university of groningen

faculty of science and engineering

Renewable Energy Valley Management using DreamerV3: a Multi Agent Implementation

Matthias Drijfhout

university of groningen

faculty of science and engineering

University of Groningen

Renewable Energy Valley Management through DreamerV3: a Multi-Agent Implementation

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
Prof. dr. D. Grossi (Artificial Intelligence, University of Groningen)
and
Robin Hermes (Repowered, Groningen)

Matthias Drijfhout (s3786196)

August 29, 2025

Contents

Acknowledgments

I am grateful to Davide Grossi, my first supervisor and professor at the Faculty of Science and Engineering, University of Groningen, for his guidance, thoughtful feedback, and steady support throughout this learning process. I also wish to thank Robin Hermes, tech lead at Repowered, for the freedom to explore my own ideas and make independent implementation choices, as well as for his insightful guidance in the energy sector. My collaboration with Lucas Velvis on MARLOES was both instructive and efficient, and his constructive peer reviews in code and writing have been greatly appreciated. Finally, I am thankful to Repowered for granting access to their proprietary software SIMON and for the opportunity to contribute to the REFORMERS project, which allowed me to connect academic research with a current and relevant practical application.

Abstract

This thesis investigates the application of multi-agent model-based reinforcement learning (MBRL) to the energy distribution problem (EDP). Building on DreamerV3, a state-of-the-art single-agent MBRL algorithm, the study develops MADreamer, an extension to multi-agent settings. The algorithm is evaluated in MARLOES, a simulation environment representing Renewable Energy Valleys (REVs). We present MARLOES as a flexible environment with adaptable objective functions and a playground to test RL algorithms. MARLOES models multiple assets, such as solar, wind, and storage, that must coordinate to balance demand and supply locally. The research highlights the theoretical advantages of MBRL and empirically exposes its limitations in an EDP modelled in MARLOES. Performance is compared to a heuristic baseline, PrioFlow, which optimizes self-sufficiency. The results reveal the weaknesses of MBRL and display the issues by visualizing the intended actions. This research discusses the causes of model bias, a common phenomenon in MBRL, and underlines the challenges and weaknesses of applying a model-based method to a multi-agent environment.

1 Introduction

The need for sustainable energy sources becomes more urgent with the consistent increase in demand for electricity [1, 2], and with it comes the need for flexible energy availability throughout the day[3]. Not only are fossil fuel sources being depleted, but electricity from these sources also emits large amounts of greenhouse gases (GHG), such as CO_2, that contribute to climate change. Goals set by the European Climate Law (ECL) [4] to reduce GHG emissions to by 55% in 2030 compared to 1990, are unlikely to be met. The most recent estimations show that the GHG emissions of 2023 (2.9GtCO_2[5]) are still well above the target, which stresses the need to abolish energy generation through fossil fuels. To this end, the European Commission set up the RepowerEU plan [6, 7] that should diversify energy supplies and accelerate the clean energy transition. An alternative to the fossil fuels are renewable energy sources (RES) that emit significantly less CO_2 [8, 9]. Sources like solar, wind, hydro, and biomass offer environmental benefits, and are renewable which ensures a more future-proof energy supply. However, energy distribution poses a challenge in transitioning to these sustainable forms of energy, as their production profiles are less predictable than fossil fuels. Solar and wind heavily depend on weather conditions. Because these sources are bound by their characteristic profiles, efficient Energy Storage Systems (ESS) are essential. ESS such as batteries and electrolysers enable the storage of excess energy during times of abundant production, making it possible to meet demand when primary sources fall short [10].

Energy Hubs (EHs) have emerged as a potential solution to the energy distribution problem (EDP) [11, 12]. An EH is an integrated system that manages multiple energy carriers—production, consumption, and storage—in a geographically confined area. EHs can incorporate various renewable sources and storage systems and serve as a testing ground for innovations and smart grid technologies that enhance energy efficiency and reliability [13, 14]. By managing energy carriers locally, an EH can reduce strain on the main grid caused by the sudden influx of RES.

The REFORMERS project led by New Energy Coalition (NEC) [15, 16] attempts to set up such an EH which they call a Renewable Energy Valley (REV). The project aims to create a CO_2 neutral and 100% self-sufficient REV integrating multiple sustainable energy carriers further contributing to the same goal as RepowerEU [7]. Such a decentralized utilization of locally produced RES is generally regarded as more efficient and future-proof than a centralized approach for energy distribution [17, 18, 19]. The experiences gained from the blueprint Flagship Valley, located in Alkmaar, will be used to further roll-out more REV in other countries in Europe, reducing fossil fuel dependencies GHG emissions in another step to reach the 2030 target.

An essential component of an REV is the implementation of an advanced control system that strives to achieve predetermined objectives. With the possibility of different goals or contracts in mind, it could greatly complicate the rules of control. To solve a complex environment with unpredictable nature, such as a REV, Reinforcement Learning (RL) [20] has shown promising results [11].

1.1 Related Work

Model-based RL (MBRL) has emerged as a subset of RL algorithms that offers advantages over model-free methods, particularly considering sample efficiency and explainability [20, 21, 22]. Unlike model-free RL, which learns a policy directly from experiences, MBRL leverages a learned or known model of the environment. This allows sample efficient planning, efficient exploration, and the ability to reason about the consequences of control strategies in the real world [23]. Algorithms like PILCO [22], MAMBA [24] MARIE [25], MBVD [26] tested on Atari environments [27], all rely on encoders that transform the observations into capturable environment dynamics. PILCO uses

Gaussian Processes (GP) to create a probabilistic world model, MBVD the Variatonal Auto-Encoder (VAE) with a Recurrent Neural Network (RNN), a dyna-style architecture [28], whereas MAMBA and MARIE operate on the popular DreamerV2 model [29], and Transformer world models, respectively. All models introduce additional samples to learn from, reducing gradient variance and increasing learning efficiency. However, with the additional samples comes a risk of introducing bias; the classic bias-variance trade-off [30, 11, 31]. If the samples generated from model transitions do not accurately represent the real world, learning from these samples negatively impacts learning.
A key advantage of MBRL is the ability to learn from relatively small amounts of data [32, 22, 25]. By learning a dynamics model, the agent can simulate future trajectories and use the simulated experiences to improve its policy [21, 24, 26]. Moreover, the learned dynamics model offers interpretability and insights into the system's behaviour [21] and can even be used for tasks beyond control such as forecasting or anomaly detection. Once the dynamics are learned, they can be used efficiently in other REVs through transfer learning [33, 34].
However, the accuracy of the learned model is crucial for the performance of the agent. An inaccurate model will result in poor decision making and suboptimal results [22, 24]. Capturing the dynamics of the environment can be challenging in high-dimensional or stochastic environments [23, 32, 35]. Furthermore, MBRL algorithms can be computationally expensive, as they often involve planning and optimization over a long horizon [35].
MARL has been applied several times in an attempt to solve the EDP in environments similar to the REV [11, 36, 12]. [37] simulates a power system to test a simple Q-learning algorithm in MAS and shows that MARL can improve system stability, whereas [38] investigates battery performance in a simulated smart grid [17]. [35] compares the performance of model predictive control (MPC) and MBRL and combines them to face the challenges in both approaches for a building energy management system (BEMS). [39] uses networks to assist Monte Carlo Tree Search (MCTS) [20] in gathering and updating the environmental dynamics model for a microgrid that includes Heating Ventilation Air Conditioning (HVAC), PV, and ES systems. An MBRL algorithm can provide a robust and adaptive scheduling system dealing with the uncertainty and randomness of RES [39]. [40] propose a control system to enhance flexibility and reduce the total cost for an environment where the multi-energy hub is formed as multi-agent cooperative control. Extending energy management with energy markets further explores the complexity of the real-world REV [41].

1.2 Contributions

Considering the EDP and the aforementioned RL algorithms we aim in this paper to develop a MBRL capable of managing the energy flows in a closed system such as the REV. The objective guided by the goal of Reformers, is tailored to improve self-sufficiency and emit as little GHG emissions as possible. The research question is formulated as follows: ”*Is a multi-agent MBRL algorithm capable of managing energy flows in a REV, to minimize CO_2 emissions?*”
In order to answer this research question, this work introduces MARLOES, a high-resolution simulation environment that captures the dynamics of an energy system with a diverse set of renewable energy sources. The framework allows experimenting with both single- and multi-agent reinforcement learning algorithms to solve the EDP, and optimize steerable assets on different objective functions. It enables experimentation and evaluation of advanced control strategies for an environment with the dynamics of a REV, including custom limits and constraints. Additionally, we introduce a heuristic baseline distributing energy flows based on a simple set of priorities, that serves as an evaluation for new algorithms. With an increasing need to solve the EDP with RES [7], we propose an extension of the state-of-the-art MBRL algorithm, DreamerV3 [29], in the energy system from MARLOES

which we call MADreamer (Multi-Agent Dreamer). MADreamer is a novel approach to solve the EDP through MBRL. Furthermore, DreamerV3 has shown success in single-agent environments but is yet to be applied to multi-agent setting, where complexity rises and dynamics are less predictable due to other agents' actions, and the issue of non-stationarity is introduced. The MADreamer implementation assesses the capabilities of a single-agent MBRL algorithm applied in a multi-agent environment.
The design and implementation of MARLOES were done in collaboration with Lucas Velvis, whereas the other contributions are the work of the author.

1.3 Thesis Outline

This thesis stated its goals in previous section and follows up with the introduction of the MARLOES environment for RL algorithms. Section 2 provides the mathematical framework for the formalization as a Multi-agent markov decision process (MMDP), more details of the different assets and introduces the suggested MBRL MADreamer, with implementation details about its inspiration DreamerV3 [42] as introduction. Then the experimental details and configurations are provided in Section 3, introducing the main experiment comparing MADreamer with a heuristic baseline, and an additional experiment to investigate arising instabilities. Section 4 present and interpret the findings of each experiment, considering the variance between algorithms and configurations, and the emergent behaviour of the assets. This is further discussed in Section 5, that reflects on challenges in MBRL and concludes with the implications of this thesis and outlines future research directions for MARLOES and MBRL in an EDP.

2 Methods

2.1 MARLOES - a RL environment

For this project we introduce a framework around the proprietary SIMON package provided by Repowered [43], that allows the implementation of different RL algorithms. This environment is called MARLOES (Multi-Agent Reinforcement Learning for Optimized Energy Solutions), which encompasses the algorithm and facilitates communication actions and information between SIMON and the algorithm. A schematic diagram is given in Figure 1.

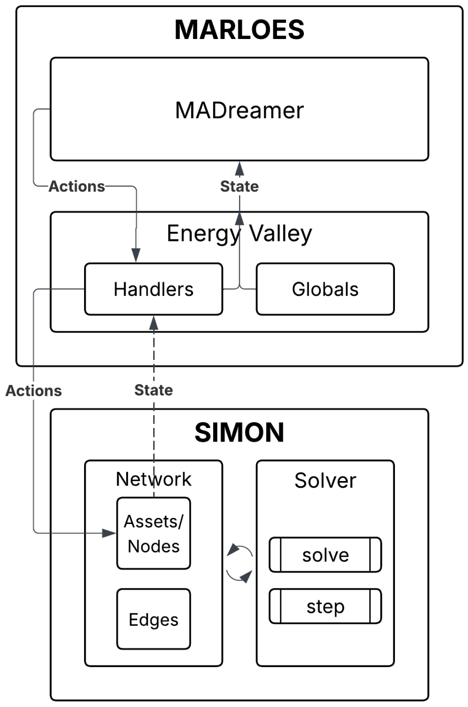

Figure 1: Diagram of the MARLOES Environment and the interaction with Repowered package SIMON.

2.1.1 SIMON

In this section, we shortly describe the role of the package 'SIMON' provided by Repowered [43], without providing the inner workings of the simulations. The package defines energy system components, such as solar panels or batteries as *Assets*. Adding assets to a model-simulation also requires the specification of the 'targets'; the assets to which energy flows can be directed. E.g. solar panels are solely supply assets, and will likely target a battery asset, a demand asset and the grid, whereas a demand asset has no targets as they are pure consumers. The network can be visualized as a directed graph $G = (V, E)$, with nodes (V) and edges (E). Figure 2 shows an example configuration with a random selection of components.
Each asset has their own state (s_t^i) with asset-specific and time-dependent information (Section A). These states are used by the model to calculate the next states. Lastly, actions can be given in the form of setpoints. The asset will try to reach that setpoint in production, whether this is positive

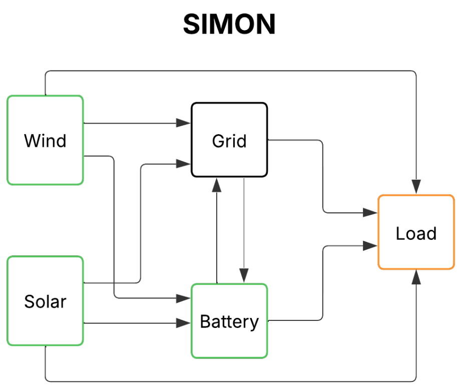

Figure 2: Directed graph of an example network in SIMON, where each edge has a priority $prio_x$.

(producing) or negative (consuming). The targets in these simulations are used to set up a structure and enforce constraints, but in reality the energy flows would not directly move from a solar asset to a Demand asset, but would result in a cumulative balance. Flow diagrams (Section B) are used to validate the behaviour of the assets in the model.
The SIMON package contains solver that distributes energy flows based on priorities, $prio_x$ with $x \in \mathbb{R}$. This Priorities Solver (PS) solves the distribution problem based priorities, ordering the edges based on the priorities and energy volumes. All available power is distributed among the assets with the highest priorities first, and the grid compensates any shortage or surplus. MARLOES assigns a fixed priority value to each possible edge, which can be found in Section C. We use this solver with a small adaptation for ESS to validate the MARLOES environment and set a heuristic baseline (Section 2.2.1).

2.1.2 Environment Formalization

We can formulate the EDP as a Markov Decision Process (MDP) because every next state (S') is only dependent on the current state S and the action $\mathcal{A}$. Since the control system will have access to information about the full state space, the problem is fully observable and given that the reward is shared amongst each agent, we can extend the MDP, with multiple agents to formalize the Energy Valley as a Multi-agent MDP (MMDP). An MMDP is a powerful formalization to theoretically derive an optimal policy for joint actions and dealing with uncertainty [44].

Definition 1. *A MMDP is a tuple: $\langle \mathcal{S}, \mathcal{N}, \mathcal{A}, \mathcal{T}, R, \gamma \rangle$ where $\mathcal{S}$ and $\mathcal{N}$ are finite sets of states (S) and agents respectively, $\mathcal{A} : a_1 \times a_2 \times \cdots \times a_N$ is a set of joint actions where a_i is the action space for agent $i \in \mathcal{N}$ and $\mathcal{T} : S \times \mathcal{A} \times S' \to [0,1]$ is a transition function, which gives the probability of transitioning to state $S' \in \mathcal{S}$ given that the system is in state S and joint action a^* is taken. $\mathcal{R} : S \to R$ is a real-valued reward function for reaching state S, and $\gamma \in [0,1]$ is the discount factor controlling the importance of future rewards [45].*

The objective in an MMDP is to find an optimal policy $\pi^*(a|S)$ that maximizes the expected cumulative reward, given in Equation 1.

$$V^{\pi}(S) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, a_t^*) \mid S_0 = S, \pi\right] \tag{1}$$

where $V^{\pi}(s)$ is the value function under policy π, with $S, S_t, S_0 \in \mathcal{S}$ and a_t^* is a joint action at timestep t.

2.1.3 MARLOES as MMDP

Below we show how MARLOES fits into the MMDP framework, by presenting the definition of each element of Definition 1.

Agents ($\mathcal{N}$) The set of assets (nodes) in SIMON directly map to the set of agents in MARLOES. For all assets i in SIMON with $i \in \mathcal{N}$ we introduce a wrapper agent, which we call a handler in MARLOES. The set of agents $\mathcal{N}$ consists of all asset handlers that also require an action a_i, therefore, the grid and the load assets are not included in this set.

State Space ($\mathcal{S}$) MARLOES introduces an asset handler that allows flexible handling of the existing state, modifying, or adding relevant information, and is given in the handler state s_i^m for $i \in \mathcal{N}$ from MARLOES. The source of information per handler is shown in Table 1 and their definition can be found in Section A. The state space (Equation 2) consists of the set of states (s_i^m) of all handlers ($i \in \mathcal{N}$) in the energy system and any additional external features or global information $\mathcal{C}$. The additional information in this paper was limited to the netto forecasted power for the next time step, and the temporal time information of the state, encoded using a trigonometric transformation (Section D) to maintain the cyclical nature of time-based features. This encoding ensures values within [-1,1] intervals making them suitable for neural network models and allows the model to learn patterns that vary cyclically over time which are likely occurrences in energy profiles such as solar and demand. More potentially relevant information regarding energy prices or weather forecasts can be added to globals in future iterations of MARLOES, in order to diversify objective functions.
The final state space is shown in Equation 2:

$$\mathcal{S} = \{s_i^m \mid i \in \mathcal{N}\} \cup \mathcal{C} \tag{2}$$

where each agent $i \in \mathcal{N}$ has a local state s_i^m with relevant state information. The full system state is defined as the set of all agent states together with the global information $\mathcal{C}$.

	SIMON			**MARLOES**	
Handler	`power` (kW)	`available_power` (kW)	`state_of_charge` (%)	`forecast` (kW)	`nomination` (kWh)
Wind	✓	✓		✓	✓
Solar	✓	✓		✓	✓
Battery	✓		✓		
Electrolyser	✓		✓		
Demand	✓			✓	✓
Grid	✓				

Table 1: Handler types and the origin of their state elements.

Action space The action space ($\mathcal{A}$) defines the control inputs to adjust the energy production or consumption of each asset:

- $a_i \in [-1, 1]$ for assets that can both produce and consume energy.
- $a_i \in [0, 1]$ for assets restricted to production.

Here $a_i = -1$ indicates an asset consuming at its full potential power, constrained by operational bound maximum power input. On the other hand $a_i = 1$ indicates an asset producing at its full potential power constrained by operational bound maximum power output. Any values in between represents a scaled adjustment of the potential power. No discretization is applied to the action space to avoid artifacts and maintain high-precision control over the assets.

Transition Dynamics ($\mathcal{T}$) Constraints and dynamics such as start-up time, ramp-up- and ramp-down limits, are modelled by SIMON (Section 2.1.1) [43] which we treat as a stationary Markov kernel. The environment moves from one state to one of the next state possible states (S') through probability $P(S' \mid S, a^*)$ only dependent on current state (S) and joint actions (a^*). The theoretical dynamics are handled by the simulator and therefore not mentioned here, but we can use this Markov property to handle the environment as an MMDP.

Reward (R) The MARLOES platform supports modular reward configurations to accommodate different optimization objectives. At each time step t, the total reward R_t is defined as the weighted sum of a configurable subset of individual reward components:

$$R_t = \sum_{k \in \mathcal{R}} w_k \cdot r_t^k \tag{3}$$

Where:

- $\mathcal{R}$: The set of selected reward components, which consists of the CO_2 reward (Section E.1) and an additional battery incentive $i_t^{\mathcal{B}}$ (Section E.2) as given in Equation 15
- w_k: A scalar weight controlling the influence of reward component $k \in \mathcal{R}$.
- r_t^k: The value of reward component k at time step t.

This general reward formulation supports both single- and multi-objective optimization by adjusting the set $\mathcal{R}$ and the associated weights w_i. Details on the calculations of each reward component r_t^k are provided in Section F.

Discount factor γ This is a scalar factor in [0,1] responsible for the handling of future rewards, which use case is described in Equation 9. The value is of this factor is determined via hyperparameter search in Section 3.

2.1.4 Asset handler details

MARLOES contains handler's that instantiate their own asset in SIMON. The handlers are responsible for providing the asset with the correct action in the form of a setpoint and to provide them with the necessary data. They share functionality to keep track of time, but differ in default parameters, action spaces and degradation models. Below is a general description of the different asset handler's in MARLOES. All production and consumption profiles, with their corresponding forecasts, were provided by Repowered. For a more detailed description of each handler, see Section A.

Solar/Wind handler The handler's for these typical production assets are very similar. They provide their asset with the production forecast and actual profile for a fixed horizon, and send the required setpoints. The asset in SIMON will produce based on the setpoint provided. The only difference between solar and wind handlers is the production profile they provide. The solar profile is based on its orientation (default is East-West), while the wind profile is an onshore profile of a wind turbine. Both datasets come from a location close to Alkmaar.

Battery/Electrolyser handler These handlers provide setpoints to the flexible storage units in SIMON. Both the battery and electrolyser handler instantiate a battery asset in SIMON, but differ in the default parameters and degradation function. Since the electrolyser stores the energy in the form of hydrogen (H_2) we apply a conversion factor of 33 kWh per kg H_2 to the capacity of the battery asset [46]. The differences model the respective characteristics of a battery and electrolyser (e.g. the battery response time is instant, while the electrolyser requires some warm-up time).

Demand handler The demand handler models a consumption asset that is not flexible, and therefore does not require setpoints. The handler is only responsible for providing the load profile and forecast for a fixed horizon. By default, a demand profile of a dairy farm is used in the simulations.

2.1.5 Data manipulation

We introduce some data manipulation functions to create an environment with uncertainty and imperfections, which approximates real world imperfections and improve robustness in the learning process of algorithms. The forecasts and real energy production or consumption profiles can be manipulated with the following functions:

1. Drop-out: Randomly selected timesteps are set to zero mimicking communication failures, or outages. Both single timesteps, and longer periods can be dropped, simulating more substantial errors.

2. Noise injection: simulates forecasts- or measurement errors with added gaussian noise, ensuring variability.

More details are provided in Section G.

2.1.6 Validation

To validate whether the framework interacts correctly with SIMON, we analyze the behaviour of PrioFlow (Section 2.2.1) and a strategy called SimpleSetpoint. This strategy produces random setpoints for all steerable assets. We visualize the energy distribution in a flow diagram. These diagrams translate the setpoints into proportional energy flows between different handlers, allowing us to verify that PrioFlow and SimpleSetpoint interact with SIMON's assets. The diagrams with their interpretations are presented in Section B.

2.2 Algorithm

2.2.1 PrioFlow

The existing PS based on energy distribution rules and priorities required a small adaptation for ESS. The ESS in SIMON can only act on provided setpoints, consequently, without a method to determine these setpoints the battery cannot be included in the PS. PrioFlow provides a heuristic to determine and provide the ESS with meaningful setpoints. The addition of a simple rule to establish the required setpoints (Section C.1), based on the forecasts of all assets, ensures charging and discharging at moments that would improve self-sufficiency. Therefore, with perfect forecasts, this algorithm will achieve perfect distribution of energy but introduces a vulnerability to unreliable forecasts or unforeseen changes in production or consumption. Optimizing for self-sufficiency is already in line with the objective of REFORMERS, which makes it suitable for this research. The algorithm is not built to learn other objective functions, but the strategy can be evaluated with other objective functions. A depiction of the energy flows can be found in Section C.2.

2.2.2 DreamerV3

DreamerV3 [42] is a state-of-the-art single agent Model-Based RL algorithm that achieved impressive performances in Atari environments [27], and as the most notable achievement was able to find diamonds in Minecraft. The algorithm is sample efficient, shows strong performance in domains with discrete- and continuous action spaces, and has been shown to be scalable [42]. The algorithm consists of three neural modules; the world model (WM) parameterized by ϕ, the actor parameterized by θ, and the critic parameterized by ψ, where the goal is to learn an accurate world model, to predict actual transitions and their rewards in a latent state. The WM is trained on real-world trajectories, whereas the actor and critic (AC) are trained solely on trajectories sampled from the WM. This makes the algorithm very sample efficient as real samples are more computationally and time expensive than trajectories imagined through the latent dynamics model in the WM.

WM learning: The world model is implemented as a Recurrent State-Space Model (RSSM), which learns latent representations $\mathbf{s}_t = (h_t, z_t)$ from sensory inputs x_t using variational autoencoding. It captures the environment's dynamics and predicts the transition through the latent state (z_t), rewards ($\hat{r}_t$) and reconstructed observations ($\hat{x}_t$) through different modules presented in Equation 4

$$\begin{aligned}
\text{Sequence Network:}\quad & h_t \sim f_\phi(h_t \mid z_{t-1}, x_{t-1}, a_{t-1}) \\
\text{Encoder:}\quad & z_t \sim q_\phi(z_t \mid h_t, x_t) \\
\text{Dynamics Predictor:}\quad & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\
\text{Reward Predictor:}\quad & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\
\text{Decoder:}\quad & \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, \hat{z}_t)
\end{aligned} \tag{4}$$

The loss function for the WM is presented in Equation 5[29] as composition of three loss functions, prediction loss ($\mathcal{L}_{pred}(\phi)$), representation loss ($\mathcal{L}_{rep}(\phi)$) and dynamics loss ($\mathcal{L}_{dyn}(\phi)$), minimized during training period T.

$$\mathcal{L}(\phi) \doteq \mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T}\left(\beta_{\text{pred}}\mathcal{L}_{\text{pred}}(\phi) + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn}}(\phi) + \beta_{\text{rep}}\mathcal{L}_{\text{rep}}(\phi)\right)\right]. \tag{5}$$

with weights $\beta_{pred} = 1$, $\beta_{dyn} = 1$ and $\beta_{rep} = 0.1$. The prediction loss trains the Decoder heads to reconstruct the input x_t and the Reward predictor head to reconstruct reward r_t.

$$\mathcal{L}_{\text{pred}}(\phi) \doteq -\ln p_\phi(x_t \mid z_t, h_t) \;-\; \ln p_\phi(r_t \mid z_t, h_t) \tag{6}$$

The dynamics loss trains the Dynamics predictor to align the prior ($p_\phi(z_t|h_t)$) with the encoder's posterior ($q_\phi(z_t \mid h_t, x_t)$) by minimizing KL-divergence [47] while clipping below b_{free}, disabling these loss functions whenever they are minimized to a meaningful extend.

$$\mathcal{L}_{\text{dyn}}(\phi) \doteq \max\Big[b_{free},\ \text{KL}\big(\text{sg}\cdot\big[q_\phi(z_t \mid h_t, x_t)\big] \,\big\|\, p_\phi(z_t \mid h_t)\big)\Big] \tag{7}$$

The representation loss is identical to the dynamics loss, but different in the location of the stop-gradient ($\text{sg}\cdot[]$) operator, which decouples the gradients from the loss function to prevent unwanted updates based on the wrong gradients. The dynamics loss is responsible for the updates to Dynamics predictor, while the representation loss updates the Encoder to achieve better encoding from the observation to the latent state.

$$\mathcal{L}_{\text{rep}}(\phi) \doteq \max\Big[b_{free},\ \text{KL}\big(q_\phi(z_t \mid h_t, x_t) \,\big\|\, \text{sg}\cdot\big[p_\phi(z_t \mid h_t)\big]\big)\Big] \tag{8}$$

Actor learning: The actor network $\pi_\theta(a_t \mid s_t)$ is trained to maximize the entropy-regularized λ-returns (Equation 9 [20]) by minimizing the surrogate loss function $\mathcal{L}(\theta)$ in Equation 10.

$$R_t^\lambda = r_t + \gamma c_t\big[(1-\lambda)\, v_\psi(s_{t+1}) + \lambda R_{t+1}^\lambda\big], \quad R_T^\lambda = v_\psi(s_T), \tag{9}$$

The λ-returns function provides a tradeoff and scaling between the immediate return and the temporal difference targets. This method expresses the return at timestep t as the weighted sum of the estimates and future returns controlled by $\lambda \in [0,1]$. This provides stable and sample-efficient updates, while retaining assignment of future rewards to actions taken at timestep t.

$$\mathcal{L}(\theta) \;=\; -\sum_{t=1}^{T} \frac{\big(R_t^\lambda - v_\psi(s_t)\big)}{\max(1,\, S)} \log \pi_\theta(a_t \mid s_t) \;+\; \eta\, \mathcal{H}\big[\pi_\theta(a_t \mid s_t)\big] \tag{10}$$

where

- $v_\psi(s_t)$: the critic's value estimate,
- S is the return-normalization range (the 5th–95th percentile span of R^λ),
- and η is a decaying entropy weight for the Shannon entropy (Equation 11), punishing deterministic actions.

Normalization is done over returns instead of advantages to prevent the exploration incentive from vanishing due to sparse rewards. By dividing returns by their standard deviation (S) we maintain stable gradient magnitudes, preserving a fixed entropy bonus weight relative to the return scale.

$$\mathcal{H}\big[\pi_\theta(a_t \mid s_t)\big] \;\approx\; -\pi_\theta(a_t \mid s_t)\log\big(\pi_\theta(a_t \mid s_t)\big). \tag{11}$$

Critic learning: The critic network $v_\psi(s_t)$ is trained to approximate the distribution of λ-returns by minimizing the negative log-likelihood over imagined trajectories:

$$\mathcal{L}(\psi) \;\doteq\; -\sum_{t=1}^{T} \ln p_\psi\big(R_t^\lambda \mid s_t\big), \tag{12}$$

where $p_\psi(R \mid s)$ is a categorical distribution over exponentially spaced bins, and R_t^λ is defined as above. This objective encourages accurate multistep return predictions under the current policy. Using imagined trajectories the critic learns to integrate multistep λ-return targets, capturing long-range dependencies in rewards, especially helpful in learning the battery strategy - storing energy for future use. Parameterizing $p_\psi(R \mid s)$ as a categorical decouples gradient magnitudes from raw return scales, solving vanishing and exploding gradients. This method also handles big differences in return distributions and results in more stable value learning.

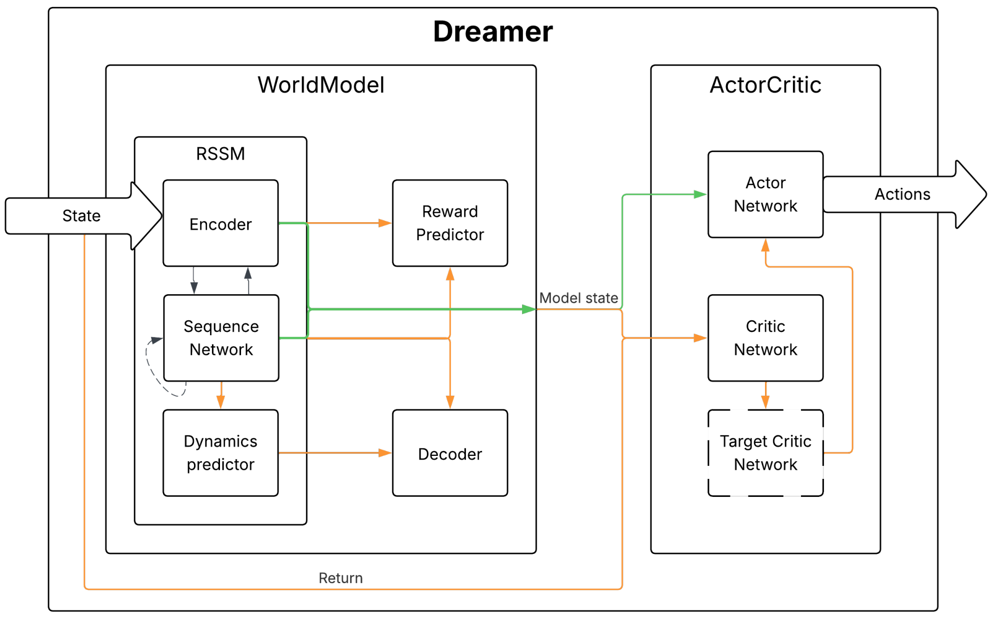

Figure 3: The Dreamer architecture for in training (orange + green) and evaluation (green). The world model produces the model state from the encoder and the sequence network, which is used to generate actions.

2.2.3 Multi-Agent Dreamer

We introduce an adaptation of a DreamerV3 [42]: MADreamer, a Model-Based RL algorithm that utilizes the Dreamer architecture in a MAS. Figure 3 shows a diagram of the modular components and their connections.

Paradigm In this project, we adopt a Centralized Training, Centralized Execution (CTCE) paradigm, where a central controller makes decisions for all agents based on full system observability and a shared reward function. This paradigm is well suited for fully cooperative (MARL) settings [48][49], where optimizing global objectives (such as self-sufficiency or grid balancing) is preferred over agent-specific optimization. SIMON provides a complete view of the energy system which naturally fits the CTCE paradigm. Furthermore, the cooperative nature leads to shared objectives that can be coordinated and planned in actions by a central controller. This paradigm allows for simplifications on the reward and observation level. The reward is shared and only needs to be calculated once, whereas the observation is fully observable, resulting in a simple concatenation of all agent states. A key challenge in CTCE is the exponential growth in complexity as the number of agents increases [49]. The centralized controller must process an increasingly large state space, making training computationally expensive and potentially intractable for large-scale systems. We investigate the scalability of the algorithm in Section 3.2.

Implementation details For the optimization step, we used the `AdamW` optimizer [50] that decouples the weight decay from the estimates and applies the decay step after the gradient-based update, whereas original `Adam` optimizer includes the decay step in the gradient-update. In a CTCE setting where gradients from all agents' actions flow into shared layers, `AdamW`'s decoupled weight decay applies a uniform shrinkage step instead of possible uneven regularization across different agent heads.

As a result, update gradients per agent remain properly scaled without possible interference from other agent's variations.
The Shannon entropy in a MAS sums the entropy over the joint actions (a^*) of each agent as in Equation 13.

$$\mathcal{H}\big[\pi_\theta(a_t \mid s_t)\big] = -\sum_{a^*} \pi_\theta(a^* \mid s_t) \log\big(\pi_\theta(a^* \mid s_t)\big) \tag{13}$$

Model architecture The network consists of two main components: a World Model and an Actor–Critic. The World Model uses a simple MLP encoder to map observations x_t into a latent representation z_t, maintains a hidden state h_t via a GRU-based RSSM, drawing from a stochastic latent space ($\hat{z}_t$), and decodes back to observations via an MLP decoder. A separate MLP reward predictor maps (h_t, z_t) to $\hat{r}_t$. The world model is the backbone of the algorithm, since the actor learns based on samples generated from this model. Table 2 shows the submodules in the world model and their roles. The temporal memory of the sequence model is important for the energy systems as storage systems achieve delayed rewards. The reward predictor is unbounded to allow an expressive reward predictor, and allows to model the actual reward from the environment directly. This does require some clamping in the learning modules in the World Model to avoid exploding gradients. Clamping the update gradients was done at (-5,5) for each module. The modules were kept relatively small, with limited number of layers to reduce the risk of model bias.

Submodule	**Purpose**	**Details**
Encoder	Encode (x_t, h_t) into latent z_t (posterior)	1 layer MLP (64) mapping to stochastic z_t of size 64 with two heads (μ and `logvar`)
Sequence Model	Temporal memory	GRU (64), with 0.01 dropout
Dynamics predictor	predicts prior $p(z_{t+1} \mid h_{t+1})$	2 heads from the Sequence model (μ and `logvar`)
Decoder	Reconstruct observation $\hat{x}_{t+1}$ from z_{t+1}	2 layer MLP (64)
Reward predictor	Predict r_t from (h_t, z_t)	1 layerMLP (64) predicting single unbounded output

Table 2: World Model architecture and training components. The size for each layer is given in brackets.

The Actor–Critic takes the concatenated (h_t, z_t) model state as input. The Actor of the original MADreamer is a three-layer MLP producing a Gaussian policy, whereas the MADreamer with separate heads, MADreamer (SH), shares one initial layer, and splits to a separate network for each agent with 2 MLP producing one action. The Critic, remains the same for both implementations for actors, and is a two-layer MLP outputting logits over $n = 50$ value-bins, from which a scalar value $v(s_t)$ is computed as a two-hot encoded expectation [42].

Rollout for WM-training The WM is trained on a batch of sampled from real states (s_t, a_t, s_{t+1}). For each sequence, it infers the posterior latent z_t via the encoder on (s_t, h_t), predicts the prior $\hat{z}_{t+1}$

and next hidden h_{t+1} via the GRU and latent heads, encodes the true next state to z_{t+1}, and collects prior and posterior sequences. These are stacked to compute the prediction- (Equation 6), dynamics- (Equation 7), and representation loss (Equation 8) that update the WM.

Imagination for AC-training The AC is trained solely on samples from the WM, starting of by sampling a batch of starting points, using the actor to sample actions for each handler and generate transitions. At each $t \in T$ steps, it concatenates $\{h_t, z_t\}$ to form the model state s_t, samples $a_t \sim \pi_\theta(a_t \mid s_t)$, uses the sequence network to predict next states, $\mathrm{RSSM}(h_t, z_t, a_t) \rightarrow (h_{t+1}, \hat{z}_{t+1})$, and predicts the reward ($\hat{r}_{t+1}$) using the reward predictor. The resulting imagined trajectories $\{z_t, a_t, \hat{r}, \hat{z}_{t+1}\}$ are used exclusively to train the Actor–Critic, decoupling policy learning from the real environment.

2.2.4 Dealing with instability

During early experiments, we observed instability in the Dreamer implementation for the environment. Even though the loss converged consistently over runs, the behaviour and performance was sub-optimal indicating convergence to local optimum without the ability to escape this plateau. Several strategies were added in an attempt to reduce this instability issue.

Before training, the replay buffer was filled with a full day ($24 \cdot 60 = 1440$) of experiences sampling random actions. The multitude of experiences is critical to ensure the controller has access to a diverse set of states, actions and rewards. The broad set of experiences prevents overfitting to specific actions and encourages exploration in the early stages of training. Additionally, the entropy coefficient η was initialized to 1, matching the scale of our normalized rewards to encourage early exploration. A slow linear decay schedule with decay factor $d_\eta = 0.999$ was applied to reduce exploration over time [51, 52].

Empirical evaluation showed persistent premature convergence to mirrored behaviour, where the battery copied the solar handler setpoint schedule and vice versa. To reduce parameter interference, we introduce separate actor heads with only one initial shared layer. With this change, the difference in tasks for different assets should be more distinguishable by the central controller[53]. Furthermore, a small dropout layer ($p = 0.1$) was added between the fully connected layers and the actor heads to further combat overfitting and remove meaningless weights [54].

Lastly, gradient normalization [55] was used to improve update stability. Gradient-norm clipping was applied with a threshold $c_{threshold} = 10$ to avoid exploding gradients, but allow large updates to get out local optima.

2.2.5 Notation summary

The important variables and notations discussed in this section are presented in Table 3, with their name and description.

Variable	**Name**	**Description**
$\mathcal{S}$	State space	Finite set of all possible environment states.
$\mathcal{N}$	Agents	Finite set of agents in the system.
$\mathcal{A}$	Joint action space	Cartesian product of action spaces $A_1 \times \cdots \times A_N$.
$\mathcal{T}$	Transition dynamics	$P(S' \mid S, a)$: prob. of next state S'.
$\mathcal{R}$	Rewards	Finite set of sub-rewards.
A_i	Agent i's action space	Actions available to agent i.
$R(S)$	Reward function	Scalar reward for reaching state S.
γ	Discount factor	Future-reward weighting, $\gamma \in [0,1]$.
$\pi(a \mid S)$	Policy	Probability of choosing action a in state S.
$V^{\pi}(S)$	Value function	Expected return under π from s.
C	Globals	Additional global/contextual features.
s_i^m	Handler state	MARLOES handler state for asset i
s_i^s	Asset state	SIMON's internal state for asset i.
R_t	Reward	Total reward at timestep t.
w_k	Reward weight	Scalar weight for reward component k.
r_t^k	Component reward	Reward of component k at timestep t.
x_t	Observation	Sensory input at time t (Dreamer).
h_t	Hidden state	RSSM's recurrent hidden state at t.
z_t	Latent state	Stochastic latent embedding at t.
$\hat{z}_{t+1}$	Predicted latent	Prior latent predicted by the world model.
$\hat{r}_t$	Predicted reward	World model's reward prediction at t.
$\hat{x}_t$	Reconstruction	Decoder's reconstructed observation at t.

Table 3: Key variables in MARLOES and MADreamer.

3 Experimental Setup

Dreamer-general		**WorldModel**		**ActorCritic**	
Param	**Value**	**Param**	**Value**	**Param**	**Value**
`horizon`	8	α_{WM}	$1 \cdot 10^{-4}$	α_{A}	$1 \cdot 10^{-4}$
`batch_size`	64	ω_{WM}	$1 \cdot 10^{-3}$	α_{C}	$5 \cdot 10^{-4}$
γ	0.999	b_{free}	1	ω_{A}	$1 \cdot 10^{-3}$
λ	0.99	—	—	ω_{C}	$1 \cdot 10^{-3}$
η	1	—	—	—	—

Table 4: Selected hyperparameter values for MADreamer, with formalized notation for learning rates α and weight decay coefficients ω.

Preliminaries The hyperparameters (Table 4) of MADreamer were selected by a grid search, covering 1% of the space defined in Table 5. Across all experiments, we employed a fixed replay buffer capacity of $1.0 \cdot 10^5$ transitions which is sufficient to store the entire trajectory of 30.000 training steps and to retain temporally distant experiences that are still valuable due to the cyclical nature of energy profiles. This mitigates overfitting to a temporal or seasonal pattern. All experiments described below were run on Intel® Core™ i7-9750H×12 processor of a Dell Inc. XPS 15 7590 with operating system Ubuntu 24.04.1 LTS, using Python `3.11.3`. The code for the experiments, and the implementation of MADreamer can be found at `https://github.com/repowerednl/marloes`.

Algorithmic performance was evaluated using two key metrics: the cumulative episodic return and total CO_2 emissions (Equation 14). This performance is compared against the heuristic baseline PrioFlow (Section 2.2.1).

$$\mathrm{CO_2} = p^{\text{grid}} \times \mathrm{GWP}_{\text{grid}} \tag{14}$$

The $\mathrm{GWP}_{\text{grid}}$ is calculated in Section E.1. The final reward (Equation 15) used in the experiments consists of a combination of the CO_2 reward and the battery incentive ($i_t^{\mathcal{B}}$) as described in Section E.2. Since the CO_2 reward is the main objective, we multiply it by two, to prevent the incentive from becoming too important and interfering with the actual objective.

$$R_t = 2 \cdot r_t^{CO_2} + i_t^{\mathcal{B}} \tag{15}$$

The experiments are summarized and presented in graphs using the mean (μ) and 95% confidence interval over $N = 10$ runs calculated as given in Equation 16, with σ as the standard deviation.

$$\mathrm{CI} = 1.96 \cdot \frac{\sigma}{\sqrt{N}} \tag{16}$$

Furthermore, to provide insight into the stability of the experiments, we include the variance (σ^2) into the analysis. This captures the spread of the data, but is sensitive to outliers as the deviations are squared. For an additional method, we present the interquartile range (IQR) as in Equation 17 to show the variance without extreme outliers, and describes the spread of the middle 50% of the data.

$$\mathrm{IQR}(X) = Q_3(X) \; - \; Q_1(X) \tag{17}$$

where:

- $Q_3(x)$: The value below which 75% of the data lies.
- $Q_1(x)$: The value below which 25% of the data lies.

Lastly, we use the coefficient of variance (CV), the ratio of the standard deviation (σ) to the mean (μ) as in Equation 18, which allows to compare variability across different configurations as they have different scales and magnitudes.

$$\mathrm{CV} = \frac{\sigma}{\mu} \tag{18}$$

Dreamer-general	
`horizon`	$\{8, 16\}$
`batch_size`	$\{32, 64\}$
γ	$\{0.995, 0.997, 0.999\}$
λ	$\{0.98, 0.99\}$
η	$\{0.9, 1\}$
WorldModel	
α_{WM}	$\{1\cdot10^{-4}, 2.5\cdot10^{-4}, 5\cdot10^{-4}, 1\cdot10^{-3}, 5\cdot10^{-3}\}$
ω_{WM}	$\{0.01, 0.001, 0.005\}$
b_{free}	$\{0.5, 1.0\}$
ActorCritic	
α_{A}	$\{1\cdot10^{-4}, 2.5\cdot10^{-4}, 5\cdot10^{-4}, 1\cdot10^{-3}, 2.5\cdot10^{-3}\}$
α_{C}	$\{1\cdot10^{-4}, 2.5\cdot10^{-4}, 5\cdot10^{-4}, 1\cdot10^{-3}, 2.5\cdot10^{-3}\}$
ω_{A}	$\{0.01, 0.001, 0.005\}$
ω_{C}	$\{0.01, 0.001, 0.005\}$

Table 5: Grid-search parameter sets with formalized learning rate α and weight decay ω.

3.1 Multi Agent

Behaviour In the multi-agent experiments, all solar and battery handlers are trainable. The controller must learn to coordinate charging and discharging: storing surplus solar output in the battery during peak production and discharging when demand exceeds generation. We will visualize timeseries of setpoint trajectories to verify that the learned policies capture this intuitive cycle. All multi-agent experiments use the configurations listed in Table 6.

Performance We compare MADreamer with and without separate heads, with the PrioFlow baseline using the mean and standard deviation of cumulative CO_2 emissions, and the demand satisfaction ratio (DSR) (Equation 19) over multiple seeded runs. The DSR is measured as the percentage over the full period T, during which the demand, is met at timestep t by the supply and no power compensation from the grid (p_t^{grid}) is necessary. Any charging from ESS is incorporated into the demand. The DSR indirectly measures the dependency on the grid.

$$\mathrm{DSR} = \frac{1}{T}\sum_{t=1}^{T} D\big(p_t^{\mathrm{grid}}\big), \quad D\big(p_t^{\mathrm{grid}}\big) = \begin{cases} 1, & p_t^{\mathrm{grid}} \le 0, \\ 0, & p_t^{\mathrm{grid}} > 0. \end{cases} \tag{19}$$

3.2 Scalability

To evaluate how the number of controllable assets affects learning efficiency, we conduct two experiments for the same MADreamer configuration but in a 3-handler setup and a 6-handler setup (Table 6). For both cases, we record the wall-clock time for the full training protocol, and compare the loss convergence of both the WM and the AC, using the area under the learning curve (AULC) Equation 20.

$$\mathrm{AULC} = \sum_{i=1}^{T-1} \frac{\mathcal{L}_i + \mathcal{L}_{i+1}}{2} \cdot (t_{i+1} - t_i) \tag{20}$$

where

- t_i is the training step at time i
- T: the total number of training steps
- $\mathcal{L}_i$ is the loss value at t_i

3-Handler		
Solar	`max_power`	orientation
	3000	EW
Demand	profile	`scale`
	Farm	15
Battery	`capacity`	`max_power`
	2000	1000

6-Handler		
Solar	`max_power`	orientation
	3000	EW
	2000	EW
Demand	profile	`scale`
	Farm	15
	Farm	10
Battery	`capacity`	`max_power`
	2000	1000
	1500	800

9-Handler		
Solar	`max_power`	orientation
	3000	EW
	2000	EW
	1000	S
Demand	profile	`scale`
	Farm	15
	Farm	10
	Farm	12
Battery	`capacity`	`max_power`
	2000	1000
	1500	800
	3000	100

Table 6: Asset configurations for 3-, 6-, and 9-handler experiments.

3.3 Single Agent

The credit assignment problem [56] arises when an agent is incapable of deriving the result of its actions. Whenever the agent receives a reward it must be able to assign it to one of his actions which becomes increasingly more difficult when multiple agents are acting in the same environment, influencing the same shared reward. To isolate each asset's learning dynamics, we trained two "single-agent" handlers independently without impactful interference from another handler.
The first investigates the learning pattern and behaviour of a energy production asset; a Solar handler, whereas the second evaluates the behaviour of an ESS in this environment; a Battery handler. The handlers were trained separately for 10 seeded runs, optimizing their respective setpoint policy via the same loss and reward structure used in the multi-agent experiments. Each steerable asset receives their action in the form of setpoints, which represents intended production or consumption (in kW). We visualize the trajectory of the policy through plotting the mean setpoints during the training phase. During evaluation we test the learned policy under unseen demand and production data, also analysing the behaviour through setpoints, which provides insights into the intended behaviour of the assets.

Robustness In an attempt to bridge the gap between perfect simulations and reality noise functions (Section 2.1.5) were added to MARLOES, that allow data manipulation and test the robustness of the learning process of algorithms. Deploying different levels of noise to the data tests how well learning occurs in a noisy environment, and if it can sustain uncertainty. An additional experiment to test this for MADreamer was moved to the Section I, as the stability issues affects the significance and hinders drawing conclusions based on the results.

4 Results

4.1 Multi-Agent

Actor implementations Figure 4 shows the average cumulative reward and CO_2-equivalent grid production for two actor implementations in MADreamer: the standard variant and the separate-heads variant (SH). In terms of mean performance, the standard actor achieves $(-7.34 \pm 1.00) \times 10^4$ reward and $(4.70 \pm 0.80) \times 10^8$ gCO_2, while the separate-heads actor reaches $(-8.62 \pm 1.87) \times 10^4$ reward and $(5.52 \pm 1.26) \times 10^6$ gCO_2. Thus, MADreamer outperforms the SH variant by 17.5 % on both reward and emissions. However, a Wilcoxon signed-rank test, a non-parametric test, suitable for data with high variance between groups, showed no significant difference ($W = 10.0$, $p = 0.084$) between MADreamer (Median=4.86×10^8) and MADreamer SH (Median= 5.73×10^8). This architectural change also comes at a computational cost: training time rises from 765 ± 6 s to 861 ± 24 s, an increase of 12.6 %.

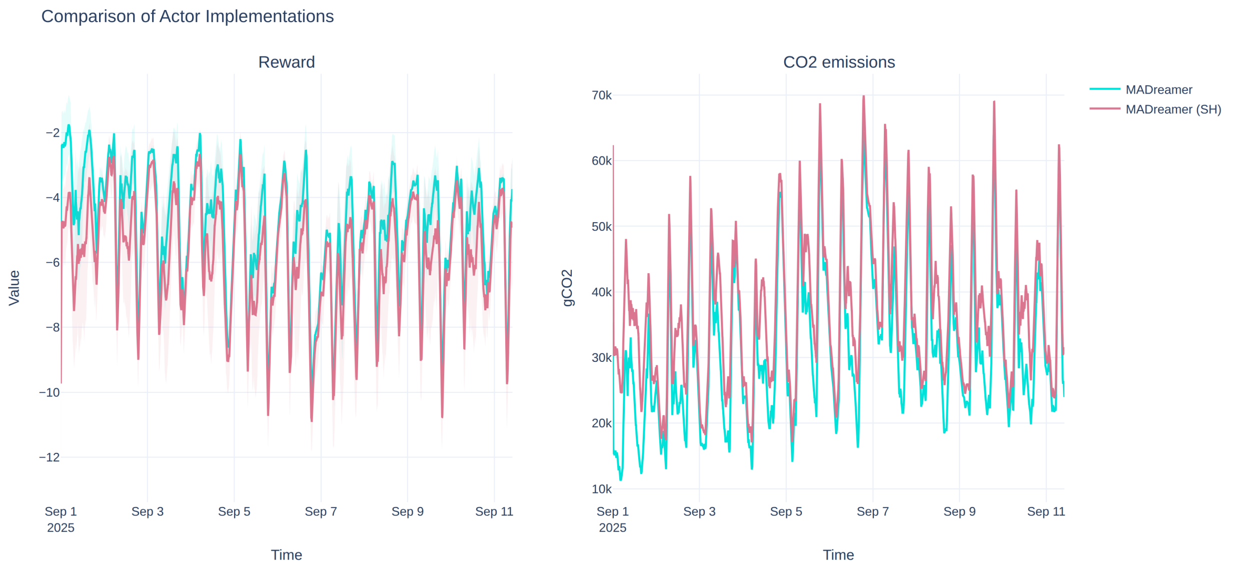

Figure 4: Average cumulative Reward and Grid Production for shared-head versus separate-heads actor implementations in MADreamer.

Configuration	**Reward** $(\times 10^4)$	**CO_2-emissions** $(\times 10^8$ **kW**$)$	**Training Time** (s)
MADreamer	-7.34 ± 1.00	4.70 ± 0.80	765 ± 6
MADreamer (SH)	-8.62 ± 1.87	5.52 ± 1.26	861 ± 24

Table 7: Mean $(\mu \pm \sigma)$ for cumulative reward, grid production and training time of the two actor variants.

Behaviour As established in Section 2.2.4 the algorithm is unstable but incidentally achieves decent performance which behaviour is presented in Figure 5. Over three days, the agent charges the battery almost exclusively during periods of high solar irradiance, producing an inverse-solar profile in its charge set-points. However, instead of tracking short-term fluctuations in demand, the battery discharges at a consistent rate, leaving residual deficits that the grid must cover, which in turn decreases the DSR metric. Nonetheless, this run achieves a DSR of 44.59 % and incurs total CO_2 emissions of 2.21×10^8 gCO_2, coming close to the PrioFlow baseline (1.89×10^8 gCO_2).

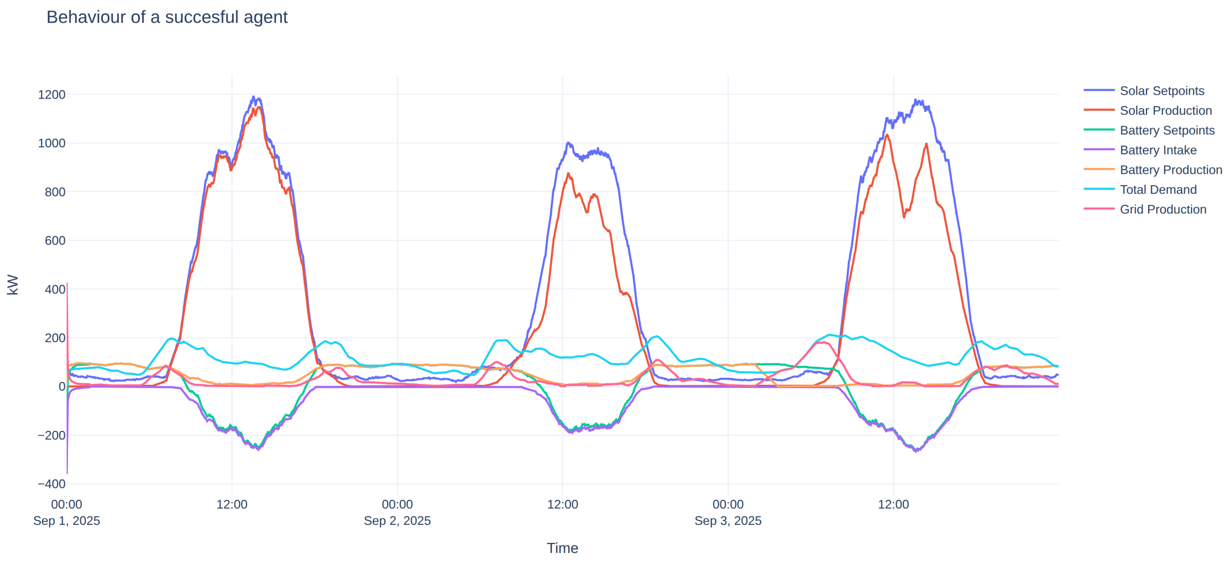

Figure 5: Behaviour visualization of a single successful run during policy evaluation over three days.

Performance Performance traces Figure 6 show that both MADreamer variants exhibit a roughly constant rate of reward loss and emissions increase whereas PrioFlow displays large spikes in reward and emissions, but otherwise maintains a shallower reward-loss slope and lower momentary emissions. This is a result of forecast errors in the data, which greatly affect the PrioFlow performance due to the adaptation for ESS Section C.1. This adaptation is a simple heuristic based on the available forecasts, where errors in the forecast cause overshoots in battery charging or discharging. Over the full evaluation window, the cumulative performance in Figure 7 confirm that PrioFlow achieves the lowest total CO_2 emissions (1.89×10^8 gCO_2), despite its dependency and vulnerability to forecast errors. A non-parametric Wilcoxon signed-rank test showed a significant difference ($W = 0.0$, $p = 0.002$) between performance regarding CO_2-emissions from PrioFlow (Median=1.89×10^8) and MADreamer (Median=4.86×10^8).

Configuration	**DSR** ($\mu \pm \sigma$, %)	**Variance**	**CV**	**IQR**
PrioFlow	**55.65** $\pm$ 0.00	0.00	0.00	0.00
MADreamer (3)	14.05 $\pm$ 9.96	99.17	0.71	17.06
MADreamer SH (3)	19.04 $\pm$ 11.26	126.72	0.59	15.47
MADreamer (6)	26.89 $\pm$ 8.15	**66.46**	**0.30**	14.05
MADreamer (9)	21.15 $\pm$ 11.00	121.05	0.52	**12.09**
Successful Run	44.59	–	–	–

Table 8: Statistics for Demand Satisfaction Ratio (DSR) across configurations (mean $\pm$ std).

Configuration	CO_2 Emissions ($\mu \pm \sigma$, $\times 10^8$)	Variance ($\times 10^{15}$)	CV	IQR ($\times 10^8$)
PrioFlow	**1.89** $\pm$ 0.00	0.00	0.00	0.00
MADreamer (3)	4.70 $\pm$ 0.80	**6.36**	**0.17**	1.33
MADreamer SH (3)	5.52 $\pm$ 1.26	15.9	0.23	2.15
MADreamer (6)	6.23 $\pm$ 1.04	10.9	**0.17**	**1.30**
MADreamer (9)	10.2 $\pm$ 2.32	53.8	0.23	2.78
Successful Run	2.21	–	–	–

Table 9: Statistics for total CO_2 emissions across configurations (mean $\pm$ std).

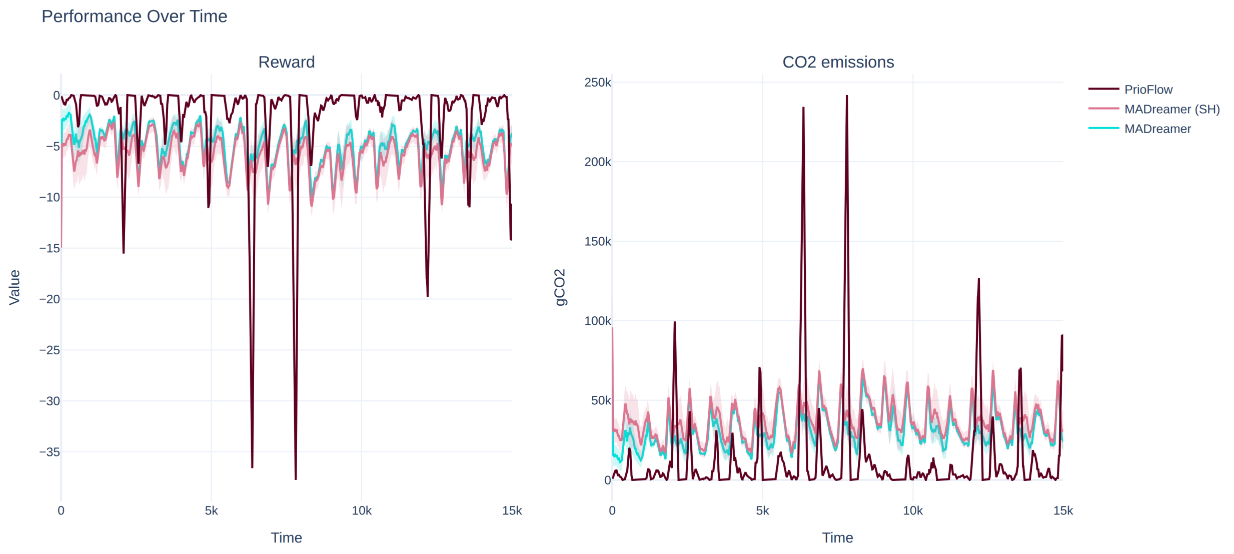

Figure 6: Performance of both MADreamer implementations compared to the PrioFlow baseline.

Instability and variance The instability of MADreamer becomes apparent when examining the variance-based metrics reported. The variance in DSR (Table 8) appears to be lower in configurations with more handlers, with the lowest variance, considering to σ^2 and CV, in a 6-handler setup. The DSR (μ) is also higher for this setup, which indicates a more stable learning process for that specific configuration under the selected hyperparameters. Since the DSR for a 9-handler setup is lower, and with a higher CV, it could also indicate a sensitivity to changes in the environment regarding the number of agents, or the ratio supply to load. However, a lower IQR for MADreamer (9) does indicate that the data might be skewed by some outliers.
Looking at the CO_2-emissions in Table 9 we do see comparable variances between MADreamer (3) and MADreamer (6), but a large increase in variance, also without extreme outliers (IQR), for MADreamer (9). Note that magnitude of the variance is 10^7 times higher than CO_2 emissions, indicating extreme differences in policies between the runs.

4.2 Scalability

Training time Table 10 reports the mean (μ) and standard deviation (σ) of key performance metrics for MADreamer with 2 trainable agents (3-handler) and 4 trainable agents (6-handler). Moving from

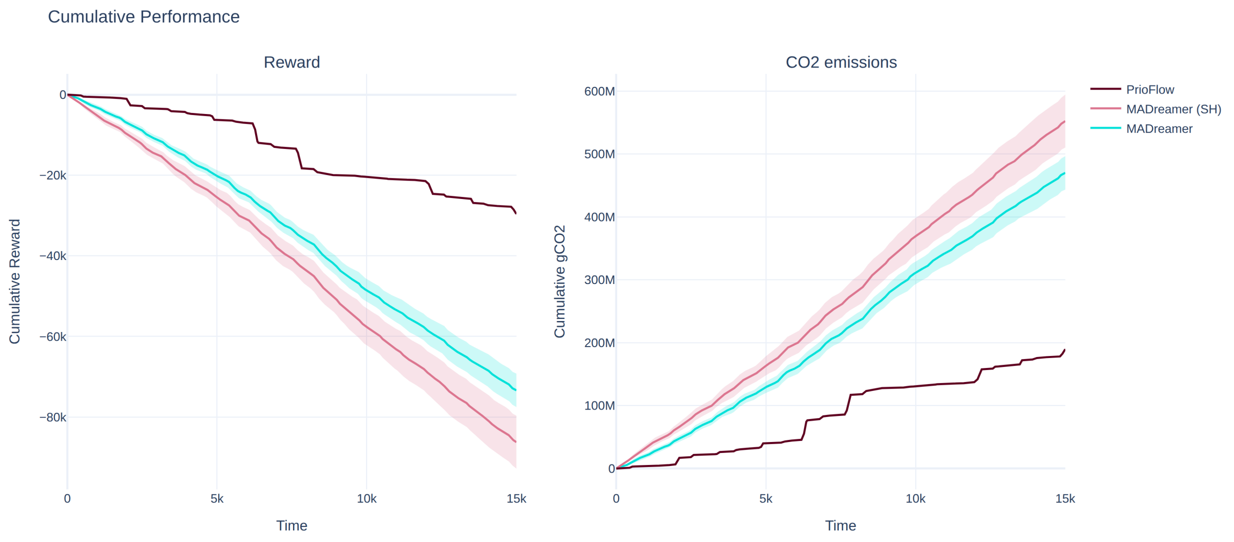

Figure 7: Cumulative performance of both MADreamer implementations compared to the PrioFlow baseline.

3 to 6 handlers increases the average training time from 765 ± 6 s to 917 ± 34 s, an increase of 19.9%.

Configuration	**$\sum$Reward** ($\times 10^4$)	**$\sum$CO$_2$-emissions** ($\times 10^8$)	**Training Time** (s)
MADreamer (3)	-7.34 ± 1.24	4.70 ± 0.80	765 ± 6
MADreamer (6)	-5.84 ± 0.98	6.23 ± 1.04	917 ± 34
MADreamer (9)	-6.44 ± 1.55	10.2 ± 2.32	1107 ± 134

Table 10: Comparison of MADreamer configurations with 3, 6, and 9 handlers (mean ± std).

AULC The difference in losses are quantified in Table 11 with the AULC method described in Equation 20. The table shows the expected lower loss values for the setup with 3 handlers and the losses increasing for more complex environments. The prediction loss (Equation 6), which represents the reconstruction (decoding) of the predicted latent state to real observations and the reward prediction of that latent state, is decreasing in a more complex environment. This may reflect bias in the world model, where the model exploits shortcuts that become more apparent with the addition of more handlers, with the same datasets. The shape of each individual loss progression and further evaluation is presented in Section H.

Configuration	**Actor Loss**	**Critic Loss**	**Dynamics Loss**	**Prediction Loss**	**Representation Loss**
MADreamer (3)	**-6.51**	**7.55**	**8.77**	5.74	**4.36**
MADreamer (6)	-10.13	7.59	11.60	4.84	4.75
MADreamer (9)	-16.53	7.64	14.08	**4.30**	6.23

Table 11: Area under the learning curve (AULC, $\times 10^4$) for each handler configuration across losses.

4.3 Single-Agent

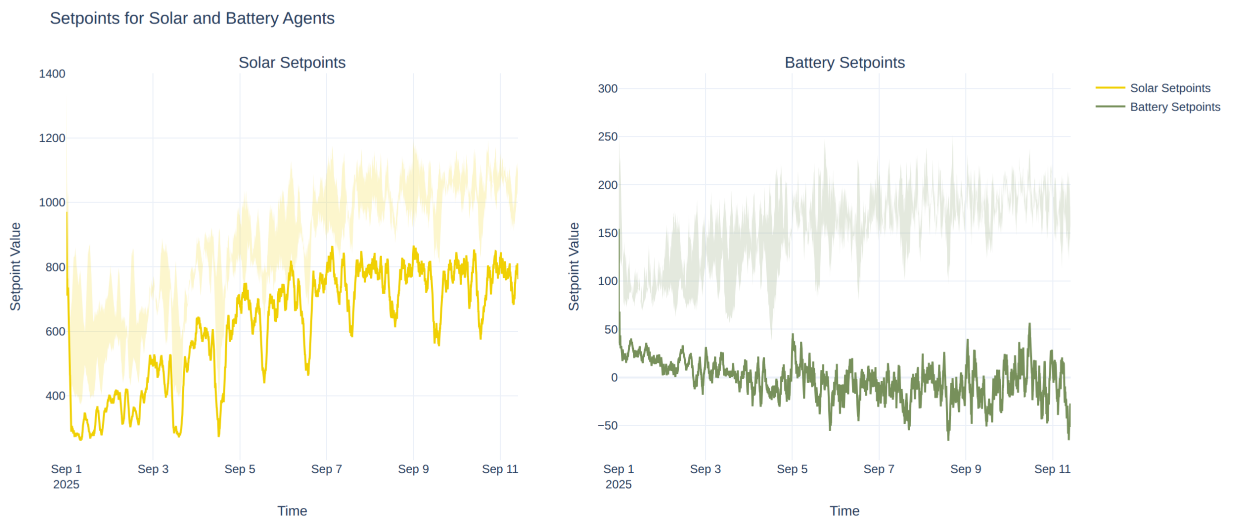

Figure 8: The mean of the setpoints during training over 10 runs for training only Solar or Battery.

Solar Handler During training (Figure 8, left), the Solar handler's mean setpoints gradually increase reflecting a correct adaptation and progressive learning towards the expected behaviour. However, the mean is well below the confidence interval (CI) band, implying that the behaviour is not learned in every run and incidental runs choose to curtail solar even though it will receive large penalties when demand is not met.

The learned policy exhibits a clear pattern during evaluation (Figure 9, left); high setpoints during the day and lower setpoints at night. The CI shows rather consistent behaviour, capturing the underlying solar production profile.

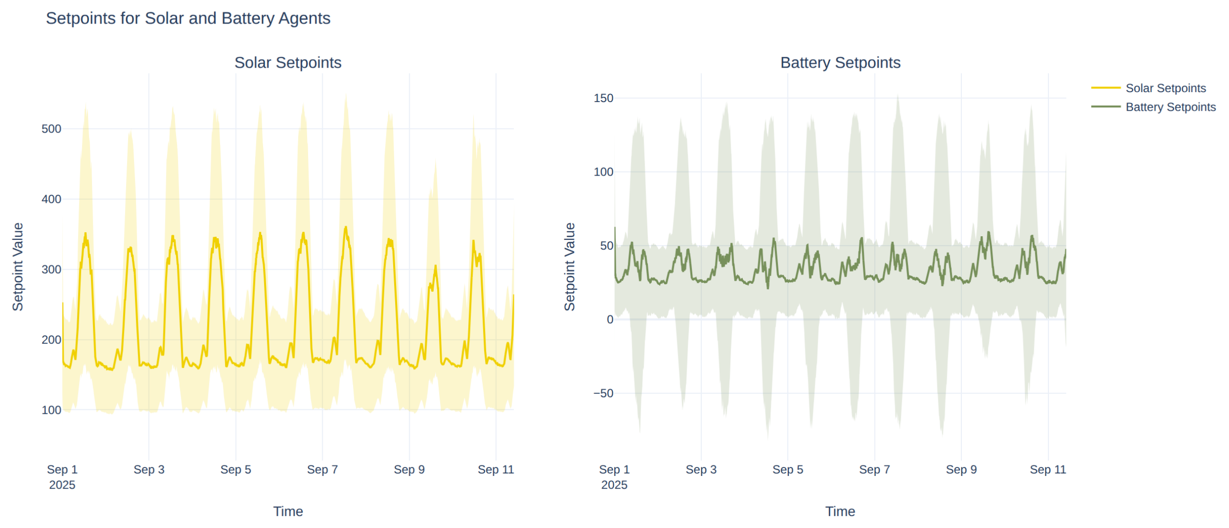

Figure 9: The mean of the setpoints for evaluation over 10 runs for training Solar and Battery in separately.

Battery Handler During the training phase (Figure 8, right), the mean Battery handler's setpoints oscillate around zero, while the CI band is much higher. This suggests that individual runs adopt varying strategies, with the most common strategy to mirror the Solar handler's setpoints, while other strategies adopt the opposite strategy; charging during solar production, sending negative setpoints reducing the mean towards zero.

Under evaluation (Figure 9, right), a wide variety of policies is employed, shown by the large CI ranging from -70kW to 150kW. Such a broad interval demonstrates a persistent instability in the battery handler's policy. The mean settles above zero indicating a slight charging bias, which is likely due to mirrored behaviour from the Solar handler's setpoints.

Table 12 shows an especially high variance for the experiment training the solar handler. This is expected as the effect of the solar handler on CO_2 emissions is very clear during the time where solar production is available. A battery is merely able to store sustainable energy, because storing energy from the grid does not result in lower CO_2 emissions, even more since the efficiency is not 100%. Therefore, in the experiment where only the battery is trained, and the solar produces fully, the effect the battery can have is limited.

Configuration	**CO_2 Emissions** ($\mu \pm \sigma$, $\times 10^8$)	**Variance** ($\times 10^{15}$)	**CV**	**IQR** ($\times 10^8$)
Just Solar	4.52 ± 1.20	14.44	0.27	2.51
Just Battery	2.73 ± 0.51	2.63	0.19	0.65

Table 12: Statistics for total CO_2 emissions of single-asset experiments (mean $\pm$ std).

5 Discussion

The MBRL algorithm DreamerV3 has shown good results in complex environments, where the world model captured the environment dynamics with success [42]. In this work we investigated the performance of that algorithm for a central controller, steering multiple agents in an EDP, solely learning from imagined trajectories. Several challenges that limit the stability and performance of our multi-agent Dreamer implementation were identified regarding the algorithm and the environment.

Credit assignment problem A critical challenge in MARL is that of credit assignment [57, 58]. RL algorithms generally learn from scalar returns that is produced through a complex neural network and multiple different decisions. The question arises whether the model understands what actions are responsible or have an effect on that scalar return [57]. In cooperative multi-agent systems, the credit assignment problem is compounded because a single return is shared across agents, each with their own, possibly asynchronous, actions influencing the return value [58]. This mismatch between the global feedback and 'separate' actions inflates the variance of gradient estimates and reducing convergence speed, or even misdirects policy optimization. The credit assignment problem appears to be an issue in this algorithm since part of the optimized policies show reverse roles for solar and battery handlers. However, the stability issue remains in the single agent experiments (Section 4.3) that greatly reduces complexity and isolates the effect of the actions of a single agent on the scalar. Therefore, even though it might play a role in instabilities presented in this paper we can conclude that the credit assignment problem is not the main concern in MADreamer and look for other causes.

5.1 Bias-variance trade-off

Since the majority of MADreamer training runs result in a sub-optimal policy for both single- and multi-agent experiments, there seems to be an imbalance regarding the bias-variance trade-off [30]. Too much bias is a well-known phenomenon in MBRL [59], which can be introduced through multiple processes within the MBRL framework described below.

Compounding error In MBRL, even small one-step prediction errors can escalate when the rollout procedure samples transition from the model for multiple steps, which is known as compounding error [60, 61]. The environment amplifies this error in such a small resolution environment, where a long rollout is required to receive valuable feedback on the actions taken at time t. In the context of an REV achieving self-sufficiency, the battery would charge during the day but only discharge at night, leaving hours between the the action and the eventual return. In MARLOES, due to the minute-resolution, the time until a reward is long into the future. Due to the minimal effect in short-term return, the model would ideally learn from longer rollouts, but as a result the compounding error would be ever more prevalent [59, 60]. Good rollout planning while mitigating the compounding error requires precise balancing and may be an extremely difficult task in such a high resolution environment as MARLOES.

Model exploitation Another pitfall related to the intricacies of training an effective world model is model exploitation [62]. Whenever the actor learns planning in an imperfect model it may discover 'short-cuts' that look rewarding in imagination, either short-term or without sufficient exploration to find more rewarding actions. This pattern appears in the majority of multi-agent experiment runs, where the battery handler charges too much and receives a penalty for grid-usage, and therefore opts to discharge the battery instead. Such behaviour suggests that the planner does not recognize the long-term reward of using the battery to prevent grid usage at a later time. This sub-optimal strategy can be seen as the exploitation of imperfections in the world model.

Synthetic data bias In this paper, we exclusively train the actor and the critic on imagined rollouts in the world model. From this design choice a possible synthetic data bias may arise, where the optimization gradients inherit a bias proportional to the rollout length [59]. To reduce this bias, the horizon was shortened from the original 16 [42] to 8. In the final experiments, the remaining bias still overpowered the ability to accurately update the world model. This is in line with the findings from Lucas Velvis, who looked at a hybrid approach. His results show a decrease in performance with a greater dependency on the world model [63].

Primacy bias Another source of bias is the same element of MBRL that can make it very succesful; sampling [64]. MBRL is sample efficient when the world model allows the generation of meaningful synthetic samples [59, 21, 62]. The world model, however, is trained on real samples and can be biased to the initial training samples that steer the world model into the wrong direction [64, 65]. MARLOES has a high resolution environment, where the replay buffer is filled with samples from random actions to ensure many different samples, but when the model is optimized on initial bad samples, it is pushed towards early misconceptions, resulting in a faulty world model. Figure 13 indicates a much faster convergence of the world model compared to the actor-critic model (Figure 14, showing a vulnerability in early sampling and indicating primacy bias.

5.2 Limitations and threats to validity

Before drawing conclusions this section will provide some reflection on algorithmic and environmental choices, that could have had an effect on the results presented in this paper. It is important to note, that these are the result of simplifications and abstractions to fit into the requirements of this project.

Algorithmic validity Correctly setting up an environment together with an algorithm comes with several challenges. The performance of MADreamer in MARLOES could have been hindered by several factors regarding the algorithm.
Each model with configurations listed in Table 10 was trained on the same set of hyperparameters. However, since training a world model has shown to be a challenging task, each configurations would perform better with their own set of optimized hyperparameters. Additionally, the hyperparameter search done in Section 3 is not enough to state that the optimal parameters were found. Even though the options for each parameter were kept to a minimum, the search space remained too large for a thorough search. This leaves the potential for more optimal hyperparameters, as RL algorithms are highly dependent on tuned parameters [66, 67].
Furthermore, the model is trained for 30k steps (minutes), roughly 21 days, which does not capture seasonal changes in the profiles. Since seasonality is especially prevalent when modeling RES, and the goal of Reformers is to be self-sustainable over the course of a full year, the number of training steps is insufficient to learn those patterns.

Environment validity The environment was set up with certain simplifications as described in **??** which would have increased the gap between simulation and reality [68, 11], and therefore requires caution when drawing conclusions about real-world applications based on the findings presented in this paper. The noise functions as described in Section 2.1.5 are based on a simple Gaussian distribution and, together with the dropout functions, applied randomly with a certain probability. No studies were conducted to base the noise or dropouts on the occurrences of outtages, or noisy measurements in real-world applications, as it was beyond the scope of this project.

Reproducibility MARLOES is built on SIMON (Section 2.1.1), the proprietary software provided by Repowered. Without access to SIMON, the experiments presented in this paper cannot be reproduced, even though the code and the experiments are open source and can be found at `https://github.com/repowerednl/marloes`.

5.3 Implications

The EDP is a research field that lends itself well to a reinforcement learning environment. The real-world application becomes rather complex considering the number of stakeholders involved. In this project we constructed an objective function based on CO_2 emissions, but the financial aspect could also be formulated as an objective, after which a multi-objective function is required to balance the interests of different stakeholders. This research provides a flexible environment and showcases common pitfalls of MBRL. This section presents insights into possible sources for bias and other model vulnerabilities. These issues must be resolved in MARLOES before transfer to an application in a real REV can be made.

5.4 Future work

This paper is an introduction of the MARLOES environment, and a presentation of an initial MBRL algorithm, to showcase the workings and possibilities in MARLOES. For this specific high-resolution environment, policy-based MARL might be more appropriate due to the aforementioned vulnerabilities that come with longer rollouts. A more sophisticated approach would come in the form of a hybrid algorithm, utilizing the strengths of both policy-based and model-based algorithms to arrive at a more robust algorithm. MARLOES, together with the baseline PrioFlow (Section 2.2.1) remains a solid playground to test the feasibility of such algorithms to solve the EDP. MARLOES is also equipped with a wind- and electrolyser handler, which are not used for this research, but introduce unique data profiles, and different dynamics to further investigate the road to a self-sustaining REV.

MADreamer improvements An improvement for future iterations of MADreamer, or other model-based algorithms is policy optimization as presented by [62], in the form of Model-Based Policy Optimization (MBPO). This algorithm reduces the bias that arises from long horizons, by limiting the imagined rollouts to short segments branched from real-data and maintaining a measure of divergence between the world and the true model [62], used to adjust the rollout horizon. The key to replicate a complex environment into a meaningful world model is to ground the optimization in true environment transitions [42, 21]. It improves overall stability and rectifies primacy bias through true trajectories, while still exploiting the sample efficiency from MBRL methods. Secondly, the exploitation-exploration balance requires tuning. Instead of linear decay schedule a more advanced mechanism like curiosity driven exploration [69] with an Intrinsic Curiosity Module (ICM) that predicts the feature representation (inverse dynamics model) and uses the prediction error as an incentive for exploration. Furthermore, to build on the sample efficiency property of MBRL, adding prioritized replay can further stabilize the optimization steps [70, 71, 72], and guide the training process towards an expressive world model. Adding a priority measure, such as the temporal difference error [70], to each experience allows for more effective sampling [71]. This modification would be particularly useful in the MARLOES environment, where sampling greatly affects the algorithm policy (Section 4.3, Section 4.1).

To ensure optimal hyperparameters, [67] suggest using separate training and testing seeds to improve fairness of comparisons. This is a rather simple improvement that greatly reduces the risk of overfitting. Comparing models with different seeds risks selecting a model based an overfitted model as the performance is limited to that data [67]. Additionaly, to reduce training time, Hyperband recognizes weak configurations of parameters, and stops training of these configurations early, leaving time for more promising configurations [66, 67]. Additionaly, the variance as described in Section 4.1 showed fluctuations between configurations with different number of handlers. For these experiments, only one set of hyperparameters was used optimized on a 3-handler setup, while tuning on a specific configuration is needed to reduce variance and stabilize policy optimization. This further expresses the importance of parameter tuning for a specific environment.

Where this research reduced the rollout length to mitigate compounding error [61], along with it the probability of recognizing the long-term reward inherent to the workings of a battery in such a REV was reduced. This asks for a more adaptive horizon strategy, with longer rollout horizon in promising trajectories. This dynamically balances the short-term stability while retaining the ability to capture long-term dependencies [62, 59].

Lastly, a larger training window would provide more insights into the loss progression, besides the additional information about seasonality in the RES data profiles. Section H showed that 30k timesteps was insufficient for a selection of the components in the algorithm to convergence and stabilize, which could have affected the results shown in the thesis. By increasing the number of training steps, we would gain valuable information about the direction of each individual loss, allowing us to further pinpoint the cause of bias and instability.

Environmental modifications As for the MARLOES environment, future research could explore adjusting the temporal resolution to make full-year simulations feasible, granting the algorithm more training samples in different seasons - further working towards the goals set by REFORMERS [16, 15]. Other than seasonality the national grid consists of different sources of energy throughout the day [73]. This is not reflected in the CO_2-reward and can be an interesting change to the reward component. A more detailed calculation of the actual CO_2-emissions of the grid can result in interesting strategies. In addition, incorporating financial or monetary objectives next to the environmental objective allows for further investigation of multi-agent reinforcement learning methods in the environment MARLOES. As a result, the CTCE approach would not fit the problem, and other paradigms can be explored.

Bibliography

[1] International Energy Agency (IEA), “World energy outlook 2022,” tech. rep., International Energy Agency, Paris, 2022. Licence: CC BY 4.0 (report); CC BY-NC-SA 4.0 (Annex A).

[2] U.S. Energy Information Administration (EIA), “International energy outlook 2023: Narrative,” tech. rep., U.S. Energy Information Administration, Washington, DC, 2023. Accessed February 28, 2025.

[3] V. Z. Castillo, H.-S. de Boer, R. M. Muñoz, D. E. Gernaat, R. Benders, and D. van Vuuren, “Future global electricity demand load curves,” *Energy*, vol. 258, p. 124741, 2022.

[4] European Union, “Regulation (eu) 2021/1119 of the european parliament and of the council of 9 june 2021 establishing the framework for achieving climate neutrality.” Official Journal of the European Union, L 243, 15.06.2021, pp. 1–17. Accessed: 2025-03-02.

[5] European Environment Agency, “Greenhouse gases viewer: Data viewers,” 2025. Accessed: 2025-03-02.

[6] European Commission, “Communication from the commission to the european parliament, the european council, the council, the european economic and social committee and the committee of the regions: *REPowerEU Plan*,” Communication COM(2022) 230 final, European Commission, Brussels, May 2022. Dated 18 May 2022. CELEX 52022DC0230.

[7] C. of the European Union, “Repowereu: energy policy in eu countries’ recovery and resilience plans,” Feb. 2025. Policy explainer; last reviewed 18 February 2025.

[8] IPCC, *Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty*. Cambridge, UK and New York, NY, USA: Cambridge University Press, 2018.

[9] U. N. E. C. for Europe, *Carbon Neutrality in the UNECE Region: Integrated Life-cycle Assessment of Electricity Sources*. Geneva: United Nations, 2022. © 2021 United Nations.

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

[11] A. Perera and P. Kamalaruban, “Applications of reinforcement learning in energy systems,” *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110618, 2021. Open access under CC BY license.

[12] A. S. Isha Das, Md. Jisan Ahmed, “Optimizing solar microgrid efficiency via reinforcement learning: An empirical study using real-time energy flow and weather forecasts,” *International Journal of Computer Applications*, vol. 187, pp. 33–38, Jun 2025.

[13] M. Mohammadi, Y. Noorollahi, B. Mohammadi-ivatloo, and H. Yousefi, “Energy hub: From a model to a concept – a review,” *Renewable and Sustainable Energy Reviews*, vol. 80, pp. 1512–1527, 2017.

[14] A. A. Eladl, M. I. El-Afifi, M. M. El-Saadawi, and B. E. Sedhom, “A review on energy hubs: Models, methods, classification, applications, and future trends,” *Alexandria Engineering Journal*, vol. 68, pp. 315–342, 2023.

[15] New Energy Coalition, “Reformers: Regional ecosystems for multiple-energy resilient system,” 2025. Accessed on January 25, 2025.

[16] R. Hueting and F. Lovati, “How renewable energy valley can impact our future,” 2025. Accessed on January 25, 2025.

[17] E. van der Sar, A. Zocca, and S. Bhulai, “Multi-agent reinforcement learning for power grid topology optimization,” in *23rd Power Systems Computation Conference (PSCC)*, (Paris, France), pp. June 4–7, IEEE, 2024.

[18] R. Roche, B. Blunier, A. Miraoui, V. Hilaire, and A. Koukam, “Multi-agent systems for grid energy management: A short review,” in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pp. 3341–3346, 2010.

[19] G. Halhoul Merabet, M. Essaaidi, H. Talei, M. R. Abid, and N. Khalil, “Applications of multi-agent systems in smart grids: A survey,” vol. 0, pp. 1088–1094, 04 2014.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[21] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, “Model-based reinforcement learning for atari,” 2024.

[22] M. P. Deisenroth and C. E. Rasmussen, “Pilco: a model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML’11, (Madison, WI, USA), p. 465–472, Omnipress, 2011.

[23] D. Ha and J. Schmidhuber, “Recurrent world models facilitate policy evolution,” 2018.

[24] V. Egorov and A. Shpilman, “Scalable Multi-Agent Model-Based Reinforcement Learning,” in *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022)*, pp. 381–390, International Foundation for Autonomous Agents and Multiagent Systems, 2022. Online Conference, May 9–13, 2022.

[25] Y. Zhang, C. Bai, B. Zhao, J. Yan, X. Li, and X. Li, “Decentralized transformers with centralized aggregation are sample-efficient multi-agent world models,” 2024.

[26] Z. Xu, D. Li, B. Zhang, Y. Zhan, Y. Bai, and G. Fan, “Mingling foresight with imagination: Model-based cooperative multi-agent reinforcement learning,” 2022.

[27] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, “Gymnasium: A standard interface for reinforcement learning environments,” 2024.

[28] R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *SIGART Bull.*, vol. 2, p. 160–163, July 1991.

[29] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, “Mastering atari with discrete world models,” 2022.

[30] S. Geman, E. Bienenstock, and R. Doursat, “Neural networks and the bias/variance dilemma,” *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[31] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3–4, p. 219–354, 2018.

[32] Z. Wang, J. Wang, Q. Zhou, B. Li, and H. Li, “Sample-efficient reinforcement learning via conservative model-based actor-critic,” 2021.

[33] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. van Hasselt, and D. Silver, “Successor features for transfer in reinforcement learning,” 2018.

[34] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[35] H. Zhang, S. Seal, D. Wu, B. Boulet, F. Bouffard, and G. Joos, “Data-driven model predictive and reinforcement learning based control for building energy management: a survey,” 2021.

[36] S. Keren, C. Essayeh, S. V. Albrecht, and T. Morstyn, “Multi-agent reinforcement learning for energy networks: Computational challenges, progress and open problems,” 2024.

[37] A. Jain, J. Sridevi, U. Dabral, A. Malhotra, and I. Kapila, “Multi-agent reinforcement learning for power system operation and control,” in *E3S Web of Conferences*, vol. 511, p. 01021, EDP Sciences, 2024. Open access under CC BY 4.0 license.

[38] P. Knap and E. Gerding, *Energy Storage in the Smart Grid: A Multi-agent Deep Reinforcement Learning Approach*, pp. 221–235. Cham: Springer Nature Switzerland, 2024.

[39] J. Yao, J. Xu, N. Zhang, and Y. Guan, “Model-based reinforcement learning method for microgrid optimization scheduling,” *Sustainability*, vol. 15, p. 9235, 06 2023.

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

[41] X. Fang, Q. Zhao, J. Wang, Y. Han, and Y. Li, “Multi-agent deep reinforcement learning for distributed energy management and strategy optimization of microgrid market,” *Sustainable Cities and Society*, vol. 74, p. 103163, 2021.

[42] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, “Mastering diverse domains through world models,” 2024.

[43] Repowered NL, “Simon: Agent-based energy simulation platform.” `https://github.com/repowerednl/simon`, 2024. Accessed: 2025-06-16.

[44] G. Lozenguez, “On the distributivity of multi-agent markov decision processes for mobile-robotics,” in *International Symposium on Swarm Behavior and Bio-Inspired Robotics*, (Kyoto, Japan), June 2021.

[45] C. Boutilier, “Planning, learning and coordination in multiagent decision processes,” in *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, (Vancouver, BC, Canada), 1996.

[46] N. Shaya and S. Glöser-Chahoud, “A review of life cycle assessment (lca) studies for hydrogen production technologies through water electrolysis: Recent advances,” *Energies*, vol. 17, no. 16, 2024.

[47] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, pp. 79–86, Mar. 1951.

[48] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” 2024.

[49] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning,” 2018.

[50] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.

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

[52] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, “Understanding the impact of entropy on policy optimization,” 2019.

[53] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, “Pathnet: Evolution channels gradient descent in super neural networks,” 2017.

[54] R. C. Castanyer, J. Obando-Ceron, L. Li, P.-L. Bacon, G. Berseth, A. Courville, and P. S. Castro, “Stable gradients for stable learning at scale in deep reinforcement learning,” 2025.

[55] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1310–1318, PMLR, 17–19 Jun 2013.

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

[57] B. J. Lansdell, P. R. Prakash, and K. P. Kording, “Learning to solve the credit assignment problem,” 2020.

[58] Y. Liang, H. Wu, H. Wang, and H. Cai, “Asynchronous credit assignment for multi-agent reinforcement learning,” 2025.

[59] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, “Model-based reinforcement learning via meta-policy optimization,” 2018.

[60] C. Xiao, Y. Wu, C. Ma, D. Schuurmans, and M. Müller, “Learning to combat compounding-error in model-based reinforcement learning,” 2019.

[61] N. Lambert, K. Pister, and R. Calandra, “Investigating compounding prediction errors in learned dynamics models,” 2022.

[62] M. Janner, J. Fu, M. Zhang, and S. Levine, “When to trust your model: Model-based policy optimization,” 2021.

[63] L. Velvis, “Hybrid multi-agent reinforcement learning for low-carbon dispatch in renewable energy valleys,” Master’s thesis, University of Groningen, 2025.

[64] Z. Qiao, J. Lyu, and X. Li, “Mind the model, not the agent: The primacy bias in model-based rl,” 2024.

[65] G. Tianci, D. D. Dmitry, K. A. Neusypin, Y. Bo, and R. Shengren, “Enhancing sample efficiency and exploration in reinforcement learning through the integration of diffusion models and proximal policy optimization,” 2025.

[66] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” 2018.

[67] T. Eimer, M. Lindauer, and R. Raileanu, “Hyperparameters in reinforcement learning and how to tune them,” 2023.

[68] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning,” 2019.

[69] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” 2017.

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

[71] S. Zhang and R. S. Sutton, “A deeper look at experience replay,” 2018.

[72] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, “Distributed prioritized experience replay,” 2018.

[73] Statistics Netherlands (CBS), “Over half of electricity production now comes from renewable sources,” September 2024. Accessed: 2025-01-07.

[74] J. Viñuelas Ferrando, “Modelling and simulation of hydrogen electrolyzers for power system applications,” master’s thesis, KTH Royal Institute of Technology, 2023. Advanced level (degree of Master (Two Years)), Independent thesis, 20 credits.

[75] P. L. Spath and M. K. Mann, “Biomass power and conventional fossil systems with and without co2 sequestration – comparing the energy balance, greenhouse gas emissions and economics,” Technical Report NREL/TP-510-32575, National Renewable Energy Laboratory, Golden, Colorado, USA, January 2004. Contract No. DE-AC36-99-GO10337.

[76] Z. Pató, “Gridlock in the netherlands,” briefing paper, Regulatory Assistance Project (RAP), February 2024.

Appendices

A Handler Information

Below is a more detailed explanation of the setup of the handlers and their setup regarding data, and parameters. All profiles and forecasts have been provided by Repowered, and are scalable through the experiment interface. At timestep t the handlers with a forecast available have access to a that forecast for a set horizon (4 hours).

Solar handler The solar handler represents a photovoltaic system, generating electricity from solar. Based on the provided orientation, either south (S) or east-west (EW), it loads in a generic profile with the corresponding forecast provided by Repowered. Both the energy profile and the forecast are in 1 hour resolution. S and EW are the most common orientations for optimal generation in The Netherlands. The solar parks are curtailable by the solver in SIMON and thus are allowed setpoints, which are given to the asset by the handler. The two available profiles are real solar profiles located near Alkmaar for 2022 and 2023, scaled to 1 MWp.

Wind handler The wind handler represents wind turbines that generate electricity. MARLOES facilitates one profile of an on-shore, 60 meter high, Vestas V80 2000 close to Alkmaar, with its corresponding forecast provided by Repowered. Both the energy profile and the forecast are in 1 hour resolution. The profile is also for 2022 and 2023, and also scaled down to 1MWp. The wind assets are curtailable and thus are allowed setpoints.

Demand handler To model the load, we introduce the demand handler that represents the energy consumption. the framework holds a demand profile for a dairy farm, with its corresponding forecast. Both the demand profile and the forecasts are in 15 minute resolution. Since the demand is not flexible, it does not require setpoints, but the demand must be met. If there is not enough production from RES the grid must supply.

Battery handler The battery handler is responsible for sending setpoints to a battery asset in SIMON. It can either charge or discharge and thus has action space [-1,1].

Electrolyser handler The electrolyser handler is modelled as a battery, with some additional parameters, simulating a slower operating time, and applying a conversion factor, for a general PEMWE [74][46], to convert hydrogen (H_2) to energy (kW). Similarly to the battery, it can be charged and discharged with action space [-1,1] .

State-properties Below is a more elaborate explanation of the features in the handler states.

1. `power` - represents the actual energy production (positive) or consumption (negative) at a given timestep.
2. `available_power` - represents the available energy production, this would be different from *power* whenever the agent received a limiting setpoint.
3. `state_of_charge` - represents the available energy in a storage unit, which is a percentage of the full capacity.

4. `forecast` - represents the predicted power production or demand of a given asset. Each forecast is given in 15 minute intervals with a configurable horizon.
5. `nomination` - represents the intended production of a supply asset that would bid on the Day Ahead (DA) market. Since DA is based on hourly data, the forecast is transformed to hourly data taking the mean of the forecast.

Producer and consumer defaults Table 13 shows the default values for initialization of the separate handlers.

Table 13: Parameter settings and notes for producers and consumers.

Parameter	**Solar Handler**		**Wind Handler**		**Demand Handler**	
	Value	**Note**	**Value**	**Note**	**Value**	**Note**
`max_power_in`	-	N/A	-	N/A	inf	No limit
`max_power_out`	3000	-	3000	-	-	N/A
`curtailable`	True	-	True	-	False	-
Action space	[0,1]	-	[0,1]	-	N/A	No setpoints

ESS defaults Table 14 shows the default parameters for energy storage systems (ESS) batteries, and electrolysers. The degradation functions are consistent with domain knowledge provided by Repowered.

Table 14: A comparison of default parameters for the storage-based handlers.

Parameter	**Battery**		**Electrolyser**	
	Default Value	**Notes**	**Default Value**	**Notes**
`max_power_in`	1000	–	1000	–
`max_power_out`	1000	–	1000	–
`max_state_of_charge`	0.95	95% of capacity	0.95	95% of hydrogen capacity
`min_state_of_charge`	0.05	5% of capacity	0.05	5% of hydrogen capacity
`energy_capacity`	2000	In kWh	2000×33	Conversion to kWh
`ramp_up_rate`	1000	Instant	$1000 \cdot 0.4$	Models slower startup
`ramp_down_rate`	1000	Instant	$1000 \cdot 0.4$	Models controlled shutdown
N_{cycles}	8000	full charge/discharge	–	N/A
T_{lifetime}	–	N/A	80000	in hours
`input_efficiency`	$0.85^{0.5}$	Round-trip (85%)	0.6	Electricity to H_2
`output_efficiency`	$0.85^{0.5}$	Round-trip (85%)	0.6	H_2 to electricity
`Degradation function`	Cycle-based	–	Hour-based	–

The battery degradation function is a cycle-based degradation function, which is a linear deterioration based on a full charge and discharge of the battery; where the energy throughput is $2{\cdot}C$, with C being the battery capacity (kWh). The battery is expected to retain 60% of its original capacity before it is considered deprecated and is modeled in the first term of Equation 21. The degradation per cycle is calculated as follows:

$$\Delta D_{\text{battery}} = \frac{1-0.6}{N_{\text{cycles}}} \times \frac{t\,|P|}{3600 \times (2C)}, \tag{21}$$

where

- t is the time step in seconds
- $|P|$ is the absolute power output (in kW)
- C is the battery capacity (in kWh)
- N_{total} is the total number of cycles before the battery is considered deprecated.

After the update, the new degradation state of the battery is given by:

$$D_{\text{new}} = D_{\text{old}} + \Delta D_{\text{battery}}. \tag{22}$$

The electrolyser degradation function is hour-based, simulating a steady loss of H_2 [74]. The function is presented in Equation 23.

$$\Delta D_{\text{electrolyser}} = \frac{1-0.6}{T_{\text{lifetime}}} \times \frac{t\,|P|}{3600\,C_{\text{max}}}, \tag{23}$$

wehere

- T_{lifetime} is the operational lifetime in hours.
- C_{max} is the maximum capacity after applying the conversion factor (from H_2 to kWh).
- t and $|P|$ as defined for the battery

The new degradation state for the electrolyser is then given by:

$$D_{\text{new}} = D_{\text{old}} + \Delta D_{\text{electrolyser}}, \tag{24}$$

B MARLOES

We validate the steering of *Simon* through MARLOES with a setup of 4 handlers; one solar handler, two battery handlers and one demand handler. We apply two different strategies and visualize the energy distributions with a flow diagram. Figure 10 quantitatively illustrates that the majority of the demand can be met by solar production, as shown by the dominant flow from the solar to the demand node. Excess solar flows into the batteries, and the remaining solar production is curtailed (Curtailment) when the batteries reached their capacity. The batteries also contribute to meeting the demand, depicted by the green flows from the battery nodes to the demand node. Finally, grid contribution is minimal; providing energy to demand if it can not be met by solar or batteries, and charging batteries or discharging batteries compensating minor imbalances or errors in forecasts.

B.1 PrioFlow

Figure 10 shows a comprehensive distribution of energy for the heuristic baseline *PrioFlow* (Section 2.2.1) based on priorities with an adaptation for batteries (Section C.1).

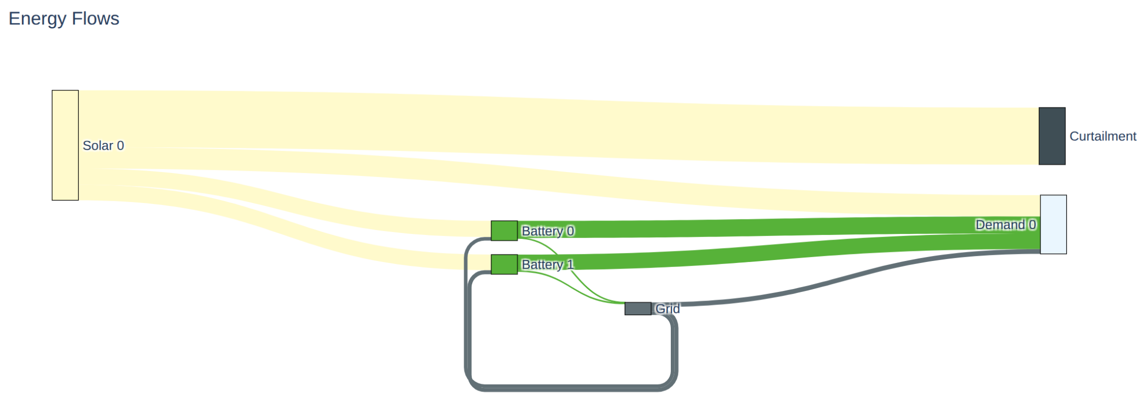

Figure 10: A flow diagram showing the distribution of energy flows under the PrioFlow strategy.

B.2 Random setpoints

The energy flow distribution between 4 assets for the random setpoint strategy *SimpleSetpoint* is given in Figure 11. In a random strategy we expect a much higher contribution from the grid, since solar production is very likely to be limited, and batteries could be charging or discharging at any time causing many inaccuracies and major imbalances to compensate. Figure 11 shows that the grid contribution is much higher compared to Figure 10 indicating imbalances and a need for compensation. The solar node minimally contributes to the demand node directly, while batteries are charged and discharged by the grid in large quantities. The battery nodes contribute to the demand with energy taken from the grid, which is in itself very inefficient.

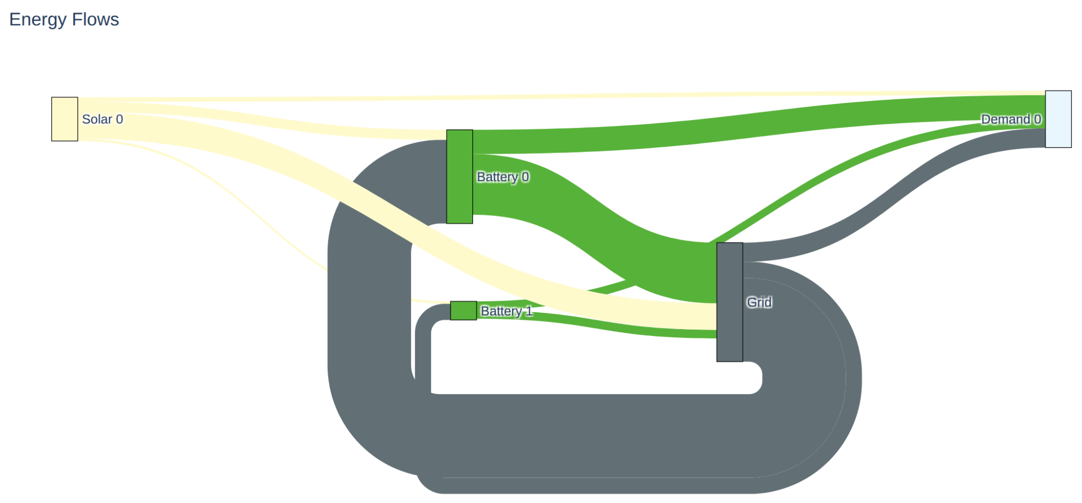

Figure 11: A flow diagram showing the distribution of energy flows under a strategy with random setpoints.

C PrioFlow

C.1 Adaptation of Priorities Solver

In priorities Solver (PS) from SIMON, energy flows are determined according to a preset priority scheme between agents (Table 15). While this mechanism is sufficient for uncontrollable assets (e.g., solar, wind) or unidirectional demand, battery agents require a dynamic decision process that reflects their dual role that makes them valuable in a REV. Energy Storage Systems (ESS) as batteries and electrolysers, both consume and produce energy, allowing them to compensate imbalances in the system.

Source Asset	Target Asset	Priority Value
Solar/Wind	Demand	3
Solar/Wind	Battery/Electrolyser	2
Battery/Electrolyser	Demand	3
Battery	Electrolyser	2
Battery/Electrolyser/Solar/Wind	Grid	1
Grid	Demand	1

Table 15: Priority values used by the Priorities Solver for directing energy flows between assets.

To accommodate this, *PrioFlow* extends PS with a strategy for ESS actions based on the net forecasted power in the system. When net forecasted power is positive (indicating surplus), ESS are instructed to absorb energy proportional to their capacity. When net power is negative (indicating shortage), ESS are discharged to cover the deficit.
However, this approach does not account for SOC of the batteries, or long-term optimization objectives with ESS. Consequently, it may result in inefficient cycling or suboptimal results. Therefore, the method is most appropriate as a baseline or fallback strategy, and mainly used to validate MARLOES as an environment.

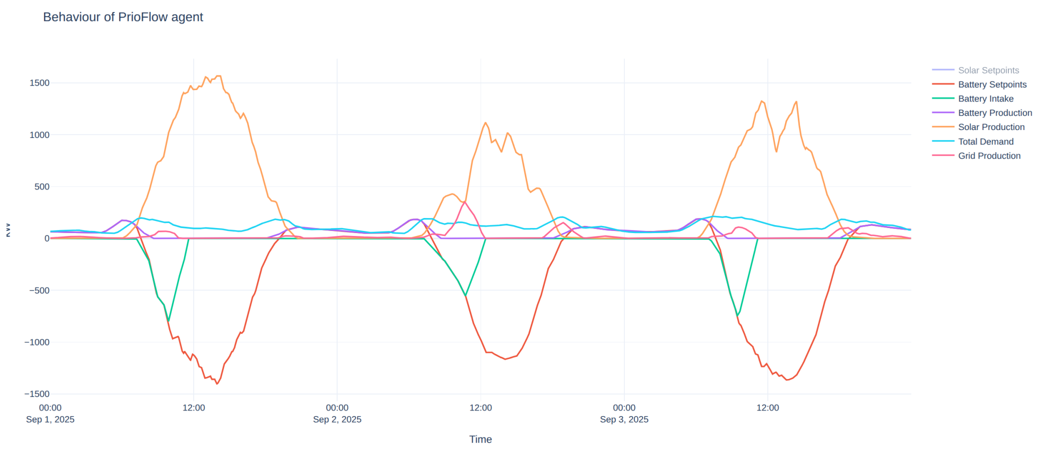

Figure 12: The battery setpoints are based on the forecasted available solar power; the red line shows an inverse solar production profile. The solar forecast is rather optimistic at the start and at times as the grid also is required to meet the demand and the charging setpoints from the battery (pink) at the start of the day. The battery is charged (green) until it has reached capacity. The battery closely meets the demand during the minutes where there is no solar production (purple).

Algorithm 1 Pseudocode for ESS action determination in PrioFlow

```
1:  function DETERMINE_ESS_ACTIONS(net_forecasted_power, batteries)
2:      total_capacity ← $\sum_{s \in \text{ESS}} s$.energy_capacity
3:      for each $s$ in ESS do
4:          share ← $b$.energy_capacity / total_capacity
5:          desired_action ← −net_power * share
6:          normalize action for $s$
7:          ess_actions ← normalized action
8:      end for
9:      return ess_actions
10: end function
```

C.2 Validation

A quantitative distribution of energy flows for *PrioFlow* is presented in Figure 10, which shows a coherent system, with minimal grid contribution to compensate unmet demand. Figure 12 shows the behaviour of 3 handlers (1 solar, 1 battery and 1 demand) and the energy production or consumption per minute over 3 days. Solar setpoints are disabled as *PrioFlow* always produces maximum solar.

D Temporal Feature Encoding

To represent time in a form that preserves its cyclical nature, each component of a timestamp is encoded on the unit circle using sine and cosine transformations (Equation 25). The resulting features are continuous, normalized to the $[-1,1]$ interval, and well-suited for machine learning models such as neural networks.

$$x_{\sin} = \sin\left(2\pi \cdot \frac{x}{P}\right), \quad x_{\cos} = \cos\left(2\pi \cdot \frac{x}{P}\right) \tag{25}$$

where, x denotes the time component and P its period:

$$P = \begin{cases} 12 & \text{for months} \\ 31 & \text{for days} \\ 24 & \text{for hours} \\ 60 & \text{for minutes} \end{cases}$$

This encoding yields 8 features per timestamp:

$$\text{month}_{\text{sin}},\ \text{month}_{\text{cos}},\ \text{day}_{\text{sin}},\ \text{day}_{\text{cos}},\ \text{hour}_{\text{sin}},\ \text{hour}_{\text{cos}},\ \text{minute}_{\text{sin}},\ \text{minute}_{\text{cos}}$$

E Reward

E.1 CO_2

Since the REV is located in Alkmaar, we estimate the CO_2 emissions or global warming potential (GWP) of the national electricity grid in the Netherlands. This estimate is derived from the emission factors per energy source provided by global environmental impact assessment [9], combined with the energy mix data for the Dutch electricity grid as reported by the national statistics bureau CBS [73].
Additionally, emission factors for biomass were obtained from a separate assessment [75] to account for its role in the composition of the Dutch grid. The resulting production shares and emission factors are summarized in Table 16.
From this data, the actual CO_2 reward used as feedback in for the training process at timestep t was calculated as follows:

$$r_t^{CO_2} = P^+_{grid} \times \text{GWP}_{grid} \tag{26}$$

where

1. $P^+_{grid_t}$: the power provided by the grid necessary to maintain the balance,
2. $\text{GWP}_{grid} = 284.73$, the result of the calculations in Table 16.

	Production (billion kWh)	**Emission Factor** (gCO_2eq/kWh)	**Weighted Contribution** (gCO_2eq/kWh)
Renewable			
Wind	29.166	15.5	4.07
Solar	19.993	45.5	8.19
Biomass	6.776	49.0	2.99
Hydro	0.068	8.55	0.005
Non-renewable			
Natural Gas	44.873	458	185.11
Coal	10.146	923	84.35
Total	111.022	–	**284.73**

Table 16: Breakdown of Production, Emission Factors, and Weighted Contributions to Grid Emissions

E.2 Battery Incentive

In early experiments, the battery agent struggled to learn effective behaviour due to sparse rewards and ambiguity as a result of the nature of the battery. If the battery is at max capacity or empty, certain actions/setpoints will have no effect, while it would, in fact, still be the best course of action. To guide learning, an auxiliary incentive was introduced to align battery actions with system-level surplus or deficit conditions, separate from the consequences of the actions.

At each timestep t, the total battery action a_t^{bat} is computed by aggregating the setpoints of all individual batteries, where charging actions are negative and discharging actions are positive. The system surplus v_t is defined as the difference between available renewable supply and total demand.
The battery incentive is then given by:

$$i_t^{\mathcal{B}} = -\frac{\left|\sum_{b\in\mathcal{B}} a_t^b + v_t\right|}{\sum_{b\in\mathcal{B}} P_b^{\max}} \tag{27}$$

where:

- $\mathcal{B}$: the set of all battery agents,
- a_t^b: action (power setpoint) of battery b at time t,
- v_t: system surplus, defined as $v_t = \text{available\_supply}_t - \text{demand}_t$,
- $C_b^{\max}$: maximum charging or discharging power of battery b.

This incentive encourages batteries to send a charge setpoint when there is excess renewable energy ($s_t > 0$) even when there is no capacity and to discharge during energy shortages ($s_t < 0$). The goal of this incentive is to improve system-wide flexibility and learning stability.

F Additional Rewards

MARLOES allows optimization on multiple objectives through additive sub-rewards. This section contains sub-rewards that were implemented but not used in the experiments for this paper.

F.1 NB (Net Balance)

This reward encourages the system to maintain a net-positive energy balance. At each timestep t, the reward is based on the previous net power balance with respect to the grid:

$$r_t^{\text{NB}} = -P_{t-1}^{\text{net}} \tag{28}$$

where P_{t-1}^{net} denotes the net power imbalance at the previous timestep, which is positive if more energy was taken from the grid and negative if more energy was supplied to the grid.

F.2 NC (Net Congestion)

This reward penalizes energy exports to the grid, which may contribute to local congestion issues [76]. The reward is calculated based on the grid state:

$$r_t^{\text{NC}} = \min(0,\ P_t^{\text{grid}}) \tag{29}$$

where P_t^{grid} is the net power delivered to the grid. Positive values (exports) are penalized, while extracting from the grid is not penalized.

F.3 NE (Nomination Error)

This reward penalizes mismatches between the nominated and actual energy delivered. This becomes especially relevant when MARLOES is extended with price information, as deviating from your nomination can be very costly depending on market prices. It is computed hourly as:

$$r_t^{\text{NE}} = -\min\left(1,\ \frac{|\bar{P}^{\text{prod}} - \bar{P}^{\text{nom}}|}{P_{\text{nom}}^{\text{max}}}\right) \tag{30}$$

where:

- $\bar{P}^{\text{prod}}$: mean production over the past hour (solar + wind − demand),
- $\bar{P}^{\text{nom}}$: mean nominated power over the past hour,
- $P_{\text{nom}}^{\text{max}}$: maximum expected nomination deviation based on capacity estimates.

G Noise functions

We implement two simple manual perturbation functions to simulate sensor dropouts and measurement noise, allowing the agent to train and evaluate under more realistic and uncertain scenarios. The function in Algorithm 2 adds sporadic single drop out data, and longer outtages with probabilities $drop_{prob}$ and $long\_drop_{prob}$ respectively. The function in Algorithm 3 adds noise to the forecast series proportional to its variance, simulating measurement uncertainties.

Algorithm 2 Simulating Random Dropouts (`drop_out_series`)

```
 1: function DROP_OUT_SERIES(forecast, drop_prob, long_drop_prob)
 2:     for each value in forecast do
 3:         forecast $\leftarrow$ drop value with probability drop_prob
 4:     end for
 5:     for each value in forecast do
 6:         with probability long_drop_prob:
 7:             $\ell \leftarrow \text{randint}(1, 1440)$
 8:             forecast $\leftarrow$ drop $\ell$ values
 9:     end for
10:     return forecast
11: end function
```

Algorithm 3 Adding Gaussian Noise (`add_noise_to_series`)

```
1: function ADD_NOISE_TO_SERIES(forecast, noise)
2:     $\sigma \leftarrow \text{forecast.std}() \cdot \text{noise}$
3:     return forecast + normal(0,$\sigma$)
4: end function
```

H Experiment Losses

The loss convergence can tell us a lot about the inner workings of the learning process in a model-based algorithm. Due to the separate modules, and loss functions we can base conclusions about instabilities in the loss graphs.

Loss convergence Figure 13 and Figure 14 depict the training trajectories of the world-model (WM) losses (dynamics, prediction, representation) and the actor–critic losses, respectively. Both configurations show qualitatively similar curves, with a rapid initial decrease in representation and prediction losses, a peak in dynamics loss around 5–10 k steps, and eventual decay toward lower values. The 6-handler setup exhibits a slightly higher plateau in dynamics loss (≈ 5.8 to ≈ 4.4) and a more unstable actor loss, indicating that the increased state- and decision-space size among a larger set of agents especially affects the central controller while learning via the world model, and the evaluation via the critic remain unchanged with minor differences.

In the WM, we see a rapid convergence for the representation loss, which reflects the encoder's ability to compress raw input observations into a latent state. The fact that this loss stabilizes quickly at a low value suggests that the encoder consistently learns a compact representation of the environment, even as the number of agents increases.

The prediction loss, derived from the decoder and reward predictor, decreases steeply during the first few thousand steps but then shows a slow increase. This indicates that while the model initially learns to reconstruct and predict well, the accuracy of these reconstructions decreases over time, which could be an indicator of compounding error in latent rollouts generated by the dynamics model.

The dynamics loss which captures how well the model predicts latent transitions, continues to decrease over the entire 30k step horizon without convergence within the time window. This trend suggests that the dynamics component has not converged yet and would likely benefit from a longer training schedule.

Extending training would also allow further investigation into the actor and critic losses (Figure 14). The actor loss seems to stabilize after a while, but also shows sudden fluctuations just before the end of the training time. The critic on the other hand, shows a rather stable process, but we cannot conclude that this has converged yet.

Together we see fast convergence of some components of the world model, but the transitioning in latent space and the policy evaluation remain undertrained in the current training window.

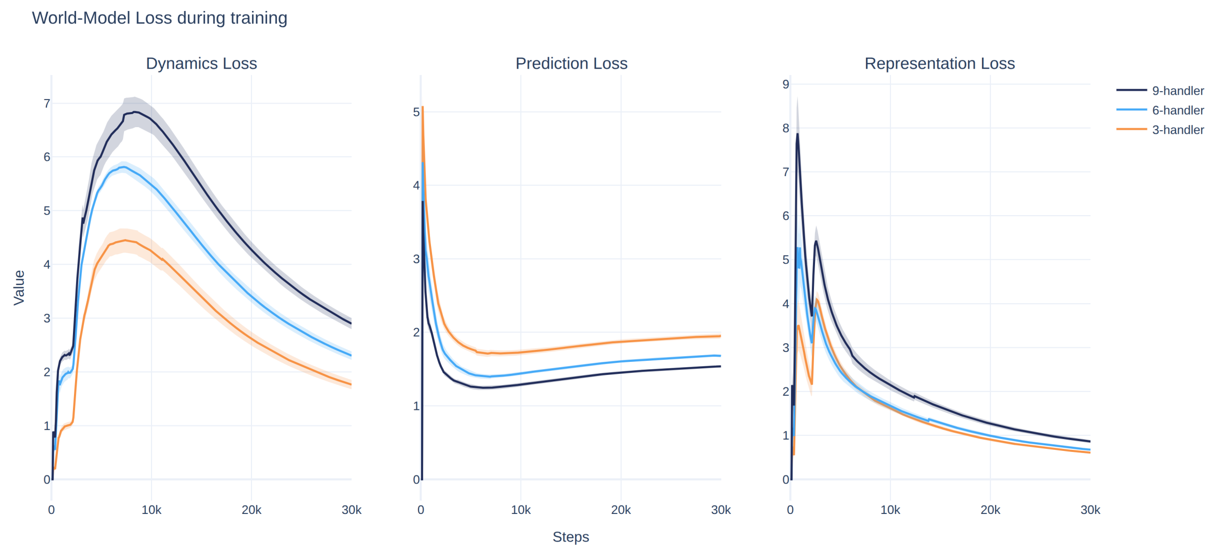

Figure 13: World Model loss convergence over 30k training episodes for 2 (3-handler), 4 (6-handler) and 6 (9-handler) trainable agents.

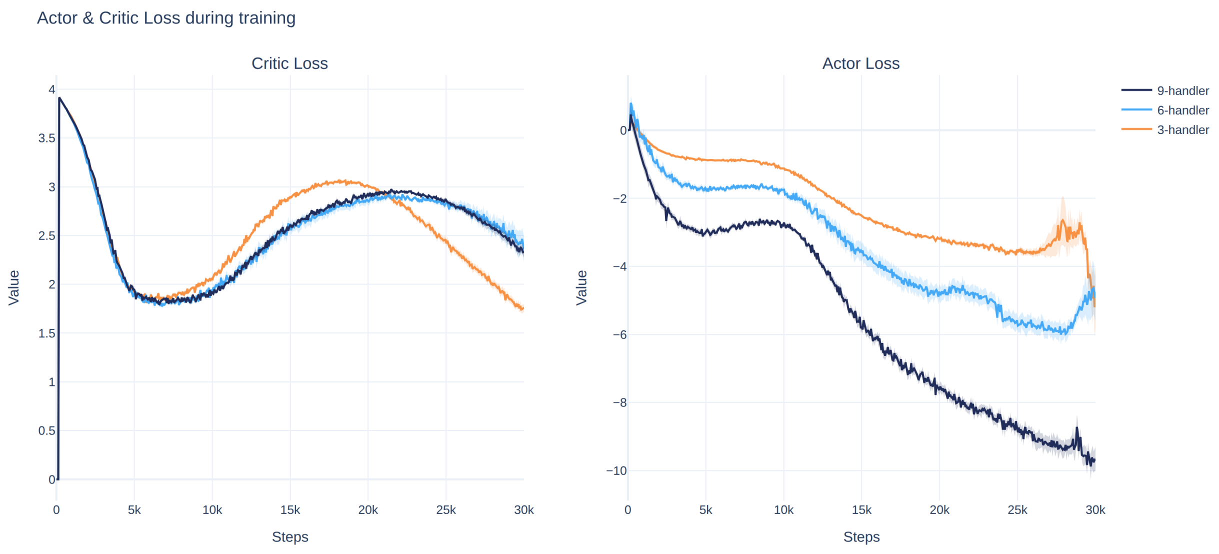

Figure 14: Actor and Critic loss convergence over 30k training episodes for 2 (3-handler), 4 (6-handler) and 6 (9-handler) trainable agents.

I Robustness

Experiment To evaluate how well MADreamer generalizes beyond idealized simulations, we alter the data with seeded noise using the functions described in Section G. The goal is to see whether the learning process of MADreamer can withstand world imperfections that can occur in a real REV. The 3-handler configuration in Table 6 was used.

- Gaussian noise (`noise`=0.01).
- Random dropouts (`drop_prob`=0.001).
- Extended outages (`long_drop_prob`=0.0001, up to 1 day).

By comparing performance metrics under clean and predictable data to the same data under noisy conditions, we quantify robustness to sensor errors, outtages and other reasons for missing information. The functions are applied to all forecast and power data.

Results To assess the training robustness Figure 15 compares the reward and grid production per timestep during evaluation of the policy of MADreamer trained with and without artificial noise. The policy trained on noise-free data has a consistently lower grid dependency, therefore also a higher reward.

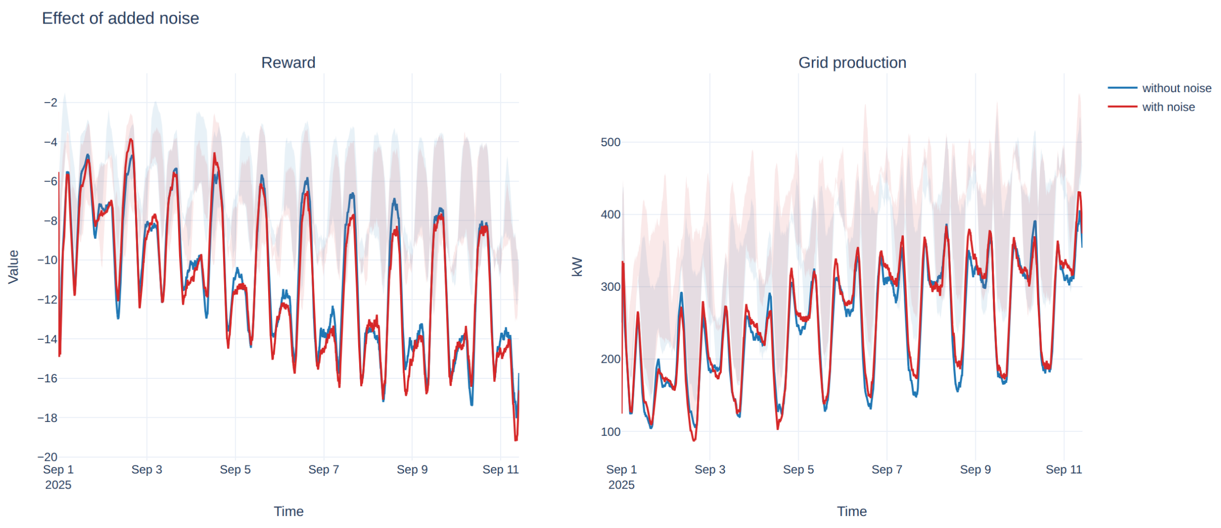

Figure 15: Policy evaluation comparison of the MADreamer algorithm trained on data with and without the addition of artificial noise.

Table 17 summarizes longer-term performance metrics. The DSR for the policy trained on noisy data (18.88 %) exceeds that of the policy trained on clean data (14.05 %), while its total CO_2 emissions increase only by 0.43%. One explanation would be a more consistent policy for solar handlers, sending more correct setpoints meeting demand with solar production, increasing DSR, while the battery is not used or used less at times where the grid would have to compensate, therefore increasing grid production. Another possibility revolves around the battery; it may be charged during hours where the grid is already supplying energy, resulting in higher grid production, but resulting in more capacity for the battery to discharge at times with less demand, increasing DSR and increasing the total grid production.

Configuration	**DSR** (%)	**Total CO_2 Emissions** ($\times 10^8 gCO_2$)
MADreamer	14.05	4.70
MADreamer (with noise)	18.88	4.72

Table 17: Demand satisfaction ratio and total CO_2 emissions for MADreamer, and MADreamer with the addition of noise.