university of groningen

faculty of science and engineering

Hybrid Multi-Agent Reinforcement Learning for Low-Carbon Dispatch in Renewable Energy Valleys

Lucas Velvis

university of groningen

faculty of science and engineering

University of Groningen

Hybrid Multi-Agent Reinforcement Learning for Low-Carbon Dispatch in Renewable Energy Valleys

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)

Lucas Velvis (s3993582)

July 18, 2025

Contents

Acknowledgments

I would like to thank my university supervisor, Davide Grossi, for his valuable feedback and guidance during this project. I would also like to express my sincere gratitude to my external supervisor, Robin Hermes, who was always available to offer insights through enjoyable and constructive conversations.

A special thanks goes to Matthias Drijfhout, with whom I had the pleasure of carrying out part of this research. I thoroughly enjoyed our collaboration, which was truly superb in my opinion.

Moreover, I would like to thank Marleen Schippers, the MSc project coordinator, for her responsiveness and support.

Finally, I am also grateful to Repowered, and Jeroen van den Berg in particular, for giving me the opportunity to carry out my thesis within their organization, and for providing a stimulating environment and valuable resources throughout the project.

Abstract

The European Union must cut 55% of 1990-level emissions by 2030. However, intermittent renewables complicate electricity-flow optimization in the recently emerging Renewable Energy Valleys. We present MARLOES, a simulation benchmark for energy hubs, and propose a hybrid multi-agent Dyna-SAC controller which combines model-free robustness with model-based sample efficiency. Over ≈ 35 day simulations, using 20% synthetic rollouts, the algorithm cuts CO_2-equivalent emissions by 32.7% versus a heuristic baseline and 23.4% relative to pure SAC. Under additional sensor and forecast noise the hybrid advantage vanishes, confirming that world-model benefits depend on model reliability. These results show that carefully calibrated model-based augmentation can advance low-carbon dispatch in distributed energy systems aligned with EU decarbonization goals, and that MARLOES provides a flexible platform for future approaches.

1 Introduction

In 2023 the European Union emitted an estimated 2.9 Gt CO_2-eq of greenhouse gases [1], remaining substantially above its 2030 target of approximately 2.09 Gt CO_2-eq[1] set by the European Climate Law (ECL) [2] in 2021. To accelerate achievement of the targets set by the ECL, the European Union has proposed the REPowerEU Plan [4]. Its intention is to reduce the EU's dependence on fossil fuels and further diversify its ways of energy generation, built upon the following three pillars: save energy, increase clean, "green" energy production, and diversify ways of energy supply.

Renewable Energy Valleys (REVs) are geographically concentrated clusters of renewable energy producers, storage assets, and consumers, enabling direct local balancing of supply and demand, in literature often described as Energy Hubs [5, 6], or, in the more strict sense limited solely to electricity, micro-grids [7–9]. Within REVs, the pillars of the REPowerEU plan intersect. They show potential in integrating distributed renewable energy production, demand and storage, aiding in reducing fossil fuel dependence. Such integration of local sources allows for a decentralized energy management structure that is commonly regarded as more future-proof than a centralized one [7, 8].

The REFORMERS project [10] falls exactly within this context. Their goal is to create a REV in Boekelermeer, next to the city of Alkmaar in the Netherlands, which will serve as a laboratory for further development of state-of-the-art renewable technologies. This REV would further serve as a pilot where a framework can be constructed to streamline the process of establishing multiple such REVs all throughout Europe.

However, integrating renewable energy sources comes with challenges, specifically the *energy dispatch problem* (EDP) [11]. Due to the stochasticity that is intrinsic to most forms of green energy generation, blending them interferes with the system's flexibility. Notable sources such as photovoltaic (PV) systems and wind turbines produce intermittently and unstable compared to traditional fossil-fuel based generators. In combination with the increase in inter-dependencies with emerging energy forms [12], this poses substantial challenges for effective optimization and management of energy systems like REVs, for which this work aims to provide a suitable approach.

2 Related Work and Background

2.1 The Energy Dispatch Problem (EDP)

The focal point of the EDP is to deliver energy demand through optimal energy flow management of renewable energy technologies, combining energy storage with (dispatchable) energy sources [11].

Traditionally, model-based optimization techniques have shown to be a popular approach to the EDP. Such techniques include model predictive control (MPC), such as by Shi *et al.* [13], where MPC was used to enhance the flexibility of dispatch scheduling. Likewise, in Li *et al.* [14], a two-stage mixed-integer linear programming (MILP) approach was employed, which also explicitly formulates

[1]A 55% reduction as mentioned in *Regulation (EU) 2021/1119 of the European Parliament and of the Council of 30 June 2021 establishing the framework for achieving climate neutrality and amending Regulations (EC) No 401/2009 and (EU) 2018/1999 ('European Climate Law')* [2] of the 4.64 Gt CO_2-eq as emitted in 1990 according to (EEA) [3], leaves 2.09 Gt CO_2-eq remaining.

a mathematical model of the system.
However, traditionally, these approaches come with an inherent limitation: they require explicit mathematical models and complete parameter information. Additionally, their potential to scale to larger REVs is limited, since the computational complexity increases exponentially with the increase in variables. Therefore, these large, detailed physical models tend to become exceedingly computationally demanding and inefficient, resulting in difficulties in regards to their practical application [11].

2.2 Machine Learning Approaches for EDP

As a result, more recently, machine-learning (ML) methods have become a more popular approach to the EDP. The data-driven nature of ML minimizes dependence on detailed physical models. Moreover, several ML methods have shown to outperform traditional approaches in terms of efficiency [15], as well as handling uncertainty [16]. Consequently, they offer several advantages in addressing the previously mentioned challenges.

In the context of energy systems, ML methods have been frequently extended to a multi-agent setting [7–9]. In these studies, each energy *asset*[2] is modeled as an autonomous agent with its own state and decision making process. This decentralized framework aligns naturally with the actual distribution of energy resources. Within this same domain, Merabet *et al.* [8] suggest that a (hierarchical) multi-agent framework can be used to extend beyond a single REV, enabling interconnections between multiple REVs.

Among ML methods, Reinforcement Learning (RL) has emerged as an especially promising approach. It enables agents to learn optimal actions through trial-and-error, by interacting with the environment, in order to maximize a reward (signal) over time [17]. The multi-agent setting of RL is commonly known as *Multi-Agent Reinforcement Learning* (MARL). Furthermore, deep Reinforcement Learning (DRL) involves the use of multi-layered neural network as function approximators in RL and has shown strong potential in tackling high-dimensional, complex state and action spaces [17]. By learning different levels of abstractions from data, its ability to generalize is improved, allowing it to make more effective decisions. Moreover, the concept of RL learning allows for the discovery of optimal strategies without necessarily requiring an explicit model of the system. Within the context of REVs therefore, RL and MARL are particularly relevant due to the highly distributed and dynamic nature of energy sources. The nature of RL allows agents to learn optimal coordination strategies in real-time, and can flexibly adapt to address unforeseen uncertainties such as production intermittency or fast-changing demand to a greater extent than the previously mentioned traditional approaches.

In addition, a multi-agent DRL (MADRL) approach can offer an efficient learning process for large-scale REVs, where ‘single‘ agent methods often suffer from high dimension joint action and state spaces, making training extensive or even intractable [18]. This is a result of the distribution of the learning process over the agents. As a result, MADRL provides a natural, scalable approach to aligning asset-level objectives with overarching EDP goals. However, one of the main challenges of MARL is that of nonstationary environments [19]. Due to the fact that agents are independently updating their policies during the learning process, to any single particular agent the environment

[2]Any entity within the system that can produce, store, or consume energy. This includes renewable energy generators (e.g., solar PV), storage (e.g., batteries), and load (e.g., demand).

appears non-stationary, complicating the learning process.

2.3 Model-Based and Model-Free Methods for the EDP

Reinforcement learning methods are generally categorized into *model-free* or *model-based* approaches [17]. In model-free RL the agent learns a value function to approximate the policy (value-based methods, e.g., Q-Learning, Deep Q-Networks), or optimizes the policy directly (policy-based methods, e.g., REINFORCE, Proximal Policy Optimization). Generally it deals better with stochasticity than model-based RL (MBRL) because it does not make use of an explicit model of the environment, which also makes it more effective in high-dimensional and complex state/action spaces [17]. Since the uncertainties in demand, such as renewable energy potential, energy market, and grid conditions can be regarded as major problems in energy dispatch [7], model-free approaches offer potential for resolving the EDP in REVs. On the other hand, in general its planning and predictive capabilities are limited and it has high sample complexity, which is often considered a critical shortcoming [11].

The survey by Perera and Kamalaruban [11] does however show that model-free RL is a popular approach within this context, as 69.5% of the reviewed papers that focused specifically on the energy dispatch problem made use of model-free RL. Interestingly, only 2.2% of the papers focusing on the EDP made use of policy-based RL methods, with state-of-the-art methods such as PPO missing from them entirely. The same goes for state-of-the-art actor-critic methods such as TD3 and SAC, which has shown robust performance in handling high stochasticity through the use of off-policy learning and maximum entropy objectives [20]. More extensive research reveals, more recently, works like that of Das *et al.* [21] have started exploring these advanced methods, though their application is still scarce.

MBRL has been very rarely employed in this context (13% according to Perera and Kamalaruban [11]), also seldom utilizing state-of-the-art approaches such as Dreamer [22], which uses recurrent neural networks to learn a less complex latent representation of high-dimensional inputs, or MBPO [23] which uses short model-generated rollouts to achieve strong sample efficiency, which is a typical characteristic of MBRL [17, 24]. Although MBRL approaches can struggle in high-dimensional or stochastic environments, through its sample-efficiency it shows potential within the EDP context.

Notably, methods that make use of function approximators such as neural networks seem to be greatly outnumbered by other tabular-based methods within the EDP context [11], indicating a potential for further research in that direction.

Ultimately, EDP scenarios feature uncertainties such as fluctuating renewable availability and variable loads [7], and benefit from planning and sample efficiency [11]. Therefore, neither purely model-free nor purely model-based RL addresses the challenges of robustness to stochasticity and sample efficiency perfectly. Ideally, a hybrid strategy could be employed, combining the strengths of both paradigms, utilizing model-based techniques to enhance data efficiency and using model-free methods to handle high-dimensional, unpredictable settings [25].

2.4 Proposed Hybrid Approach: A Dyna-Q Style Algorithm

An example of such a hybrid approach to the EDP is the *Dyna-Q style* family of algorithms [26]: a foundational and promising hybrid approach which learns a model of the environment to generate synthetic experiences, using it to make the learning process of a model-free RL agent more efficient. In energy dispatch contexts, Dyna-Q has already shown potential in other contexts [27], but its specific application to the EDP scenario is still largely unexplored.

In this work, we therefore adopt a Dyna-Q style hybrid approach for MARL, aiming to combine model-based efficiency through rapid learning via simulated samples and model-free flexibility for robust behavior in a stochastic setting. This hybrid design addresses the limitations of purely model-free methods (high sample complexity) and purely model-based methods (limited scalability in uncertain settings) by aiming to find a balance that is well suited to the dynamics and uncertainties of the EDP in REVs.

3 Research Objectives

In conclusion, the goal of this study is to develop a Dyna style hybrid approach for electricity flow management within a simulated Renewable Energy Valley. The aim is to achieve high self-sufficiency and low emissions, which is in line with the objectives formalized in the REFORMERS proposal [10]. Consequently, central to this work is the research question: *"Does incorporating a learned world model (dyna-style) significantly improve multi-agent coordination and CO_2 emission minimization in energy hub management compared to purely model-free MARL?"*

Furthermore, to simplify the evaluation of algorithms within a consistent context, this work aims to establish an environment framework that enables reliable comparative analysis of MADRL energy flow optimization algorithms. Therefore, this paper's novel scientific contributions are as follows:

1. We release MARLOES, a flexible REV simulation interface for evaluating MARL energy flow management algorithms in a consistent context.
2. We design DynaSAC, a Dyna-style SAC algorithm with a hierarchical world model.
3. We provide a systematic benchmark of model-based, model-free, and heuristic controllers on a realistic REV scenario aligned with the REFORMERS pilot.

The design and implementation of the first contribution, the MARLOES framework, was done in collaboration with Matthias Drijfhout. All further algorithm development, experimental benchmarking, and analysis presented in this thesis, are the sole work of the author.

The remainder of this study is structured as follows. Firstly, definitions are established in section 4. Consequently, the energy dispatch problem is contextualized into this formal framework in section 5. Given the formalization, the approach is detailed in section 6. We outline the outcomes in section 7 and discuss their interpretation, impact and potential future work in section 8.

4 Preliminaries

In order to ensure clarity and consistency throughout the paper, as well as ensure definitions are provided before the theoretical formalization, we define general key concepts and notations below.

4.1 Partially Observable Stochastic Games (POSG)

First of all, we introduce the most general environment definition for multi-agent reinforcement learning problems: Partially Observable Stochastic Games (POSGs) [28]. Each multi-agent reinforcement learning problem can be formulated as a POSG, which is given by the tuple $(n, \mathcal{S}, \mathcal{A}, \mathcal{Z}, p, r, o)$, where:

- n is the number of agents,
- $\mathcal{S}$ is the state space, which is continuous and therefore infinite,
- $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the joint action space and $\mathcal{A}_i$ the continuous infinite action set of agent i,
- $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$ is the joint observation space and $\mathcal{Z}_i$ the infinite observation set of agent i,
- $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function, where $\Delta(\mathcal{S})$ denotes the set of all probability densities over the set $\mathcal{S}$.
- $r = (r_1, \ldots, r_n)$, where $r_i : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the reward function for agent i,
- $o = (o_1, \ldots, o_n)$, where $o_i : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{Z}_i)$ is the observation function for agent i, where $\Delta(\mathcal{Z}_i)$ denotes the set of all probability densities over the set $\mathcal{Z}_i$

At each time step t, the environment is in state $s_t \in \mathcal{S}$. Then all agents (simultaneously) select an action, which together form the joint action $a_t = (a_t^1, \ldots, a_t^n) \in \mathcal{A}$. The environment then transitions to the next state s_{t+1} according to the transition function $p(s_{t+1} \mid s_t, a_t)$. As a result, each agent i receives a reward $r^i(s_t, a_t)$, and a new observation $z_{t+1}^i \sim o_i(s_{t+1}, a_t)$.

Formalizing the environment according to this structure enables capturing the full complexity of multi-agent interactions in environments that can be uncertain. As a result, this forms a solid theoretical foundation to analyze reinforcement learning algorithms.

4.2 Multiagent MDP (MMDP)

A subclass within POSGs are Multiagent Markov Decision Processes (MMDPs) [29], in which the environment is fully observable to all agents, and they work together in a cooperative settings towards a common goal with a shared reward. An MMDP can be described by the tuple $(n, \mathcal{S}, \mathcal{A}, p, r)$, where:

- n is the number of agents,
- $\mathcal{S}$ is the infinite state space,
- $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the joint action space and $\mathcal{A}_i$ the continuous infinite action set of agent i,
- $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function, where $\Delta(\mathcal{S})$ denotes the set of all probability densities over the set $\mathcal{S}$.
- $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the shared reward function.

For each time step t, all agents observe the environment state $s_t \in \mathcal{S}$. After selecting joint action $a_t = (a_t^1, \ldots, a_t^n) \in \mathcal{A}$, they receive the reward $r(s_t, a_t)$, and the environment transitions to the next state s_{t+1}, which happens according to $p(s_{t+1} \mid s_t, a_t)$.

Under the assumption of full observability, as in this work, adopting the MMDP framework is beneficial as it reduces the complexity associated with partial observability while at the same time retaining sufficient power to model the dynamics of a fully observable cooperative system.

4.3 Training and Execution Paradigms

The approach to training and executing agents on the environment, critically affects performance and scalability. There are several such paradigms within MARL that define the centralization or decentralization of both learning and action selection within an approach, which affect its achievable coordination and scalability. Below we detail two paradigms relevant to this study.

Centralized Training for Centralized Execution (CTCE) In the standard CTCE paradigm, both training and execution are centralized. This results in a single central controller (or "puppeteer" agent) that learns and makes the decisions [28]. This central controller observes a global state $\mathcal{S}$. During training the central controller learns a global policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which assigns a probability distribution $\pi(\cdot \mid s)$ over the joint action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ to each global state $s \in \mathcal{S}$. This learned policy is then used to select joint actions based on the observed global state during execution time. The use of CTCE is generally most effective for problems that require complete coordination and global optimization, as it ensures that all agents' actions are synchronized towards a common objective. However, it scales poorly with the number of agents, as the size of the joint state and action spaces increases exponentially [28, 30].

Sequential Centralized Training for Decentralized Execution (sCTDE) The sCTDE paradigm, based on CTDE ([30–32]), also includes centralized training, but improves stability by sequential policy updates. During such an update, the policy of a single agent is updated at a time, while the policies of all other agents are kept fixed, making the environment (temporarily) stationary. Training, thus, is a sequence of single-agent updates. For each agent i, a policy $\pi_i : \mathcal{S} \to \mathcal{A}_i$ is learned, mapping the global state $\mathcal{S}$ to the agent's individual action space $\mathcal{A}_i$. After training, execution is decentralized: each agents acts using its individual policy. The sequential decomposition still allows agents to learn coordinated strategies, while reducing the complexity increase with scalability. Additionally, it specifically addresses the non-stationarity issue that often arises in MARL problems: by breaking down the training process into sequential stages where agents are trained one at a time whilst the policies of the other agents stay fixed, the learning process is made stationary for the current agent.

4.4 Soft Actor–Critic (SAC)

Having defined the environment and paradigm, below we outline the two main algorithmic building blocks of this study: SAC and dyna-style model-enhanced planning. The SAC (Soft Actor-Critic)-algorithm is a state-of-the-art, model-free reinforcement learning framework that is especially well suited for continuous action spaces [20]. It is off-policy, meaning it learns from data collected from past policies, which results in a high sample efficiency. The idea behind the SAC framework is to succeed at a task while acting as randomly as possible. High entropy, a measure of randomness, is encouraged in its policy by adding an explicit term, leading to the following objective:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} [r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{1}$$

where:

- π is the policy, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which is a mapping from states to probability densities over actions: for any state s_t, policy $\pi(a_t|s_t)$ gives the density of selecting action a_t.
- ρ_π is the distribution over possible state-action pairs when the agent follows policy π (in the environment). Formally, for any set $B \subseteq \mathcal{S} \times \mathcal{A}$,

$$\rho_\pi(B) = \mathbb{P}\big[(s_t,a_t) \in B \mid a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t,a_t)\big]. \tag{2}$$

- $r(s_t,a_t)$ is the reward the agent receives for taking action a_t in state s_t at time t.
- $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy of the policy at state s_t, given by:

$$\mathcal{H}(\pi(\cdot|s_t)) = -\int \pi(a_t|s_t) \log \pi(a_t|s_t)\, da_t \tag{3}$$

  for continuous actions. This term encourages exploration.
- α is the temperature parameter which controls the trade-off between reward maximization and entropy maximization. Higher α means acting more randomly and vice versa.
- The expectation $\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}$ is taken over all the states and actions the agent might encounter while following the policy.
- The sum over T aggregates the rewards and entropy bonuses over the full trajectory.

The entropy bonus helps the policy avoid converging to suboptimal, deterministic behaviors, which makes SAC more robust to local minima.

Extending SAC to the multi-agent domain differs based on the paradigm that is employed. Under CTCE the single central controller can be treated exactly as a large single-agent SAC whose action vector is an accumulation of all n individual actions. Architecturally, that only results in a wider final layer in the actor network. Under CTDE/sCTDE, the critic ($Q_\theta(s,a)$) stays shared, but the actor is split into separate heads: $\{\pi_{\phi_i}(a_i|s)\}_{i=1}^{n}$, which each produce their individual component of the joint action. At a higher level, this architecture mirrors the CTDE framework that is used in Pu *et al.* [33], which also combines a centralized critic with decentralized actors. A similar approach to off-policy learning is further detailed in Zhang *et al.* [30].

4.5 Dyna Framework

The dyna framework supplements real experiences with synthetic rollouts from a learned world model [26]. Its ‘best-of-both-worlds’ hybrid approach, dynamically combines model-free and model-based RL. As a result, each training cycle consists of:

1. Updating the world model using real transitions.

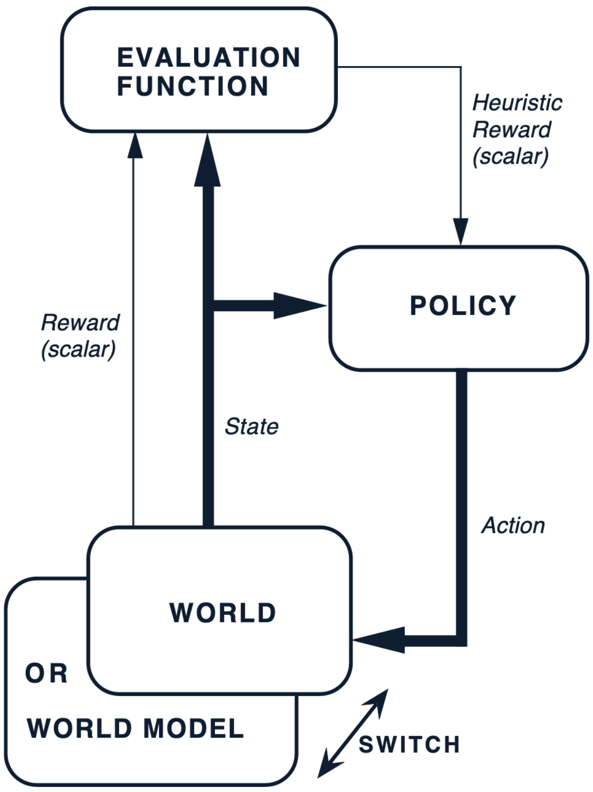


Figure 1: The original design of Dyna style algorithms [26].

2. Generating *k*-step synthetic rollouts starting from real states.
3. Updating the model-free agent according to its own update rule with a mix of real and synthetic experiences sampled from a replay buffer, with ratio ρ, where $\rho = 1$ indicates purely real experience sampling.

This loop, visualized in Figure 1 enables improved data efficiency and a more efficient form of exploration.

The adaptation of dyna to a multi-agent system does not have large consequences for the CTCE and CTDE/sCTDE paradigms. Since both paradigms rely on centralized training, a single, centralized world model suffices.

4.6 Notation summary

Table 1 outlines the notations and descriptions that are relevant to this study.

Symbol	**Description**
n	Number of agents
$\mathcal{S}$	State space
$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$	Joint action space; $\mathcal{A}_i$ for agent i
$\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$	Joint observation space; $\mathcal{Z}_i$ for agent i
$s_t \in \mathcal{S}$	Global state at time t
s_t^i	Local state of asset i at time t
$a_t = (a_t^1, \ldots, a_t^n)$	Joint action at time t
a_t^i	Action taken by agent i at time t
z_t^i	Observation for agent i at time t
$p(s_{t+1} \| s_t, a_t)$	Transition probability function
$o_i(z_{t+1}^i \mid s_{t+1}, a_t)$	Observation function: probability of agent i's observation at t given next state and joint action
r_t	Shared (global) reward at time t
π, π_i	Policy (joint, individual)
T	Episode length/horizon
$V^\pi(s)$	Value function under policy π for global state s
$Q^\pi(s, a)$	State-action value function under policy π
α	SAC entropy temperature parameter
ρ	Dyna real/synthetic sample ratio
γ	Discount factor for future rewards ($0 < \gamma \leq 1$)

Table 1: Summary of relevant notations used in this study.

5 Problem Formulation

With the definitions and overarching objectives established, we now formalize the Energy Dispatch Problem in a theoretical framework. This explicit formulation will consists of transitioning the informal definition of our problem into a formal basis on which the proposed approach is built.

Note that, in this work “asset” refers to one of the (renewable) assets that is capable of producing/intaking energy within our REV, whereas an agent is a reinforcement learning, autonomous decision-making agent. In our multi-agent formulation, each flexible asset is associated with an agent responsible for its control.

5.1 Informal Problem Statement

The Renewable Energy Valley (REV) we aim to optimize consists of several production and consumption assets. Each asset has the potential to create an energy “flow” by producing or consuming a certain amount of kW at each time-step. Our focus is on those assets that are flexible in their consumption and/or production: assets qualified for agents that can act as autonomous decision makers. As in Figure 2, the REV can be visualized as a graph where each node represents an asset, and the edges represent the potential energy flows between them, which are constrained by the capacity of the connecting power network cables. For each time-step, the objective is to decide how much power to produce or consume, ensuring that the net power produced and consumed is always balanced, as that is a fundamental requirement for any energy system. Moreover, the ultimate goal is to withdraw as little power as possible from the national grid, aiming to emit as little CO_2 as possible and achieve a high-level of self-sufficiency [10]. Therefore, we aim to resolve as much of our power demand and supply within the REV itself.

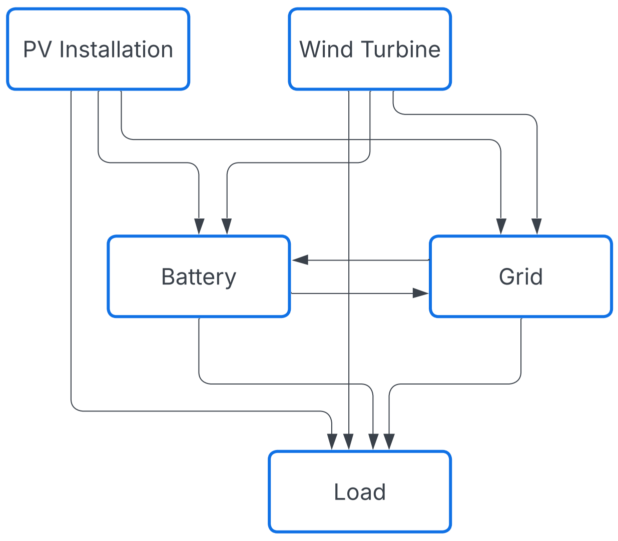


Figure 2: Abstracted graph form of an example REV, the edges denote the flows between the assets.

5.2 Formal Problem Statement

We formalize the EDP problem by the Partially Observable Stochastic Game (POSG) structure as defined in subsection 4.1. Below we will divide it into its most relevant POSG components and provide detailed explanations for its states, observations, actions, and rewards.

States Let s_t^i be the state of asset i at time t. This state always includes the observed power p_t^i (which is negative for consumption and positive for production). Depending on the asset type, s_t^i can also include additional attributes such as available power, state of charge, or other time-dependent information. This modular representation enables the framework to handle the diverse asset types present in REVs. Full details for each asset's states as formulated in this framework are provided in Appendix A and Appendix B.

Observations In this work, we assume full observability of the system, justified by the existence of a single controlling entity that would integrate the information from all assets to steer the assets in an REV in practice, serving as the responsible party for the system performance. Therefore, the controller has access to global information about the entire energy system. Accordingly, each agent i at time-step t observes not only its own asset's state but also the states of all other assets, along with any additional global contextual information. As a result the global state $s_t \in \mathcal{S}$, observed by each agent is given by:

$$s_t = \{s_t^1, s_t^2, \ldots, s_t^n, g_t\}, \tag{4}$$

where:

- s_t^j: The state of asset j at time t, for $j \in \{1, 2, \ldots, n\}$.
- g_t: Global contextual information at time t, such as the date and time.

Actions Actions are defined as adjustments to an asset's power production or consumption, and are constrained by the asset's physical capabilities. Formally, each agent i at time-step t selects an action $a_t^i \in \mathcal{A}^i$, where

$$\mathcal{A}^i = \begin{cases} [-1, 1], & \text{if the asset is flexible in consumption and production (e.g. batteries),} \\ [0, 1], & \text{if the asset is production-only and flexible (e.g., solar panels),} \end{cases} \tag{5}$$

and

- $a_t^i = 1$ corresponds to the asset producing its maximum potential power $[p_{\text{potential},t}^i]^+$, constrained by its maximum power output $p_{i,\text{max out}}$.
- $a_t^i = -1$ corresponds to the asset consuming its maximum potential power $[p_{\text{potential},t}^i]^-$, constrained by its maximum power input $p_{i,\text{max in}}$.
- Any values that are intermediate ($a_t^i \in [-1, 1]$) represent scaled adjustments of the power output, where 0 is the boundary between production and consumption, which itself does not lead to any production or consumption.

The joint action space is then given by:

$$\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n. \tag{6}$$

Transition function We treat the environment as an unknown, stationary Markov kernel, which describes how the system moves from one state to the other by a probability density over next possible states, given (only) the current state and action: $P(s'|s,a)$. Since this Markov kernel is implemented by the simulator, the environment can therefore be regarded as a Markov Decision Process (MDP). For conciseness, we deliberately omit its theoretical dynamics here.

Rewards Given the goals as formalized in the REFORMERS proposal [10], the reward function r_t should be designed to penalize CO_2 emissions, as well as maximize self-sufficiency. Coincidentally, minimizing CO_2 emissions and maximizing self-sufficiency both lead to the same goal: minimizing take-off from the grid, as this is the CO_2 emitting factor in an Energy Valley with renewable assets, and also the definition of maximizing self-sufficiency. The base reward r_b at timestep t can therefore be formalized as:

$$r_b = -\frac{p_t^{\text{grid}}}{d_{\max}}, \tag{7}$$

where p_t^{grid} is the grid power supplied at t in kW, normalized by the maximum system demand $d_{\max}$ in kW, to limit the reward to a reasonable range regardless of the size of the REV. However, optimal behavior of fully flexible assets can be complex to learn, due to their high degree of flexibility and the resulting large action space, including both supply and consumption. Therefore, a storage incentive r_i was introduced to encourage desirable behavior:

$$r_i = -\frac{|a_t^{\text{stor}} + \Delta_t|}{p_{\max}^{\text{stor}}}, \tag{8}$$

where:

- a_t^{stor} is the net battery action at t in kW
- $\Delta_t = s_t^{\text{avail}} - d_t$ is the production surplus in kW, with s_t^{avail} the available supply and d_t the total demand at time t in kW
- $p_{\max}^{\text{stor}}$ is the normalizing factor given by the total maximum system storage power in kW.

This incentive stimulates storage assets to capture any production surplus and try to fill any production deficit in order to emit minimal CO_2 emissions. The use of available (rather than actual) supply in the production surplus serves to decouple the incentive from the actions of other flexible supply assets. This addresses a potential credit assignment problem (accurately attributing global rewards to individual agents' actions [34]), where, if actual supply were used, the penalty could also be reduced by curtailing flexible supply, which is opposite to the purpose of the incentive of rewarding proper storage use.

As a result, the complete reward at timestep t is formulated as follows:

$$r_t = \kappa[r_b + r_i], \tag{9}$$

where the reward is scaled by a real-valued factor κ as parameter that can be tuned (later specified in Appendix C).

5.2.1 Multiagent Markov Decision Process (MMDP)

Given that the reward function is shared among agents and every agent has access to global observations of the system, the energy flow optimization problem is cooperative and fully observable. These characteristics allow us to narrow down our general POSG framework into a more specialized subclass, namely into a Multiagent Markov Decision Process (MMDP) [29] (subsection 4.2), since the collaborative cooperation towards a shared objective inherent to the MMDP framework matches this approach to the EDP in REV exactly.

5.3 Proposed Algorithmic Paradigms

Building on the cooperative, fully observable MMDP formulation, there are two main paradigms we will consider for this study: a Centralized Training for Centralized Execution (CTCE) framework, as well as a sequential Centralized Training for Decentralized Execution (sCTDE) approach (see subsection 4.3).

CTCE offers an effective framework for scenarios that require complete coordination. By leveraging the global state, CTCE can synchronize strategies toward a common objective, making it a relevant approach for our cooperative fully observable MMDP [28], where policy alignment is essential for success.
However, a main limitation of CTCE is its poor scalability [28]. Consequently, we also consider the sCTDE paradigm, as outlined in subsection 4.3. Its sequential approach allows agents to coordinate strategies while reducing the computational complexity that comes with joint policy optimization in larger-scale systems. Given their respective strengths, both CTCE and sCTDE are therefore included in this study. The inclusion of both paradigms allows us to explore the trade-offs between coordination and scalability within our EDP setting.

6 Methodology

This section covers the practical aspects of our work. Below, we will outline our methodology in the two parts that correspond to the objectives of this study: (1) the design of the simulation environment, including its high-level architecture, assumptions, and limitations, and (2) the implementation of the proposed algorithm alongside the experimental setup that was used to evaluate its performance.

6.1 Simulation Environment Architecture

6.1.1 High-Level Architecture

Overall, the entire simulation environment was set-up to be object-oriented and modular, in order to retain a high-level of flexibility for alternative approaches. This framework was developed in collaboration with Matthias Drijfhout and is structured around the proprietary SIMON package developed by Repowered. It consists of two main components:

1. SIMON: a directed-graph-based simulation package that models and solves energy flows.[3]
2. MARLOES: a multi-agent reinforcement learning (MARL) environment that builds on SIMON's functionalities, providing control over each energy asset to RL agents.

6.1.2 SIMON: Core Simulation Engine

SIMON is a simulation package used to model energy flows within an energy system. The energy system is modeled as a directed graph where its nodes correspond to energy assets and its links denote the energy flow between them. These flows do not represent the physical movement of electricity at the particle level but instead serve as an administrative representation of the energy flows, and showcase which amount of produced energy is consumed somewhere else within the system. This abstraction allows for a straightforward interpretation and smooth visualization of an energy system, such as the example shown in Figure 3.
The simulation can be run by iteratively solving the distribution of the flows within the energy system for each time-step, while taking into account each asset's operational constraints (e.g., produced power, maximum intake power, etc.). The solving process consists of determining the flow constraints between asset connections for each asset, while prioritizing "target" assets according to a manually defined priority ranking. After considering both the constraints of the target and source asset, the solving algorithm then allocates energy flows within these boundaries. After iterating over each edge, the system production and consumption must balance to finally commit the flows, which consists of propagating the computed flows through the entire system and updating the asset states accordingly. This whole process is visualized in Figure 4.

Asset states, then, are a form of time-dependent information on the assets constraints, including its potential to supply or consume energy, its projected future potential (e.g. state of charge), and other characteristics. On top of this, assets may be assigned a setpoint. This setpoint is an additional constraint on an asset set by an external party, serving as an indication for which time the asset should preferably produce or consume a certain amount of energy. This approach to energy system management utilizing setpoints is a common way to "steer" assets in practice. Formally, the whole process can be described as follows.

[3]This package is proprietary, the code is therefore not open-source.

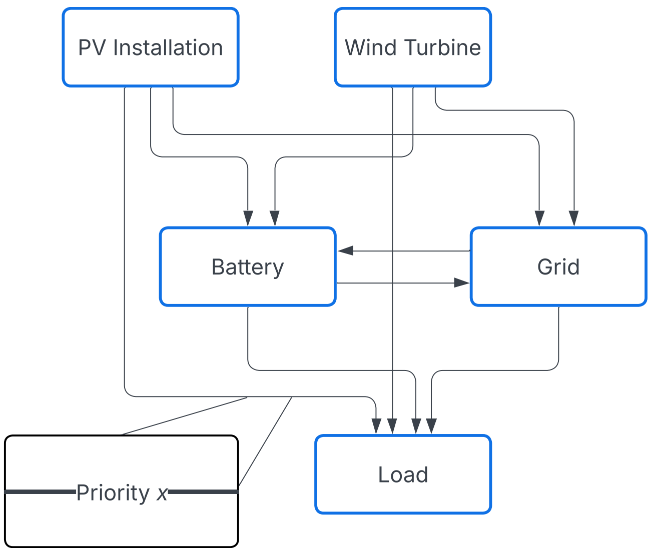


Figure 3: Adaptation of Figure 2, which visualizes how an energy system is abstracted in SIMON. The edges each are assigned their own priority, which can be any real number.

Let the energy system be a directed graph $G = (\mathcal{V}, \mathcal{E})$. For each asset $i \in \mathcal{V}$, $p^i_{\text{max out, t}}$ forms the maximum power out, $p^i_{\text{max in, t}}$ the maximum power in, $c^i_{\text{in},t}$ the residual import capacity, and $c^i_{\text{out},t}$ the residual export capacity, all at timestep t. If an asset is assigned a setpoint $s_{i,t}$, the solver simply replaces those capacities with those assigned by the setpoint if the setpoints is lower:

$$c^i_{\text{out},t} = \min\bigl(c^i_{\text{out},t}, \max(0, s_{i,t})\bigr), \qquad c^i_{\text{in},t} = \min\bigl(c^i_{\text{in},t}, \max(0, -s_{i,t})\bigr). \tag{10}$$

Each edge $f^{ij}_t, i \to j$ then, must respect the constraints:

$$\sum_{j:(i,j)\in\mathcal{E}} f^{ij}_t \le p^i_{\text{max out, t}}, \qquad \sum_{j:(j,i)\in\mathcal{E}} f^{ji}_t \le p^i_{\text{max in, t}}. \tag{11}$$

The edges of each asset are then sorted by descending priority p_{ij} and we assign each edge (i, j) in that order:

$$f^{ij}_t = \min\{c^i_{\text{out},t}, c^j_{\text{in},t}\} \qquad c^i_{\text{out},t} \leftarrow c^i_{\text{out},t} - f^{ij}_t,\ c^j_{\text{in},t} \leftarrow c^j_{\text{in},t} - f^{ij}_t. \tag{12}$$

As a result, a single pass maximizes all edges sequentially in priority order, which the simulator commits and propagates if every asset accepts the flow. Any asset can softly reject a flow, e.g. in case a flow is not possible but not necessary, which re-runs the allocation until every asset accepts or a loop limit is hit. In case any critical failures occur, e.g. not enough energy is allocated for a non-downward-dispatchable consumer, the simulator exits and reports failure to resolve.

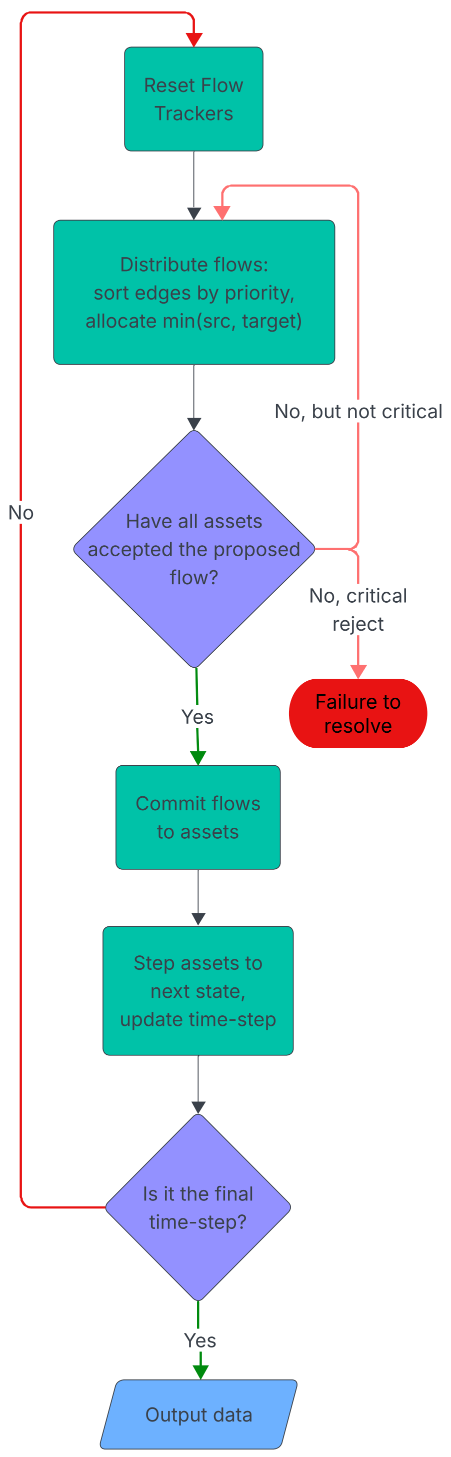

Reset Flow Trackers
Distribute flows: sort edges by priority, allocate min(src, target)
No, but not critical
Have all assets accepted the proposed flow?
No
No, critical reject
Failure to resolve
Yes
Commit flows to assets
Step assets to next state, update time-step
Is it the final time-step?
Yes
Output data

Figure 4: A flow diagram demonstrating the solving process in SIMON. Min(src, target) chooses the largest flow that satisfies both sides simultaneously: no more than either can consume/produce.

6.1.3 MARLOES: Multi-Agent RL Extension

Building upon SIMON, we developed the *Multi-Agent Reinforcement Learning for Optimal Energy Solutions (MARLOES)* framework in this study, which provides a structured environment for training and optimizing reinforcement learning agents in (multi-agent) energy systems.

In MARLOES, each asset modeled in SIMON is interacted with through an asset handler. The asset handler is responsible for initializing the asset within the SIMON framework and allows control over flexible assets by imposing passed setpoint constraints. Furthermore, the asset handler's state extends the asset's state, enabling the incorporation of additional parameters such as forecasted power demand or supply.

A simulation within the MARLOES framework progresses in discrete steps. During each step, setpoints are applied to each flexible asset, the graph is solved and the resulting energy flows are committed. Subsequently, the framework computes the manually user-defined feedback metrics, saves those, and then returns observations, rewards, termination conditions, and additional information for each asset. The observation consists of all asset states and the global context, which is comprised of the net forecasted power across all assets and a normalized representation of the current time using cyclical encoding[4].

The actions can be passed to the environment by any controlling algorithm. The framework provides a base algorithm skeleton, which as a result of MARLOES' modular nature can adhere to any centralized or decentralized paradigm in order to either form independent or joint actions. Figure 5 illustrates the simplified overall system flow, showing how the handlers interact with SIMON's solver, which updates asset states and returns feedback.

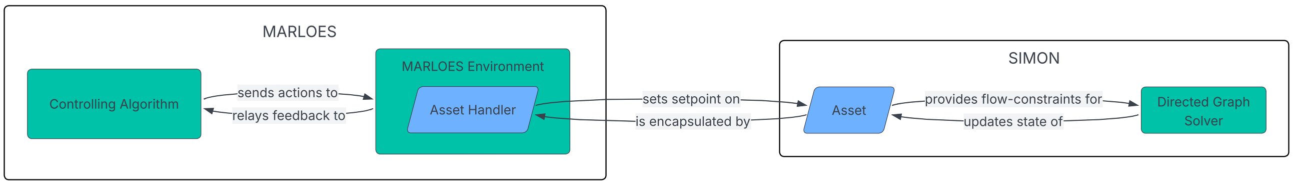


Figure 5: A diagram showcasing the interaction between SIMON and MARLOES.

6.1.4 Asset Handler Types and Configurations

Having established the high-level overview of the MARLOES framework, the following few paragraphs will go into detail about certain assumptions, design choices and asset configurations within the framework.

One of the main components of the framework are all the energy asset handlers. All energy assets (batteries, demands, solar panels, wind turbines, and electrolysers) are instantiated by the handlers in MARLOES. They share basic functionalities, such as keeping track of time-dependent attributes within their states, but differ in terms of default parameters, degradation modeling, and action spaces.

[4]This encoding maps periodic variables (in this case each time component: month, day, hour & minute) onto both a sine and cosine component to preserve their cyclical nature, which makes it easier for machine learning models to learn temporal patterns.

The descriptions below outline the key distinctions, with additional technical details provided in the appendices. Furthermore, in Table 2, a quick outline is given of the main characteristics of each asset handler type.

Battery Handler The battery represents a typical energy storage node. Its handler's configuration limits the maximum power in and out, round-trip efficiency (default to 85%), and ramp rates. It has a continuous action space spanning [-1, 1], which allows for a shift between charging and discharging. Battery aging is modeled by cycle-based degradation, where the maximal capacity of the battery degrades linearly with each charge-discharge cycle. A more extensive explanation of the battery's configuration can be found in Appendix B.

Solar and Wind Handlers Asset handlers controlling solar and wind assets provide their assets with time-series data regarding their energy production, which they can curtail (limit) if necessary. The solar production series are orientation-based (East-West or South), while the wind production series corresponds to a typical onshore turbine, each localized around Alkmaar. Both asset handlers are furthermore equipped with power forecast data, which are available over a predefined future horizon. Further details are provided in Appendix A.

Demand Handler A demand asset models end-use energy consumption. It operates without set-points (i.e., it cannot decide to use more or less energy), which reflects how most real-world demands lack direct flexibility. Therefore the asset handler is only concerned with initializing the asset and providing it with a consumption profile and its forecast. By default a profile of the energy use of a dairy farm (provided by Repowered) is included. See Appendix A for more details.

Electrolyser Handler Though the electrolyser is modeled as a type of battery within the framework (and thus sharing core functionalities), its handler diverges in some aspects. In practice, an electrolyser converts electricity into stored hydrogen and back instead of charging and discharging electrons. This is modeled by treating it as a battery, but applying a conversion factor (defaulting to 33 kWh per kg of hydrogen) to the consumed or produced energy. Furthermore, the asset has a startup delay (to imitate warm-up time) and an hour-based degradation model based on operational hours, to more closely imitate typical electrolyser functionality. A more thorough look into the electrolyser configuration is provided in Appendix B.

Asset type	Produces	Consumes	Is flexible	Forecast present
PV	✓	✗	✓	✓
Wind Turbine	✓	✗	✓	✓
Demand	✗	✓	✗	✓
Battery	✓	✓	✓	✗
Electrolyser	✓	✓	✓	✗

Table 2: Overview of asset (handler) features in MARLOES.

6.1.5 Data Preprocessing: Noise Injection and Dropout

To more closely approximate real-world scenarios and enhance robustness in the learning process, functionality was added to be able to pollute power forecast time-series data with noise and dropout.

Noise Injection In order to ensure variability and mimic forecast error, normally distributed noise can be added to both the energy profiles and the forecast data.

Short-Term Dropouts Randomly selected time-steps can be replaced with zeros, which simulates sensor blips or short-term communication failures.

Longer Dropouts More infrequently, this dropout can be extended to a sequence spanning up to several days, mimicking more substantial technical failures.

The full data handling pipeline of each time-series is described in Appendix D.

6.1.6 Design Limitations and Assumptions

Although this design provides a flexible, object-oriented foundation, certain simplifications and assumptions should be noted:

1. Simplified Asset Modeling: The behavior of the assets has been abstracted to some point and comes with simplified assumptions. For example, the battery degradation function assumes linear deterioration over time, not accounting for any more complex realistic degradation behaviors. Additionally, some physical constraints (such as temperature effects on battery performance) are not modeled explicitly.

2. Data Assumptions: The datasets provided in the framework (see Appendix A for more details) are derived from actual production and consumption data recorded in 2022 and 2023. Since these profiles are inherently time-dependent, they exhibit significant variations and, particularly demand patterns, can be quite specific to their context.

These assumptions should be taken into account in any attempt to draw further conclusions on the outcome of any MARLOES simulation.

6.1.7 PrioFlow: Baseline Control Algorithm

The final component of the MARLOES framework consists of a baseline controlling algorithm (*PrioFlow*), against which other, more advanced, algorithms can be compared in order to evaluate their performance. PrioFlow is designed to be a relatively simple, heuristic-based approach that minimizes emissions and maximizes self-sufficiency in line with the REFORMERS project objectives. It works on the assumption that production assets should maximally produce and storage assets should capture any production surplus and try to fill any production deficit in order to emit minimal CO_2 emissions. The tactic is outlined in pseudocode in Algorithm 1, and is optimal given perfect foresight: it leads to optimal results when supplied with perfect forecasts, but its performance degrades proportionally to forecast inaccuracies. PrioFlow reduces mean final cumulative CO_2 emissions by 88.2% compared to random control over $N = 10$ runs, using the protocol and set-up as described in subsection 6.3. For a detailed insight into PrioFlow and the logic behind it, see Appendix E.

Algorithm 1 PrioFlow: Priority-based control

```
 1: Reset environment to initial state
 2: for each timestep do
 3:     compute net forecasted power over all assets
 4:     for each storage device do
 5:         compute storage device's share of total capacity
 6:         assign action:
 7:             $action \leftarrow -$net forecasted power $\times$ device share / device max power
 8:     end for
 9:     return actions for storage devices to environment (production assets fully produce when assigned no action)
10: end for
```

6.2 Multi-Agent RL Algorithms

6.2.1 Research Hypothesis and Evaluation Goals

Building upon the constructed MARLOES framework, this study aims to perform an in-depth evaluation on whether augmenting the Soft Actor-Critic (SAC) [20] algorithm with a model-based world component, according to a dyna-style integration, leads to superior performance in managing energy flows within our simulated REV environment. The objective being to minimize cumulative CO_2 emissions and, in line with that, maximize self-sufficiency. We expect the explicit world model enhancement will diversify and enrich SAC's transition buffer and therefore yield policies that outperform its model-free counterpart. The experimental framework is designed to not only compare these algorithms' performances, but also benchmark them against the heuristic PrioFlow baseline in order to properly evaluate their potential. Below we outline how each algorithmic component was built and exactly how we went about analyzing its performances and what measures were taken to ensure reproducibility and experimental precision.

6.2.2 Algorithmic Design

Below we detail the algorithmic design choices that were made in the implementation.

SAC We use Soft Actor-Critic (SAC) [20] as our model-free component, due to its state-of-the-art robustness in continuous action spaces. The implementation closely follows the original algorithm, with some modifications as outlined below.

Instead of using a fixed α, as in the original paper [20], we automated entropy adjustment by a learnable-α, identical to the one introduced in SAC's follow-up paper [35]. This implementation uses gradient descent to match the policy's entropy with a target entropy, which is set proportional to the action space dimensionality (i.e., $-\dim(\mathcal{A})$), a common choice which was found to strike a reasonable balance of randomness per action dimension.

Additionally, several other refinements were made to the vanilla SAC to better handle the relatively high level of variance and noisiness that was found to be present in the environment throughout testing. First of all, the value and critic network updates were altered to use Huber loss[5] instead of standard mean squared error, as its reduced sensitivity to outliers led to more stable gradients. Additionally, to even further prevent exploding gradients, gradient-norm clipping (1.0 for critics and value networks, 0.5 for actors[6]) was applied. On top of that, the usual $1:1$ update ratio (one critic update per actor update) was adapted to be variable ($x:1$). This operates equivalently to the delayed policy updates of the state-of-the-art TD3 algorithm [36], as it was found that performing multiple Q-network updates for every policy update could stabilize the Q-estimates and benefit actor learning. Finally, the actions that were output by the actor were passed through a tanh squashing function in order to match our $[-1,1]$ action space. During evaluation, the policy learned during training was converted to a deterministic one by using the mean output directly as action: $\pi : \mathcal{S} \to \mathcal{A}$.

As mentioned before, the sequential Central Training and Decentralized Execution (sCTDE) paradigm offers advantages as opposed to the singular CTCE implementation. Therefore the SAC implementation was also adapted to adhere to the sCTDE framework (further called *MASAC*; Multi-Agent SAC), where each asset is now assigned a dedicated actor network. In this version, critics and value estimators are shared during training, where each actor is sequentially trained while treating the other actors as fixed. During evaluation, however, each agent operates independently. For exact details on this adaptation, see subsection 4.4.

Dyna Augmentation We further build upon SAC by enhancing it with a dyna extension [26]. As outlined in subsection 4.5, this implies a hybrid approach that dynamically combines model-free and model-based RL through a world model.

World Model Architecture In designing the world model component of the dyna architecture, two design paradigms were considered: Markovian models and recurrent (RNN-based) models. The Markovian structure assumes that the future state depends solely on the current state, drastically simplifying the learning process and typically leading to efficient and stable training dynamics [17, 34]. However, purely Markovian approaches can struggle to capture essential temporal dependencies; RNNs (specifically, Gated Recurrent Units; GRUs [37]) are specifically modeled to capture these [17, 34, 38], but can incur extra costs in increased complexity and potential instability because of its vanishing and exploding gradient problems [39].

Our solution is a hybrid, hierarchical model: temporal dependencies are modeled locally and selectively for each agent by using a shallow GRU on power forecasts. The global backbone, on the other hand, remains Markovian for stability, together allowing the model to capture essential temporal relationships, without leading to the full training instabilities and computational overhead.

The complete modular structure, fit to the inherent structure of REVs, is visualized in Figure 6 and further expanded on in Appendix F. Each asset's state is separated and split into its temporal at-

[5]

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{13}$$

where a is the prediction error and δ is a positive threshold parameter (1.0). Leads to reduced sensitivity to outliers, as for large errors the loss is not quadratic, but linear.

[6]Actor, critic and value networks are implemented as fully connected neural networks following the original SAC paper.

tributes and non-temporal ones. The temporal attributes (its power forecast) are fed through a simple one-layered GRU and rejoined with the other aspects into an asset state encoding. All asset state encodings together form the world state encoding together with any global context. The world dynamics are then modeled by combining this encoding with the joint actions in order to predict the next state and the reward through separate heads.

By combining the strengths of both Markovian and recurrent architectures, in combination with the hierarchical and therefore modular nature, this world model provides a robust and computationally efficient solution, specifically suited to modeling the dynamics of an REV.

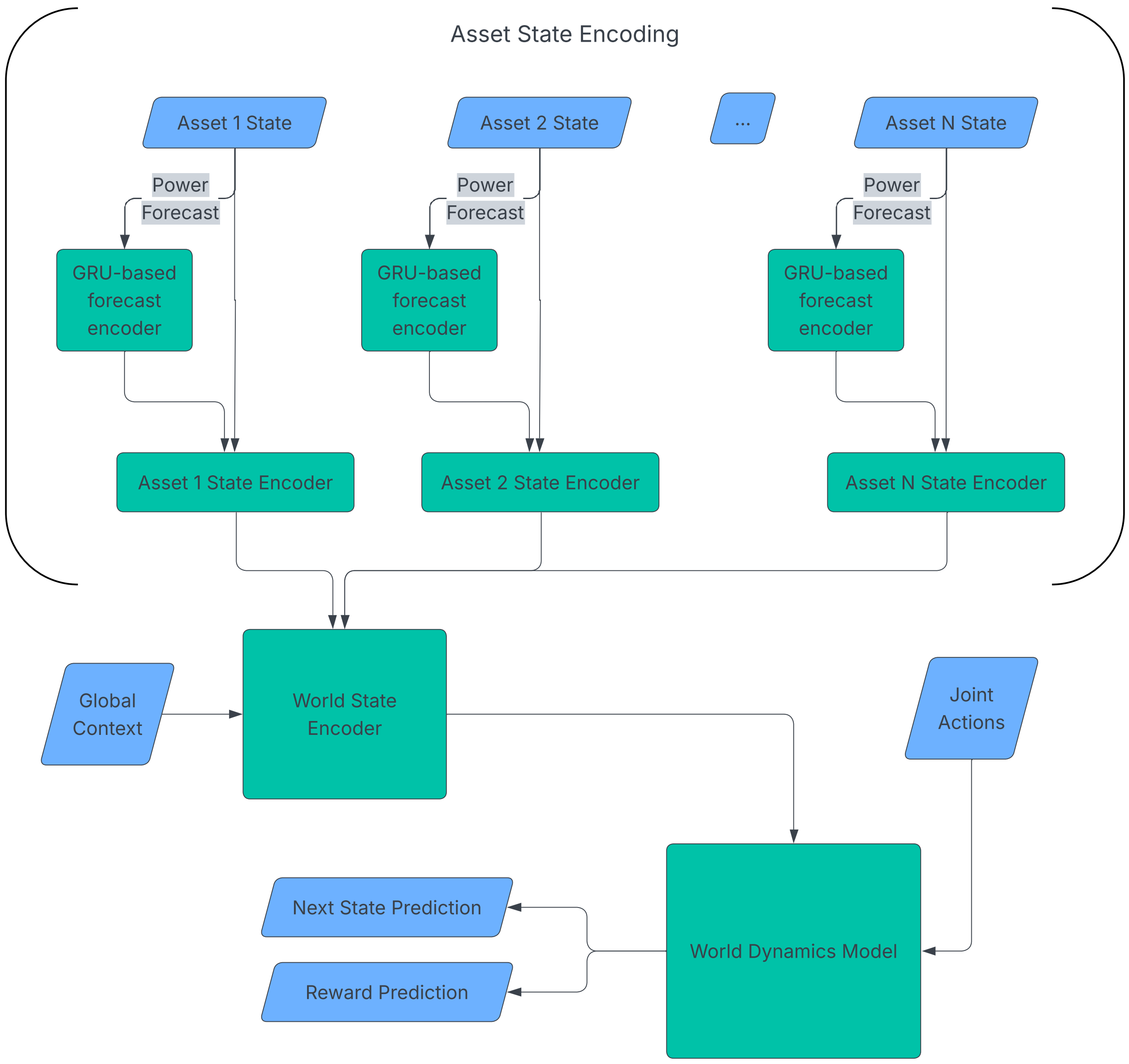


Figure 6: Hierarchical World Model architecture: agent-specific recurrent encoders for temporal attributes (e.g., forecasts), combined with global state and action encoding to predict next states and rewards.

6.2.3 Resulting Algorithms

In summary, this study evaluates the following algorithms, as desribed in detail above:

- PrioFlow: The heuristic baseline controller.
- SAC: Standard Soft Actor-Critic.
- MASAC: Multi-agent extension of SAC (sCTDE paradigm).
- DynaSAC: SAC with a model-based world component (Dyna-style).
- DynaMASAC: Multi-agent Dyna-style SAC.

6.3 Training Protocol and Experimental Setup

6.3.1 Training Details

Having defined our architecture, we now detail the further training procedure. For all experiments, each configuration was trained for 50 000 consecutive time-steps at 1-minute resolution. Each configuration was then also evaluated on 50 000 consecutive time-steps of unseen data. All hyperparameters, unless otherwise specified can be found in Appendix C. The results of each configuration are averaged over 10 runs, and in order to ensure strict comparability, across and within all experiments the same sequence of random seeds ($10 - 19$) is used.

Unless specified otherwise, all experiments were run in a configuration with the following three assets:

- A 3 MWp (both AC and DC), East-West oriented solar park;
- a demand profile based on a dairy farm (scaled up by a factor of 15);
- and a battery with 2000 kWh energy capacity and 1000 kW power rating.

The battery and solar park are flexible, meaning RL agents can control them through setpoints, while the demand is not flexible.
This deliberately simple composition of one supply, one demand, and one flexible storage asset, was chosen to ensure that any effects of learned policies can be directly attributed to each asset type and maximizes interpretability. Furthermore, through preliminary testing we found that this specific configuration provides a good balance between production and consumption, enabling the battery to meaningfully influence system behavior. Thus, this setup was found to be a useful and transparent base for analyzing the MARL algorithms.

All experiments were performed on the Hábrók high-performance computing cluster at the University of Groningen. In general for each run, we allocated CPU cores and RAM via the SLURM job scheduler (10 workers/16 GB by default and 32 workers/32 GB for the hyperparameter search). All jobs used Python 3.11.3. The hyperparameter search was distributed across 10 jobs, with each job running on a separate standard compute node (2 × AMD EPYC 7763 CPUs, 128 cores, 512 GB RAM), whereas the less intensive training jobs were executed on a single compute node.

6.3.2 Hyperparameter Optimization

In order to finetune all hyperparameters present within our models, we employ a single-run random hyperparameter search over 200 trials. The search space includes all major parameters regarding SAC and the world-model enhancement. Full details are provided in Appendix C. The optimal configuration is held fixed for all subsequent experiments. Furthermore, all searched configurations made use of the standard DynaSAC configuration and the optimal parameters are directly applied to SAC, MASAC and DynaMASAC, as we assume these hyperparameters will be largely transferable due to the similar core mechanisms between SAC and MASAC (the shared critic and similar policy learning mechanisms).

6.3.3 Ablation Study: Recurrency

In subsubsection 6.2.2 we hypothesize that, within our hybrid world model, local recurrency is necessary to capture the temporal dependencies of energy forecasts, while the Markovian backbone suffices for global state transitions. In order to test this hypothesis we analyzed the contribution of the specific recurrent architectural element and conducted a targeted ablation study. In order to do so, we removed the local GRU and expanded the input layer of the agent encoder to fit the complete power forecast directly. We compared its performance with the original design to be able to analyze the modules' effects on the world model representation. The ablation is performed only on DynaSAC, for consistency and to isolate the effect of this specific component.

6.3.4 Preliminary Paradigm Comparison: SAC vs. MASAC

Prior to full-scale experimentation, we performed a paradigm comparison to select between the CTCE-adhering SAC and sCTDE structured MASAC counterparts. In order to evaluate and analyze performance differences between these paradigms, we ran a (preliminary) two-fold experiment, consisting of an analysis on performance and robustness, as well as scalability as the number of assets increased (3, 6, and 12 assets[7]). This two-way approach offers both valuable information for potential future research in larger REVs and informed our final choice of paradigm for the performance analysis hereafter.

6.3.5 Performance Analysis

In the main experiments, we compare:

- **SAC/MASAC** (based on preliminary experiment);
- **DynaSAC/DynaMASAC**, at both $\rho = 0.8$ and $\rho = 0.5$ (ρ = real transition sample ratio, where $\rho = 1$ denotes only real transitions), in order to gain detailed insight into the extent to which model-based enhancements affect performance;
- **PrioFlow**: the heuristic baseline.

In order to thoroughly compare main performance of our algorithms, an extra noisy data-quality regime was created. It introduces realistic data imperfections and mimics typical operating conditions, as follows: additive Gaussian noise ($\sigma = 0.01$) is applied to each value in both forecast and power data time series to model sensor or forecast uncertainty; and random missing data in actual power

[7]These configurations were constructed by doubling and quadrupling the base configuration outlined in subsection 6.3.

measurements is simulated both as single-point dropouts ($p_{drop} = 0.005$) and as longer continuous dropout periods ($p_{long} =$ 1e-4) with a maximum length of two days ($T_{\max} = 2880$ minutes).
The modifications are performed as described in Appendix D, and ensure that all models are evaluated under both standard as well as challenging and uncertain data conditions.

6.4 Evaluation Metrics and Statistical Analysis

To thoroughly assess the overall performance of each algorithm, we structure the evaluation around the following four core concepts and associated questions:

1. **Final-performance:** How well is each algorithm able to reduce CO_2 emissions during evaluation?
2. **Uncertainty & variability:** How consistent is the performance of each algorithm (across random seeds)?
3. **Pairwise comparison (statistical robustness):** Are the observed differences between methods significant?
4. **Compute and learning efficiency:** How long does each algorithm training take and how efficient is the learning process?

To answer these questions, we selected fitting metrics for each question, which are outlined below.

6.4.1 Final-performance

In order to assess the final performance, we consider cumulative CO_2 emissions, which are defined as follows:

$$e_{i,t} = \varepsilon_{\text{grid}}\, p_{i,t}, \tag{14}$$

where

- $p_{i,t}$ is the grid electricity production (in kWh) at timestep t of run i,
- $\varepsilon_{\text{grid}}$ is the estimated emission factor (gCO_2eq/kWh) of the Dutch national grid, of which the estimation details are provided in Appendix G.

To summarize this across N runs, we use the mean of $e_{i,t}$ and the 95% confidence interval (CI) given by:

$$\text{CI} = 1.96\,\frac{s}{\sqrt{N}}, \tag{15}$$

where s is the sample standard deviation of the N runs. To give a full picture of performance, the median and IQM are also reported, where the IQM serves as a trimmed mean, given by:

$$\text{IQM} = \frac{1}{N-2k}\sum_{i=k+1}^{N-k} E_{(i)}, \tag{16}$$

where $E_{(i)}$ are the sorted final cumulative emissions values as given in Equation 14 and $k = \lfloor 0.25N \rfloor$.

6.4.2 Uncertainty & Variability

We further quantify relative variability by the coefficient of variation (CV):

$$\text{CV} = \frac{s}{\mu}, \tag{17}$$

where s is the standard deviation and μ the mean over N runs. Supplementary, we measure the uncertainty with a more robust nonparametric bootstrap CI, by resampling each set $\{E_{(i)}\}$ 5000 times, computing the mean each time, and then taking the 2.5th and 97.5th percentiles of the bootstrap means.

6.4.3 Pairwise comparison

As the seeds match between the runs of the methods, we can apply pairwise statistical testing on the final cumulative CO_2 emissions of each method. Before comparing method A with method B, we test normality of the paired differences through the Shapiro-Wilk test. If $p_{SW} < .05$, we apply a paired t-test, otherwise we use the non-parametric Wilcoxon signed-rank test. As a measure of effect size, Cohen's d is reported, a measure of how many standard deviations apart two means lie. On top of that, for increased direct and practical interpretability, we report the Probability of Improvement (PoI):

$$\text{PoI} = \frac{1}{N^2} \sum_{i,j} \mathbf{1}(E_i^A < E_j^B), \tag{18}$$

where E_i^A is the final cumulative emission of algorithm A on seed i, and $\mathbf{1}$ denotes the indicator function (returns 1 if its argument is true and 0 otherwise).
For multi-group comparisons, we first perform ANOVA (or Kruskal–Wallis if non-normal) and only proceed to pairwise tests if $p < 0.05$. We adjust the raw p-values via the Holm–Bonferroni procedure to control the family-wise error rate, which indicates the probability of incorrectly finding a significant result just by chance, when performing multiple tests.

6.4.4 Compute and learning efficiency

We then measure computational efficiency by wall-clock training time in hours and capture learning efficiency by the area under the emissions learning curve (AULC), given by:

$$\text{AULC}_i = \sum_{t=1}^{T} e_{i,t}, \tag{19}$$

where $e_{i,t}$ is the CO_2 emission at timestep t for run i, and T is the total number of timesteps during training.

Collectively, these metrics represent a detailed overview of the performance of each algorithm and will therefore be applied to every experimental condition, except the hyperparameter optimization, unless specified otherwise.

6.5 Threats to Validity

The analysis naturally comes with some threats to the validity of the outcome, this includes:

- The simulation-reality gap: real world stochasticity is not completely modeled and captured by the simulator.
- Transferability of parameters: hyperparameters that are tuned for 3-asset systems might not generalize to other scales and configurations.

- Constant grid CO_2 emission factor: we do not account for seasonal variation in the generation mix of the grid, as that will not be fixed in practice, but differ based on the availability of resources.

These threats contextualize the outcome, and allow us to clearly define the scope in which our analysis takes place: our results hold under these assumptions and extrapolation to any broader context should be done with caution, as is detailed in section 8.

6.6 Reproducibility

To support reproducibility, all code including the code for the MARLOES framework, datasets, a `README.md`, experiment configurations and SLURM scripts are provided at `https://github.com/repowerednl/marloes/tree/DynaSAC`.

7 Results

Following the established approach in order to answer our research question, this section covers the outcomes and analysis of our experiments. Specifically, we evaluate the main hypothesis: *Model-based augmentation (Dyna) improves the performance of SAC/MASAC agents.* The further per-experiment analysis is structured as detailed in subsection 6.4, concluded with a summary of key insights.

7.1 Hyper-parameter Optimization Outcome

7.1.1 Objective

First of all, we performed the randomized hyperparameter search (subsubsection 6.3.2) in order to find the best performing configuration.

7.1.2 Results

The variable hyperparameters of the configuration that achieved the lowest final cumulative CO_2 emissions are outlined in Table 3, more details are provided in Appendix C. Among the best-performing models, some notable patterns emerged, which give some insight into the inner workings of the methods in the simulated REV environment, among which are:

- The highest critic actor update ratio of $3:1$ is the most common, giving some indication that the more frequent critic update refinement is beneficial.
- A batch size of 128 is used in more than half of the best runs, suggesting larger batch sizes are required for stability or performance, as they likely reduce variance in gradient updates.
- Relatively high gamma values of 0.99 and 0.995 are much more common, indicating that long-term returns are important.

All subsequent experiments fix these parameters.

Dyna Parameters		
Parameter	**Best Value**	**Description**
U	18	No. model updates per update step (I_{upd})
k	6	Model rollout length
SAC Parameters		
λ_s	6.38×10^{-6}	Optimizer weight decay
τ	0.02	Target smoothing coefficient
η_π	5.35×10^{-5}	Policy network learning rate
η_Q	8.43×10^{-5}	Q-network learning rate
η_V	5.83×10^{-6}	Value network learning rate
η_α	6.65×10^{-5}	Entropy coefficient learning rate
ε	1.07×10^{-7}	Optimizer epsilon
B	64	Mini-batch size
r	3	Critic to actor update frequency
γ	0.99	Discount factor
d_h	64	Hidden layer size
L	3	Number of hidden layers
World Model Parameters		
η_w	4.81×10^{-3}	World model learning rate
$d_{\text{h}}^{\text{world}}$	128	World model hidden layer size
$d_{\text{h}}^{\text{forecast}}$	8	Forecasting module hidden size

Table 3: Best hyperparameter values found in the 200 trial random hyperparameter sweep, along with parameter descriptions. Further details and symbol definitions are given in Appendix C.

7.2 Ablation Study: Role of Local Recurrency

Below, the objective and results of the ablation study are outlined.

7.2.1 Objective

The objective was to evaluate the necessity and impact of the GRU recurrent layer.

7.2.2 Results

a. Final-performance summary Figure 7 shows how the emission of CO_2 gasses (gCO_2eq) of the two models progressed over the time steps of the evaluation, whereas Table 4 reports final cumulative CO_2 emissions over $N = 10$ runs, summarized by mean $\pm$ 95% confidence interval (CI), median, and interquartile mean (IQM). Table 4 shows that, while both methods perform similarly in terms of median and IQM, the GRU-inclusive model does exhibit slightly lower mean emissions with a narrower confidence interval, which can also be visually derived from Figure 7.

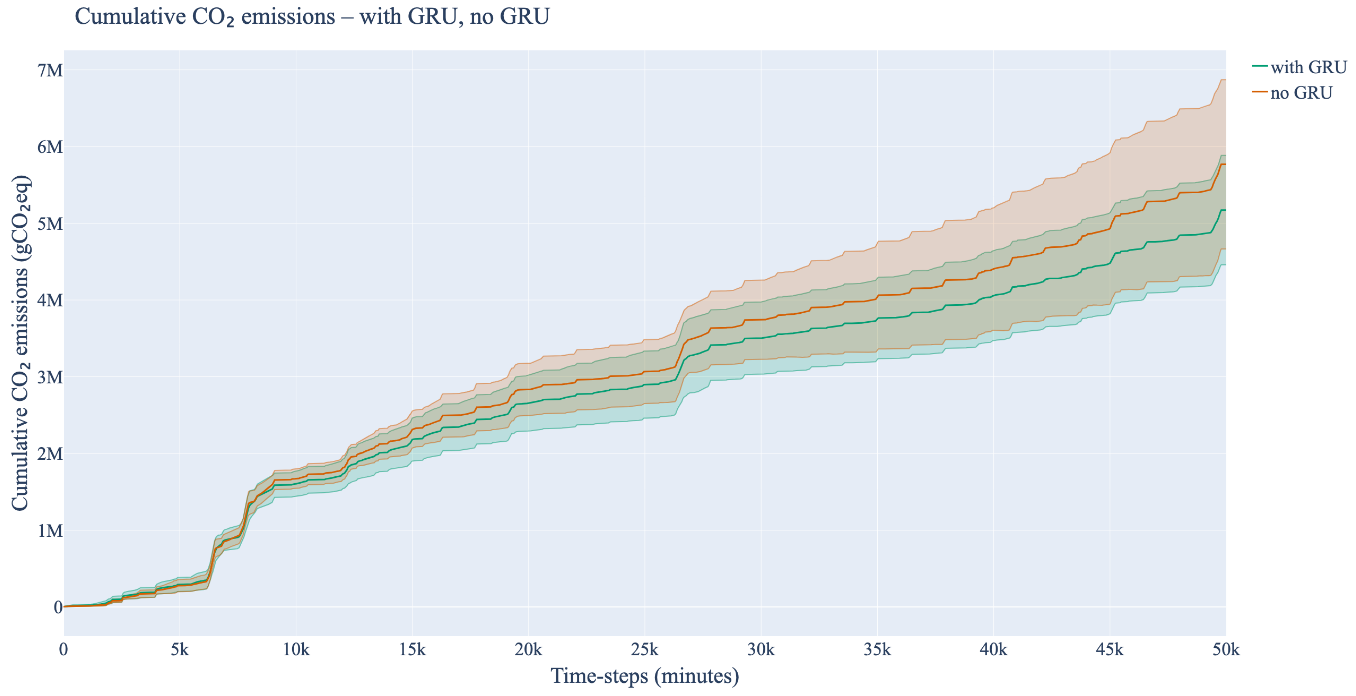


Figure 7: The progression of cumulative CO_2 emissions of the DynaSAC model with and without GRU in the world model, showing means and 95% confidence interval (CI) averaged over 10 runs.

Method	**Mean $\pm$ 95% CI** ($\downarrow$)	**Median** ($\downarrow$)	**IQM** ($\downarrow$)
with GRU	5.17 ± 0.71	5.31	5.27
no GRU	5.77 ± 1.10	5.25	5.34

Table 4: Final cumulative CO_2 emissions (gCO_2eq) for the ablation study ($N = 10$ runs). Results as mean $\pm$ 95% confidence interval (CI), median, and interquartile mean (IQM). All values are in tCO_2eq ($\times 10^6$ gCO_2eq). The arrow in each column header indicates whether higher ($\uparrow$) or lower ($\downarrow$) values are better.

b. Uncertainty & variability We further analyzed the uncertainty and the variability of the final emissions, by means of the coefficient of variation (CV) and bootstrap 95% confidence intervals (see Table 5). The bootstrap confidence interval for the GRU-inclusive model is narrower than that of the no-GRU model, which reflects more consistent outcomes. Similarly, the GRU model also resulted in a lower CV (CV = 0.22) compared to the non-GRU variant (CV = 0.31), which supports the indication of reduced variability across runs for the GRU model. These results suggest that the GRU module does contribute to increased stability and reduced uncertainty in the model's behavior.

Method	CV (↓)	Bootstrap 95% CI Lower	Upper (↓)
with GRU	0.22	4.48	5.85
no GRU	0.31	4.92	6.94

Table 5: Uncertainty and variability metrics for final cumulative CO_2 emissions in the ablation study. CV: coefficient of variation. Bootstrap 95% CIs over $N = 10$ runs. Bootstrap values are in $\times 10^4$ gCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

c. Pairwise comparison After confirming normality through a Shapiro-Wilk test ($p > .05$), a paired t-test revealed no significant difference in performance in regards to cumulative CO_2 emissions between the two models ($p = 0.394$, Cohen's $d = -0.28$). The small effect size, measured by Cohen's d, also confirms that the observed difference is small relative to the variability of the data. Furthermore, the probability of improvement (PoI), was 0.52 in favor of the GRU model, meaning the GRU model outperformed no-GRU in 52% of bootstrap comparison, which indicates minimal improvement.

d. Compute and learning efficiency In Table 6, the computational costs and learning efficiency of the two model variants are compared, by wall-clock time and area under the learning curve (AULC), with means and 95% confidence intervals. The GRU-inclusive model required an average wall-clock time of 1.32 ± 0.02 hr, which is roughly 32% longer than the no-GRU variant's 1.00 ± 0.03 hr: a significant difference, as confirmed by a paired t test ($p = 2.6 \times 10^{-7}$, Cohen's $d = 4.31$), with substantially large effect size. On the other hand, the AULC, an indication of learning efficiency, was near identical between the two models, indicating no meaningful difference in learning performance.

Method	Wall Time (hr, ↓)	AULC (↓)
with GRU	1.32 ± 0.02	$1.78 \times 10^4 \pm 106$
no GRU	1.00 ± 0.03	$1.79 \times 10^4 \pm 83$

Table 6: Computational efficiency (mean wall time in hours ± 95% CI) and area under the learning curve (AULC, mean ± 95% CI) for the ablation study ($N = 10$). The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

7.2.3 Summary

While the GRU-inclusive model shows a lower mean final cumulative CO_2 emission, the statistical tests did not reveal any significant difference in the emissions reduction performance between the two methods. The incorporation of a GRU model was shown to be computationally more expensive, as it led to a significant increase in wall-clock time. Nonetheless, the GRU model does exhibit lower run-to-run variability (CV = 0.22 vs. 0.31) and a narrower bootstrap 95% CI ([4.48–5.85]10^6 vs. [4.92–6.94]10^6 gCO_2eq). As a result of this, and building upon existing literature demonstrating that GRUs can enhance the modeling of temporal aspects ([38]), we choose to retain the GRU in the architecture for successive experiments, despite the lack of statistical significance under these circumstances. We believe larger-scale studies are desirable to more thoroughly evaluate the impact of GRUs on performance.

7.3 Pre-study: Paradigm Selection

7.3.1 Objective

The objective was to identify which paradigm between the CTCE-based DynaSAC and sCTDE-based DynaMASAC is superior in this multi-asset energy management context. The evaluation objective was two-fold:

1. Evaluate the absolute performance of each model in the several asset configurations.
2. Analyze the relative performance change as the number of assets within the REV increased.

7.3.2 Results

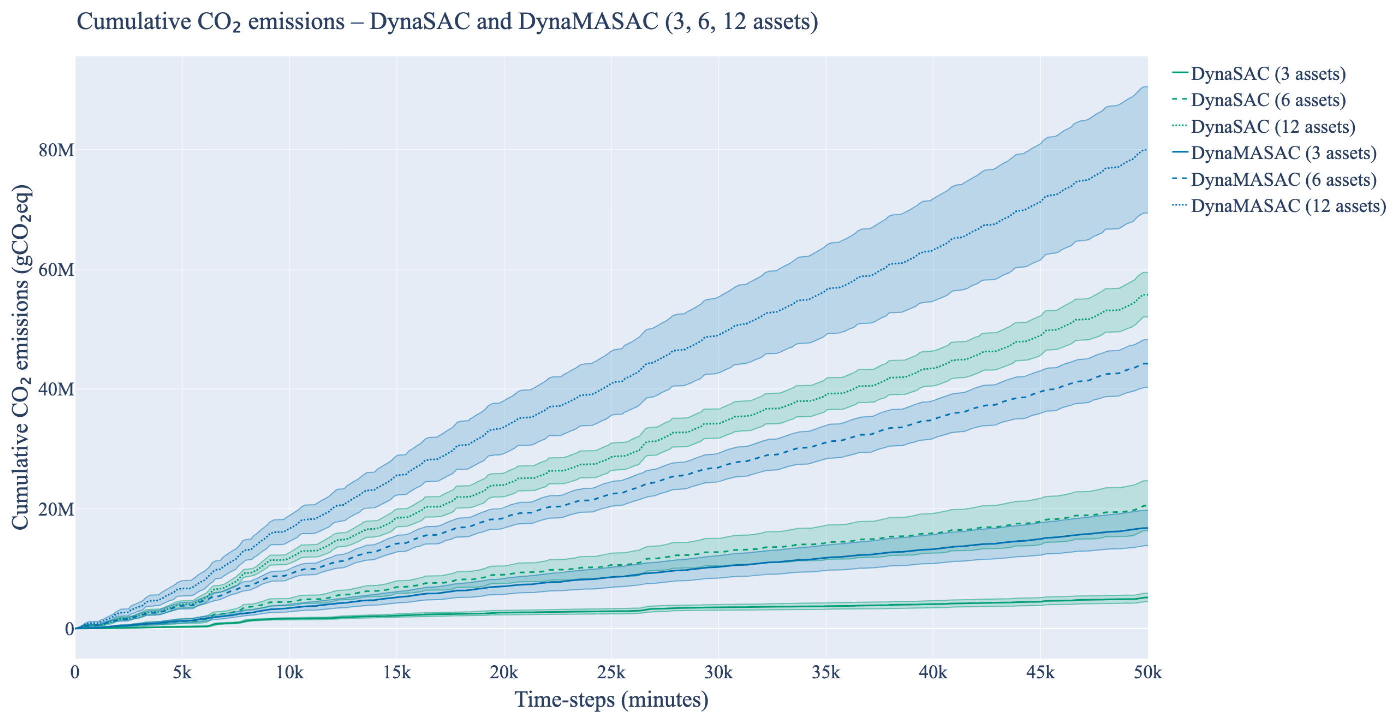


Figure 8: The progression of cumulative CO_2 emissions of both the DynaSAC and DynaMASAC model in a configuration with 3, 6 and 12 energy assets, showing means and 95% confidence interval (CI) averaged over 10 runs.

a. Final-performance summary Figure 8 visualizes the CO_2 emission evolvement as they unfolded during evaluation, whereas Table 7 summarizes the final results. In the standard 3-asset scenario, the DynaSAC paradigm yields substantially lower emissions (5.17×10^6 gCO_2eq), than the sCTDE-based DynaMASAC (1.68×10^7 gCO_2eq), indicating a clear advantage for DynaSAC in smaller-scale settings. As the number of assets increases, this pattern stays present across all summary statistics, with DynaSAC achieving consistently lower emissions than DynaMASAC. In terms of scalability, within the tested asset range, DynaSAC continues to outperform DynaMASAC, though the difference narrows in the largest, 12-asset configuration.

Method	**Mean $\pm$ 95% CI** (↓)	**Median** (↓)	**IQM** (↓)
DynaSAC (3 assets)	5.2 ± 0.7	5.3	5.3
DynaSAC (6 assets)	20.5 ± 4.2	20.8	19.8
DynaSAC (12 assets)	55.7 ± 3.7	54.9	54.8
DynaMASAC (3 assets)	16.8 ± 3.0	16.9	17.0
DynaMASAC (6 assets)	44.2 ± 4.0	43.7	43.6
DynaMASAC (12 assets)	80.0 ± 10.6	78.7	79.7

Table 7: Final cumulative CO_2 emissions (gCO_2eq) for each method and asset group ($N = 10$ runs). Results are reported as mean $\pm$ 95% confidence interval (CI), median, and interquartile mean (IQM). All values are in tCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

b. Uncertainty & variability Table 8 summarizes what the uncertainty metric evaluated to for each method and asset scale. While DynaSAC shows a lower level of variability (lower CV and narrower confidence intervals) at both the smallest and largest asset scales (3 and 12), DynaMASAC performs better in those terms at the intermediate, 6 asset level. Overall, there is no clear, consistent trend which favors one of either methods across all scales with regards to uncertainty.

Method	**CV** (↓)	**Bootstrap 95% CI Lower**	**Upper** (↓)
DynaSAC (3 assets)	0.22	4.5	5.8
DynaSAC (6 assets)	0.33	16.6	24.5
DynaSAC (12 assets)	0.11	52.5	59.4
DynaMASAC (3 assets)	0.28	14.1	19.6
DynaMASAC (6 assets)	0.15	40.6	48.2
DynaMASAC (12 assets)	0.21	69.9	90.1

Table 8: Uncertainty and variability metrics for final cumulative CO_2 emissions, grouped by method and asset size. CV denotes coefficient of variation. Bootstrap 95% CIs are for the mean, over $N = 10$ runs. Bootstrap values are in tCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

c. Pairwise comparison For statistical pairwise comparison, we did two separate analyses, corresponding to the two sub-objectives of this experiment.

First of all we compared final cumulative CO_2 emissions per assets scale through statistical testing. The initial Kruskal-Wallis test, due to the rejection of normality via the Shapiro-Wilk test ($p < .05$), revealed statistically significant difference between the methods' performances ($H = 54.70$, $p = 1.5 \times 10^{-10}$). Therefore, we performed paired tests between both methods at every asset scale. Given that for each individual comparison normality assumptions were confirmed by the Shapiro-Wilk test, we performed paired t-testing adjusted by Holm-Bonferroni correction, as we do multiple comparisons. The results are reported in Table 9, which shows that at every scale DynaSAC significantly outperforms DynaMASAC, with very large effect sizes (Cohen's $d > 1.5$) and probabilities of improvement (PoI) near 1.00, indicating a highly consistent performance advantage under these conditions, only slightly narrowing at the largest (12-asset) scale.

Comparison	**No. Assets**	**Test**	**Holm *p***	**Cohen's *d***	**PoI**
DM vs DS	3	paired *t*	**3.01e-5**	2.65	1.00
DM vs DS	6	paired *t*	**2.69e-7**	4.87	1.00
DM vs DS	12	paired *t*	**0.00078**	1.57	0.92

Table 9: Pairwise statistical comparisons of final cumulative CO_2 emissions for DynaMASAC (DM) and DynaSAC (DS) across asset sizes. p-values are Holm–Bonferroni adjusted; significant results ($p < 0.05$) are shown in bold. Cohen's d and PoI (Probability of Improvement) quantify effect size and likelihood of superior performance, respectively. PoI indicates DynaSAC over DynaMASAC.

In order to assess scalability, we analyzed the relative increase in the mean of cumulative CO_2 emissions as the number of assets increased, within each method. As detailed in Table 10, for DynaSAC, emissions rose by approximately 297% from 3 to 6 assets and by 172% from 6 to 12 assets. DynaMASAC showed a 163% increase from 3 to 6 assets and an 81% increase from 6 to 12 assets. These results reveal that DynaSAC's emissions increase more sharply as the scale increases, whereas DynaMASAC shows a more gradual (though consistently higher) emission profile under the same circumstances, suggesting that DynaMASAC may be less sensitive to increases in system size.

Method	**3 $\rightarrow$ 6 assets**	**6 $\rightarrow$ 12 assets**
DynaSAC	+297%	+172%
DynaMASAC	+163%	+81%

Table 10: Relative growth in mean cumulative CO_2 emissions when the number of assets increases.

d. Compute and learning efficiency As is shown in Table 11, across all asset scales, DynaSAC outperformed DynaMASAC on both wall time and area under the learning curve (AULC). This was confirmed through statistical testing, as all pairwise comparisons resulted in Holm–Bonferroni adjusted p-values < 0.05 and large effect sizes (Cohen's $d > 0.8$), leading to the conclusion that DynaSAC outperforms DynaMASAC consistently and significantly in computational efficiency for each tested asset scale.

Method	**Wall Time (hr)** (↓)	**AULC** (↓)
DynaSAC (3 assets)	1.28 ± 0.04	$1.78 \times 10^4 \pm 106$
DynaSAC (6 assets)	1.83 ± 0.05	$3.55 \times 10^4 \pm 133$
DynaSAC (12 assets)	3.09 ± 0.06	$7.72 \times 10^4 \pm 718$
DynaMASAC (3 assets)	1.34 ± 0.02	$1.95 \times 10^4 \pm 618$
DynaMASAC (6 assets)	2.33 ± 0.03	$4.29 \times 10^4 \pm 1184$
DynaMASAC (12 assets)	4.84 ± 0.05	$8.53 \times 10^4 \pm 1694$

Table 11: Computational efficiency (wall time in hours ± 95% CI) and area under the learning curve (AULC, mean ± 95% CI) for each method and asset group ($N = 10$ runs). The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

7.3.3 Summary

To conclude, DynaSAC showcased superior superior emissions and computational performance compared to DynaMASAC across all tested asset scales. While DynaMASAC showed a relatively smaller proportional increase in emissions with larger asset scales, DynaSAC did consistently achieve lower absolute emissions. Altogether, these results demonstrate that DynaSAC is the preferred approach for this setting, and will thus be used for the main performance comparison hereafter.

7.4 Main Performance Comparison

7.4.1 Objective

The objective of this experiment is two-fold:

1. To determine whether augmenting Soft Actor-Critic (SAC) with a world model leads to improved performance, and to quantify the extent of any such improvement.
2. To assess the performance of both the vanilla SAC and its model-augmented variants relative to the heuristic PrioFlow baseline.

To comprehensively evaluate these objectives, we compare two variants of DynaSAC against the original SAC and the PrioFlow heuristic. The two DynaSAC variants differ in ρ, which controls the sample ratio between synthetic transitions and real environment transitions ($\rho = 1.0$ is real transitions only). Specifically, we investigate $\rho = 0.5$ and $\rho = 0.8$, which enables us to analyze the impact of the varying degree of model-based planning on overall agent performance. As stated in subsubsection 6.3.5, these models were also evaluated under more challenging, noisy data conditions in subsubsection 7.4.4.
In line with the objectives, the following pairwise comparisons will be performed, grouped by objective:

1. **To assess the benefit of world model augmentation:**
   - DynaSAC ($\rho = 0.8$) vs. SAC
   - DynaSAC ($\rho = 0.5$) vs. SAC
   - DynaSAC ($\rho = 0.8$) vs. DynaSAC ($\rho = 0.5$)

2. **To evaluate performance relative to the heuristic baseline:**
   - SAC vs. PrioFlow
   - DynaSAC ($\rho = 0.8$) vs. PrioFlow
   - DynaSAC ($\rho = 0.5$) vs. PrioFlow

7.4.2 Results: Standard Data Conditions

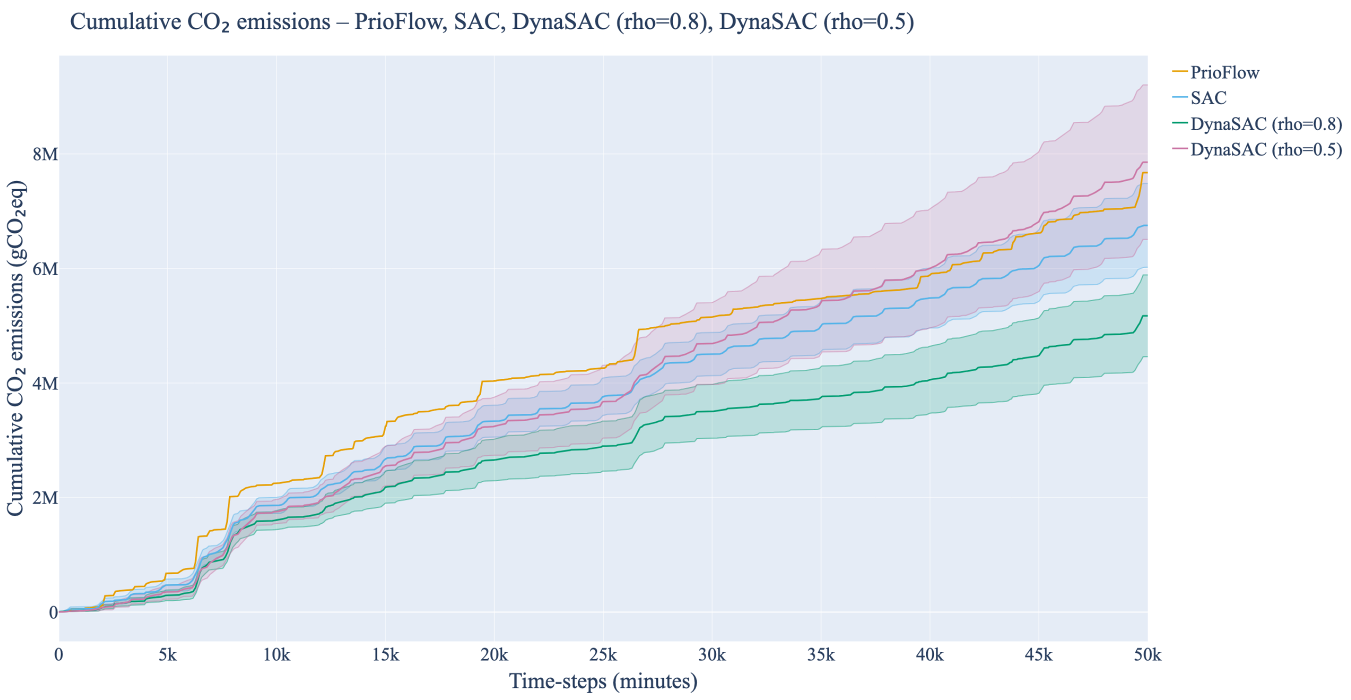


Figure 9: Performance evaluation: The progression of cumulative CO_2 emissions for each method under standard data conditions (mean ± 95% CI, $N = 10$).

a. Final-performance summary The progression of cumulative CO_2 emissions is shown in Figure 11. The final values are summarized in Table 12. DynaSAC ($\rho = 0.8$) achieved the lowest mean cumulative emissions, followed by SAC and PrioFlow. The DynaSAC ($\rho = 0.5$) variant exhibited higher mean emissions than both SAC and DynaSAC ($\rho = 0.8$), with greater variability reflected by its larger confidence interval. Both median and interquartile mean (IQM) statistics show the same consistent ranking, also ranking DynaSAC ($\rho = 0.8$) the highest and all other methods relatively close to each other, as can be confirmed by visual interpretation of Figure 11.

Method	**Mean ± 95% CI** (↓)	**Median** (↓)	**IQM** (↓)
PrioFlow	7.67 ± 0	7.67	7.67
SAC	6.75 ± 0.73	6.98	6.87
DynaSAC ($\rho = 0.8$)	5.17 ± 0.71	5.31	5.27
DynaSAC ($\rho = 0.5$)	7.86 ± 1.35	7.57	7.43

Table 12: Final cumulative CO_2 emissions (gCO_2eq) for each method under standard data conditions ($N = 10$ runs). Results are reported as mean ± 95% confidence interval (CI), median, and interquartile mean (IQM). All values are in tCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

b. Uncertainty & variability Estimates regarding the uncertainty for each method across seeds are reported in Table 13. As expected from a deterministic heuristic method, PrioFlow exhibited no variability across runs (CV = 0). Among the learning-based methods, both the CV and bootstrapped 95% confidence intervals display the same trend in variability: DynaSAC ($\rho = 0.8$) achieves the narrowest interval and a moderate degree of variability (CV = 0.22), SAC also showed moderate, though slightly lower variability (CV = 0.17), with a wider interval, only topped by DynaSAC ($\rho = 0.5$) wide interval and highest level of variability (CV = 0.28), reflecting its greater uncertainty.

Method	**CV** (↓)	**Bootstrap 95% CI Lower**	**Upper** (↓)
PrioFlow	0.00	7.67	7.67
SAC	0.17	6.04	7.43
DynaSAC ($\rho = 0.8$)	0.22	4.50	5.83
DynaSAC ($\rho = 0.5$)	0.28	6.75	9.26

Table 13: Uncertainty and variability metrics for final cumulative CO_2 emissions. CV denotes the coefficient of variation. Bootstrap 95% CIs are for the mean, computed over $N = 10$ runs. Bootstrap values are in tCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

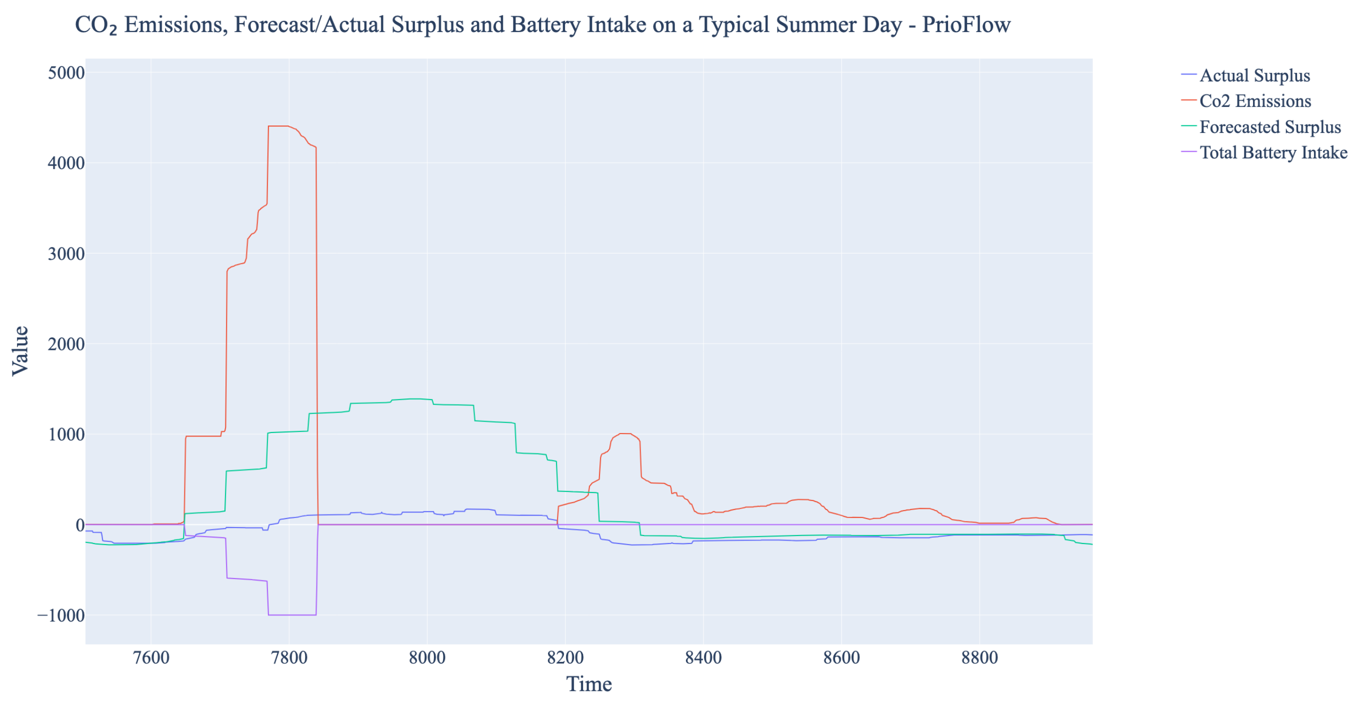
c. Pairwise comparison Overall evaluation of normality with the Shapiro-Wilk test indicated a significant deviation from normality ($p < 0.05$). Therefore, a Kruskal-Wallis test was employed, which indicated significant differences in the final cumulative CO_2 emissions across methods ($p = 7.5 \times 10^{-4}$). The ensuing pairwise comparisons with Holm-Bonferroni correction (see Table 14), showed that DynaSAC ($\rho = 0.8$) significantly outperformed both SAC (Holm-adjusted $p = 0.038$, Cohen's $d = -1.08$, PoI = 0.21) and PrioFlow (Holm-adjusted $p = 4.4 \times 10^{-4}$, $d = -2.17$, PoI = 0.00). Apart from that, none of the pairwise differences were statistically significant after correction. Notably the original SAC variant did not lead to significant improvements over PrioFlow, and the even more world-dependent DynaSAC ($\rho = 0.5$) variant did not yield any increase in performance over SAC or PrioFlow. The effect size (Cohen's d) was the largest for the DynaSAC ($\rho = 0.8$) comparisons, indicating the substantial benefit of incorporating the world model with a moderate

sample ratio under these circumstances.
Another notable outcome is that, although the PoI of DynaSAC ($\rho = 0.8$) vs DynaSAC ($\rho = 0.5$) is low (0.08), indicating that DynaSAC ($\rho = 0.8$) nearly always outperforms DynaSAC ($\rho = 0.5$) in the sampled runs, the difference in means did not reach statistical significance ($p = 0.091$) in the paired t-test. This disparity may be a result of limited sample size or variability within runs, which reduces the statistical power of the test despite a large observed effect size ($d = 0.87$).

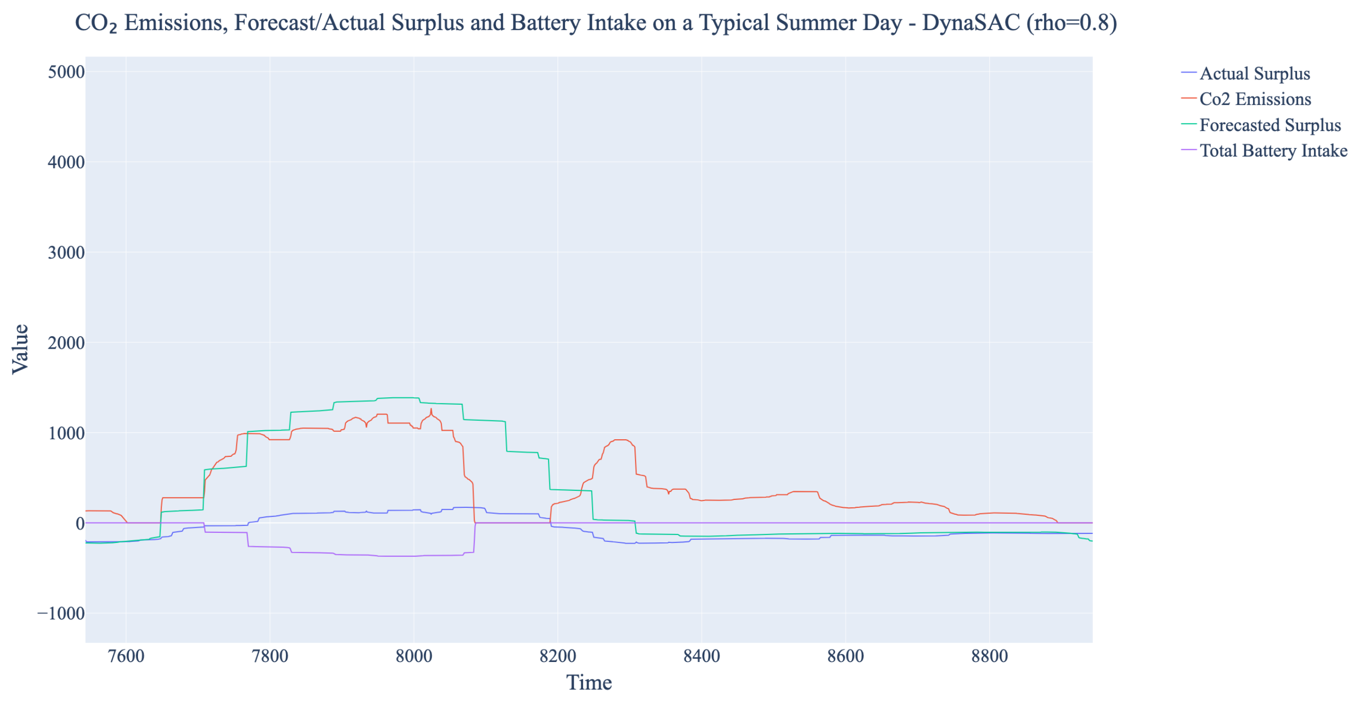
Comparison	**Test**	**Holm p**	**Cohen's d**	**PoI**
SAC vs PrioFlow	paired t	0.106	−0.78	0.30
DynaSAC ($\rho = 0.8$) vs SAC	paired t	**0.038**	−1.08	0.21
DynaSAC ($\rho = 0.5$) vs SAC	Wilcoxon	0.262	0.41	0.64
DynaSAC ($\rho = 0.8$) vs PrioFlow	paired t	**0.00044**	−2.17	0.00
DynaSAC ($\rho = 0.5$) vs PrioFlow	paired t	0.798	0.08	0.50
DynaSAC ($\rho = 0.8$) vs DynaSAC ($\rho = 0.5$)	paired t	0.091	−0.87	0.08

Table 14: Pairwise statistical comparisons of final cumulative CO_2 emissions for the list methods. p-values are Holm–Bonferroni adjusted; significant results ($p < 0.05$) are shown in bold. Cohen's d and PoI (Probability of Improvement) quantify effect size and likelihood of superior performance, respectively. PoI indicates B over A for A vs. B.

In-depth analysis: A more in depth look into how DynaSAC is able to consistently outperform the baseline is visualized in Figure 10, which gives a detailed insight into some of both models' actions on a representative day. As is shown in Figure 10a, PrioFlow controls the battery charging directly opposite to the forecasted surplus as a result of its heuristic (subsubsection 6.1.7). In this case, however, the forecast is too optimistic, leading to high CO_2 emissions. In contrast, DynaSAC ($\rho = 0.8$) acts more conservatively (Figure 10b), leading to the actual battery charge being a lot closer to the actual surplus than the forecasted surplus, resulting in less emissions. These results illustrate how DynaSAC is able to leverage real-time information to outperform the static heuristic PrioFlow.



(a) Metrics for the PrioFlow baseline.



(b) Metrics for the DynaSAC model.

Figure 10: Battery intake, the forecasted and actual production surplus and CO_2 emissions on a representative day for two models: (a) PrioFlow, (b) DynaSAC ($\rho = 0.8$). DynaSAC adapts battery-use to real-time information, resulting in reduced emissions compared to the heuristic baseline, which bases itself purely on the over-optimistic forecast.

d. Compute and learning efficiency The computational efficiency and learning curves metrics of each method are shown in Table 15. The average wall time is the shortest for SAC (0.94 ± 0.01 hr), with both DynaSAC methods performing comparably (DynaSAC ($\rho = 0.8$): 1.27 ± 0.02 hr; DynaSAC ($\rho = 0.5$): 1.25 ± 0.02 hr). In terms of learning efficiency, as measured by the AULC, the same pattern can be observed, with SAC achieving the lowest AULC values followed by DynaSAC ($\rho = 0.8$) and DynaSAC ($\rho = 0.5$). For both metrics, pairwise t-tests (after confirming normality) show that SAC exhibits significantly better performance in wall time and AULC values (p-values < 0.05 and large effect sizes (Cohen's $d > 0.8$)).
These results indicate that, despite their final-performance benefits, the model-augmented DynaSAC variants learn more slowly and result in a higher computational cost than vanilla SAC.

Method	**Wall Time (hr)** ($\downarrow$)	**AULC** ($\downarrow$)
SAC	0.94 ± 0.01	$1.63 \times 10^4 \pm 65$
DynaSAC ($\rho = 0.8$)	1.27 ± 0.02	$1.78 \times 10^4 \pm 106$
DynaSAC ($\rho = 0.5$)	1.25 ± 0.02	$1.84 \times 10^4 \pm 251$

Table 15: Computational efficiency and area under the learning curve (AULC) for each method. Wall time is reported as mean hours $\pm$ 95% CI. AULC is reported as mean $\pm$ 95% CI. All over $N = 10$ runs. The arrow in each column header indicates whether higher ($\uparrow$) or lower ($\downarrow$) values are better.

7.4.3 Summary

Under the standard (noise-free) conditions, DynaSAC with the moderate world model reliance ($\rho = 0.8$), achieved the lowest final cumulative CO_2 emissions, significantly outperforming both SAC and the PrioFlow baseline ((Holm-adjusted $p < .05$, large effect sizes). Notably, it exhibited a PoI of 0.00 versus PrioFlow, showcasing the virtually zero probability that DynaSAC's true mean emissions exceed those of heuristic baseline, confirming its superiority: a result of its ability to adaptively respond to forecast errors detailed in the in-depth analysis. The more aggressive world-model dependent variant ($\rho = 0.5$), on the other hand, did not show any significant benefit and showed the greatest run-to-run variability. Computationally speaking, SAC was the most efficient (with lowest wall-time and fastest learning). Altogether, this indicates that under these conditions, integrating a world model at $\rho = 0.8$ leads to a meaningful reduction in emissions at increased computational costs, whereas a higher model reliance ($\rho = 0.5$), does not does not provide any additional performance benefits.

7.4.4 Results: Noisy Data Conditions

a. Final-performance summary As both Figure 11 and Table 16 demonstrate, the performance gap between the methods disappears with the introduction of noise into the environment: means, medians, and IQMs are all relatively similar, though the model-enhanced methods trail behind slightly. Furthermore, the average mean emissions also increased by a considerable factor: 60.7% on average, lead by a 113.5% increase for DynaSAC ($\rho = 0.8$), whereas PrioFlow was the least affected (25.2% increase). This showcases how the addition of extra noise into the environment worsened performance for each method noticeably.

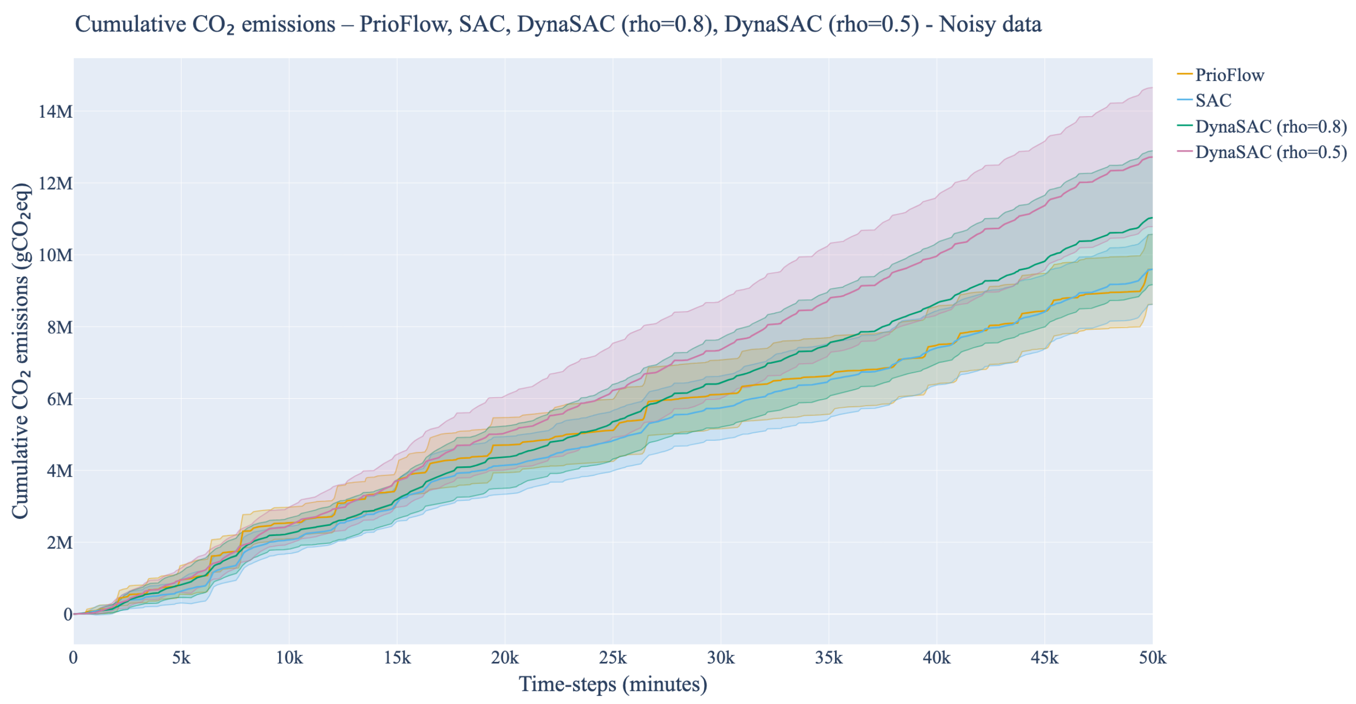


Figure 11: Performance evaluation: The progression of cumulative CO_2 emissions for each method under noisy data conditions (mean $\pm$ 95% CI, $N = 10$).

Method	**Mean $\pm$ 95% CI (↓)**	**Median (↓)**	**IQM (↓)**
PrioFlow	9.6 ± 1.0	9.2	9.6
SAC	9.6 ± 1.0	9.1	9.2
DynaSAC ($\rho = 0.8$)	11.0 ± 1.9	10.8	11.3
DynaSAC ($\rho = 0.5$)	12.7 ± 1.9	14.0	13.2

Table 16: Final cumulative CO_2 emissions (gCO_2eq) for each method under noisy data conditions ($N = 10$ runs). Results are reported as mean $\pm$ 95% confidence interval (CI), median, and interquartile mean (IQM). All values are in tCO_2eq. The arrow in each column header indicates whether higher (↑) or lower (↓) values are better.

b. Uncertainty & variability Table 17 shows that no method is immune to the noisy conditions, as all CIs widened and all CVs are at least 0.16 in the noisy settings. PrioFlow's heuristic has shifted from fully deterministic to notably variable, and both model-enhanced methods show a substantially wider CI, with DynaSAC ($\rho = 0.8$) also seeing a substantial increase in CV. Notably, the original model-free SAC showed the greatest level of robustness against noise, with no dramatic change in either CV or CI.

Method	**CV** ($\downarrow$)	**Bootstrap 95% CI Lower**	**Upper** ($\downarrow$)
PrioFlow	0.16	8.7	10.5
SAC	0.16	8.8	10.6
DynaSAC ($\rho = 0.8$)	0.27	9.2	12.8
DynaSAC ($\rho = 0.5$)	0.25	10.9	14.4

Table 17: Uncertainty and variability metrics for final cumulative CO_2 emissions, grouped by method. CV denotes the coefficient of variation. Bootstrap 95% CIs are for the mean, computed over $N = 10$ runs. Bootstrap values are in tCO_2eq. The arrow in each column header indicates whether higher ($\uparrow$) or lower ($\downarrow$) values are better.

c. Pairwise comparison The overall evaluation of normality was not rejected by the Shapiro-Wilk test and the subsequent one-way ANOVA indicated significant differences in final cumulative CO_2 emissions across methods ($F = 3.72$, $p = 0.020$). Therefore pairwise comparisons with Holm-Bonferroni correction were performed and its results summarized in Table 18. The results demonstrate that the only significant differences were found in performance of DynaSAC ($\rho = 0.5$), which was outperformed by both PrioFlow (p =0.0434) and SAC ($p = 0.0361$) with large effect sizes (Cohen's d of 0.82 and 0.81 respectively). Apart from that, no differences in performance were found to be significant. However, the PoI statistic did demonstrate a large shift, causing all statistics to favor SAC and PrioFlow, which perform nearly identical in this regard (PoI = 0.49). These results show that under noisy conditions, model-based DynaSAC variants lose their advantage, and may even underperform against SAC and PrioFlow.

Comparison	**Test**	**Holm** p	**Cohen's** d	**PoI**
SAC vs PrioFlow	paired t	0.995	0.00	0.49
DynaSAC ($\rho = 0.8$) vs SAC	paired t	0.614	0.43	0.71
DynaSAC ($\rho = 0.5$) vs SAC	paired t	**0.0361**	1.13	0.81
DynaSAC ($\rho = 0.8$) vs PrioFlow	paired t	0.478	0.54	0.66
DynaSAC ($\rho = 0.5$) vs PrioFlow	paired t	**0.0434**	1.06	0.82
DynaSAC ($\rho = 0.8$) vs DynaSAC ($\rho = 0.5$)	paired t	0.655	-0.33	0.37

Table 18: Pairwise statistical comparisons of final cumulative CO_2 emissions under noisy data conditions. p-values are Holm–Bonferroni adjusted; significant results ($p < 0.05$) are shown in bold. Cohen's d and PoI (Probability of Improvement) quantify effect size and likelihood of superior performance, respectively. PoI indicates B over A for A vs. B.

d. Compute and learning efficiency As demonstrated in Table 19, when compared to Table 15 the addition of noise did affect wall time. Remarkably though, in contrast with what the performance measurements seem to show, the mean AULC slightly increased for SAC and decreased for the DynaSAC variants, indicating their improved learning efficiency compared to SAC. The confidence intervals for AULC also increased excessively for each method, indicating exceedingly less consistency in learning efficiency between runs.

Method	**Wall Time (hr)** ($\downarrow$)	**AULC** ($\downarrow$)
SAC	0.94 ± 0.01	$1.68 \times 10^4 \pm 833$
DynaSAC ($\rho = 0.8$)	1.28 ± 0.01	$1.72 \times 10^4 \pm 851$
DynaSAC ($\rho = 0.5$)	1.19 ± 0.02	$1.75 \times 10^4 \pm 709$

Table 19: Computational efficiency and area under the learning curve (AULC) for each method under noisy data conditions ($N = 10$ runs). Wall time is reported as mean hours $\pm$ 95% CI. AULC is reported as mean $\pm$ 95% CI. The arrow in each column header indicates whether higher ($\uparrow$) or lower ($\downarrow$) values are better.

7.4.5 Summary

The introduction of noisy data conditions limited the performance differences between methods, though increasing their mean cumulative CO_2 emissions by 61% on average, with the greatest relative increase observed for DynaSAC ($\rho = 0.8$). The pairwise statistical tests revealed that DynaSAC ($\rho = 0.5$) was significantly outperformed by both SAC and PrioFlow, each with large effect sizes, while no other pairwise differences were significant. Probability of improvement statistics further confirmed the convergence in performance, particularly between SAC and PrioFlow. The noise also lead to decreased learning efficiency consistency across runs.
Overall, these results indicate that under substantial input noise, the performance advantages of model-based approaches largely disappear.

7.5 Key Findings

Conjointly, we observed the following key findings:

- The GRU-enhanced model did not show significantly better results than the non-recurrent model, though slightly less variability.

- DynaSAC outperforms DynaMASAC in every tested configuration in emission reduction and computational efficiency.

- DynaMASAC showed a relatively smaller proportional increase in emissions with larger asset scales than DynaSAC.

- DynaSAC with moderate world model reliance ($\rho = 0.8$) outperforms SAC and the heuristic PrioFlow baseline in emission reduction under standard (noise-free) conditions.

  Higher world model reliance ($\rho = 0.5$), on the other hand, did not provide performance benefits.

- With the introduction of noisy data conditions, the performance of model-based approaches degraded and the advantages disappeared, whereas SAC showed the greatest level of robustness.

8 Discussion

8.1 Interpretation of key findings

Given the outcomes of our experiments, we further scrutinize the key findings of this study.

Stochastic environment dynamics First of all, the hyperparameter optimization (subsection 7.1), showcased that high discount factors (γ) were prevalent in the best performing configurations. This is in line with expectations, as retaining surplus energy within energy storage assets can lead to long-term CO_2 savings. It further demonstrated that most parameters regarding stability in environments with stochastic dynamics, particularly the large batch size and delayed policy updates, generally resulted in the best performance, indicating high transition stochasticity is present in the simulated REV environment as is in line with literature [7]. These findings are reflected by the reduced performance benefits of the higher world-reliant model DynaSAC ($\rho = 0.5$) (subsection 7.4), as generally, model-based methods suffer under increasing uncertainty as the model error propagates [11, 17, 23, 25]. The experiment under noisy circumstances (subsubsection 7.4.4) reinforces this pattern, illustrating the greater noise-resistance of the model-free SAC against the model-enhanced methods.

Real-time adaptation Notably, under standard circumstances, moderately world-model enhanced DynaSAC ($\rho = 0.8$) did outperform SAC, significantly and with a large effect size (see Table 14). Consequently, it seems that diversifying the agent's replay buffer with synthetic experiences to a certain extent leads to improved performance. Under the same circumstances, DynaSAC ($\rho = 0.8$) also outperforms the heuristic baseline (Table 14), demonstrating the advantage RL methods can have over static heuristic solutions. A result of the fact that static methods generally rely to a great extent on the accurate prediction of energy consumption and production and are unable to adapt its policy in real-time (such as displayed in Figure 10).

Centralized versus decentralized paradigm On the other hand, the more classically decentralized sCTDE extension of SAC did not show better absolute performance (Table 14). However, as detailed in Table 10, the relative increase in final cumulative CO_2 emissions of the sCTDE variant, with an increasing number of assets within the REV, was substantially lower than that of the default CTCE variant. For large-scale REVS then, utilizing the sCTDE paradigm may lead to improved performance compared to CTCE, though more extensive research is necessary to fund any such claim.

Role of recurrency Finally, the unobserved performance benefits of including the local recurrent component within the world model, lead to believe that either the experiment scale within this study is too small to find significant benefits, or perhaps more likely, the temporal dependencies within the energy forecasts are so weak that they can be captured by simple feed-forward layers.

In conclusion, hybrid methods (like DynaSAC with moderate world model reliance) may be best for complex, semi-predictable REV environments. Though its usage requires balance: there should be enough synthetic data to diversify, but not so much that model errors dominate. However, in domains that exhibit a high level of dynamics stochasticity, model-free SAC suffers the smallest relative degradation, and might therefore be preferred. Additionally, given that DynaMASAC (multi-agent) never beats DynaSAC (single-agent dynamics), even though MASAC might intuitively seem "more correct" for a multi-agent environment, at least on a smaller scale, the extra coordination steps evidently introduce overhead difficulties that outweigh the theoretical benefits.

Finally, to answer the research question central to this study: *"Does incorporating a learned world model significantly improve multi-agent coordination and CO_2 emission minimization in energy hub management compared to purely model-free MARL?"*: Yes, under noise-free conditions the DynaSAC ($\rho = 0.8$) agent significantly reduced mean final cumulative CO_2 emissions by 23.4% relative to SAC and 32.7% relative to the heuristic baseline (Holm-Bonferroni $p < 0.05, d > 1$). Under additional sensor/forecast noise the advantage vanished. Therefore, world-model augmentation is beneficial only when the model is sufficiently accurate relative to the level of environmental noise.

8.2 Comparison with prior work

Several recent studies in closely related energy-management contexts, have arrived at conclusions aligned with ours: reinforcement learning can effectively reduce carbon dispatch [40], state-of-the-art RL approaches can outperform heuristic baselines [21], and model-based enhancements can surpass model-free approaches [27]. Below we elaborate further on prior related works.

(Multi-agent) Reinforcement learning approaches within system energy management As discussed before in (section 2), Dyna-style optimization approaches remain largely unexplored within the energy management context, with specific appliances to the EDP lacking. However, some parallels can be drawn from closely related domains. For instance, within the vehicle energy system context, Liu *et al.* [27] employ a tabular Dyna-Q approach in order to reduce fuel consumption in hybrid vehicles, achieving performance improvements over purely model-free Q-learning. Their findings align with those demonstrated here, confirming that model-based enhancements can lead to increased performance at the expense of an additional computational costs.

Similarly, the work of Sun *et al.* [40] utilizes Q-learning to minimize both operations costs and emissions within an energy system. Though the integrated system is not purely electrical but also introduces heating, the behavior of the energy storage device does exhibit recognizable similarities to that in our study, charging surplus PV (photovoltaic) generation in the middle of the day for later exploitation. This provides evidence for the applicability of reinforcement learning for energy system emission minimization, in line with this work.

Within the multi-agent domain, Zhang *et al.* [5] approach REVs as individual, energy-type-specific agents, using a fully decentralized DTDE (decentralized training for decentralized execution) paradigm. Accordingly, each agent maintains their own actor and critic components, but the agents learn collaboratively through a shared experience replay buffer during training. They found this approach to demonstrate more stable performance than the single-agent DDPG (deep deterministic policy gradient) method. Notably, these benefits attributed to DTDE are not reflected directly in our CTDE approach, suggesting the advantages of decentralization may be context-dependent or might benefit from a greater extent of decentralization during training.

The bias-variance trade-off On the other hand, our results do reaffirm a classic deep reinforcement learning concept: the bias-variance trade-off. Specifically, the world model in model-based reinforcement learning (MBRL) can offer many additional (synthetic) transitions, which reduces gradient variance, and can accelerate learning, but only as long as it is truthful and mimics real dynamics, otherwise it introduces bias [11, 17]. This balance is made explicit in the comparison of (low-bias,

high-variance) SAC and (higher bias, low variance) DynaSAC ($\rho = 0.8$) under standard (see Table 14) and noisy circumstances (Table 18).
To keep experiences in line with reality, Janner *et al.* [23] therefore suggest that lower ks (roll-out lengths) offer advantages, because shorter roll-outs are more likely to reflect actual dynamics. Although we did not explicitly explore roll-out lengths in detail, the optimal $k = 6$ found here is modest compared to state-of-the-art model-based DRL frameworks such as Dreamer V3 ($k = 16$; [22]), which further highlights the challenge of accurately modeling highly stochastic environments.

Model-based sampling ratio Furthermore, regarding the degree of world-model integration during training, we keep the transition sampling ratio ρ fixed. However, previous studies, such as Gu *et al.* [25] and Kalweit and Boedecker [41], indicate that model-based enhancement is most effective at the start of training, and becomes less helpful as estimates become more certain.
Gu *et al.* [25] suggest using a strategy which turns off model-based rollouts after a number of steps within training. In line with that suggestion, Kalweit and Boedecker [41] adapt the ratio for which model based synthetic transitions are sampled based on the uncertainty: higher uncertainty increases the chance of sampling a synthetic transition. The consistent heavy reliance on synthetic transitions throughout training may explain the diminishing returns observed in our higher world-model-dependent variant DynaSAC ($\rho = 0.5$) (Table 14).

Centralized versus decentralized paradigms Additionally, in the consideration of CTCE versus (s)CTDE paradigms, the literature offers nuanced insights. As noted by Zhang *et al.* [32] and Oliehoek and Amato [28], centralized frameworks generally perform well in fully observable cooperative context at smaller agent scales, due to their ability to learn optimal joint policies, deteriorating with increasing agent scales. In this light, given the outcome of the paradigm experiment (subsection 7.3), the complexity at the tested agent scales may be manageable for a CTCE approach, whilst the performance decrease suggests challenges in complexity for the CTDE approach, preventing it from surpassing single-agent performance. Though further research would be required to pinpoint the cause of the increased complexity, as the traditional multi-agent issues of credit-assignment [34] and nonstationarity [19] are already largely mitigated in this work through tailored a“per-agent” reward and sequential agent updates.

Static baseline out-performance Finally and notably, a recent similar study by Das *et al.* [21], employing state-of-the-art model-free RL techniques in an identical 3-asset configuration (solar, storage & demand), reports similar outcomes: the RL methods consistently out-perform the heuristic baseline, which is a simple PV surplus-based charge/discharge strategy, similar to PrioFlow. This parallel reinforces our findings and further confirms the effectiveness of RL in the EDP context.

8.3 Limitations and threats to validity

Having established the main conclusive findings of this study, below we reflect on the limitations and threats of validity that should be taken into account before extrapolation to any broader context.

Construct validity Given that we optimize solely for cumulative CO_2 emissions, we assume emissions are the dominant stakeholder objective. Though that is in line with the objectives outlined by the REFORMERS project [10], other objectives, especially those regarding costs, could be of importance and yield different policies. This would be resolved by including a multi-objective reward

which would incorporate such factors as well.
Additionally, as mentioned in subsection 6.5, the fixed grid emissions factor might misrepresent seasonal or marginal effects, which could be resolved with actual hourly grid mix data. Furthermore, the simplified asset modeling as mentioned in subsection 6.1 could widen the simulation-reality gap, that could be mitigated by adding additional physics-informed constraints to the SIMON simulation environment.

Internal validity Evaluating the internal consistency of the results, two main threats can be identified. Firstly, the 50 000 one-minute steps (which is approx. 35 days), is most likely insufficient for agents to pick up on seasonal patterns. Training over longer horizons, or increasing the representation of such patterns within the dataset would help mitigate this issue. Moreover, the hyperparameter sweep fine-tuned the parameters on a 3-asset configuration for SAC. These parameters might not transfer well to MASAC and the larger REV configurations. Ideally, a per-configuration tuning would ensure a more optimal composition of hyperparameters for each situation.

External validity Generalizing the findings of this study to other systems comes with some limitations. Notably, the asset and data-mix used in this study might not transfer to other REV configurations. Specific behavior of a dutch dairy-farm demand profile, may not capture the inconsistencies or behavioral patterns as other loads. Furthermore, the relative small-scale of the tested REV (3 assets, of which 2 controllable), may not surface emergent coordination phenomena that can arise in larger energy systems.

Conclusion validity Regarding the soundness of the statistical inferences in this study, the use of ten random seeds per algorithm may not fully capture the true variability of the underlying distribution. Likewise, the multiple comparison across algorithms, even with Holm-Bonferroni correction, does increase the risk of false positives: detecting apparent differences due to chance rather than true differences.

Reproducibility The MARLOES code, configurations and experiments are open-source, but SIMON is proprietary. Consequently, only users with SIMON access will be able to fully reproduce the experiments as conducted within this study.

These limitations should be taken into account when interpreting the results of this study, and point out the need for caution in generalizing these findings beyond its settings and assumptions.

8.4 Practical implications

Given the conclusions and associated limitations, the practical implications and recommendations are as follows. Firstly, reflecting on the outcome of this work, first of all we recommend operators could start with model-free SAC and start using synthetic roll-outs once forecast quality is known. For very noisy REVs, it would be best to stick to a model-free approach, or invest in better forecasting quality. Furthermore, given the fact that MARLOES is specifically set-up to facilitate fast real-world deployment by imitating the commonly-used “steering” architecture (see subsection 6.1), operators could efficiently utilize the framework to simulate performance before deploying any MARL steering algorithms in actual REVs.

8.5 Future work

In addition to the aforementioned mitigations in regard to the threats to validity, we suggest the following alleys for future work:

- Employ a world model that is more specifically designed to handle uncertainty, such as an ensemble similar to that in MBPO [23].
- Evaluate the performance of the CTCE and CTDE paradigms in larger scale REVs, to find whether the expected pattern of improved performance under increasing decentralization emerges.
- Implement an automated adaptive control of the sample ratio between synthetic and real experiences, such as that used in Kalweit and Boedecker [41], to test the effects and potential benefits of an adaptive sample ratio within the EDP context.
- Compose an adaptive rollout horizon (k) in order to automatically minimize k to the extent to which it remains beneficial, keeping the learning process efficient and in line with the findings of Janner *et al.* [23].
- Evaluate other state-of-the-art RL methods that are still missing within the EDP context in the MARLOES environment.
- Incorporate a multi-objective reward to analyze how policies develop, when e.g. market costs are incorporated as well.

8.6 Summary and outlook

In this work, we addressed key gaps within the field of multi-agent reinforcement learning as a solution to the energy dispatch problem in closed energy systems such as Renewable Energy Valleys, by: (1) introducing MARLOES, a simulation interface for consistent evaluation of MARL algorithms; (2) developing a Dyna-style SAC algorithm with a hierarchical world model that is suitable for REV energy management; and (3) systematically benchmarking model-based, model-free, and heuristic control algorithms on a realistic REV scenario aligned with the REFORMERS pilot.

Our results provide insight into the circumstances under which hybrid RL methods contribute to substantial emissions reductions and demonstrate both their potential and limitations. By benchmarking the state-of-the-art SAC actor-critic approach, we contribute new empirical evidence in this context, where such methods have been underexplored.

We hope that these findings will support future research and progress in adaptive energy management aligned with EU climate goals.

References

[1] E. E. A. (EEA), *Greenhouse gas data viewer*, https://www.eea.europa.eu/en/analysis/maps-and-charts/greenhouse-gases-viewer-data-viewers, Provisional 2023 data; accessed 1 July 2025., 2025.

[2] *Regulation (EU) 2021/1119 of the European Parliament and of the Council of 30 June 2021 establishing the framework for achieving climate neutrality and amending Regulations (EC) No 401/2009 and (EU) 2018/1999 ('European Climate Law')*, en, Legislative Body: CONSIL, EP, Jun. 2021. [Online]. Available: http://data.europa.eu/eli/reg/2021/1119/oj/eng.

[3] E. E. A. (EEA), "Annual european union greenhouse gas inventory 1990–2022 and inventory report 2024," European Environment Agency, Tech. Rep. EEA Report No 8/2024, 2024. DOI: 10.2800/025423. [Online]. Available: https://www.eea.europa.eu/en/analysis/publications/annual-european-union-greenhouse-gas-inventory.

[4] E. Commision, "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," English, European Commission, Brussels, Tech. Rep. COM(2022) 230 final, May 2022. [Online]. Available: https://www.reteambiente.it/repository/normativa/48230_comunicazione18_maggio_2022repowereu.pdf.

[5] X. Zhang, Q. Wang, J. Yu, Q. Sun, H. Hu, and X. Liu, "A Multi-Agent Deep-Reinforcement-Learning-Based Strategy for Safe Distributed Energy Resource Scheduling in Energy Hubs," *Electronics*, vol. 12, no. 23, p. 4763, 2023, Publisher: MDPI. [Online]. Available: https://www.mdpi.com/2079-9292/12/23/4763.

[6] A. Jabbary, R. Noroozian, and G. B. Gharehpetian, "Optimum utilization of hub energy micro-grids with micro-networking strategy of local biogas productions," *Heliyon*, vol. 9, no. 11, e20995, Oct. 2023, ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2023.e20995. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10637905/.

[7] 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*, ISSN: 1553-572X, Nov. 2010, pp. 3341–3346. DOI: 10.1109/IECON.2010.5675295. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5675295.

[8] G. H. Merabet *et al.*, "Applications of multi-agent systems in smart grids: A survey," in *2014 International conference on multimedia computing and systems (ICMCS)*, IEEE, 2014, pp. 1088–1094. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6911384/.

[9] M. W. Khan, J. Wang, M. Ma, L. Xiong, P. Li, and F. Wu, "Optimal energy management and control aspects of distributed microgrid using multi-agent systems," *Sustainable Cities and Society*, vol. 44, pp. 855–870, 2019, Publisher: Elsevier. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670718315233.

[10] H. E. R. Consortium, *Description of Action (DoA): Regional Ecosystems for Multiple Energy Resilient Systems (REFORMERS)*, English, Access note: Internal project document, not publicly available., 2023.

[11] A. Perera and P. Kamalaruban, “Applications of reinforcement learning in energy systems,” en, *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110 618, Mar. 2021, ISSN: 13640321. DOI: 10.1016/j.rser.2020.110618. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1364032120309023.

[12] S. M. Rinaldi, J. P. Peerenboom, and T. K. Kelly, “Identifying, understanding, and analyzing critical infrastructure interdependencies,” *IEEE control systems magazine*, vol. 21, no. 6, pp. 11–25, 2001, Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/969131/.

[13] M. Shi, H. Wang, P. Xie, C. Lyu, L. Jian, and Y. Jia, “Distributed energy scheduling for integrated energy system clusters with peer-to-peer energy transaction,” *IEEE Transactions on Smart Grid*, vol. 14, no. 1, pp. 142–156, 2022, Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9857842/.

[14] C. Li *et al.*, “Optimal planning of islanded integrated energy system with solar-biogas energy supply,” *IEEE Transactions on Sustainable Energy*, vol. 11, no. 4, pp. 2437–2448, 2019, Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8930985/.

[15] D. Bhardwaj *et al.*, “AI-Driven Energy Management Systems for Microgrids: Optimizing Renewable Energy Integration and Load Balancing,” in *E3S Web of Conferences*, vol. 591, EDP Sciences, 2024, p. 01 005. [Online]. Available: https://www.e3s-conferences.org/articles/e3sconf/abs/2024/121/e3sconf_icrera2024_01005/e3sconf_icrera2024_01005.html.

[16] L. Sun and F. You, “Machine Learning and Data-Driven Techniques for the Control of Smart Power Generation Systems: An Uncertainty Handling Perspective,” en, *Engineering*, vol. 7, no. 9, pp. 1239–1247, Jan. 2021, Number: 9 Publisher: Higher Education Press, ISSN: 2095-8099. DOI: 10.1016/j.eng.2021.04.020. [Online]. Available: https://www.engineering.org.cn/engi/EN/10.1016/j.eng.2021.04.020.

[17] 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, pp. 219–354, 2018, Publisher: Now Publishers, Inc. [Online]. Available: https://www.nowpublishers.com/article/Details/MAL-071.

[18] G. Zhang *et al.*, “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. 115 340, 2022, Publisher: Elsevier. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890422001364.

[19] A. Plaat, *Deep Reinforcement Learning*. Springer Nature Singapore, 2022, ISBN: 9789811906381. DOI: 10.1007/978-981-19-0638-1. [Online]. Available: http://dx.doi.org/10.1007/978-981-19-0638-1.

[20] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, arXiv:1801.01290 [cs], Aug. 2018. DOI: 10.48550/arXiv.1801.01290. [Online]. Available: http://arxiv.org/abs/1801.01290.

[21] I. Das, J. Ahmed, and A. Shukla, “Optimizing Solar Microgrid Efficiency via Reinforcement Learning: An Empirical Study Using Real-Time Energy Flow and Weather Forecasts,” en, *International Journal of Computer Applications*, vol. 187,

[22] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, *Mastering Diverse Domains through World Models*, arXiv:2301.04104 [cs], Apr. 2024. DOI: `10.48550/arXiv.2301.04104`. [Online]. Available: `http://arxiv.org/abs/2301.04104`.

[23] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to Trust Your Model: Model-Based Policy Optimization," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2019/hash/5faf461eff3099671ad63c6f3f094f7f-Abstract.html`.

[24] W. Sun, N. Jiang, A. Krishnamurthy, A. Agarwal, and J. Langford, "Model-based RL in Contextual Decision Processes: PAC bounds and Exponential Improvements over Model-free Approaches," en, in *Proceedings of the Thirty-Second Conference on Learning Theory*, ISSN: 2640-3498, PMLR, Jun. 2019, pp. 2898–2933. [Online]. Available: `https://proceedings.mlr.press/v99/sun19a.html`.

[25] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International conference on machine learning*, PMLR, 2016, pp. 2829–2838. [Online]. Available: `http://proceedings.mlr.press/v48/gu16.html`.

[26] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*, Elsevier, 1990, pp. 216–224. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9781558601413500304`.

[27] T. Liu, Y. Zou, D. Liu, and F. Sun, "Reinforcement Learning–Based Energy Management Strategy for a Hybrid Electric Tracked Vehicle," en, *Energies*, vol. 8, no. 7, pp. 7243–7260, Jul. 2015, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1996-1073. DOI: `10.3390/en8077243`. [Online]. Available: `https://www.mdpi.com/1996-1073/8/7/7243`.

[28] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs* (SpringerBriefs in Intelligent Systems), en. Cham: Springer International Publishing, 2016, ISBN: 978-3-319-28927-4 978-3-319-28929-8. DOI: `10.1007/978-3-319-28929-8`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-28929-8`.

[29] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, ser. TARK '96, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 195–210.

[30] Q. Zhang, D. Zhao, and F. L. Lewis, "Model-Free Reinforcement Learning for Fully Cooperative Multi-Agent Graphical Games," in *2018 International Joint Conference on Neural Networks (IJCNN)*, ISSN: 2161-4407, Jul. 2018, pp. 1–6. DOI: `10.1109/IJCNN.2018.8489477`. [Online]. Available: `https://ieeexplore.ieee.org/document/8489477/`.

[31] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html`.

[32] K. Zhang, Z. Yang, and T. Başar, *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*, arXiv:1911.10635 [cs], Apr. 2021. DOI: `10.48550/arXiv.1911.10635`. [Online]. Available: `http://arxiv.org/abs/1911.10635`.

[33] Y. Pu, S. Wang, R. Yang, X. Yao, and B. Li, *Decomposed Soft Actor-Critic Method for Cooperative Multi-Agent Reinforcement Learning*, arXiv:2104.06655 [cs], May 2021. DOI: `10.48550/arXiv.2104.06655`. [Online]. Available: `http://arxiv.org/abs/2104.06655`.

[34] R. S. Sutton and A. Barto, *Reinforcement learning: an introduction* (Adaptive computation and machine learning), en, Second edition. Cambridge, Massachusetts London, England: The MIT Press, 2020, ISBN: 978-0-262-03924-6.

[35] T. Haarnoja *et al.*, *Soft Actor-Critic Algorithms and Applications*, arXiv:1812.05905 [cs], Jan. 2019. DOI: `10.48550/arXiv.1812.05905`. [Online]. Available: `http://arxiv.org/abs/1812.05905`.

[36] S. Fujimoto, H. v. Hoof, and D. Meger, *Addressing Function Approximation Error in Actor-Critic Methods*, arXiv:1802.09477 [cs], Oct. 2018. DOI: `10.48550/arXiv.1802.09477`. [Online]. Available: `http://arxiv.org/abs/1802.09477`.

[37] K. Cho *et al.*, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, arXiv:1406.1078 [cs], Sep. 2014. DOI: `10.48550/arXiv.1406.1078`. [Online]. Available: `http://arxiv.org/abs/1406.1078`.

[38] D. Ha and J. Schmidhuber, "Recurrent World Models Facilitate Policy Evolution," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1` `Abstract.html`.

[39] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training Recurrent Neural Networks*, arXiv:1211.5063 [cs], Feb. 2013. DOI: `10.48550/arXiv.1211.5063`. [Online]. Available: `http://arxiv.org/abs/1211.5063`.

[40] Q. Sun, D. Wang, D. Ma, and B. Huang, "Multi-objective energy management for we-energy in Energy Internet using reinforcement learning," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov. 2017, pp. 1–6. DOI: `10.1109/SSCI.2017.8285243`. [Online]. Available: `https://ieeexplore.ieee.org/document/8285243/`.

[41] G. Kalweit and J. Boedecker, "Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning," en, in *Proceedings of the 1st Annual Conference on Robot Learning*, ISSN: 2640-3498, PMLR, Oct. 2017, pp. 195–206. [Online]. Available: `https://proceedings.mlr.press/v78/kalweit17a.html`.

[42] Statistics Netherlands (CBS). "Over half of electricity production now comes from renewable sources." Accessed: 2025-01-07. (Sep. 2024), [Online]. Available: `https://www.cbs.nl/en-gb/news/2024/39/over-half-of-electricity-production-now-comes-from-renewable-sources`.

Appendices

A Demand, Solar, and Wind Asset Handlers

This appendix describes the design and configuration of the demand, solar, and wind asset handlers as implemented in our framework. Each of these asset handlers works by reading in an energy profile as a time-series as well as its corresponding forecast, and then scaling both by a configurable factor. During each time-step the handler only has access to its current actual production/consumption, but does have access to its future forecasted power for a certain horizon (default is 4 hours). All time-series and their corresponding forecasts have been provided by Repowered.

A.1 Asset Handler Specifications

Demand Handler The demand asset models energy consumption. The framework provides a 15-minute resolution energy profile of a dairy farm spanning both 2022 and 2023, with corresponding forecast. Furthermore, within the framework, the demand asset is not assigned any setpoints by its handler because consumption is treated as a must-serve load.

Solar Handler The solar asset represents photovoltaic generating systems. Based on the provided orientation (East-West or South), the production profile is retrieved, scaled according to the passed DC capacity and capped at the inverter (AC) capacity. Solar parks are considered curtailable (downward dispatchable) by the solver, and thus are allowed setpoints. The production profiles that are provided by the framework are hourly production data of two (East-West and South) 1 MWp solar parks spanning the years 2022 and 2023, located in Alkmaar.

Wind Handler The wind asset models wind power production. There is one production profiles provided based on the production of typical onshore turbine (a 60 meter high Vestas V80 2000) within the neighbourhood of Alkmaar, scaled to 1 MWp. Just as for the solar profiles the data spans 2022 and 2023 and the data is scaled using the available power rating and capped at AC capacity. Equivalently, wind generation is assumed to be curtailable by setpoints.

States Each asset handler maintains a state composed of the following three elements: (1) the current power, (2) the available power (for solar and wind, indicating potential production), and (3) a forecast vector for the next four hours.

A.2 Default Parameters

Table 20 summarizes the default parameters for the demand, solar, and wind assets.

Parameter	**Demand**		**Solar**		**Wind**	
	Default Value	**Notes**	**Default Value**	**Notes**	**Default Value**	**Notes**
`max_power_in`	∞	No strict upper limit	–	N/A	–	N/A
`max_power_out`	–	N/A	3000	-	3000	-
Curtailable?	false	must-serve load	true	-	true	-
Action Space	N/A	No setpoints for demand	$[0,1]$	-	$[0,1]$	-

Table 20: Comparison of the default parameters for demand, solar, and wind assets.

B Battery and Electrolyser Handlers

This appendix provides detailed information on the configuration, state and degradation functions for the storage-based handlers implemented in the MARLOES framework: the BatteryHandler and the ElectrolyserHandler. Both handlers share common functionality and states, but are configured with different parameters to reflect their distinct characteristics. Their respective configurations are outlined in detail below.

B.1 Default Parameters

Table 21 below compares the default parameters for the two handler types. Each handler has a state consisting of the elements: (1) power (the current power), (2) state of charge (currently filled capacity ranging from 0-1), and (3) degradation (result of the degradation functions as outlined below in subsection B.2).

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*0.4	Models slower startup
`ramp_down_rate`	1000	Instant	1000*0.4	Models controlled shutdown
`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 Model	Cycle-based	-	Hour-based	-

Table 21: A comparison of default parameters for the storage-based handlers.

B.2 Degradation Functions

To also model the deterioration of batteries and electrolysers, our framework incorporates distinct degradation models for either.

Battery Degradation The battery degradation is modeled as a linear function of the number of complete charge–discharge cycles. A full cycle is defined as a complete charge and a complete discharge: equivalent to an energy throughput of $2C$, where C is the battery capacity (in kWh). The degradation per cycle is given by:

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

where:

- t is the time step in seconds,

- $|P|$ is the absolute value of the power output (in kW),
- C is the battery capacity (in kWh),
- N_{total} is the total number of cycles until the battery is considered deprecated (default is 8000).

As a result, the battery's updated degradation state is:

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

Electrolyser Degradation For the electrolyser, degradation is modeled based on cumulative operating hours. The degradation per hour is defined as:

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

where:

- T_{lifetime} is the total operational lifetime in hours (default is 80 000 hours),
- C_{max} is the maximum capacity after applying the conversion factor (i.e., the effective capacity in kWh),
- t and $|P|$ are defined as above.

The degradation state for the electrolyser is then also:

$$D_{\text{new}} = D_{\text{old}} + \Delta D_{\text{electrolyser}}.$$

The use of these formulas ensure that both assets degrade in manner that is consistent with the domain knowledge as provided by Repowered: the battery is expected to retain 60% of its capacity and be completely degraded after the specified number of cycles, whilst the the electrolyser's efficiency declines over its operational lifetime in hours.

C Hyperparameter details

The combination of SAC and the Dyna-style enhancement have several hyperparameters that can be tuned in order to further align the model. In order to decide on the final parameters to use for the experiments we performed a hyperparameter search. The search was done over the most influential hyperparameters in a range that was found to be reasonable through intensive testing. We did a randomized hyperparameter sweep of 200 trials over the search space as outlined in Table 22, consisting of 50.000 trainings steps and 50.000 evaluation steps each, and evaluated on cumulative CO_2 emissions. The best performing configuration, which are further used in this study, unless mentioned otherwise, is shown in Table 3. Other relevant parameters which were not included in the search are detailed in Table 23.

General Parameters		
Parameter	**Range/Choices**	**Description**
B	{32, 64, 128}	Mini-batch size
SAC Parameters		
Parameter	**Range/Choices**	**Description**
η_α	$[10^{-6}, 10^{-4}]$	Entropy coefficient LR
η_π	$[10^{-6}, 10^{-4}]$	Policy network LR
η_Q	$[10^{-6}, 10^{-4}]$	Q-network LR
η_V	$[10^{-6}, 10^{-4}]$	Value network LR
γ	{0.98, 0.99, 0.995, 0.999}	Discount factor
τ	{0.005, 0.01, 0.02, 0.025}	Target smoothing coefficient
λ_s	$[10^{-8}, 10^{-5}]$	Optimizer weight decay
ε	$[10^{-8}, 10^{-6}]$	Optimizer epsilon
r	{1, 2, 3}	Critic to actor update frequency
d_h	{64, 128, 256}	Hidden layer size
L	{1, 2, 3}	Number of hidden layers
Dyna Parameters		
Parameter	**Range/Choices**	**Description**
U	{15, 18, 20}	No. model updates/I_{upd}
k	{4, 6, 8, 10}	Model rollout length
World Model Parameters		
Parameter	**Range/Choices**	**Description**
η_w	$[10^{-4}, 10^{-2}]$	World model learning rate
$d_{\text{h}}^{\text{world}}$	{32, 64, 128}	World model hidden layer size
$d_{\text{h}}^{\text{forecast}}$	{8, 16, 32}	Forecasting module hidden size

Table 22: Hyperparameter search space that was used in this study (LR = learning rate).

General Fixed Parameters		
Parameter	**Value**	**Description**
N_{init}	2000	Steps before learning starts
M_{model}	100000	Capacity of model replay buffer
M_{real}	100000	Capacity of real replay buffer
κ	10	Reward scaling factor
SAC Fixed Parameters		
Parameter	**Value**	**Description**
$\log \sigma_{\max}$	2	Maximum log-std for action distribution
$\log \sigma_{\min}$	-5	Minimum log-std for action distribution
Dyna Fixed Parameters		
Parameter	**Value**	**Description**
ρ	0.8	Fraction of real vs. model data per update
I_{upd}	10	Env. steps between model updates
F_w	1	World-model updates per planning cycle
World Model Fixed Parameters		
Parameter	**Value**	**Description**
$d_{\text{out}}^{\text{agent}}$	8	Output size for agent encoder
d_h^{agent}	32	Hidden layer size in agent encoder
L_{forecast}	1	Layers in forecast GRU module
λ_w	0.0	Weight-decay (L_2 regularization)
d_h^{dyn}	64	Hidden size of world-dynamics network
$d_{\text{out}}^{\text{world}}$	32	Output size for world encoder

Table 23: Fixed hyperparameters (not included in search space).

D Data Processing Pipeline

This appendix outlines the data processing pipeline that is applied on all energy profiles and their corresponding forecasts. This pipeline ensures that the data used is temporally consistent. Furthermore its goal is to more closely represent real world circumstances. It consists of two main phases:

1. Conversion and alignment: standardizing and synchronizing the input data.
2. Noise injection and dropout simulation: introducing noise and some stochastic variation to more closely reflect uncertainty during real-time operation.

D.1 Conversion and Alignment

Before any time-series data can be used within the simulation, it must be processed into a consistent format. This phase consists of the following steps, which are visualized in Figure 12:

1. **Unit Conversion:** Input data could either be in kilowatt-hours (kWh) or kilowatts (kW). To ensure consistency, any kWh-based data is converted to kW using:

   $$P_{\text{kw}}(t) = \frac{E_{\text{kWh}}(t)}{\Delta t}, \tag{20}$$

   where $E_{\text{kWh}}(t)$ is the energy recorded in a given time step, and Δt is the duration of that step, since the SIMON framework requires power data (kW) rather than cumulative energy (kWh).
2. **Temporal Resolution Adjustment:** To maintain temporal consistency across all series, all data is converted to a minutely resolution by distributing the energy evenly across each minute.
3. **Time Alignment:** To ensure the actual times within the series match, they are shifted to a common time frame (i.e., aligned to the year 2025). This is done by mapping timestamps from the original dataset to corresponding timestamps in the target year, which ensures that cyclical and seasonal patterns are preserved.

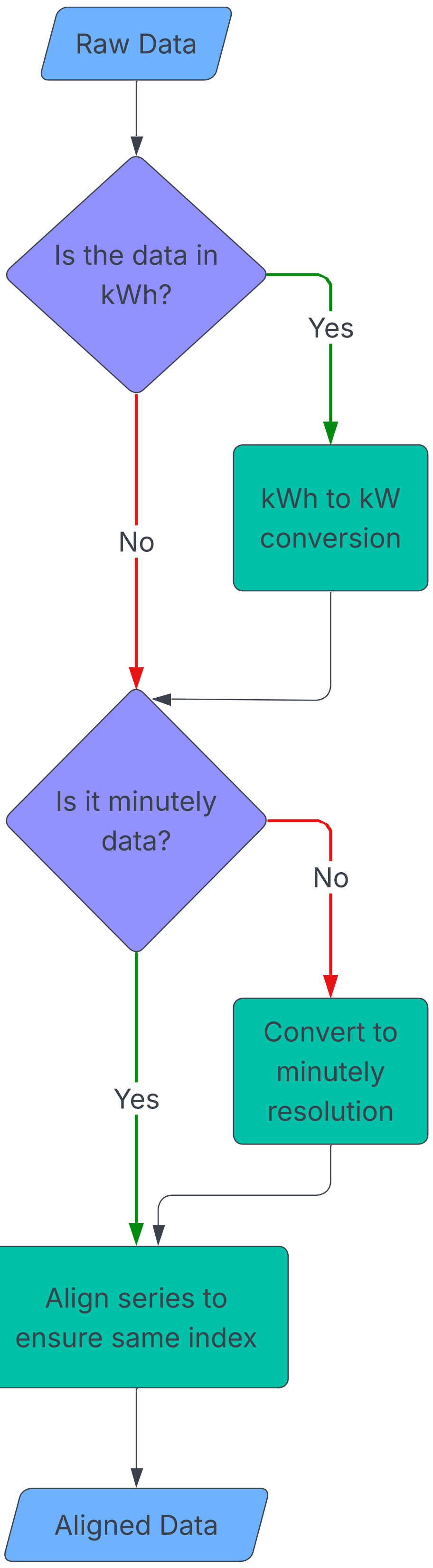


Figure 12: Data conversion and alignment is the first part in the processing pipeline which standardizes the incoming data.

D.2 Noise Injection and Dropout Simulation

After the alignment, the data is further processed in order to simulate real-world inconsistencies (see Figure 13):

1. **Noise Injection:** Noise is added to the series using normally distributed random fluctuations, which simulates forecast errors and ensures variability in training:

$$P_{\text{noisy}}(t) = P_{\text{aligned}}(t) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma) \tag{21}$$

   where σ is a variable standard deviation.

2. **Dropout Simulation:** After noise addition, dropouts are applied by default to the original measurement series, though not to the forecasts, since they are meant to simulate operational issues. This consists of:

   - **Short-term Dropouts:** A fraction of the data points is randomly set to zero with probability p_{drop}.
   - **Longer Period Dropouts:** Occasionally (probability p_{long}), longer periods (up to several days) are replaced with zeros, mimicking extended sensor or communication failures.

   The length of a long dropout period is sampled from a uniform distribution:

$$T_{\text{long}} \sim U(T_{\text{min}}, T_{\text{max}}) \tag{22}$$

   where T_{min} and T_{max} define the minimum and maximum dropout durations (by default $T_{\text{min}} = 1$ minute), and $U(a,b)$ denotes the uniform distribution on the interval $[a,b]$.

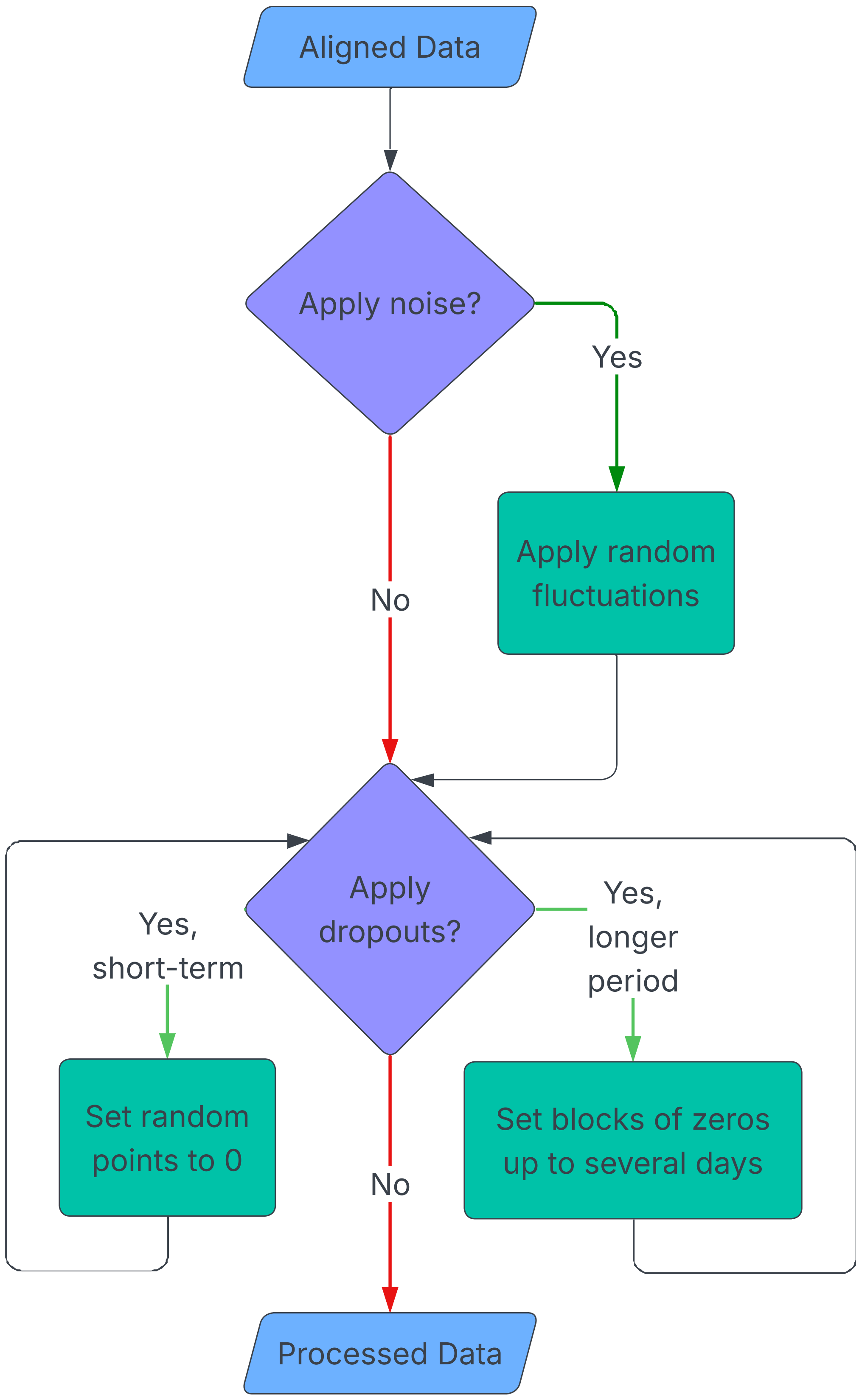


Figure 13: Data dropout and noise injection is the second part of the processing pipeline which can apply noise and dropout modifications to simulate real-world inconsistencies.

E PrioFlow: Priorities-Based Control Algorithm for Energy Management

The *PrioFlow* algorithm is a heuristic rule-based energy management strategy that makes use of priority values to determine the allocation of energy flows within the MARLOES environment. Since the connections between assets are sorted on priority when solving the SIMON model, this provides a simple way to prioritize flows between assets. This appendix gives details about the heuristics behind it.

E.1 Heuristics PrioFlow Algorithm

Overall the heuristics behind the PrioFlow algorithm are based on the simple rule that as much renewable energy produced as possible should be retained within the energy system. Intuitively, applying this rule maximally satisfies both the requirements for the self-sufficiency of the REV as well as that for emitting the fewest carbon-emissions [10]. The grid serves as a last priority target, that fills the necessary flows that would otherwise unbalance the system.
The resulting priority mapping has each asset first fulfilling demand, then attempting storage before making use of the grid. The exact values are shown in Table 24. Priorities are sorted in descending order, therefore connections with a higher priority value get allocated energy before the other connections.

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 24: The predefined priorities for used to rank target assets in PrioFlow.

E.2 Storage Devices

Storage devices such as batteries and electrolysers, however, always require a setpoint in order to perform any action by definition of the design within Simon. Therefore, a simple heuristic rule was designed for those assets specifically. The rule is as follows: if the net forecasted power balance over the entire system is positive (there is an energy surplus), the storage devices should attempt to charge that amount of energy in order to retain that energy. On the other hand, if the net forecasted power balance is negative, the storage devices should discharge the deficit to compensate for the missing energy. The decision process within PrioFlow is thus as follows: The net system power is distributed among all available storage devices based on their respective capacities. Then the setpoint for each device is computed in proportion to its capacity and as a fraction of its maximum power output or input. Specifically the action a_i for device i in the set of all storage devices is determined by:

$$a_i = \frac{-P_{net}^{for}(c_i/C)}{P_i^{max}}, \tag{23}$$

where C is the total capacity of all devices, c_i that of device i with max power P_i^{max} and P_{net}^{for} the net forecasted power. This method of allocation ensures that the “missing” net system power is shared equally among storage devices in proportion to their capacities. Given a perfect forecast, this method achieves optimal behavior when it comes to the minimization of CO_2 emissions.

F World Model details

Given the hierarchical structure of the developed world model, below we will give a bottom-up detail of its structure, supplied with its loss function and how states are flattened and reconstructed.

F.1 Structure

The lowest layer present in the model is the ForecastEncoder, a one-layered GRU with L_{forecast} hidden layers of size h_f, which is part of the AssetStateEncoder. The AssetStateEncoder further consists of two feedforward layers with ReLU activation, a hidden size of d_h^{asset} and an output size of $d_{\text{out}}^{\text{asset}}$. The forecast is passed through the ForecastEncoder before it is rejoined with the other state attributes and passed through the layers. All asset state encodings are merged in the WorldStateEncoder, together with the global context provided by the environment. This has the same layout as the AssetStateEncoder (hidden size $d_{\text{h}}^{\text{world}}$), with output size $d_{\text{out}}^{\text{world}}$. The final component is the WorldDynamicsModel, which is comprised of the world state encoding together with the joint actions and has the same double-layered feedforward structure, with hidden size d_h^{dyn}. It has two output heads: a reward head and a next state head which delivers the final predictions. The values of each of the parameters are detailed in Appendix C.

F.2 Loss Function

The world model is trained using mean squared error (MSE) for both the predicted next state and the predicted reward. For a minibatch of size B,

$$\mathcal{L} = \frac{1}{B}\sum_{b=1}^{B}\left(\left\|\hat{s}'^{(b)} - s'^{(b)}\right\|_2^2 + \left\|\hat{r}^{(b)} - r^{(b)}\right\|_2^2\right)$$

where $\hat{s}'$ is the predicted next state (after flattening), s' is the ground-truth next state, and $\hat{r}$, r are the predicted and true rewards, respectively. The next state vector is always flattened before loss computation, of which we will outline the procedure below.

F.3 State Flattening and Forecast Shifting

As stated above, the model operates on flattened state vectors. An exception on the default flattening method is the forecast. For efficiency the model predicts only the next value in each forecast sequence, rather than re-predicting the entire series. This reduces prediction complexity and preserves temporal continuity.

Example: For a forecast array $[f_1, f_2, ..., f_T]$ at time t, after unflattening, the next state's forecast becomes $[f_2, ..., f_T, \hat{f}_{T+1}]$ where $\hat{f}_{T+1}$ is the predicted value for the new time step.

G Grid Emission Factor

To determine the emissions for the grid, we make simplifications based on data from the Dutch CBS [42] about the national production in 2023. The breakdown of production and emission factors to determine the GWP of the grid is shown in Table 25

Source	**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	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 25: Breakdown of production, emission factors, and weighted contributions to grid emissions of the Dutch national grid in 2023.

Thus, the total emissions from the grid are:

$$\text{Grid Emissions} = 284.73\,\text{gCO}_2\text{eq/kWh}.$$

This value represents the average greenhouse gas emissions for the Dutch grid in 2023 based on the provided energy mix and emission factors.