 or 3 results in improving or equalling the performance of rMDQL without planning.

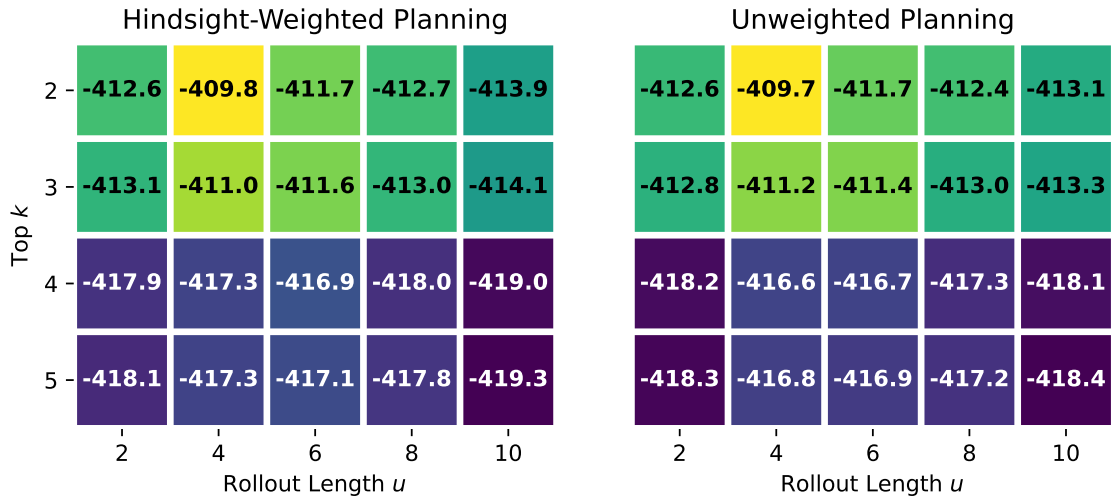

FIGURE 7.5: Heatmap of the cumulative reward received on the test environment for different hyperparameter configurations. At the left, we denote the scores obtained with rMDQL and Hindsight-Weighted Planning, while on the right we deployed unweighted planning (i.e., the HWP rollout-strategy without reweighting the estimated returns). We report on the mean over three replications for each (k, u)-combination. As context, rMDQL obtained a cumulative reward of **-413.8** without planning.

To identify what component causes the subpar planning performance, we ran the planning strategy with the ground truth values on the state transitions and/or reward function. The results are denoted in Table 7.2, where we compared the different planning configurations with $k = 4$ and $u = 8$. In addition, we also include MDQL where none of the weights are shared between the approximators (referred to with 'without $\mathcal{F}$'), with which we can verify whether the lack of performance is inherent to the network architecture.

TABLE 7.2: Mean cumulative reward on the test environment for different planning configurations. We also report on the standard deviation in the second row of each entry. The hyperparameters k and u are set to 4 and 8, respectively. The best obtained scores per column are highlighted in *italic*. In addition, scores that outperform MILP (-400.7) are underlined. Note: for the ground truths, we made use of unweighted planning.

Planning Configuration	**MDQL**	**rMDQL**	**MDQL w/o $\mathcal{F}$**
No Planning	-405.76	-413.84	-410.93
	1.06	3.64	1.17
HWP Planning	-415.15	-417.95	-414.79
	2.95	2.07	0.74
Ground Truth			
Transitions	-409.45	-415.69	—
	1.14	1.52	
Rewards	-401.71	-402.20	—
	1.57	0.89	
Rewards and transitions	*-393.79*	*-394.64*	—
	0.28	0.53	

The results show that with the ground truth on the transitions and rewards, (r)MDQL is able to outperform MILP by a significant margin. Moreover, only providing the true rewards also results in an improvement upon the non-planning performance, and is equal or slightly worse than MILP. Utilising the true next states does result in an improvement, but is still worse than no planning. When compared, providing true rewards has a greater benefit than true states. Therefore, we can say that the main bottleneck lies in the accuracy of the reward approximator.

We also evaluated MDQL without any weight sharing, in order to identify whether the network architecture is the main cause of the performance deterioration with planning. The table shows that, also without weight sharing, the planning seems to suffer from inaccurate predictions. Based upon this, we believe that the main cause in the lack of performance is not inherent to the main characteristics of MDQL. We hypothesise that incorporating the aleatoric uncertainty into the approximators might yield a significant performance boost, since the two stochastic components of the environment — the outdoor temperature and PV production — play a substantial role in the heatpump dynamics and battery charging, respectively.

For instance, mispredicting the outdoor temperature could result in that the expected heatpump input power is underestimated, such that the agent opts for `DISCHARGE` to supply power from the battery to the heatpump. While in reality, the input power is higher, and then it can be the case that the battery SoC is too low, resulting in a wasted action instead. This results in significantly different reward signals.

All in all, planning does show to be promising in the problem setting of battery dispatch optimisation, with it even able to outperform MILP (with perfect foresight) given the ground truth over the whole rollout. But, more accurate next state and reward approximators are essential in order to observe any gains. Appendix F describes how the features attribute to the loss of the dynamics models. We have to note that planning with the ground truths ignores the aleatoric uncertainty, and does not exactly quantify the gains of having a perfect model. A perfect model can predict the transition probabilities perfectly, but the actual predicted next state can still be different to the environment, and will lead to rollouts that can still deviate from reality.

7.2.3 Ablation Study

Next, we investigate the performance attribution of each component of MDQL by means of an ablation study. For all entries holds that we omit planning altogether, as it has already been discussed in the previous experiments. Six different configurations are evaluated, namely:

- **Default**: MDQL with all components enabled. All ablations are applied to this model, which means that most configurations will not use LSTM layers.
- **With LSTM layers**: MDQL with LSTM layers in the feature extractor $\mathcal{F}$, i.e.: rMDQL.
- **No Feature Extractor** $\mathcal{F}$: None of the weights are shared between any of the approximators. In order to ensure a fair comparison, the layers that were originally in the feature extractor are now added to all approximators. For instance, given $\mathcal{F}$ and $\hat{D}_i$ with [1024, 1024] and [256, 256] as their network architectures, respectively, we now set the architecture of $\hat{D}_i$ to [1024, 1024, 256, 256]. The same applies to $\hat{R}$ and the Q-heads.
- **No Ensemble**: The ensemble size of MDQL is set to one.
- **No Dynamics**: The $\hat{D}$-ensemble and reward approximator are omitted. Thus, we are dealing with an ensemble of DDQNs, which still make use of weight sharing.
- **DDQN**: No dynamics learning, no ensemble, no weight sharing.

We have to note that setting τ to zero, when the convergence point has been reached, does not apply to 'No Dynamics' and DDQN. Subsequently, these configurations may suffer from diverging q-values, as the learning curve of DQN demonstrates in Section 7.1. In addition, Diverse Dynamics Training is not deployed for configurations that do not contain any dynamics models. Instead, the whole training procedure follows Q-Ensemble Training.

Figure 7.6 contains the learning curves of the six configurations on the validation environment. First of all, the 'No Dynamics' and DDQN configurations also seem to suffer from diverging q-values to different extents. However, the configuration without ensembles also experiences a drop off in validation performance, which has been contained by setting τ to zero around the 500k timesteps. Moreover, the peak performance of 'No Ensemble' is also significantly lower than the default. This highlights how ensembles positively contribute to the overall performance by being able to quantify the epistemic uncertainty, and also delay the issue of extreme q-values.

Notably, MDQL without a feature extractor also performs worse than the default, indicating that sharing state representations is beneficial to learning the optimal Q-approximators. This is also highlighted by the difference in peak performance of DDQN and 'No Ensemble', where the latter only uses the dynamics model for weight regularisation. Overall, each component, with the exception of including LSTM layers, has a positive attribution to the performance without planning.

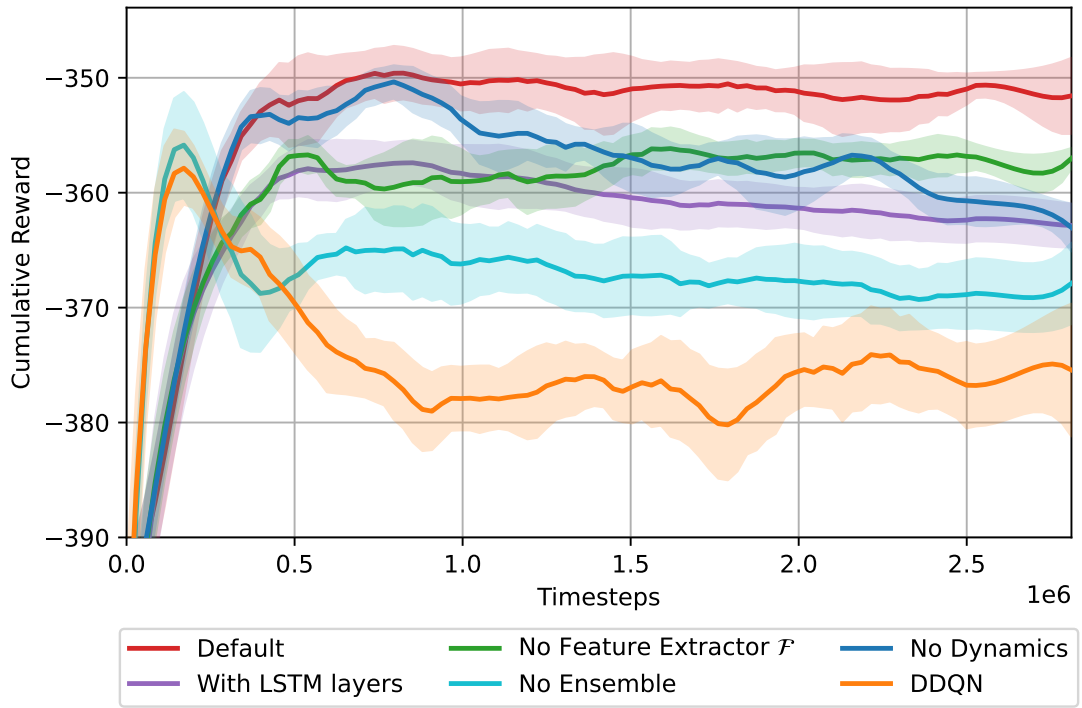

FIGURE 7.6: Learning curves of the ablations on the validation environment. We report on the mean and standard deviation over three replications. We refer the reader to the enumeration denoted above for a description of each ablation. The main takeaway is that each component of MDQL has some positive attribution to the validation performance, with the exception of utilising LSTM layers. For instance, not sharing weights between all approximators results in a significant performance drop. Moreover, weight regularisation through the dynamics models also shows an improvement, which has been derived from the difference in peak performance of DDQN and 'No Ensemble'.

7.2.4 Training Data Sensitivity

With this experiment, we aim to determine how much training data is required to learn an adequate policy, and whether adding more data would have improved the Q-heads and/or dynamics models. During this experiment, we only alter the training set, i.e.: the validation and test sets remain constant, as described in the experimental setup. The train set contains 26 months of data (January 2017 — February 2019), from which we create the following splits:

- **22 months**: May 2017 — February 2019
- **18 months**: September 2017 — February 2019
- **14 months**: January 2018 — February 2019
- **10 months**: May 2018 — February 2019
- **6 months**: September 2018 — February 2019

The n most recent months are kept in each split. This does mean that the distribution of months over the four seasons changes significantly to the validation and test set. For instance, in the extreme case of taking the most recent six months, the dataset mostly contains datapoints that lie in the fall/winter seasons. Subsequently, training an agent on this split, after which it is evaluated on months from all four seasons, should result in a suboptimal performance.

Figure 7.7 visualises the Q-ensemble performance on the validation environment at the left, and the mean dynamics/reward loss over the whole ensemble at the right. The majority of the results are in line with expectations; more data results in more accurate dynamics models and reward approximator, less overfitting and in a higher cumulative reward. However, we are dealing with diminishing returns, with insignificant gains from 22 months onwards.

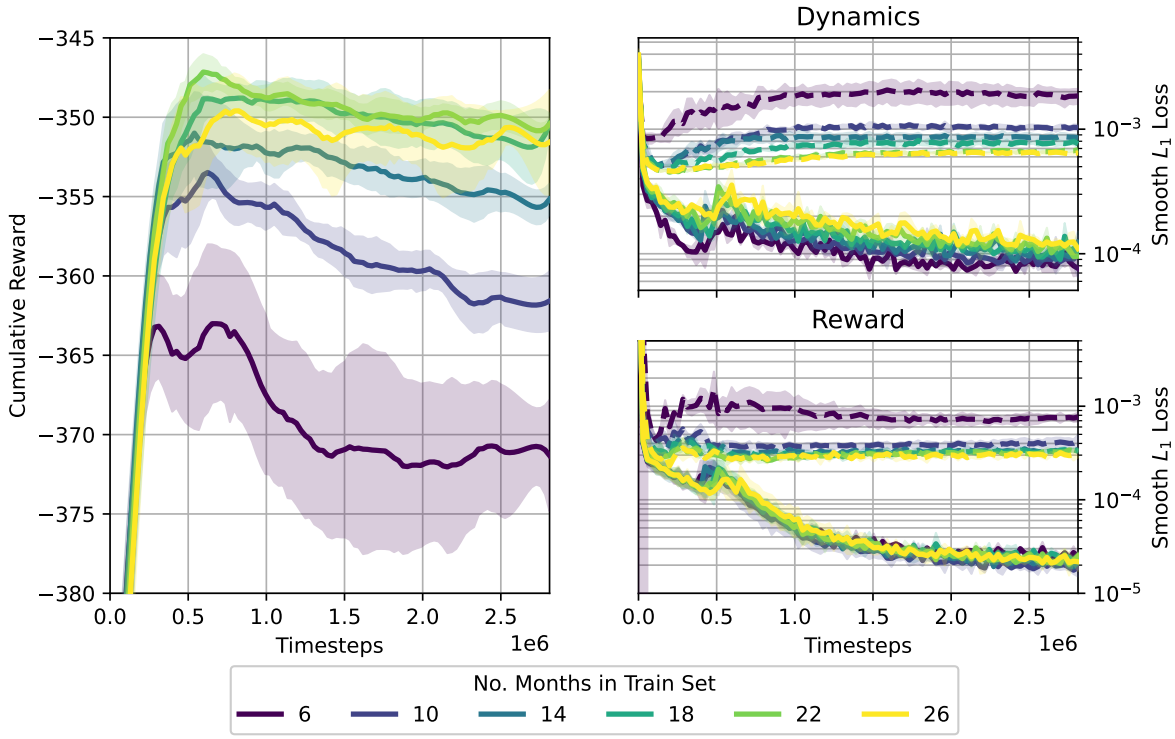

FIGURE 7.7: Validation performance (left) and mean dynamics/reward loss (right) for different training splits. For the losses, we report on the training loss (solid) and validation loss (dashed) over the traces generated by the Q-ensemble.

Notably, MDQL obtains a higher validation score with 18 and 22 months, compared to the original 26 months. This is surprising, since the 26 months of data is the superset of all other training splits. Moreover, this difference is also somewhat present on the test environment, where MDQL is able to obtain a mean score of -407.91, -404.69 and -405.76 when trained on 18, 22, or 26 months, respectively. We suspect that those additional four months contain some signals/attributes that are not in line with the validation/test environment, which can be explained by the fact that there is two years between the four months and the validation/test months. A lot can change in that period of time, especially regarding the power market, and may cause that the learned knowledge of that time period is not applicable to more recent months.

As for the dynamics and reward loss, we are dealing with diminishing returns on the validation environment. Adding more months to the training set would likely not yield any significant performance boosts in terms of accurate state- and reward-estimations.

7.3 Insights into the EcoGenie Environment

In the last set of experiments, we investigate how a reinforcement learning agent behaves in the EcoGenie environment, along with how the optimal behaviour changes given different reward function configurations. For all experiments holds that we deployed MDQL without planning.

7.3.1 MDQL Behaviour Analysis

In order to analyse the behaviour and reasoning of the agent, we have plotted two weeks from the test environment: one week from July and one from November. These two weeks vary significantly in terms of outdoor temperature and PV radiation, and should provide some insights as to how the agent acts under various circumstances. Figure 7.8 shows the two weeks for one replication, with the top and bottom plot corresponding to the weeks from July and November 2019, respectively.

In the first week of July, the agent mainly charges the battery via the PV panels. As the figure shows, the majority of the available PV power has been covered by the agent. Then, this power is consumed by the heatpump in the hours where there is little to no daylight. In addition, the agent sometimes opts for a quick charge from the grid at midnight, possibly since the grid tariffs are often relatively low during that point in time. Unfortunately, these drops in tariff are not fully exploited. For instance, the agent does not opt to charge the battery during the low tariff at the end of the 6^{th} of June, despite the battery being nearly depleted.

During the second week of November, there is little to no PV power available. Here, the agent mainly relies on the power grid to charge the battery, and mainly provides the heatpump with power from the battery during periods with a relatively high grid tariff. For instance, during the tariff peak in the afternoon of the 8^{th} and 13^{th} (but unfortunately not at the absolute peak) of November. However, there are also timesteps where the agent selects an action that does not change the battery SoC, e.g.: around noon on the 9^{th}, in the afternoon/evening of the 11^{th}, and in the evening of the 14^{th}. It shows that, despite the reward multiplier, the agent still misinterprets the limitations of the battery, and still opts to charge/discharge under some circumstances despite it being not possible.

Overall, the agent tries to maximise its PV-utilisation and supply this power to the heatpump during periods with no sunlight. In case there is an overall lack of PV radiation, the battery is used to a lesser extend and the required HP-power is mainly supplied by the grid. Moreover,

these two weeks show that there are still some points in time where the decision making was suboptimal, and highlight how there is still some improvements to be made over short-term periods.

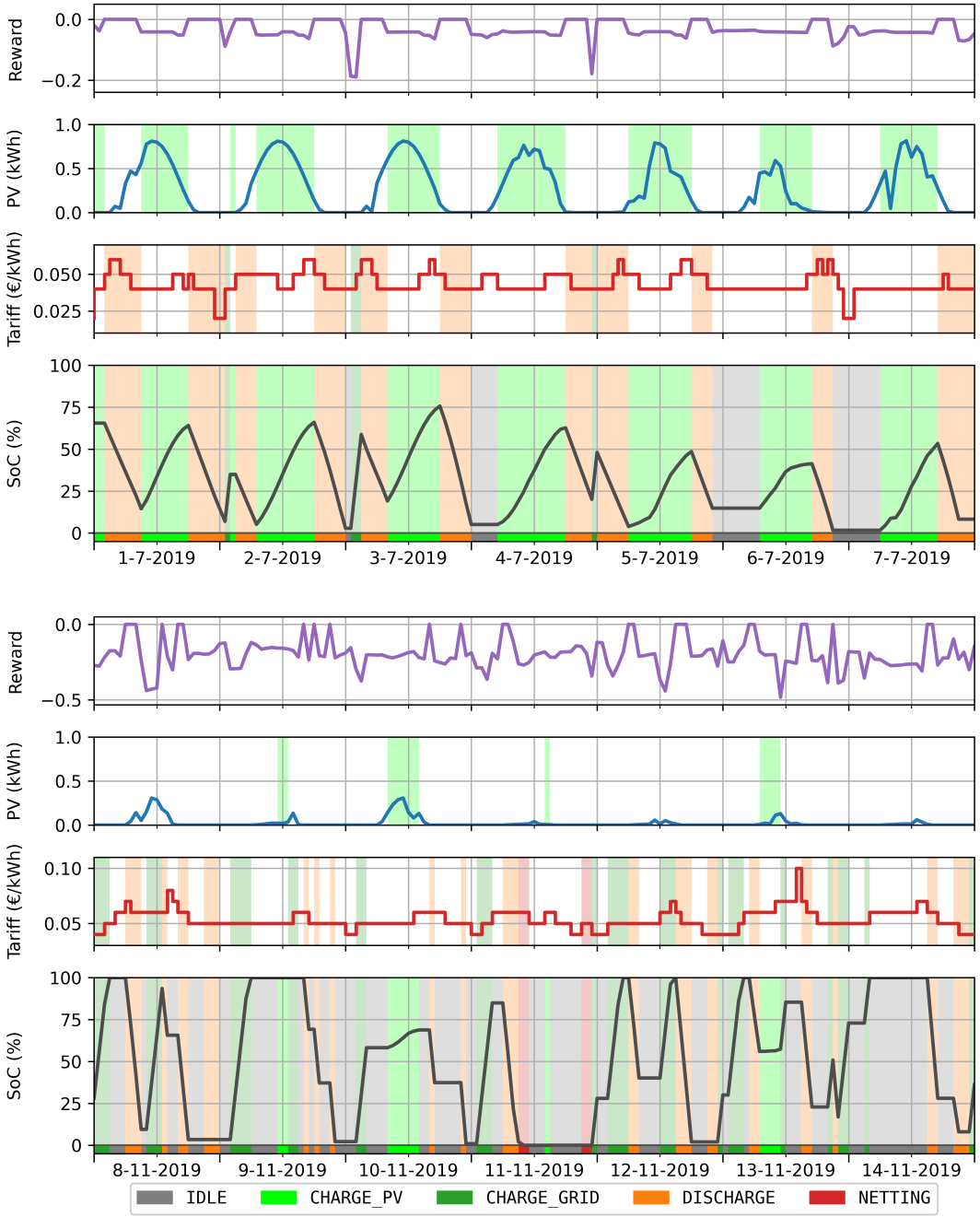

FIGURE 7.8: MDQL acting over two weeks in July (top) and November (bottom). Each plot consists out of four subplots describing, from top to bottom, the reward signal, PV radiation, grid tariff and progression of the battery SoC. Moreover, the background color denotes the action that has been selected at that point in time. The two weeks show a different approach in supplying power to the heatpump, due to a different amount of PV being available during those two periods.

7.3.2 Influence Reward Function Configuration

As the final experiment, the influence of the battery SoH reward-component on the MDQL behaviour has been investigated. We introduce three scenarios {S1, S2, S3} for the comparison:

S1. The default reward function configuration, as described in Section 6.2, with $\lambda_{\text{SoH}} = 1.0$. The battery SoH penalty is only applied to `NETTING`, which is an action outside of the main objective of providing power to the household.

S2. The SoH penalty is applied to all discharge actions, i.e.: `DISCHARGE` and `NETTING`. The idea is to use the battery when it is absolutely beneficial to the household. The scalar λ_{SoH} is set to 1 for both actions.

S3. Lastly, we omit the SoH-penalty altogether such that the health of the battery does not matter. In this scenario, `NETTING` should become more attractive in order to minimise the household costs.

Figure 7.1 shows the learning curves of MDQL on each of the scenarios. For scenarios 2 and 3, we used the optimal hyperparameter settings we obtained from the grid search on scenario 1. The plot shows significant differences in cumulative reward after convergence; scenario 3 has the highest and is closely followed by scenario 1. Notably, scenario 2 is below the 'Idle'-baseline.

Without the SoH-penalty in place, the agent is able to make substantial profits by selling its power during price spikes. Subsequently, the overall power consumption (and with it the carbon footprint) of the household will be higher. The number one priority becomes to minimise the costs of the household. As for scenario 2, the SoH-penalty can not be overcome by the agent with either discharge-actions. As a result, the highest cumulative reward is obtained by selecting the action `IDLE`. The gap between the 'Idle'-baseline and scenario 2 can be explained by the reward multiplier in case of wasting actions.

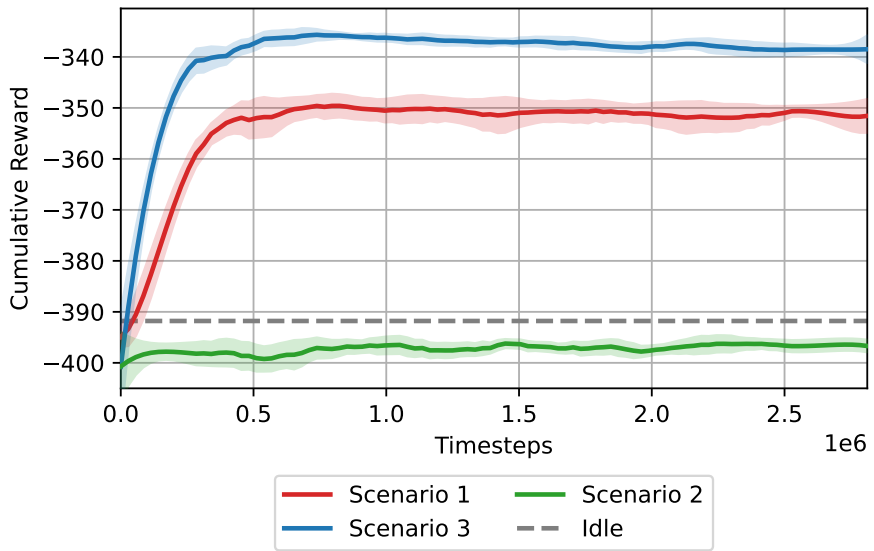

FIGURE 7.9: Learning curves of MDQL on the validation environment for the three scenarios. We report on the mean and standard deviation over three replications. The cumulative reward at convergence differs significantly between the three scenarios. Most notably, applying the SoH-penalty to all discharge actions results in a cumulative reward lower than the 'Idle'-baseline. This means that it is rarely possible to overcome the penalty for both `DISCHARGE` and `NETTING`.

Figure 7.10 confirms the insights regarding scenarios 2 and 3 for one replication of MDQL. In scenario 2, the actions `IDLE` and `CHARGE_PV` are mostly selected, since those do not decrease the reward signal any further (given the action is not wasted). MDQL still selects the other actions at some point during the evaluation, but it is classified as a wasted action in almost all of the occurences.

With scenario 3, the problem resembles more a power trading setting; the agent mostly discharges its power back to the grid for small profits. In addition, more charging from the grid, compared to scenario 1, occurs. This indicates that the agent tries to buy power cheaply, and then sell it back for a higher tariff. Most likely, this also occurs when no profits are made at all, i.e.: charge the battery with n kWh's for a tariff of x EUR/kWh, and discharge the same amount for the same tariff in the following timestep. In the reward function is no mechanism in place where this is punished, so the agent receives the same reward for the earlier described action sequence, as for selecting `IDLE` twice.

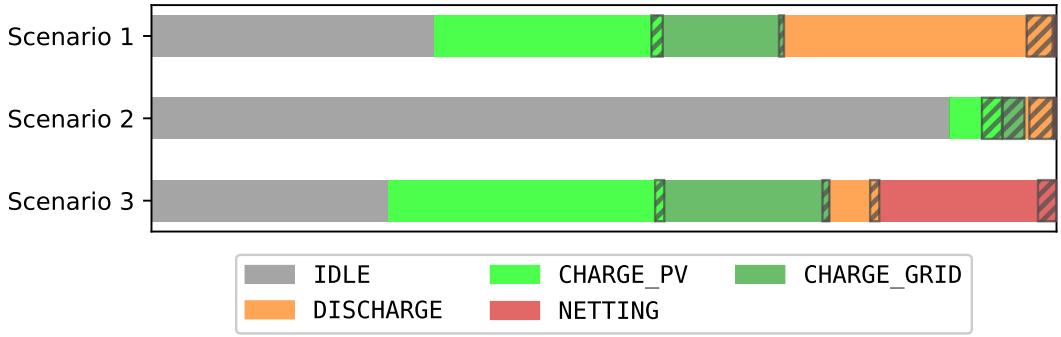

FIGURE 7.10: Action distribution of MDQL on the three scenarios. The distributions are taken from an evaluation run on the test environment with one replication. The inclusion of the SoH-scalar has a great impact on the final behaviour of MDQL; with scenario 2, the two discharge actions are almost never able to overcome the penalty. Consequently, the way to obtain the highest possible cumulative reward is to remain idle. As for scenario 3, the majority of the discharging is directed to the grid. Moreover, the battery is more frequently charged from the grid. Based upon this, we can say that the problem becomes a power trading optimisation problem, where the main objective is to make as much profits as possible; the household consumption is almost completely disregarded.

Overall, the final behaviour of the three scenarios demonstrates that the SoH-penalty is difficult to overcome. In case of the current battery setup, the battery shows to be too expensive in order to consider it during the decision making process. This does raise the question as to whether utilising a Behind-the-Meter battery is cost-effective in 2023 (given that our approach for SoH-approximation is somewhat accurate). A more realistic approach to the battery dynamics, along with a more sophisticated SoH-approximation should clarify this. Otherwise, the main benefit that remains is a lower carbon footprint, until cheaper and/or more durable batteries enter the market.

8 Discussion

With the energy transition in mind, there is a push for the electrification of various appliances and the use of Behind-the-Meter (BtM) smart batteries. In combination of photovoltaic (PV) panels, the smart batteries can be deployed to lower the carbon footprint and expenses of the household. This can be achieved by intelligently charging the battery from the PV panels and power grid, along with discharging it to the household at appropriate timeslots. The aim is to provide more affordable and renewable power to a heatpump of a residential household, namely Shell's EcoGenie house at The Hague.

In this work, we provide a new take on the Home Energy Management System (HEMS) problem setting by introducing the EcoGenie environment, developed to accurately simulate the setting of the EcoGenie house. It consists out of a BtM battery, access to the grid, a heatpump and PV panels. The HEMS is able to select one action every hour, with which it can control the power flow around the battery; charge it from the grid or PV, or discharge to the heatpump or to the grid (i.e., netting). Historical data regarding outdoor temperature, PV radiation, weather forecasts and power grid tariffs are used to implement the environment, along with rule-based dynamics for the heatpump and heat loss of the house itself. Subsequently, we are dealing with an environment where epistemic uncertainty can not be mitigated by continuously sampling new transitions, unlike traditional reinforcement learning environments.

Given the environment, we have conducted a comparison between baselines from various domains; Mixed-Integer Linear Programming (MILP) with perfect foresight, a Heuristic-Based System and a set of reinforcement learning (RL) algorithms.

In addition, we propose a novel reinforcement learning network architecture, Multi Dynamics- and Q-Learning (MDQL), that shares state representations between two ensembles of q-value and dynamics model approximators, along with a shared reward function approximator. The Q-ensemble is utilised to quantify the epistemic uncertainty by computing the confidence lower bound over the ensemble. In addition, a recurrent variant, rMDQL, is also proposed, which makes use of LSTM layers in the shared layers of the network architecture.

The main drawback of MDQL is the fact that all approximators must be trained simultaneously, which in the case of RL algorithms means that the training samples must stay diverse. Otherwise, catastrophic forgetting may occur for the dynamics models and reward approximator, because the Q-heads will generate traces that become less diverse due to convergence and the exploration strategy.

Moreover, we introduce Hindsight-Weighted Planning (HWP), a planning strategy consisting of weighted rollouts with the dynamics models and Q-approximators. After each planning iteration, the accuracy of each dynamics model is evaluated on the next state in the environment. With this accuracy per dynamics model, hindsight-weights are computed to weight each rollout during the next planning step.

Benchmarking MDQL. Based upon the experiments, we demonstrate that MDQL outperforms all RL algorithms (DQN, BCQ, PPO, recurrent PPO) and the Heuristic-Based System. However, it still falls short of MILP with perfect foresight, which quantifies an upper bound[1]

[1]Note: this is not the absolute upper bound, due to a set of constraints (time budget, limiting the horizon) that have been imposed on MILP.

on the performance.

When enabling HWP, the performance of MDQL slightly deteriorates, which is due to the inaccurate next state and reward predictions. Given the ground truth on the next state and reward, MDQL plus planning is able to outperform MILP by a significant margin. Therefore, we can say that planning is a promising approach in the battery dispatch optimisation setting, but it requires more accurate predictions. However, we have to consider the fact that planning with the ground truth results in omitting the aleatoric uncertainty of the environment, thus the actual performance with a perfect model would be less extreme than demonstrated by planning over the ground truths. Moreover, we were able to demonstrate with a set of ablations that the issue is not inherent to the network architecture, nor to the limited amount of data or weighting the rollouts.

Moreover, we have shown that each component of MDQL attributes positively to its performance through an ablation study. In particular, the use of ensembles and sharing weights between the approximators contribute the most to the performance improvements. Recurrent layers have shown to result in a slight performance deterioration, which might indicate that recurrent layers and long-range dependencies are not as important as we initially expected. However, some replications of recurrent PPO have shown significant improvements over the default PPO, but they also come with more instability. In addition, the environment already provides features with regards to the short-term future (e.g., PV production over the next 24 hours, maximum grid tariff over the next day-ahead, etc.), which might result in less pronounced improvements by incorporating recurrency or transformers. Intuitively, there still remain important dependencies over long periods of time (e.g., the weather might turn bad in a couple of days time, resulting in less PV production and lower outdoor temperatures). However, charging the battery is achieved quite quickly, with as a result that it would be sufficient to start preparing for the bad weather when it arrives in a one or two days. Overall, we hypothesise that it should be sufficient to look one or two days ahead in the decision making process (this is also highlighted by the diminishing returns when increasing the horizon of MILP), which can be achieved by incorporating it into the state space.

Linear Programming versus Reinforcement Learning. The main shortcoming of this study is the fact that MILP has access to the perfect information, resulting in an unfair comparison against the other approaches. In addition, a set of constraints have been imposed on MILP (e.g., limiting the decision horizon, time budget) due to its computational properties. This does raise the question as to how its performance would be without any constraints and making its decisions based on forecasts or the information of the previous n days.

On the other hand, we have observed that MDQL with planning over a shorter horizon on the ground truths, for which we argue that it is similar to MILP with perfect foresight, is able to outperform MILP by a significant margin. The main drawback is that MILP is not able to incorporate the short-term future into its decision making process, which we have tried to mitigate by introducing an execution horizon.

We believe that given accurate forecasts, MILP would be able to obtain a similar performance compared with access to perfect information; a minor error in, for instance, the outdoor temperature or PV radiation would not be detrimental in its decision making process. The main uncertainty lies in how accurate the forecasts can be. As an example, the Numerical Weather Predictions models we utilised for the temperature and PV radiation forecasts were often quite accurate (e.g., a reasonable root mean squared error). But there were datapoints with a large error, which would result in suboptimal decisions by MILP.

When compared to RL approaches, the main question to answer is which approach would be best to deploy in the real world. For this, we have to consider a set of factors:

(i) **Computational requirements**: The main difference between RL and MILP is the fact that the former requires a significant amount of training time initially, after which its deployment costs are relatively low. Once the neural network weights are optimised, a HEMS has to simply forward the network to receive its next action, which does not require a powerful machine. On the other hand, MILP has the opposite: no initial training costs, but with the drawback that a significant amount of computational power goes into computing the next set of actions. The computation time would not exceed its time budget of a few hours, but running it for a prolonged period of time might result in MILP having used more resources than RL at the end of it. The length of this period is dependent on various factors, such as MILP horizon, RL training time, and machine specifications, which makes it difficult for quantification.

(ii) **Robustness**: Weather and power grid tariffs have shown to be non-static, i.e.: it slowly changes over time (e.g., higher mean temperatures due to climate change), for instance. As a consequence, the optimal behaviour will also change over time, which has been demonstrated by comparing MDQL's behaviour in July and November, and also by reducing the training data. This in itself is not a major issue, since those mean values will increase very slowly over the years. However, extreme weather occurrences might occur more frequently, upon which a RL approach is trained too little or not at all. This raises the question as to whether it is required for a RL approach to be retrained after a certain period of time, in order to ensure its robustness against extreme weather. Similarly, to what extent can RL be deployed in other countries with different climates; to what extent is transfer learning a feasible solution for this? As for MILP, it would simply require a different forecasting model per country, its actual program remains unchanged.

(iii) **Increasing complexity**: The problem setting can be expanded upon, for instance by adding more household appliances, making the action space continuous, or having multiple actions to select at once (e.g., one action for managing the battery flow and one for appliance scheduling). This often results in a larger state-action space, and due to the curse of dimensionality, MILP with the current horizon might not be feasible anymore. In these cases, RL might be the preferred choice.

All in all, we argue that in the current problem setting, MILP seems to be the preferred choice due to its performance and the lack of a high-dimensional action space in the environment. But in case of more complex problem settings, we believe that RL would be able to outperform MILP, when given an equal amount of computational resources.

EcoGenie Environment. The strategy of MDQL in the EcoGenie environment mainly consists of maximising the PV utilisation, and deploying it to the household outside sun-hours. In addition, the battery is often charged from the grid at midnight since the power is relatively cheap at that point in time. However, an in-depth analysis shows that there are still gains to be made by exploiting the grid tariff spikes more effectively.

Moreover, the final experiment has shown that incorporating the battery State-of-Health (SoH) into the reward function results in the agent not utilising the battery at all. With the current setup, it is too difficult to overcome the penalty for both discharging to the household or netting to the grid. This does highlight as to whether the current setup is cost-effective, since the costs of the battery would effectively nullify the savings. The remaining benefit becomes minimising the carbon footprint of the household. Additional research is required in the form of incorporating more advanced and realistic battery dynamics along with a more accurate SoH, and would indicate whether the setup of a BtM battery with PV panels is worthwhile.

8.1 Future Work

Algorithmic Work. The experiments regarding MDQL have shown that there is potential in planning over the dynamics models. So one of the main suggestions would be to improve the dynamics models by supplying more diverse data, or more hyperparameter and network architecture tuning. In addition, Hindsight-Weighted Planning might be more suited to true offline environments, where we do not have access to an environment, so it would be interesting to confirm whether the approach holds up in these settings. Another idea would be to utilise a different planning scheme altogether, such as Monte-Carlo Tree Search.

In general, it would be interesting to investigate for how long historical data is relevant to reinforcement learning (or deep learning altogether) approaches. With what frequency would an agent have to be retrained with relevant data, if at all. Different train/validation/test splits might already provide more intuition in this area. Similarly, is there a need to have an agent trained specifically for each climate/country? As we have demonstrated, the optimal behaviour differs for each season, which might extend to different climates.

As mentioned earlier, the main weakpoint of this study is the fact that MILP makes use of perfect information, resulting in an unfair comparison against other approaches. Future work should benchmark MILP that makes use of forecasts or historical data. This would demonstrate whether the current performance gap still holds in a more realistic setting for MILP, and whether reinforcement learning or linear programming is the preferred approach given the current problem setting.

Moreover, the computational costs of deploying MILP versus reinforcement learning should also be factored into deciding between the two approaches. MILP requires more compute time and/or better hardware, which both result in higher operating costs.

Environment Work. When we consider the current state of the EcoGenie environment, its main shortcomings are the battery dynamics with its State-of-Health approximation, along with the discrete action space. More realistic battery dynamics might make the SoH-penalty more dependent on the previous actions that have been selected (e.g., using a battery frequently over a short interval results in a high operating temperature, which can have various side effects). In addition, this would also indicate whether the current setting of having PV panels with a BtM battery is cost-effective. If not, then the optimisation objective can also be changed such that the carbon footprint is minimised (i.e., maximising the PV-utilisation) instead, which is currently implicitly incorporated into the reward function.

As for the action space, making it continuous will give the HEMS more control over the battery power flow. Currently, the charge/discharge rate is maximised, but we believe that it can be beneficial to reduce the rate in order to keep the battery operating temperature low (in case the battery dynamics have been improved) and reducing the costs of drawing power from the grid during higher tariffs.

In addition, the user profile currently assumes that the household is occupied at all times. A stochastic method could be developed to make this aspect more realistic and simulate empty households during vacations/travel.

Lastly, it might be interesting to apply the algorithms to more complex problem settings, such as optimising the power consumption of a microgrid consisting of multiple households, which can be achieved through, for instance, the use of a cooperating multi-agent RL approach or graph neural networks. In case of a single household, new appliances (e.g., washing machine, dishwasher, induction stove) can be added to the problem setting, such that the HEMS would also be able to take those activities into account, or even schedule them (only applicable to deferrable appliances).

References

Ahmed, Maytham S., Azah Mohamed, Raad Z. Homod, and Hussain Shareef (2016). "Hybrid LSA-ANN Based Home Energy Management Scheduling Controller for Residential Demand Response Strategy". In: *Energies* 9.9. ISSN: 1996-1073. DOI: 10.3390/en9090716. URL: https://www.mdpi.com/1996-1073/9/9/716.

Argenson, Arthur and Gabriel Dulac-Arnold (2021). "Model-Based Offline Planning". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=OMNB1G5xzd4.

Arroyo, Javier, Carlo Manna, Fred Spiessens, and Lieve Helsen (2021). "An OpenAI-Gym Environment for the Building Optimization Testing (BOPTEST) Framework". In: *Proceedings of the 17th IBPSA Conference*.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath (2017). "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Process. Mag.* 34.6, pp. 26–38. DOI: 10.1109/MSP.2017.2743240. URL: https://doi.org/10.1109/MSP.2017.2743240.

Bahrami, Shahab, Vincent W. S. Wong, and Jianwei Huang (2018). "An Online Learning Algorithm for Demand Response in Smart Grid". In: *IEEE Trans. Smart Grid* 9.5, pp. 4712–4725. DOI: 10.1109/TSG.2017.2667599. URL: https://doi.org/10.1109/TSG.2017.2667599.

Beal, Logan, Daniel Hill, R Martin, and John Hedengren (2018). "GEKKO Optimization Suite". In: *Processes* 6.8, p. 106. DOI: 10.3390/pr6080106.

Bellman, Richard (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37.

Benth, Fred Espen and Juratė Šaltytė-Benth (2005). "Stochastic modelling of temperature variations with a view towards weather derivatives". In: *Applied Mathematical Finance* 12.1, pp. 53–85.

Berner, Christopher, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang (2019). "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* abs/1912.06680. arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

Blum, David, Javier Arroyo, Sen Huang, Ján Drgoňa, Filip Jorissen, Harald Taxt Walnum, Yan Chen, Kyle Benne, Draguna Vrabie, Michael Wetter, et al. (2021). "Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings". In: *Journal of Building Performance Simulation* 14.5, pp. 586–610.

Botev, Zdravko I, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer (2013). "The cross-entropy method for optimization". In: *Handbook of statistics*. Vol. 31. Elsevier, pp. 35–59.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "OpenAI Gym". In: *CoRR* abs/1606.01540. arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

Buckman, Jacob, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee (2018). "Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 8234–8244. URL: https://proceedings.neurips.cc/paper/2018/hash/f02208a057804ee16ac72ff4d3cec53b-Abstract.html.

C3S, Copernicus Climate Change Service (2022). "ERA5-Land hourly data from 1950 to present." In: *Climate Data Store*. Accessed on 24-03-2023. DOI: 10.24381/cds.e2161bac. URL: https://cds.climate.copernicus.eu/cdsapp#!/dataset/10.24381/cds.e2161bac.

Camacho, Eduardo F and Carlos Bordons Alba (2013). *Model predictive control*. Springer science & business media.

Chaslot, Guillaume Maurice Jean-Bernard Chaslot (2010). *Monte-Carlo Tree Search*. Vol. 24. Maastricht University.

Chen, Xiong-Hui, Yang Yu, Qingyang Li, Fan-Ming Luo, Zhiwei (Tony) Qin, Wenjie Shang, and Jieping Ye (2021). "Offline Model-based Adaptable Policy Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, pp. 8432–8443. URL: https://proceedings.neurips.cc/paper/2021/hash/470e7a4f017a5476afb7eeb3f8b96f9b-Abstract.html.

Chen, Yujiao, Leslie K. Norford, Holly W. Samuelson, and Ali Malkawi (2018). "Optimal control of HVAC and window systems for natural ventilation through reinforcement learning". In: *Energy and Buildings* 169, pp. 195–205. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2018.03.051. URL: https://www.sciencedirect.com/science/article/pii/S0378778818302184.

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine (2018). "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 4759–4770. URL: https://proceedings.neurips.cc/paper/2018/hash/3de568f8597b94bda53149c7d7f5958c-Abstract.html.

Clavera, Ignasi, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel (2018). "Model-Based Reinforcement Learning via Meta-Policy Optimization". In: *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*. Vol. 87. Proceedings of Machine Learning Research. PMLR, pp. 617–629. URL: http://proceedings.mlr.press/v87/clavera18a.html.

Crawley, Drury B, Linda K Lawrie, Curtis O Pedersen, and Frederick C Winkelmann (2000). "Energy plus: energy simulation program". In: *ASHRAE journal* 42.4, pp. 49–56.

Deisenroth, Marc Peter and Carl Edward Rasmussen (2011). "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, pp. 465–472. URL: https://icml.cc/2011/papers/323_icmlpaper.pdf.

Eysenbach, Ben, Ruslan Salakhutdinov, and Sergey Levine (2019). "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 15220–15231. URL: https://proceedings.neurips.cc/paper/2019/hash/5c48ff18e0a47baaf81d8b8ea51eec92-Abstract.html.

Findeis, Arduin, Fiodar Kazhamiaka, Scott Jeen, and Srinivasan Keshav (2022). "Beobench: a toolkit for unified access to building simulations for reinforcement learning". In: *e-Energy '22: The Thirteenth ACM International Conference on Future Energy Systems, Virtual Event, 28 June 2022 - 1 July 2022*. Ed. by Sebastian Lehnhoff, David E. Irwin, and Dan Wang. ACM, pp. 374–382. DOI: 10.1145/3538637.3538866. URL: https://doi.org/10.1145/3538637.3538866.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1126–1135. URL: http://proceedings.mlr.press/v70/finn17a.html.

Fujimoto, Scott, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau (2019a). "Benchmarking Batch Deep Reinforcement Learning Algorithms". In: *CoRR* abs/1910.01708. arXiv: 1910.01708. URL: http://arxiv.org/abs/1910.01708.

Fujimoto, Scott, David Meger, and Doina Precup (2019b). "Off-Policy Deep Reinforcement Learning without Exploration". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2052–2062. URL: http://proceedings.mlr.press/v97/fujimoto19a.html.

Girshick, Ross B. (2015). "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169. URL: https://doi.org/10.1109/ICCV.2015.169.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1856–1865. URL: http://proceedings.mlr.press/v80/haarnoja18b.html.

Hasselt, Hado van, Arthur Guez, and David Silver (2016). "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, pp. 2094–2100. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

Hernandez, Cesar A., Ricardo Romero, and Diego Giral (2010). "Optimization of the Use of Residential Lighting with Neural Network". In: *2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1–5. DOI: 10.1109/CISE.2010.5677018.

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580. arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

Hu, Maomao and Fu Xiao (2018). "Price-responsive model-based optimal demand response control of inverter air conditioners using genetic algorithm". In: *Applied Energy* 219, pp. 151–164. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2018.03.036. URL: https://www.sciencedirect.com/science/article/pii/S0306261918303672.

Huang, Zhiao, Fangchen Liu, and Hao Su (2019). "Mapping State Space using Landmarks for Universal Goal Reaching". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 1940–1950. URL: https://proceedings.neurips.cc/paper/2019/hash/3b712de48137572f3849aabd5666a4e3-Abstract.html.

IEA (2021). "Net Zero by 2050". In: License: CC BY 4.0. Paris. URL: https://www.iea.org/reports/net-zero-by-2050.

Janner, Michael, Justin Fu, Marvin Zhang, and Sergey Levine (2019). "When to Trust Your Model: Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 12498–12509. URL: https://proceedings.neurips.cc/paper/2019/hash/5faf461eff3099671ad63c6f3f094f7f-Abstract.html.

Kidambi, R., A. Rajeswaran, P. Netrapalli, and T. Joachims (2020). "MOReL: Model-Based Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/f7efa4f864ae9b88d43527f4b14f750f-Abstract.html.

Kim, Byung-Gook, Yu Zhang, Mihaela van der Schaar, and Jang-Won Lee (2016). "Dynamic Pricing and Energy Consumption Scheduling With Reinforcement Learning". In: *IEEE Trans. Smart Grid* 7.5, pp. 2187–2198. DOI: 10.1109/TSG.2015.2495145. URL: https://doi.org/10.1109/TSG.2015.2495145.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, pp. 6402–6413. URL: https://proceedings.neurips.cc/paper/2017/hash/9ef2ed4b7fd2c810847ffa5fa85bce38-Abstract.html.

Lange, Sascha, Thomas Gabel, and Martin A. Riedmiller (2012). "Batch Reinforcement Learning". In: *Reinforcement Learning*. Ed. by Marco A. Wiering and Martijn van Otterlo. Vol. 12. Adaptation, Learning, and Optimization. Springer, pp. 45–73. DOI: 10.1007/978-3-642-27645-3_2. URL: https://doi.org/10.1007/978-3-642-27645-3_2.

Leibfried, Felix and Peter Vrancx (2018). *Model-Based Regularization for Deep Reinforcement Learning with Transcoder Networks*. arXiv: 1809.01906 [cs.LG].

Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu (2020). "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *CoRR* abs/2005.01643. arXiv: 2005.01643. URL: https://arxiv.org/abs/2005.01643.

Lokeshgupta, B. and S. Sivasubramani (2019). "Multi-objective home energy management with battery energy storage systems". In: *Sustainable Cities and Society* 47, p. 101458. ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2019.101458. URL: https://www.sciencedirect.com/science/article/pii/S2210670718312496.

Loshchilov, Ilya and Frank Hutter (2019). "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Lugo-Cordero, Hector M., Abigail Fuentes-Rivera, Ratan K. Guha, and Eduardo I. Ortiz-Rivera (2011). "Particle Swarm Optimization for load balancing in green smart homes". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*. IEEE, pp. 715–720. DOI: 10.1109/CEC.2011.5949689. URL: https://doi.org/10.1109/CEC.2011.5949689.

Luo, Yuping, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma (2019). "Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=BJe1E2R5KX.

Mason, Karl and Santiago Grijalva (2019). "A review of reinforcement learning for autonomous building energy management". In: *Comput. Electr. Eng.* 78, pp. 300–312. DOI: 10.1016/j.compeleceng.2019.07.019. URL: https://doi.org/10.1016/j.compeleceng.2019.07.019.

Matallanas, E., M. Castillo-Cagigal, A. Gutiérrez, F. Monasterio-Huelin, E. Caamaño-Martín, D. Masa, and J. Jiménez-Leube (2012). "Neural network controller for Active Demand-Side Management with PV energy in the residential sector". In: *Applied Energy* 91.1, pp. 90–97. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2011.09.004. URL: https://www.sciencedirect.com/science/article/pii/S0306261911005630.

Matsushima, Tatsuya, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu (2021). "Deployment-Efficient Reinforcement Learning via Model-Based Offline Optimization". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=3hGNqpI4WS.

Mattsson, Sven Erik and Hilding Elmqvist (1997). "Modelica—An international effort to design the next generation modeling language". In: *IFAC Proceedings Volumes* 30.4, pp. 151–155.

Mbuwir, Brida V., Frederik Ruelens, Fred Spiessens, and Geert Deconinck (2017). "Battery Energy Management in a Microgrid Using Batch Reinforcement Learning". In: *Energies* 10.11. ISSN: 1996-1073. DOI: 10.3390/en10111846. URL: https://www.mdpi.com/1996-1073/10/11/1846.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Moerland, Thomas M., Joost Broekens, and Catholijn M. Jonker (2020). "Model-based Reinforcement Learning: A Survey". In: *CoRR* abs/2006.16712. arXiv: 2006.16712. URL: https://arxiv.org/abs/2006.16712.

Moon, Seungyong, JunYeong Lee, and Hyun Oh Song (2022). "Rethinking Value Function Learning for Generalization in Reinforcement Learning". In: *NeurIPS*. URL: http://papers.nips.cc/paper_files/paper/2022/hash/e19ab2dde2e60cf68d1ded18c38938f4-Abstract-Conference.html.

Moriyama, Takao, Giovanni De Magistris, Michiaki Tatsubori, Tu-Hoa Pham, Asim Munawar, and Ryuki Tachibana (2018). "Reinforcement Learning Testbed for Power-Consumption Optimization". In: *CoRR* abs/1808.10427. arXiv: 1808.10427. URL: http://arxiv.org/abs/1808.10427.

Nogueira, Fernando (2014). *Bayesian Optimization: Open source constrained global optimization tool for Python*. URL: https://github.com/fmfn/BayesianOptimization.

Oh, Junhyuk, Satinder Singh, and Honglak Lee (2017). "Value Prediction Network". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, pp. 6118–6128. URL: https://proceedings.neurips.cc/paper/2017/hash/ffbd6cbb019a1413183c8d08f2929307-Abstract.html.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, pp. 4026–4034. URL: https://proceedings.neurips.cc/paper/2016/hash/8d8818c8e140c64c743113f563cf750f-Abstract.html.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Plaat, Aske, Walter A. Kosters, and Mike Preuss (2021). "High-Accuracy Model-Based Reinforcement Learning, a Survey". In: *CoRR* abs/2107.08241. arXiv: 2107.08241. URL: https://arxiv.org/abs/2107.08241.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley. ISBN: 978-0-47161977-2. DOI: 10.1002/9780470316887. URL: https://doi.org/10.1002/9780470316887.

Raboso, Javier Jiménez, Alejandro Campoy-Nieves, Antonio Manjavacas-Lucas, Juan Gómez-Romero, and Miguel Molina-Solana (2021). "Sinergym: a building simulation and control framework for training reinforcement learning agents". In: *BuildSys '21: The 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Coimbra, Portugal, November 17 - 18, 2021*. Ed. by Xiaofan Fred Jiang, Omprakash Gnawali, and Zoltan Nagy. ACM, pp. 319–323. DOI: 10.1145/3486611.3488729. URL: https://doi.org/10.1145/3486611.3488729.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations".

In: *Journal of Machine Learning Research* 22.268, pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Sabater, J. Muñoz, E. Dutra, A. Agustí-Panareda, C. Albergel, G. Arduini, G. Balsamo, S. Boussetta, M. Choulga, S. Harrigan, H. Hersbach, B. Martens, D. G. Miralles, M. Piles, N. J. Rodríguez-Fernández, E. Zsoter, C. Buontempo, and J.-N. Thépaut (2021). "ERA5-Land: a state-of-the-art global reanalysis dataset for land applications". In: *Earth System Science Data* 13.9, pp. 4349–4383. DOI: 10.5194/essd-13-4349-2021. URL: https://essd.copernicus.org/articles/13/4349/2021/.

Sak, Hasim, Andrew W. Senior, and Françoise Beaufays (2014). "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition". In: *CoRR* abs/1402.1128. arXiv: 1402.1128. URL: http://arxiv.org/abs/1402.1128.

Scharnhorst, Paul, Baptiste Schubnel, Carlos Fernández Bandera, Jaume Salom, Paolo Taddeo, Max Boegli, Tomasz Gorecki, Yves Stauffer, Antonis Peppas, and Chrysa Politi (2021). "Energym: A Building Model Library for Controller Benchmarking". In: *Applied Sciences* 11.8. ISSN: 2076-3417. DOI: 10.3390/app11083518. URL: https://www.mdpi.com/2076-3417/11/8/3518.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). "Prioritized Experience Replay". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1511.05952.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz (2015). "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1889–1897. URL: http://proceedings.mlr.press/v37/schulman15.html.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nat.* 529.7587, pp. 484–489. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis (2017). "Mastering the game of Go without human knowledge". In: *Nat.* 550.7676, pp. 354–359. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

Smit, Jordi, Canmanie T. Ponnambalam, Matthijs T.J. Spaan, and Frans A. Oliehoek (2021). "PEBL: Pessimistic Ensembles for Offline Deep Reinforcement Learning". In: *Robust and Reliable Autonomy in the Wild Workshop at the 30th International Joint Conference of Artificial Intelligence*. URL: http://resolver.tudelft.nl/uuid:2053b579-a663-4def-ad25-4bedad0169be.

Sohn, Kihyuk, Honglak Lee, and Xinchen Yan (2015). "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, pp. 3483–3491. URL: https://proceedings.neurips.cc/paper/2015/hash/8d55a249e6baa5c06772297520da2051-Abstract.html.

Souza Dutra, Michael de, Miguel Anjos, and Sébastien Le Digabel (Mar. 2019). "A realistic energy optimization model for smart-home appliances". In: *International Journal of Energy Research* 43. DOI: 10.1002/er.4454.

Sutton, Richard S. (1991). "Dyna, an Integrated Architecture for Learning, Planning, and Reacting". In: *SIGART Bull.* 2.4, pp. 160–163. DOI: 10.1145/122344.122377. URL: https://doi.org/10.1145/122344.122377.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press. ISBN: 978-0-262-19398-6. URL: https://www.worldcat.org/oclc/37293240.

Tan, Zhukui, Xiaoshun Zhang, Baiming Xie, Dezhi Wang, Bin Liu, and YU Tao (Apr. 2018). "Fast learning optimizer for real-time optimal energy management of a grid-connected microgrid". In: *IET Generation, Transmission & Distribution* 12. DOI: 10.1049/iet-gtd.2017.1983.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Vázquez-Canteli, José R. and Zoltán Nagy (2019). "Reinforcement learning for demand response: A review of algorithms and modeling techniques". In: *Applied Energy* 235, pp. 1072–1089. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2018.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0306261918317082.

Wang, John, Ping Liu, Jocelyn Hicks-Garner, Elena Sherman, Souren Soukiazian, Mark Verbrugge, Harshad Tataria, James Musser, and Peter Finamore (2011). "Cycle-life model for graphite-LiFePO4 cells". In: *Journal of Power Sources* 196.8, pp. 3942–3948. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2010.11.134. URL: https://www.sciencedirect.com/science/article/pii/S0378775310021269.

Wang, Yuan, Kirubakaran Velswamy, and Biao Huang (2017). "A Long-Short Term Memory Recurrent Neural Network Based Reinforcement Learning Controller for Office Heating Ventilation and Air Conditioning Systems". In: *Processes* 5.3. ISSN: 2227-9717. DOI: 10.3390/pr5030046. URL: https://www.mdpi.com/2227-9717/5/3/46.

Wilt, Peter van der (July 2022). "Salderen en terugleververgoeding zonnepanelen". In: Consumentenbond. URL: https://www.consumentenbond.nl/zonnepanelen/salderen-en-terugleververgoeding-zonnepanelen.

Yamada, Hiroyuki (Jan. 2019). *cpprb*. URL: https://gitlab.com/ymd_h/cpprb.

Yu, Liang, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan (2020a). "Deep Reinforcement Learning for Smart Building Energy Management: A Survey". In: *CoRR* abs/2008.05074. arXiv: 2008.05074. URL: https://arxiv.org/abs/2008.05074.

Yu, T., Dong Kim, and Sung-Yong Son (Jan. 2013). "Optimization of scheduling for home appliances in conjunction with renewable and energy storage resources". In: *International Journal of Smart Home* 7, pp. 261–272.

Yu, Tianhe, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn (2021). "COMBO: Conservative Offline Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, pp. 28954–28967. URL: https://proceedings.neurips.cc/paper/2021/hash/f29a179746902e331572c483c45e5086-Abstract.html.

Yu, Tianhe, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y. Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma (2020b). "MOPO: Model-based Offline Policy Optimization". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/a322852ce0df73e204b7e67cbbef0d0a-Abstract.html.

Yuce, Baris, Yacine Rezgui, and Monjur Mourshed (2016). "ANN–GA smart appliance scheduling for optimised energy management in the domestic sector". In: *Energy and Buildings* 111, pp. 311–325. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2015.11.017. URL: https://www.sciencedirect.com/science/article/pii/S0378778815303820.

Zafar, Usman, Sertac Bayhan, and Antonio Sanfilippo (2020). "Home Energy Management System Concepts, Configurations, and Technologies for the Smart Grid". In: *IEEE Access* 8, pp. 119271–119286. DOI: 10.1109/ACCESS.2020.3005244. URL: https://doi.org/10.1109/ACCESS.2020.3005244.

Zhang, Lunjun, Ge Yang, and Bradly C. Stadie (2021). "World Model as a Graph: Learning Latent Landmarks for Planning". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 12611–12620. URL: http://proceedings.mlr.press/v139/zhang21x.html.

Zhang, Wei-Jun (2011). "Structure and performance of LiFePO4 cathode materials: A review". In: *Journal of Power Sources* 196.6, pp. 2962–2970. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2010.11.113. URL: https://www.sciencedirect.com/science/article/pii/S037877531002104X.

Zhang, Zhiang and Khee Poh Lam (2018). "Practical implementation and evaluation of deep reinforcement learning control for a radiant heating system". In: *Proceedings of the 5th Conference on Systems for Built Environments, BuildSys 2018, Shenzen, China, November 07-08, 2018*. Ed. by Rajesh Gupta, Polly Huang, and Marta Gonzalez. ACM, pp. 148–157. DOI: 10.1145/3276774.3276775. URL: https://doi.org/10.1145/3276774.3276775.

Zhou, Bin, Wentao Li, Ka Wing Chan, Yijia Cao, Yonghong Kuang, Xi Liu, and Xiong Wang (2016). "Smart home energy management systems: Concept, configurations, and scheduling strategies". In: *Renewable and Sustainable Energy Reviews* 61, pp. 30–40. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2016.03.047. URL: https://www.sciencedirect.com/science/article/pii/S1364032116002823.

A Environment Specifications

To ensure reproducibility, we denote the exact values for all constants in order to accurately recreate the EcoGenie environment in Table A.1.

TABLE A.1: The EcoGenie specifications with corresponding symbols if used.

Attribute	Symbol	Unit	Value
BtM Sonnen Battery			
Max. battery capacity	$\text{SoC}^{\max}$	kWh	13.5
Min. battery capacity	$\text{SoC}^{\min}$	kWh	1.0
Max. charge rate	$\psi^{\max,\,c}$	kWh	3.5
Max. discharge rate	$\psi^{\max,\,d}$	kWh	3.5
Netting threshold	—	—	1.0
Rated no. discharge cycles	—	—	10,000
Purchase price	—	EUR	9,415
PV panels			
Max. production	—	kWh	2.5
Heatpump			
Max. consumption	HP^{c}	kWh	2.5
Max. coefficient of performance	$\text{CoP}^{\max}$	—	4.0
Min. coefficient of performance	$\text{CoP}^{\min}$	—	1.0
EcoGenie House			
Property size	S	m^2	192
House volume	V	m^3	576
House insulation	U	$\text{W/m}^2\text{K}$	0.4
Required heating power per degree Celcius	P	kW	0.2
Household room air heat	—	kJ/kg	1.012
User Profile			
Comfortable lower temperature	$T^{\text{comfort}}_{\text{low}}$	°C	20.0
Comfortable upper temperature	$T^{\text{comfort}}_{\text{high}}$	°C	22.0
Comfortable night temperature offset	T^{offset}	°C	4.0
Start day mean	$\tilde{t}^{\mu}_{\text{day}}$	—	07:00
Start night mean	$\tilde{t}^{\mu}_{\text{night}}$	—	23:00
Variance of Gaussian for generating shift	—	—	0.5
Max. no. hours in shift from mean start day/night	—	—	3
Miscellaneous			
SoH scalar for scenarios 1 and 2	λ_{SoH}	—	1.00
Wasted actions reward multiplier	m	—	1.05
Reasonable compensation `NETTING`	—	EUR	0.01
Min. observed reward @ scenario 1	—	—	-2.00
Max. observed reward @ scenario 1	—	—	0.10

B Heatpump and Heating Dynamics

In this appendix, we describe the computations for the heating and heatpump dynamics. They have been taken from an environment internally developed at Shell. First, the predicted temperature loss T^L at timestep t is computed via Equation B.1, where U and S are the insulation value and surface area of the house, respectively. Next, T^{in} and T^{out} are the indoor and outdoor temperatures, respectively. Lastly, P is the number of kWh required to heat the house per degree Celcius, which can be approximated by multiplying the household room air heat with the air density and house volume.

$$T_t^L \doteq \left(U \cdot S \cdot \left(T_{t-1}^{\text{in}} - T_t^{\text{out}}\right) / 10^3\right) \Big/ P \tag{B.1}$$

Given the predicted heat loss, we compute the coefficient of performance (CoP) of the heatpump for the next timestep with Equation B.2. Since the efficiency can not be 100%, we conservatively approximate the true CoP by assuming the efficiency is 80%.

$$\text{CoP}_t \doteq \max\left(\text{CoP}^{\min}, \min(\text{CoP}^{\max}, \text{CoP}'_t)\right) \text{ with } \text{CoP}'_t \doteq 0.8 \cdot \frac{T_t^{\text{in}}}{\max\left(1, \left|T_{t-1}^{\text{in}} - T_t^{\text{out}}\right|\right)} \tag{B.2}$$

Next, the temperature difference T^Δ to the lower and upper comfortable temperatures $T_{\text{low}}^{\text{comfort}}$ and $T_{\text{high}}^{\text{comfort}}$ are computed according to Equation B.3. If the indoor temperature is below the lower comfortable temperature threshold, then T_{low}^Δ will become negative. In similar fashion, T_{high}^Δ will become negative if the current indoor temperature is above the upper comfortable temperature.

$$\begin{aligned} T_{\text{low},t}^\Delta &\doteq T_{t-1}^{\text{in}} - T_{\text{low}}^{\text{comfort}} \\ T_{\text{high},t}^\Delta &\doteq T_{\text{high}}^{\text{comfort}} - T_{t-1}^{\text{in}} \end{aligned} \tag{B.3}$$

In case the indoor temperature is outside the comfortable range of temperatures, we correct it by subtracting/adding the required degrees Celcius in order to be within the range. This is achieved by Equation B.4.

$$T_t^{\text{in}} \doteq \begin{cases} T_{t-1}^{\text{in}} + \left|T_{\text{low},t}^\Delta\right|, & \text{if } T_{\text{low},t}^\Delta < 0 \\ T_{t-1}^{\text{in}} - \left|T_{\text{high},t}^\Delta\right|, & \text{if } T_{\text{high},t}^\Delta < 0 \\ T_{t-1}^{\text{in}}, & \text{otherwise} \end{cases} \tag{B.4}$$

Next, we compute the required heatpump output power HP^{out} in order to maintain the newly set indoor temperature and compensating for the earlier computed heat loss T^L. Given the required output power of the heatpump, we also compute the required input power HP^{in}.

$$\begin{aligned} \text{HP}_t^{\text{out}} &\doteq P \cdot (T_t^{\text{in}} + T_t^L) \\ \text{HP}_t^{\text{in}} &\doteq \text{HP}_t^{\text{out}} / \text{CoP}_t \end{aligned} \tag{B.5}$$

In case $\text{HP}_t^{\text{in}} > \text{HP}^c$, we deploy an auxiliary heating source (CoP = 1) to take care of the remaining heating that the heatpump is not able to deliver.

C Heuristic-Based System Insights

Due to the stochastic nature of optimising the parameters of the Heuristic-Based System (HBS), we ran three replications on scenario 1 of the environment. Table C.1 denotes the rounded parameters of each replication, along with a brief description of the purpose of each parameter. Since HBS receives normalised features, the values of the parameters (bar the offsets; ϕ_3, ϕ_5 and ϕ_7) are also normalised. The parameters ϕ_3, ϕ_5 and ϕ_7 are optimised in the range (-0.02, 0.02).

In the end, the three replications obtained a test score of -416.10, -418.47 and -419.11, respectively. In addition, the values of some parameters differ significantly between the replications. For instance, ϕ_4, the SoC-threshold to net power back to the grid, is significantly lower for replication 3.

TABLE C.1: The best found parameters per replication, optimised on the training and validation environment. The parameters are optimised with Bayesian Optimisation. Note that the values of the parameters have been rounded to three decimals.

Parameters Φ	**Replication**			**Description**
	1	**2**	**3**	
ϕ_0	0.338	0.210	0.192	Threshold for the expected PV-production. If it is above ϕ_0, then the heuristic considers charging the battery with it, which is also dependent on the battery SoC and ϕ_1.
ϕ_1	0.795	0.996	0.830	Threshold for when the battery SoC is too high to make charging the battery worthwhile.
ϕ_2	0.197	0.214	0.214	Threshold for when there is sufficient power in the battery to discharge to the residential house.
ϕ_3	-0.017	-0.001	-0.003	Offset to the moving average (MA) grid price. Indicates when it is profitable to supply the requested power of the household from the battery instead of from the power grid.
ϕ_4	0.964	0.869	0.645	SoC threshold for when there is sufficient power in the battery for netting.
ϕ_5	0.017	0.005	-0.002	Offset to the MA grid price, when it is profitable to net power back to the grid. Note that the SoH penalty also has to be overcome in this case.
ϕ_6	0.976	0.814	0.856	SoC threshold when to consider charging the battery from the grid.
ϕ_7	0.011	-0.010	0.016	Offset to the MA grid price when it is not too expensive to charge the battery from the grid.

Best parameter configuration. When we review the parameters of the best performing replication, we can see that its thresholds for the expected PV-production and maximum battery SoC are relatively high. Thus, it only opts to charge with PV when there is plenty available and possible to store in the battery. Next, it chooses to relatively quickly discharge the battery into the household, with a low SoC threshold (ϕ_2) and a low threshold for the grid tariffs. `NETTING` is only selected with for almost fully charged battery and high grid tariff. Lastly, the thresholds for `CHARGE_GRID` are also relatively strict, with a high offset to the MA tariff.

All in all, replication 1 tries to optimise the sequential decision making process by maximising its PV-utilisation, and then discharge it into the household whenever possible. Figure C.1 confirms this for replication 1, which shows the decision making of the HBS agent over one week in July 2019, which is a week sampled from the test environment. During this week, the battery is mainly charged by PV, and then deployed to supply power to the heatpump. But, it is still wasting some available PV, in these cases the battery was already nearly full. At some nights, the battery is being charged from the grid for a reasonable tariff, in order to be used later in the day. However, we also see large periods where the agent decides to do nothing with its fully charged battery (e.g., the 6th of July).

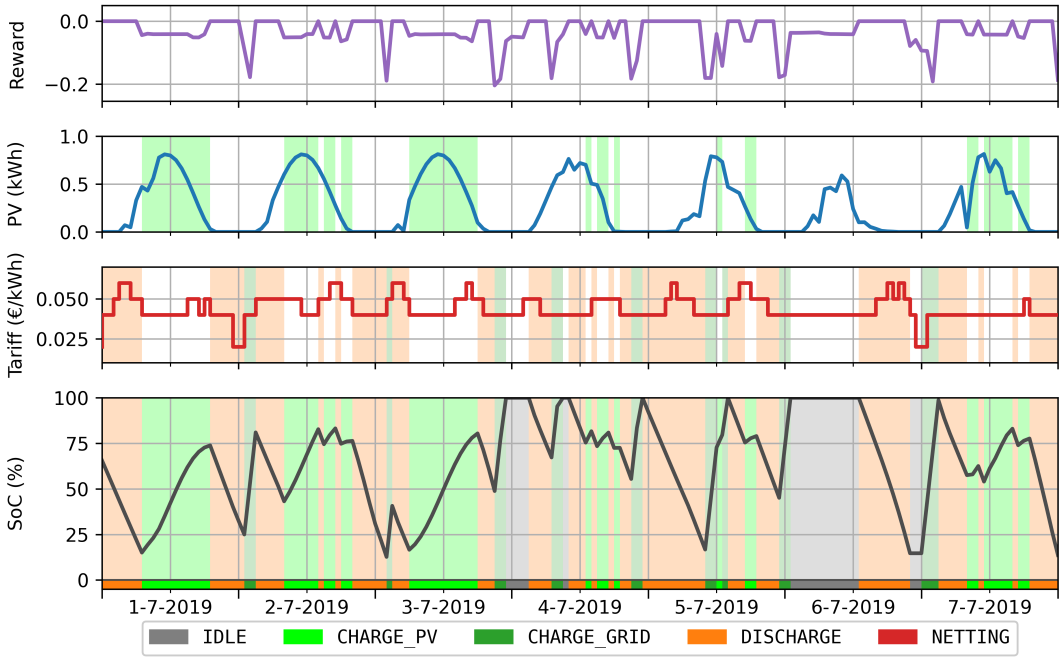

FIGURE C.1: Battery usage of the HBS over one week in July 2019 (taken from the test environment). For this, we used the best performing set of parameters, namely those from replication 1. From top to bottom, the plots show the reward signal, available PV, grid tariff, and the progression of the battery SoC. For the SoC, the agent is able to fully use it from 0 to 100%; the margins mentioned in Section 4.1.1 are outside this range. I.e.: at 0%, there is still some charge left in the battery, but this can not be accessed by the agent in order to prolong the battery lifespan. As for the background color, it denotes what action was selected at that point in time, with the relevant actions also being shown in the tariff and PV plots. Noteworthy is how the battery is mainly being charged by PV and then deployed to power the heatpump; there is no `NETTING` at all. In addition, charging from the grid mainly occurs at night, when there is no PV available but the battery SoC is still relatively low.

General trend in parameter configuration performance. In general, Table C.1 highlights that different approaches are possible. There does seem to be a trend regarding what parameter values result in relatively high cumulative rewards. In Figure C.2, we have plotted each Φ that has been evaluated. In order to visualise it, we applied Principal Component Analysis (PCA) to reduce the seven dimensions of Φ back to two. The implementation has been taken from Scikit-Learn (Pedregosa et al., 2011).

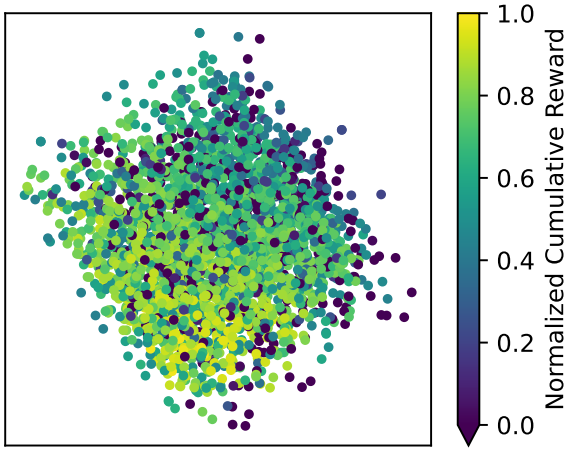

FIGURE C.2: The evaluated parameter configurations plotted in two-dimensional space with PCA. The color of the datapoints denote the obtained normalised cumulative score over both the train and validation environment. Due to extreme negative outliers, the scores have been normalised over the top 85% of datapoints. Thus, the remaining 15% of datapoints have the same color as the worst datapoint of the best 85%, which is denoted by the pointy end at the bottom of the colorbar. In general, the scatterplot shows a trend where better scores are obtained in the lower left part of the plot. However, noteworthy is the fact that minor changes in that region can still result in a significant deterioration of performance. This is likely due to a significant change in one of the first parameters that are used by the heuristic (e.g., setting ϕ_0 and ϕ_1 to a low and high value, respectively, results in that `CHARGE_PV` will often be selected as action).

Noteworthy is the fact that small changes to the parameter configuration can result in a significant deterioration in performance, as is highlighted by the purple datapoints that are surrounded by yellow. This can be explained by the nature of the heuristic: the binary decision tree is in fact a long chain of if-statements. Consequently, changes in parameters that are part of the first few statements can have a big impact on the overall performance. For instance, if we set the SoC-threshold ϕ_0 for charging from PV too strict, then the battery will never be charged with PV-power.

D Reinforcement Learning Hyperparameter Configurations

The exact hyperparameter configuration for each reinforcement learning (RL) method is denoted in this appendix. First, we show the values regarding the baselines, after which we summarise the configuration of the Multi Dynamics- and Q-Learning models. For all baseline configurations holds that if an hyperparameter is not denoted here, we have taken the default value from the corresponding implementation. We refer the reader to Section 6.3.3 for the specific codebases of each baseline. Table D.1 contains the hyperparameter configurations of all RL baselines: DQN, BCQ, PPO and recurrent PPO (rPPO).

TABLE D.1: Hyperparameter configuration of all RL baselines. The reduction period of ϵ denotes the fraction of timesteps in the training trajectory over which the ϵ is decayed from its starting value to its ending value. Stable-Baselines3 refers to this as the 'exploration fraction'. As exploration strategy, DQN and BCQ use ϵ-greedy. Lastly, '—' indicates that a certain setting did not apply to the corresponding algorithm.

Hyperparameter	**RL Algorithm**			
	DQN	**BCQ**	**PPO**	**rPPO**
Optimiser	Adam			
Learning rate α	0.001	0.00001	0.0001	0.0001
Batch size	128			
Network architecture	[256, 256]	[64, 64, 64]	[256, 256]	[256, 128]
Discount factor γ	0.99			
Replay buffer capacity	100,000	100,000	—	—
Target update coefficient τ	1.00	0.005	—	—
Target update frequency	10,000	1	—	—
Initial exploration ϵ	1.00	1.00	—	—
Final exploration ϵ	0.05	0.05	—	—
Reduction period of ϵ	0.10	0.10	—	—
BCQ threshold Φ_{BCQ}	—	0.10	—	—
GAE	—	—	0.95	0.95
Entropy coefficient	—	—	0.01	0.01

The best found hyperparameter configurations for MDQL and rMDQL are denoted in Table D.2. As the table shows, the only difference between the two architectures is the use of LSTM layers and planning configuration. For the remainder, the optimal found hyperparameters have been copied from MDQL to rMDQL.

TABLE D.2: The hyperparameter configurations of MDQL and rMDQL. The Q-network architectures do not contain any hidden layers, thus it is a direct linear mapping between the output of the feature extractor to the q-values.

Hyperparameter	**MDQL**	**rMDQL**
Optimiser		
Optimiser algorithm	AdamW	
AMSGrad	✓	
Learning rate α	10^{-4}	
Learning rate scheduler	✗	
Gradient clipping by norm	1.0	
Gradient clipping by value	100.0	
Loss function	smooth_{ℓ_1}	
Network architecture		
Ensemble size	4	
Feature extractor $\mathcal{F}$	[1024, 1024]	
Q-network	—	
$\hat{D}$-network	[256, 256]	
$\hat{R}$-network	[128, 128]	
Dropout (for $\hat{D}$ and $\hat{R}$)	0.3	
Activation function	ReLU	
Experience replay		
QET replay buffer capacity	200,000	
DDT replay buffer capacity	1,000,000	
Prioritized Experience Replay	✓	
α_{PER}	0.6	
β_{PER}	0.4	
Target networks		
Coefficient τ pre-convergence	1.0	
Coefficient τ post-convergence	0.0	
Update frequency	10,000	
Exploration		
Initial exploration ϵ	0.9	
Final exploration ϵ	0.4	
Reduction period of ϵ	0.1	
Planning		
Rollout length u	4	4
Top-k action selection	3	3
ω	10^{-7}	
Miscellaneous		
Discount factor γ	0.99	
Improvement steps for convergence	10	
Batch size	128	
Transition length l	1	8

E Q-Learning on the EcoGenie Environment

The experimental results demonstrated that DQN and variations of MDQL have similar shaped learning curves, where they reach peak performance relatively quickly, and then drop off to a performance similar to the 'Idle'-baseline. This dropoff seems to be caused by diverging q-value estimations. Since the environment mostly returns negative rewards, the q-values become more extreme over time due to bootstrapping.

In order to verify this, we have ran MDQL, without Diverse Dynamics Training nor setting τ to zero at some point in the training procedure, with four different target q-value computation strategies.

- **Default**: As described in Section 5.2, for each Q-head we take the q-values according to its target network.
- **Minimum**: For each action, we take the minimum predicted q-value over *all* target networks in the Q-ensemble.

$$q^{(a)} \doteq \min_{i \in \{0,\dots,n-1\}} Q_{\theta',i}\big(\mathcal{F}_{\theta'}(s), a\big)$$

- **Maximum**: Similarly to the minimum, but instead we take the maximum over all target networks.

$$q^{(a)} \doteq \max_{i \in \{0,\dots,n-1\}} Q_{\theta',i}\big(\mathcal{F}_{\theta'}(s), a\big)$$

- **Confidence Lower Bound (CLB)**: The confidence lower bound (introduced in Section 5.1) is computed for each action over the whole ensemble.

$$q^{(a)} \doteq Q_{\theta',\mathrm{CLB}}\big(\mathcal{F}_{\theta'}(s), a\big)$$

By taking one of the three latter strategies, we hypothesise that the difference between the ensemble heads will decrease, and consequently the accuracy of quantifying the epistemic uncertainty.

In Figure E.1, we have plotted the learning curve on the validation environment for each computation strategy, along with the trajectory of the mean estimated q-values and Q-head disagreement. We have to note that a slightly different hyperparameter configuration to Appendix D was used for these experiments, with the main differences being the network architecture of all approximators and the feature extractor, along with a different constant learning rate.

First of all, the middle plot shows that the mean q-estimations keep decreasing linearly, and notably, with maximum as computation strategy with the least steepest slope. Moreover, maximum is also the strategy that settles to the highest cumulative reward and obtains the best peak performance. Thus, these results suggest that indeed the diverging q-values cause the drop off in performance.

Next, we have also plotted the Q-head disagreement in the bottom graph. It confirms our hypothesis that taking a computation strategy where we aggregate over the whole ensemble results in a significantly lower disagreement.

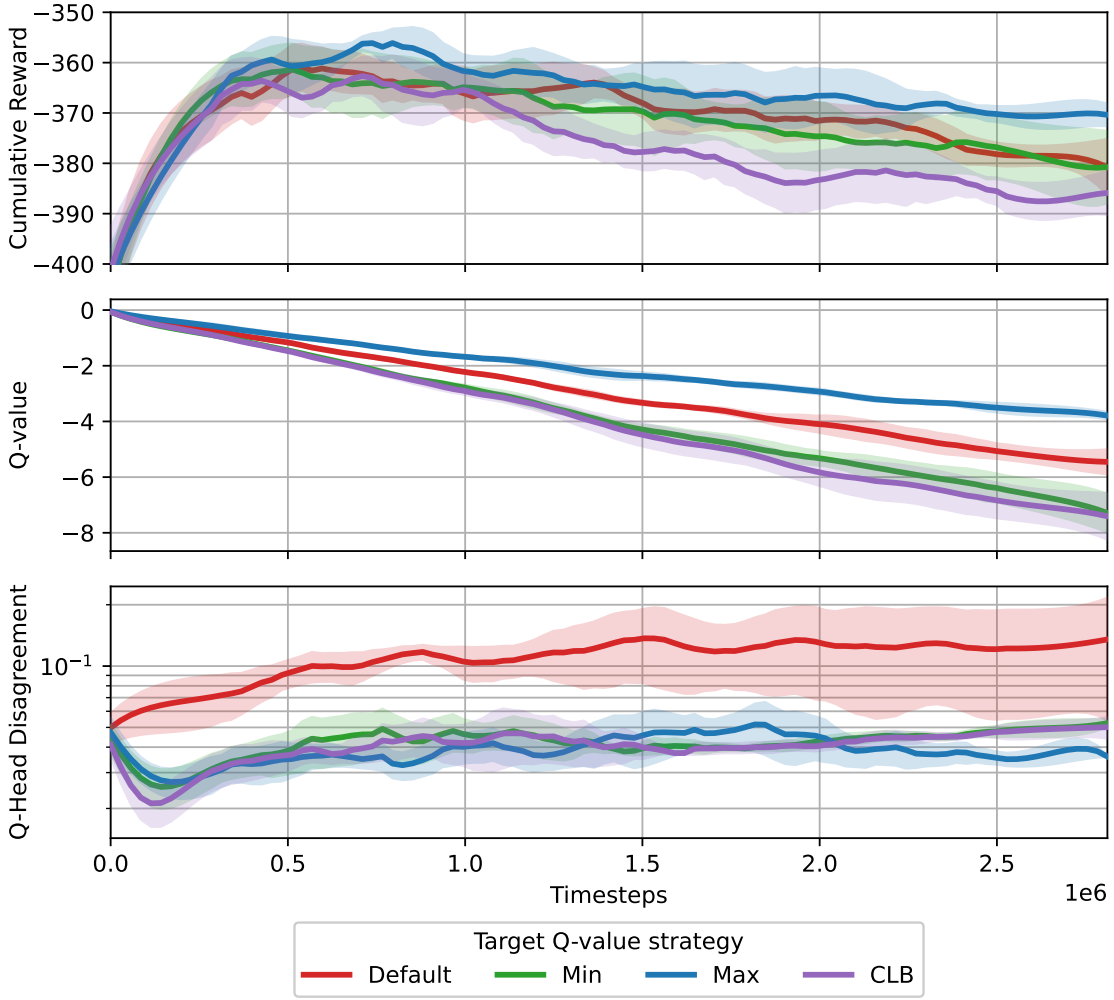

FIGURE E.1: State-action value trajectory for different target q-value strategies. The top plot shows the learning curve on the validation environment, while the bottom two plots show the mean predicted q-value over the whole ensemble and mean Q-head disagreement, respectively. The Q-head disagreement is computed by taking the action-wise standard deviation over the whole ensemble, after which the mean over all the standard deviations is taken as the Q-head disagreement for that evaluation episode.

F MDQL Dynamics Models Performance

In this appendix, we provide some additional details on the (r)MDQL dynamics models performance. Figure F.1 shows the trajectory of the loss of the dynamics models and reward function approximator.

As is shown, both approximators overfit on the training data to a certain degree. In addition, the validation loss is slightly higher for rMDQL on both the dynamics and rewards losses. Moreover, the sudden spike in training loss can be explained by the fact that a switch is made at that point from Q-Ensemble Training to Diverse Dynamics Training.

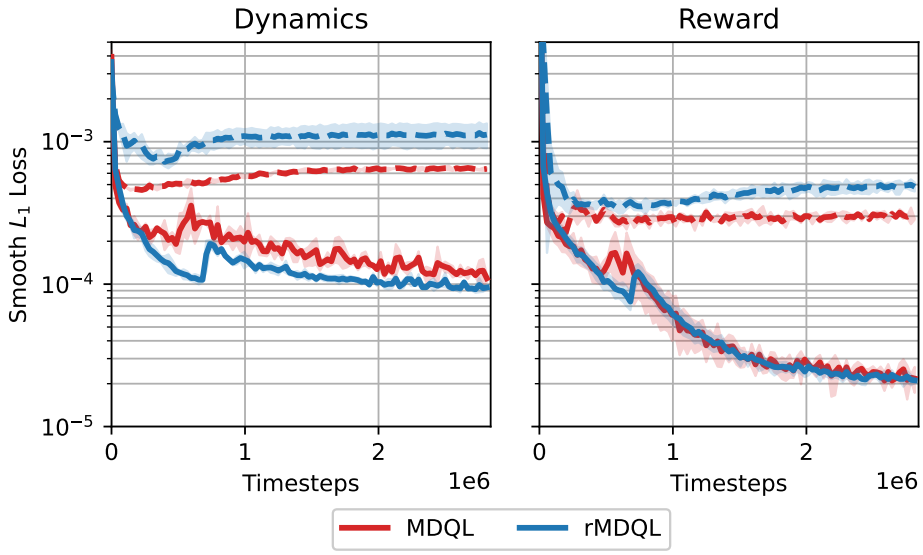

FIGURE F.1: Mean dynamics and reward loss for MDQL and rMDQL. The solid line denotes the loss on the training environment, while the dashed line is with regards to traces generated during evaluations on the validation environment.

In Figure F.2, we have plotted the feature and reward absolute error of MDQL. Both for traces generated with actions according to the Q-ensemble, as well as for random actions. The figure shows that there are some differences in accuracy between the ensemble heads. Moreover, traces generated with random actions result in slightly higher errors for some of the features (most notably battery SoH, netting and the reward). Lastly, the magnitude of the error differs significantly between each feature, with the PV-feature having the highest mean absolute error, followed by the battery SoC, grid tariff and weekday.

The main takeaway is that the deterioration in performance when using planning is most likely due to the inaccurate PV and reward estimations. We think that the following new features might be useful for the dynamics model in order to make this prediction more accurate: the PV-production of the last n hours, along with a categorical variable denoting the month of the year. In addition, the figure does not show the outliers to ensure readability, but unfortunately there were quite a few outliers of differing magnitudes.

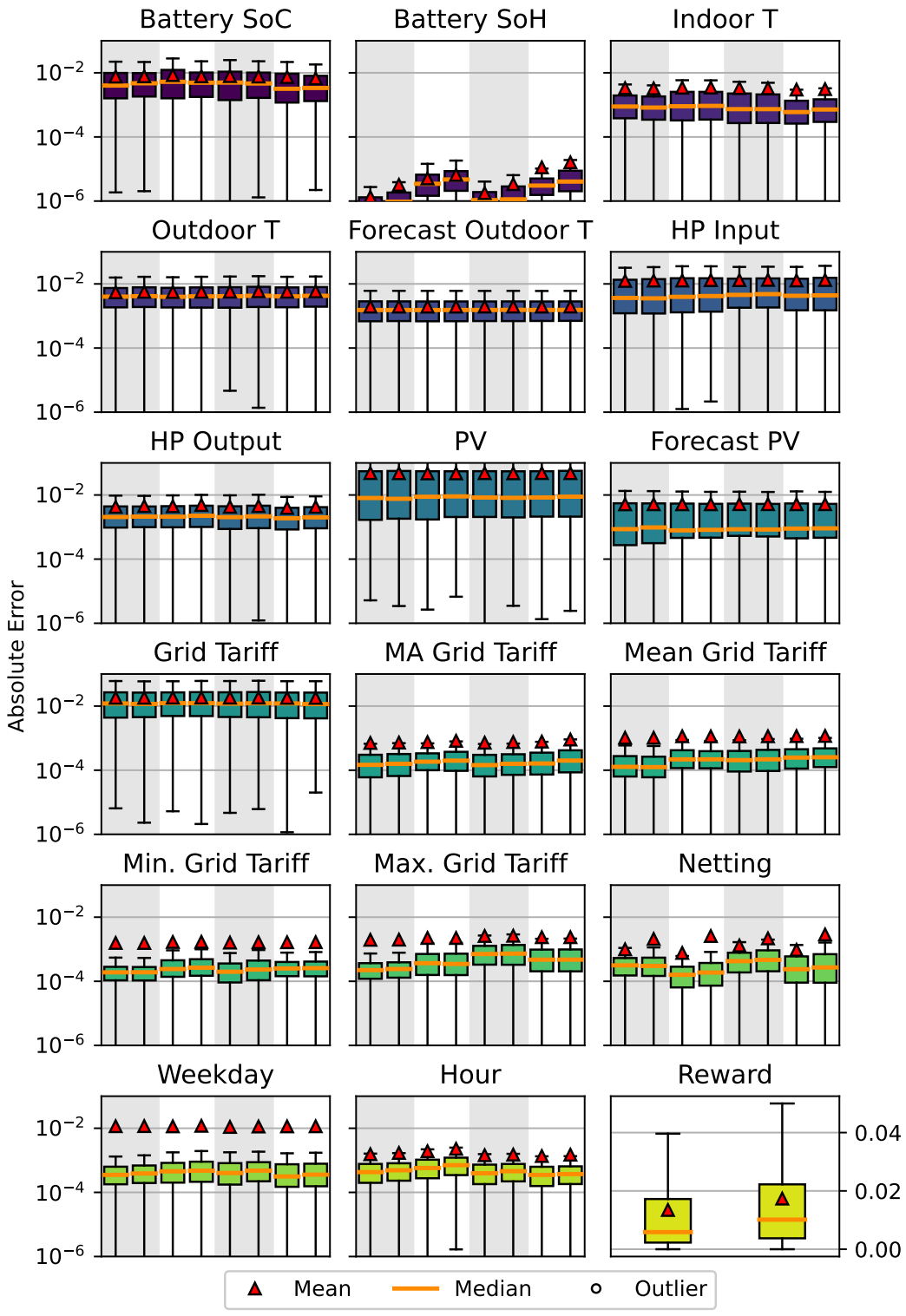

FIGURE F.2: Feature-wise absolute error for MDQL. The absolute error on the validation environment is plotted for each ensemble head separately. In addition, we have plotted each ensemble head twice, first dynamics head 1 twice, then head 2, 3 and 4 (the distinction between each head is also highlighted by the white/gray background). For each ensemble head, we plot the absolute error on traces generated by the Q-ensemble at the left, and traces generated by random actions at the right. Note: outliers have been omitted in order to ensure readability, and there is a different y-axis for the reward-error.