Shielded Controller Units for RL with Operational Constraints Applied to Remote Microgrids

Hadi Nekoei, Alexandre Blondin-Massé, Rachid Hassani, Sarath Chandar, Vincent Mai

Keywords: Applied RL, Constrained RL, Shielding, Microgrid optimization, Energy dispatch

Summary

Reinforcement learning is a powerful approach to optimize decision making for complex systems under uncertainty which could be instrumental in industrial settings. An example application is remote microgrids, where wind turbines, fuel generators and batteries must be controlled to minimize fuel consumption and battery degradation given exogenous demand and available wind power. Such systems are often subject to extensive norms with diverse sets of complex constraints. Ensuring compliance of an RL agent applied on these systems requires interpretable guarantees regarding these constraints. In this paper, we present shielded controller units, a systematic and interpretable approach to using prior knowledge of the environment's dynamics to satisfy this requirement. Designed for real world deployment, our shield synthesis methodology decomposes the environment in a hierarchical structure where each shielded controller unit explicitly addresses a subset of constraints. We apply our shielded controller units on a remote microgrid optimization problem with industrial requirements, where the RL agent outperforms other baselines while satisfying every constraint.

Contribution(s)

1. This paper presents Shielded Controller Units, a systematic approach to decomposing white-box shield design into simpler shielding problems, making it easier to apply existing shielding approaches for complex industrial constrained environments.
   Context: Prior works established white-box shielding for constraint compliance (Krasowski et al., 2023; Hsu et al., 2023). However, shield design and interpretability becomes challenging for complex critical systems, impeding RL deployment for industrial problems. A close line of work is factored shielding (ElSayed-Aly et al., 2021; Melcer et al., 2024), which improves scalability in multi-agent RL by assigning localized shields to subsets of agents. In contrast, SCUs decompose the environment hierarchically using digital twins, enabling modular and interpretable shielding for complex real-world systems.
2. This paper presents a remote microgrid optimization under uncertainty problem, aimed at training and testing an RL agent with realistic industrial considerations and constraints.
   Context: Microgrid optimization under uncertainty is a challenging decision making problem for non-learning based approaches. Prior works (Wang et al., 2024) have shown the potential of RL for such settings, but often lack realistic considerations and constraint compliance guarantees.
3. Shielded Controller Units are applied to microgrid optimization, using several simple shields designs to ensure constraint compliance, while the resulting RL agent outperforms current industry heuristics and standard constrained RL baselines for this problem.
   Context: This demonstrates the applicability of the Shielded Controller Units approach as it facilitates developing RL solutions for applications in real world, critical systems. Prior works (Eichelbeck et al., 2022) applying shielded RL to a similar problem show the complexity to synthesis a single shield layer for all constraints.

Shielded Controller Units for RL with Operational Constraints Applied to Remote Microgrids

Hadi Nekoei[1,2,3], **Alexandre Blondin-Massé**[4], **Rachid Hassani**[4], **Sarath Chandar**[1,3,5,6], **Vincent Mai**[4]
nekoeihe@mila.quebec

[1] Chandar Research Lab [2] Université de Montréal [3] Mila – Quebec AI Institute [4] IREQ – Hydro-Québec Research Institute [5] Polytechnique Montréal [6] CIFAR AI Chair

Abstract

Reinforcement learning (RL) is a powerful framework for optimizing decision-making in complex systems under uncertainty, a key challenge in real-world settings such as the energy transition. A representative example is remote microgrids that supply power to communities disconnected from the main grid. Enabling the energy transition in such systems requires coordinated control of wind turbines, fuel generators, and batteries to meet demand while minimizing fuel consumption and battery degradation under uncertain load and wind conditions. These systems must comply with extensive regulations and operational constraints, requiring interpretable guarantees when deploying RL agents. In this paper, we introduce Shielded Controller Units (SCUs), a systematic and interpretable approach that leverages prior knowledge of system dynamics to ensure constraint satisfaction. Our shield synthesis methodology decomposes the environment into a hierarchy where each SCU explicitly manages a subset of constraints. We demonstrate the effectiveness of SCUs on a remote microgrid optimization task with strict operational requirements. The resulting RL agent achieves a 24% reduction in fuel consumption without increasing battery degradation, outperforming current industry heuristics and standard constrained RL baselines while satisfying all constraints. We hope SCUs facilitate the safe deployment of RL agents. All code and supporting data are available at https://github.com/chandar-lab/SCU_RL_MicroGrid.

1 Introduction

Ensuring safe, sustainable, and reliable operation is critical for real-world control systems, particularly in the context of the energy transition. Systems such as remote microgrids serving communities without access to the main power grid must adhere to numerous operational constraints and regulations. Any deployed algorithm must satisfy these constraints with explicit and interpretable guarantees. This challenge has hindered the widespread adoption of reinforcement learning (RL) in critical industrial settings, despite RL's potential for solving complex optimization problems in real-world applications (Dulac-Arnold et al., 2019; Kober et al., 2013). Addressing this issue is essential for deploying RL in industrial environments, where incorrect decisions may lead to equipment damage, system failures, safety incidents, or significant economic losses (García & Fernández, 2015; Amodei et al., 2016).

White-box shielding is a promising approach to address these requirements (Alshiekh et al., 2018; Jansen et al., 2018). Synthesized from prior knowledge of environment dynamics, a white-box shield verifies that an agent's action does not violate constraints and replaces it with a compliant action when necessary. Because the shield is expressed in human-understandable logic, it can be audited

by system experts and tuned to handle system uncertainty. Significant progress has been made in shield synthesis (Könighofer et al., 2020; Bloem et al., 2020) and white-box shielding (Hsu et al., 2023; Krasowski et al., 2023). However, in complex systems involving many constraints across multiple devices and time horizons, designing a provable shield remains challenging. Shielding approaches typically require expressing constraints logically, simulating environment rollouts, and designing fallback policies, which can be tedious and error-prone. A methodology that decomposes shielding into simpler subproblems and proves compliance independently for each constraint would significantly facilitate this process and broaden the applicability of RL in critical settings.

In this paper, we propose *Shielded Controller Units* (SCUs) for RL, a systematic methodology designed to address this challenge. The environment is decomposed into SCUs, each linking a controller to a system while enforcing a set of constraints. Systems can be devices or groups of lower-level SCUs, forming a hierarchical structure. Each controller consists of a shielded action dispatcher and a digital twin of the corresponding system. The digital twin combines a state estimator and a simulator to predict system behavior from sensor measurements. The dispatcher uses prior knowledge and the digital twin to validate actions before executing them on the real system. Each SCU includes a shield guaranteeing compliance with its associated constraints. This structure ensures system-wide compliance while substantially simplifying the design of provable shields for complex environments. Assuming accurate digital twins and correct local shields, the SCU hierarchy guarantees that executed actions satisfy operational constraints at every level.

We demonstrate SCUs on the problem of remote microgrid optimization. Remote microgrids are isolated power systems typically powered by fossil fuel generators (*gensets*). To reduce emissions and operational costs, utilities increasingly integrate wind turbines. Due to wind intermittency, large batteries provide flexibility while gensets ensure reliable supply. The power dispatch problem aims to coordinate batteries and gensets to minimize fuel consumption and battery degradation. This sequential decision-making problem involves long-term planning, uncertain exogenous variables such as demand and wind, and many operational constraints. When modeled in a realistic industrial scenario, it is difficult for traditional optimization methods, making it a compelling use case for shielded RL. Using our SCUs, an RL agent based on the Soft Actor–Critic (SAC) algorithm outperforms baselines while guaranteeing compliance with every constraint. For comparison, we also evaluate a Lagrangian variant of Soft Actor–Critic (SAC-Lag), a common constrained RL approach that enforces safety constraints through dual penalty variables. Although SAC-Lag successfully reduces constraint violations, it learns a more conservative policy and consumes more fuel than SAC combined with our SCU-based shielding. These results highlight the limitations of penalty-based constraint enforcement for learning high-performing policies under strict operational constraints. In contrast, SCUs provide a structured and interpretable mechanism for enforcing safety, allowing the RL agent to focus on optimizing the operational objective while compliance is guaranteed by design.

This paper makes three contributions:

- We introduce Shielded Controller Units (SCUs), a systematic approach to white-box shield design that improves interpretability while ensuring constraint compliance for RL agents in complex industrial contexts.
- We present a realistic remote microgrid optimization problem under uncertainty for training and evaluating RL agents under real-world operational constraints.
- We demonstrate that SCUs enable an RL agent to outperform existing baselines while guaranteeing constraint compliance in the microgrid control problem.

2 Related work

Constrained RL seeks to optimize policies under safety constraints and is an active area of research (Wachi et al., 2024; Liu et al., 2021). Black-box approaches typically rely on cost signals to discourage unsafe behavior, including methods such as CPO (Achiam et al., 2017) and CVPO (Liu et al., 2022), as well as Lagrangian techniques (Achiam et al., 2017; Ha et al., 2020; Honari et al.,

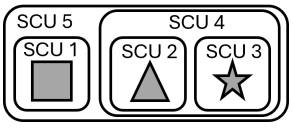
2024) that incorporate constraint violations into the objective through dual penalty variables. While flexible, these approaches often lack interpretability and can struggle to reliably learn safe policies in complex environments (Mani et al., 2025). Shielding approaches attempt to address safety more directly by blocking unsafe actions during execution, for example by training classifiers to detect violations (Waga et al., 2022). However, such methods depend heavily on environmental knowledge and may be difficult to tune in practice. White-box shielding instead leverages known dynamics. Early work (Bloem et al., 2012; Alshiekh et al., 2018) used formal methods to synthesize shields, while later efforts incorporated runtime monitoring (Fulton & Platzer, 2018), probabilistic safety constraints (Bouton et al., 2019; Yang et al., 2023), and adaptive interventions via model predictive control (Banerjee et al., 2024; Gerold & Lucia, 2025). Reviews (Krasowski et al., 2023; Hsu et al., 2023) summarize the tradeoffs between black-box and model-based approaches. However, these approaches can be difficult to deploy in practice. They often require formal logic, simulated rollouts, and/or fallback policies, which are tedious and error-prone to design in complex systems. We address this by introducing a practical, modular methodology that decomposes safety enforcement into simpler shielding sub-problems, making shield design easier. The closest line of work is factored shielding (ElSayed-Aly et al., 2021; Melcer et al., 2024), which improves scalability in multi-agent RL by assigning localized shields to subsets of agents. While factored shielding decomposes safety enforcement across agents in multi-agent environments, SCUs instead decompose safety along the physical hierarchy of the controlled system. This allows to design shields that ensure both device-level constraints and higher-level operational constraints that couple multiple components.

Microgrid optimization aims to manage energy sources for reliable, low-emission, and sustainable operation, accounting for battery degradation. This requires long-term planning under uncertainty (e.g., power demand, wind), which challenges traditional methods due to mixed variables and nonlinear dynamics. Mixed-integer programming has had some success (Hajimiragha & Zadeh, 2013; Silvente et al., 2015; Hirwa et al., 2022), but often oversimplifies constraints and ignores battery depletion. A more realistic model (Lambert & Hassani, 2023) supports real-time control of gensets but excludes batteries and wind. RL-based approaches show promise (Dimeas & Hatziargyriou, 2010; Foruzan et al., 2018; Yang et al., 2021; Wang et al., 2024), including constrained RL formulations for microgrid operation (Ye et al., 2023) and digital-twin-based optimization frameworks combining RL with high-fidelity system models (Özkan et al., 2024), though many approaches still lack industrial realism or strong safety guarantees. Eichelbeck et al. (2022) apply RL for energy dispatch under similar constraints, using an action projection shield, though replacing fuel genset dynamics with islanding robustness.

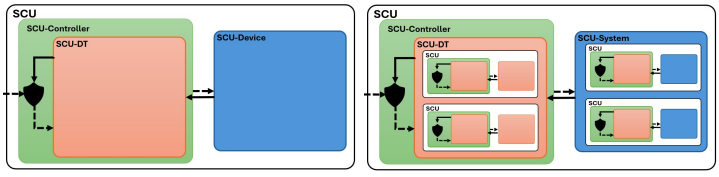
3 Shielded controller units

In complex, critical environments like real-world industrial problems, numerous constraints must be respected across systems with multiple devices. Some constraints pertain to individual devices, while other concern groups of devices, binding their control together. Ensuring compliance with a constraint sometimes requires considering future environment states to guarantee that an action is recoverable. To satisfy operational norms, any agent deployed on such complex system must interpretably ensure compliance with these constraints, despite uncertainties from exogenous variables.

Our proposed methodology addresses this problem systematically by decomposing the environment into Shielded Controller Units (SCUs). This decomposition is based on analyzing the set of con-



(a) Example of SCU hierarchy. Shapes represent devices in a complex environment.



(b) General SCU structure. Left: SCU for a single device. Right: higher-level SCU for a composite system.

Figure 1: Illustrations of Shielded Controller Units (SCUs) at different levels of abstraction.

straints. Every constraint pertains to a subset of the environment's devices, and constraints corresponding to the same subset are implemented in the same SCU. SCUs are implemented hierarchically, meaning they cannot overlap without one containing the other. For example, consider a system composed of square, triangle and star devices as shown in Figure 1a. Each devices has its own constraints, and thus its own SCU: 1, 2 and 3. Two additional constraints might concern the triangle and the star: they would be managed by SCU 4. If another constraint links the triangle and square devices, whether or not the star is involved, it cannot be treated outside of SCU 4. Therefore, SCU 5 will contain both SCU 1 and SCU 4.

3.1 Elements of shielded controller units

An SCU links a shielded controller to a real system. The controller receives an action from a higher-level SCU, ensures it respects constraints at its level, and relays it to the real system. It contains two elements: a digital twin and a shielded dispatcher. The shielded dispatcher enforces constraints by checking each action and sending a corrected one if needed for every system subcomponent. Many white-box shielding methods exist; choices depend on constraint type and prior knowledge. Some rely on state estimation (Carr et al., 2022) or model predictive shielding (Bastani, 2020; Banerjee et al., 2024). This need is met by the digital twin, combining a state estimator updating internal state from sensors and a simulator predicting system behavior from the current state and action sequence. Importantly, the digital twin must simulate all SCUs of components to ensure compliance with constraints. This includes the digital twin for the real system *and* the digital twin inside the SCU controller, as shown in Figure 1b.

3.2 Step function and interactions

The SCU structure operates hierarchically, processing the action by the agent from the top level SCU down to the lower levels, and propagating the information from the devices sensors back up to the agent while updating the various levels of digital twins. This process can be realized with a single step function called from the top SCU, as described in algorithm 1. It is divided in three substeps.

1. First, at line 3, the shield dispatcher receives action a_t from the higher level and dispatches it into complying actions $\{a_t^{\text{comp},i}\}_{i=1\ldots k}$ using function SHIELD. This function is designed ad-hoc via an interpretable shielding method, potentially with digital twin U^{DT}.
2. Next, lines 4 to 11, the safe actions are propagated to the real system's STEP function. If it is a device, it returns observation o_{t+1}. If it is a system of SCUs, it will return set $\{\hat{s}_{t+1}^i\}_{i=1\ldots k}$ of state estimations of the subcomponents' SCUs. These are aggregated in observation o_{t+1}.
3. Finally, at line 12, the controller internal state s_{t+1}^{SC} and its numerical twin's $s_{t+1}^{\text{SC·DT}}$ are updated using function UPDATECONTROLLER in algorithm 3. s_{t+1}^{SC} is updated with ad-hoc function CONTROLLERSTATE and $s_{t+1}^{\text{SC·DT}}$ with function UPDATEDT (Algorithm 2). If the SCU is of a device, UPDATEDT estimates s_{t+1}^{DT} using ad-hoc estimation function STATEESTIM. If however the SCU has k SCUs as sub-components U^j, there are 2 elements to update per U^j: (1) digital twin $U^{j,\text{DT}}$ of the real device and (2) controller $U^{j,\text{SC}}$ and its internal digital twin $U^{j,\text{SC·DT}}$. This implies a new recursive function propagating o_{t+1} to lower levels j. It is disaggregated into o_{t+1}^j which are used for UPDATEDT on $U^{j,\text{DT}}$ and UPDATECONTROLLER on $U^{j,\text{SC}}$. Finally, the digital twin new state s_{t+1}^{DT} is the aggregation of lower level SCU states s_{t+1}^j.

3.3 Integration within the RL framework

The RL agent interacts with the environment using the usual Markov Decision Process (MDP) framework, but only through the highest level SCUs' step functions. These receive their respective parts of the action space's inputs and propagate them to their subcomponents down to the devices. The agent's observation is the aggregation of high-level SCUs' controllers' state estimation.

Algorithm 1 SCU step function

function STEP(U, a_t)
  $k \leftarrow U$.NBCOMPONENTS
  $\{a_t^{\text{comp},i}\}_{i=1\dots k} \leftarrow$ SHIELD(U^{DT}, a_t)
  if U.ISDEVICE **then**
    $o_{t+1} \leftarrow$ STEP($U^{\text{dev}}, a_t^{\text{comp},1}$)
  else
    for $i \in \{1 \dots k\}$ **do**
      $s_{t+1}^i \leftarrow$ STEP($U^i, a_t^{\text{comp},i}$)
    end for
    $o_{t+1} \leftarrow$ OBS(AGG($\{s_{t+1}^i\}_{i=1\dots k}$))
  end if
  $s_{t+1} \leftarrow$ UPDATECONTROLLER(U^{SC}, o_{t+1})
  return s_{t+1}^{DT}
end function

Algorithm 2 Digital twin update function

function UPDATEDT(U^{DT}, o_{t+1})
  if U.ISDEVICE **then**
    $s_{t+1}^{\text{DT}} \leftarrow$ StateEstim(s_t^{DT}, o_{t+1})
  else
    $k \leftarrow U$.NBCOMPONENTS
    $\{o_{t+1}^j\}_{j=1\dots k} \leftarrow$ Disagg(o_{t+1})
    for $j \in 1 \dots k$ **do**
      $s_{t+1}^{j,\text{SC}\cdot\text{DT}} \leftarrow$ UpdateDT($U^{j,\text{DT}}, o_{t+1}^j$)
      $s_{t+1}^j \leftarrow$ UpdateController($U^{j,\text{SC}}, o_{t+1}^j$)
    end for
    $s_{t+1}^{\text{DT}} \leftarrow$ Agg(s_{t+1}^j)
  end if
  return s_{t+1}^{DT}
end function

Algorithm 3 Controller update function

function UPDATECONTROLLER(U^{SC}, o_{t+1})
  $s_{t+1}^{\text{SC}} \leftarrow$ ControllerState(U^{SC}, o_{t+1})
  $s_{t+1}^{\text{SC}\cdot\text{DT}} \leftarrow$ UpdateDT($U^{\text{SC}\cdot\text{DT}}, o_{t+1}$)
  $s_{t+1} = [s_{t+1}^{\text{SC}}, s_{t+1}^{\text{SC}\cdot\text{DT}}]$
  return s_{t+1}
end function

4 Shielded remote microgrid optimization

We consider a remote microgrid composed of a wind turbine, a battery and two fuel gensets, generating electric power to meet the demand of a village. The objective of the power dispatch problem is to control these four devices to minimize genset fuel consumption and battery depreciation, while ensuring that the generated power satisfies the demand at every time step. In this problem, two variables are considered exogenous and beyond the agent's control: demand and available wind power. Specialized industry models can provide predictions to the agent, but they are uncertain. This problem, when realistically modeled to target industrial deployment, is an excellent test case for the SCU approach. It is of reasonable scale for in-depth analysis but complex enough that non-learning based methods struggle with real time control. RL is a powerful alternative, but it must provably respect the industrial constraints. In this section, we describe how we modeled the environment's dynamics, rewards and constraints. We also discuss how the environment is decomposed in SCUs (See Figure 2), the shielding approach for each controller, and define the MDP the RL agent interacts with. Refer to appendix C.3 for a detailed process of applying SCUs to this microgrid and appendix C.1 for a single-line diagram.

4.1 Exogenous variables

There are two exogenous variables in the environment, over which the agent has no control: the power demand δ_t and the available wind power $p_t^{\text{wind,avail}}$. We use real data gathered from remote microgrids over one year. Demand ranges from 180 to 540 kW, averaging 320 kW, while available wind power ranges from 0 to 400 kW, averaging 272 kW. The agent observes both variables at every time step and receives realistic predictions over a 30-point forecast at 15 minutes intervals, covering 7.5 hours. These predictions were generated using industry-standard approaches. Demand forecasts were produced

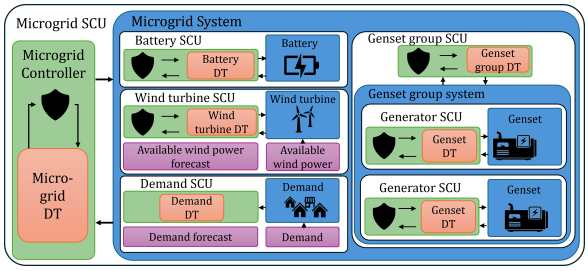


Figure 2: *SCU decomposition of the microgrid.*

using a general additive model (GAM) that combines transformations of the consumption observed the previous day and the previous week, along with observed and forecast temperatures (David et al., 2023). Wind power forecasts were generated by downscaling meteorological variables and converting them using an 8-direction power curve model. Forecasts were then fine-tuned using turbine-specific loss models, availability, and a statistical auto-adaptive model with real-time data.

4.2 Devices

The modeled microgrid includes three types of device: one wind turbine, two gensets, and one battery. Each device is managed by an SCU. The industrial norms they must comply with are described in **bold**. In this experiment, the digital twin sensor update function for devices and systems has been implemented as exact state update, which is realistic given the availability of precise sensors to measure relevant state variables for batteries, gensets and wind turbines.

Wind turbine The wind turbine generates renewable electric power for the microgrid by harnessing wind energy at no cost. At step t, it receives action $a_t = \bar{p}_t^{\text{wind}}$ containing the wind power curtailment setpoint. The resulting power p_t^{wind} is $\bar{p}_t^{\text{wind}}$ clipped between 0 and current available wind power $p_t^{\text{wind,avail}}$. In this power dispatch problem, **no operational constraints applies** on the wind turbine, so the shield directly relays $a_t^{\text{comp}} = a_t$ to the device.

Battery The battery stores chemical energy and can absorb or release electrical power p_t^{batt}. The power p_t^{batt} responds to the $\bar{p}_t^{\text{batt}}$ setpoint from action a_t, can be positive (discharge) or negative (charge), and is limited by the battery nominal power of 600 kW. The battery state of charge (SoC) indicates the accumulated energy as a percentage of the battery 672 kWh capacity. Charging and discharging are inefficient processes, leading to a 5% energy loss in each direction. We model the battery's physical behavior precisely according to Hassani & Lambert (2024). The battery's degradation over charging/discharging cycles $d_{b,t}$ is modeled in arbitrary units using an approximated online rainflow algorithm adapted for MDPs, inspired from Obermayr et al. (2021) and Kwon & Zhu (2022) and detailed in the appendix. To prevent damage to the battery, the **SoC is constrained between 90 and 10% of capacity**, or 5% as a reserve in case of emergency. The shield uses prior knowledge of the battery inner dynamics to determine the corresponding limits and accordingly clips setpoint $\bar{p}_t^{\text{batt}}$ received from a_t before relaying it as a_t^{comp} to the real battery.

Gensets Gensets can produce electric power from fuel energy. We model their behavior and constraints based on Lambert & Hassani (2023). Gensets receive action $a_t = (\Delta_t^{\text{gen}}, \bar{p}_t^{\text{gen}})$. Gensets's running status can be *on* or *off*, changed by *start* or *stop* command Δ_t^{gen}. Once *on*, their power production p_t^{gen} is controlled by setpoint $\bar{p}_t^{\text{gen}}$ ranging between 0 and maximum power of 440 kW. Fuel consumption is proportional to p_t^{gen} at 0.25 l/kWh, plus a constant 10 l/h when the genset is *on*.

To prevent long term damage, several operational constraints must be respected. When changing status, **gensets must complete routines**, like a 3-minute warm-up producing 100 kW or a 5-minute cool-down producing 0 kW. Once on, they have **minimum runtime** of 30 minutes during which they should not be turned *off*. Except during these routines, *on* gensets have a **minimal power production** of 120 kW. Producing the maximum 440 kW continuously can damage the device: **it should instead be capped at its nominal capacity** of 400 kW, except in emergencies where the 40 kW overload reserve can be used. Finally, to meet ISO 8528-1 standards (International Organization for Standardization (ISO), 2018), the **average power over the last 48 hours of operation is capped** at 70% of nominal capacity. The genset SCU controller ensures these constraints are respected by modifying violating Δ_t^{gen} and clipping $\bar{p}_t^{\text{gen}}$ from a_t before relaying them as a_t^{comp} to genset.

4.3 Systems

Two sets of operational constraints for the microgrid concern specific groups of devices, leading to two additional SCUs: the genset orchestrator and the microgrid.

Genset orchestrator The genset orchestrator manages the two gensets, its subcomponents, by dispatching action $a_t = (\Delta_t^{\text{orch}}, \bar{p}_t^{\text{orch}})$ into actions $a_t^{\text{comp},i} = (\Delta_t^{\text{gen},i}, \bar{p}_t^{\text{gen},i})$ complying with two operational constraints. First, genset i cannot be turned *on* if genset $i-1$ is turned *off*, and vice versa. This **priority order constraint** ensures compatibility with existing heuristic-based systems so they can take over if necessary, as described in Hassani & Lambert (2024). The shielded dispatcher follows a state-machine logic allowing it to turn *on* or *off* only one specific generator at the time. This allows to dispatch a_t's single status change command Δ_t^{orch} to *start* or *stop* a generator, or keep the current configuration, which the shield defaults to if Δ_t^{orch} is inapplicable.

The other constraint is the **equal power fraction** presented in Lambert & Hassani (2023): all running genset not in warmup or cooldown must run at the same percentage of their nominal powers. The orchestrator uses its digital twin to account for single genset constraints, such as the 70% maximum average power over 48h, as it may limit both gensets together. This constraint binds all $\bar{p}_t^{\text{gen},i}$ together: the orchestrator dispatches them in $a_t^{\text{comp},i}$ so that the total power produced by the gensets, p_t^{orch}, is as close as possible to the orchestrator setpoint $\bar{p}_t^{\text{orch}}$ received in a_t.

Microgrid The microgrid is the highest level SCU. Its objective is to ensure that **the balance, i.e. the difference between power generation and demand, is kept at 0 for all time steps**. A negative balance, or shortage, is a critical failure of the system, while a positive balance should also be avoided although it is easier to recover from. To enforce this constraint, the microgrid subcomponents are the battery SCU, the wind turbine SCU, and the genset orchestrator SCU. Each time step, the microgrid shield dispatches complying actions $a_t^{\text{comp},i}$ to its three subcomponents, as power setpoints $\bar{p}_t^{\text{batt}}$, $\bar{p}_t^{\text{wind}}$ and $\bar{p}_t^{\text{orch}}$, and orchestrator status change Δ_t^{orch}. The balance constraint, and a hard-coded wind over fuel priority, reduce the degrees of liberty of the received action a_t to Δ_t^{orch} and $\bar{p}_t^{\text{batt}}$.

To ensure 0 balance, shielding operates at two levels. At any given time step t, the subcomponents setpoints must ensure the sum of generated powers p_t^{batt}, p_t^{wind} and p_t^{orch} meets demand δ_t. It prioritizes respecting the agent's battery setpoint, then using wind before fuel. If the demand is not met due to the limits of wind and gensets, $\bar{p}_t^{\text{batt}}$ is adjusted. If all the wind power cannot be absorbed by the demand and the battery, it is curtailed. To ensure that every subcomponent constraint is accounted for, the microgrid digital twin simulates each SCU's response to given commands.

The shield must also avoid actions leading to irrecoverable future states, characterized by the genset orchestrator's inability to provide the power to balance the demand due to a genset being in cooldown/warm-up routine. Such blockages can take a maximum of 8 minutes. To prevent this, a *recovery shield* using a predictive shielding approach is deployed (Bastani, 2020; Hsu et al., 2023). Monitoring is done by simulating the environment with the digital twin over a 9-minute period, given a *fallback* genset orchestrator policy $\{\Delta_t^{\text{orch}}, \Delta_{t+1}^{\text{orch}}, \ldots, \Delta_{t+8}^{\text{orch}}\}$ with future $\Delta_{t+\tau}^{\text{orch}} = \text{Start}$ for $\tau \in 1, \ldots, 8$. There are three unknown variables when simulating the next steps: exogenous demand, available wind power, and agent battery commands. The recovery shield guarantee should hold irrespective of the precision of exogenous variable predictions, so they are not used here. Instead, two scenarios are considered: (1) The worst case scenario with reserve, where battery command is set to discharge to the maximum, and demand and wind are simulated as rising and decreasing, respectively, towards their historical high and low at their fastest historical rate of change. The genset and battery devices are simulated as able to access their emergency reserve power. (2) A constant scenario for wind and demand, but where simulated gensets and batteries are not allowed to access their reserves. Scenario (1) ensures a strong guarantee on the 0-balance constraint, while the (2) reduces reliance on reserve. If a negative balance is detected during one of these simulations, Δ_t^{orch} is considered as possibly non-compliant and is replaced by the least different recoverable option.

4.4 Markov Decision Process for the agent

As the microgrid SCU is at the highest level, it interfaces with the RL agent. The agent's action space is two-dimensional: a discrete value for Δ_t^{orch} and a continuous value for $\bar{p}_t^{\text{batt}}$. Its observation space includes the state estimation of the microgrid subcomponents and two time series of exogenous variable predictions – wind and demand. The reward is a negative cost combining the gensets fuel consumption $\mathbf{F}_{o,t}$ (in liters) and the battery degradation $d_{b,t}$ in arbitrary units: $r_t = -(\mathbf{F}_{o,t} + \alpha d_{b,t})$, where weight α can be adjusted based on the utility's objective. Between equally performing solutions in terms of cost, a controller minimizing reserve usage of batteries and gensets is preferred. Refer to Appendix C.2 for a more detailed definition of the microgrid's MDP.

5 Experimental results

5.1 Experimental setup

We used the Soft-Actor Critic (SAC) RL algorithm (Haarnoja et al., 2018), which is known for its robust performance in continuous domains, to train our agents. Although we also trained the RL agents using PPO (Schulman et al., 2017), we found that SAC consistently outperformed PPO in this setting. Therefore, we report the results for the SAC agents. To process the available wind power and demand prediction time series, the agent's architecture integrates LSTMs. Details about the agent's architecture can be found in the appendix. We also trained a Lagrangian variant of SAC (SAC-Lag), where the reward is modified as $r_t^{\text{lag}} = r_t - \sum_i \lambda_i w_i c_i(s_t, a_t)$, with one nonnegative multiplier λ_i per constraint component, raw cost c_i, and fixed weight w_i. The multipliers are updated by dual ascent at the end of each episode using the average episode cost $\lambda_i \leftarrow \left[\lambda_i + \eta_i(\bar{c}_i - d_i)\right]_+$, where η_i is the dual learning rate and d_i is the allowed cost limit for constraint i. The constraint costs correspond to power imbalance and genset overload. Learning agents were trained on 10^5 one-minute environment steps divided in one-day episodes. The episodes initialization involved sampling a random time during the year and the gensets and battery initial states. We explored several hyper-parameter configurations (detailed in the appendix) as well as 10 training seeds and three values of α for the SAC agent. Moreover, we implemented four control baselines providing action based on predetermined policies. Both the baselines and the trained agents were then tested in a fixed test environment consisting of the first 10-day periods of each month from February to November to represent varying conditions. To report the results, we averaged the performance across these 100 days. For the RL agent's performance, we averaged the performance and calculated the standard errors across the 10 training seeds. Note that all agents use shields during evaluation unless stated otherwise. We verified that none of the constraints are violated to confirm the validity of the shield.

5.2 Results

We designed the experimental section to address three research questions: (1) How do shielded RL agents perform compared to baselines in terms of minimizing fuel consumption and battery degradation? (2) How does the recovery shield impact the performance and compliance to constraints? (3) Given a shielded deployment environment, how to constrain an RL agent during training?

5.2.1 Performance analysis

Figure 3 compares the performance of our RL agents with four control baselines providing action $a_t = (\Delta_t^{\text{orch}}, \bar{p}_t^{\text{batt}})$ based on predetermined policies. The *random* agent uniformly samples discrete Δ_t^{orch} from (*start, stop, do nothing*), and continuous $\bar{p}_t^{\text{batt}}$ from maximum discharging to maximum charging power. It performs poorly, showing that relying on the SCU shield is not enough to guarantee good performance. The *battery greedy* agent lets the shield change the genset status and keeps the battery idle, with Δ_t^{orch} to *do nothing*, and $\bar{p}_t^{\text{batt}}$ to 0. The battery does not degrade but fuel consumption is not optimal. The *greedy* agent tries to turn the genset off with Δ_t^{orch} at *stop*, and

discharges the battery with maximal $\bar{p}_t^{\text{batt}}$. It consumes little fuel but highly degrades the battery. The *fuel-greedy* agent also tries to turn the genset off but charges the battery when available wind power is higher than the demand, and discharges otherwise.

The industry *heuristics* reproduces a policy currently deployed in industrial settings. It always keeps one genset *on*, and decides to turn the other genset *on* when current genset is at more than 90% of its available power (nominal or capped by the moving average constraint) for 5 minutes in a row, or directly if it reaches more than 100%. It turns the second genset *off* if the current total genset orchestrator power could be managed by 70% of the other genset's available power for 5 minutes in a row. On the battery side, it switches between two modes: charging, or discharging, to avoid fast cycling. In charging mode, it only charges $\bar{p}_t^{\text{batt}}$ to capture an eventual excess power from the wind. It does not use the genset power to charge. Discharge only happens in case of emergency, as enforced by the shield. The battery charges until achieving 90% SoC, and then switches to discharging mode. In discharging mode, $\bar{p}_t^{\text{batt}}$ is set to maximum discharging power until the battery reaches 10% SoC. It then switches back to charging mode. Performance-wise, it provides a relevant comparison point in terms of trade-off between both metrics.

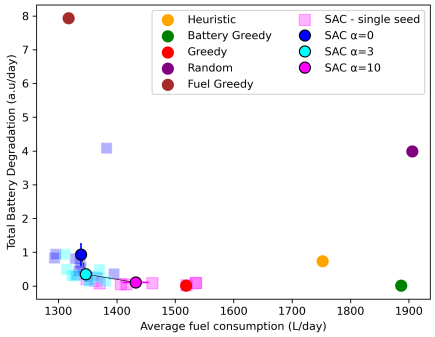
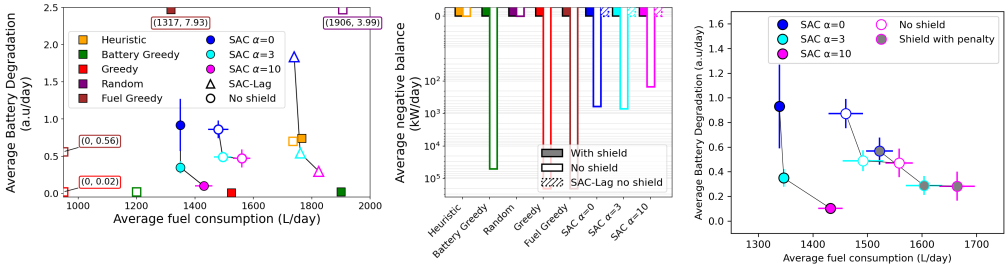


Figure 3: *The RL agent seeks to strike a balance between fuel consumption and battery degradation costs. Changing their respective importance in the reward function with α appears to trace the Pareto frontier. In contrast, all other baselines exhibit suboptimal performance in one or both metrics.*

The RL agents trained with SAC perform significantly better, improving fuel efficiency by 24% for similar battery degradation. Modifying parameter α to balance battery degradation cost and fuel consumption in the reward function seems to exhibit a Pareto frontier. This could allow the electric utilities to trade-off battery lifetime and fuel efficiency at their preference while striking the high gain zone for remote microgrid fuel consumptions.

5.2.2 Impact of the recovery shield

In this experiment, we explore the importance of recovery shielding for both RL agents and baseline algorithms. The recovery shield is the most complex shield in this system, addressing the uncertainty of exogenous variables and requiring multi-step simulation for compliance verification. To ensure



(a) *Left: Impact of the recovery shield on fuel consumption and battery degradation. Without shielding, SAC learns a more conservative policy, while SAC-Lag enforces the constraints but consumes more fuel than SAC with shielding. Right: Average negative balance violations. Several agents violate the constraint without shielding, whereas SAC-Lag and SAC with recovery shielding avoid violations.*

(b) *Shield penalties during training reduce performance. The agent trained with recovery shielding and no penalty performs best, while agents trained without shielding or with shield penalties learn more conservative policies.*

Figure 4: *Performance of agents with and without recovery shielding .*

compliance in every situation, the recovery shield considers worst-case scenario which could impair performance due to over restrictiveness. Figure 4a presents the performance metrics and negative balance violations, with and without recovery shield. Battery-greedy agent without recovery shield has a relatively good performance but it is not acceptable due to violating the negative balance constraint. Fuel-greedy agent without recovery shield does barely consume any fuel but it also show high amount of negative balance violation. On the other hand, Heuristic policy has a stable performance even without recovery shield which shows the validity of its policy.

Finally, SAC agents trained and deployed without recovery shield, but penalized for negative balance violation exhibit increased fuel consumption and slightly higher battery degradation than their shielded counterparts, while still sometimes experiencing negative balance (right subplot). We hypothesize that this occurs because they adopt an overly conservative and suboptimal policy due to the substantial penalties associated with negative balance during training. Using a constrained RL approach such as SAC-Lag, the agent successfully learned to avoid the constraints; however, at the cost of a more conservative policy, which consumes more fuel compared to SAC with our shielding approach. It is noteworthy that the RL agent with shielding shows much smaller standard errors in evaluated variables, demonstrating the robustness of training the shielded RL agent.

5.2.3 How to train an RL agent given a shield?

A recurring research question is how penalizing the use of shielding during training impacts the agent's ability to learn an optimal policy (Hsu et al., 2023). We explore this question for our use case by evaluating three variants of the RL agents in Figure 4b: the first variant (full color) was trained with recovery shielding but without penalizing the shield intervention. The second variant (gray) was trained with recovery shield and a penalty for shield intervention, and the third agent (white) was trained without recovery shield but with a penalty for violating the balance constraint.

The agent trained with recovery shielding and no penalty (solid circle) achieves best performance. Both the agents trained either with no shield (empty circle) or the agent trained with penalty (gray circle) learn to be more conservative due to the negative balance or shield penalty. Over-conservativeness is a known issue in punishing RL environments (Mani et al., 2025). Indeed, we notice these agents keep gensets *on* longer which decreases the chance of receiving a penalty. Interestingly, the shield penalty seems to encourage the agent to charge the battery more probably because the shield penalty is harder to predict than the negative balance penalty for the agent.

6 Discussion

While remote microgrid optimization is complex, it remains at a sufficiently small scale that it can be analyzed in depth. Eichelbeck et al. (2022) have shown that, for a very similar problem, a provable, single shield can be hand-designed to respect constraints without significantly impairing the performance. However, their paper shows the challenge of designing such a shield to comply with all constraints at once. Our approach's applicability stands out in comparison: for every SCU, the shield is simple as it only focuses on a specific set of constraints: the others are accounted for by the SCU structure. In most cases, fallback policies are evident. Handling uncertain dynamics, such as with exogenous variables for the microgrid SCU, can be managed with the necessary level of restrictiveness. In addition, SCUs allow to analyze the degrees of freedom that each constraints removes from the RL agent, and allow it to focus on these instead of outputting actions for every device. Although our experiments focus on remote microgrid optimization, the SCU methodology is not specific to this domain. Many real-world control systems are composed of interacting subsystems organized in hierarchical structures. Examples include robotic platforms, transportation networks, and industrial process control systems. In such settings, SCUs provide a structured way to decompose safety verification across subsystems while maintaining system-level constraint compliance.

A potential limitation of the SCU approach is the duplication of lower-level digital twins inside higher-level digital twins. This allows higher-level shielded controllers to consider the lower-level

constraints when using rolling out strategies, but could lead to an exponential number of digital twins depending on the number of SCU levels. However, in moderately complex real world cases, the number of SCU levels can be expected to be small (Siyari et al., 2019), such that scaling should not be a limiting factor. As an example, in our case the SCU hierarchy has depth 3, and each microgrid SCU step—including future projections—executes in under 0.05 seconds on a standard CPU, demonstrating that practical deployments remain well within real-time control. In many industrial control environments, digital twins are already deployed for monitoring, simulation, and predictive maintenance. The SCU framework leverages these existing models rather than introducing additional modeling requirements. In cases where the digital twin is imperfect, the shielding logic can incorporate conservative bounds or worst-case assumptions, as demonstrated in the recovery shield for the microgrid system.

7 Conclusion

This paper presented Shielded Controller Units, a decomposition approach facilitating the design of white-box shields to ensure compliance for complex constrained systems controlled by RL agents. SCUs allow to design simple shields for every constraint, which are then combined hierarchically. Digital twins pertaining to each SCU are updated at every time step so higher-level units shields can rollout simulations accounting for lower-level constraints. SCUs were applied on a realistic remote microgrid optimization problem, ensuring the RL agent complies with every operational constraints. Every shield level was simple to design, describe and interpret, showcasing the relevance of SCU for provable RL compliance in complex real-world settings.

By facilitating deployment of RL to critical systems requiring interpretable guarantees, SCUs can allow us to benefit from the potential of RL for optimizing decision-making with important social impact. We demonstrated this by showing how RL agents when supported by SCUs can reduce fuel consumption by 24% for similar battery degradation compared to industry heuristics and standard constrained RL baselines. We hope SCUs can be valuable for RL deployment in other regulated systems in the fields of energy, healthcare, transportation, etc.

Broader Impact Statement

SCUs aim at facilitating the proof of constraint compliance for RL agents in complex systems and thus enhance RL applicability to critical industrial settings. This is a necessary step towards capitalizing on the potential of RL for real world applications. In the case of power systems such as microgrids, this could facilitate the integration of renewable energy sources towards the energy transition.

Acknowledgments

The authors would like to thank Mathieu Lambert for his precious advice, and Slavica Antic for her help gathering wind turbine data. This research was conducted with the support of Mitacs Accelerated Grants and facilitated through computational resources provided by IREQ – Hydro-Québec Research Institute.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization, 2017. URL https://arxiv.org/abs/1705.10528.

Mohammed Alshiekh, Roderick Bloem, Robert Könighofer, Laura Nimmermann, Anna-Katharina Schmuck, and Stefan Weinberger. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Arko Banerjee, Kia Rahmani, Joydeep Biswas, and Isil Dillig. Dynamic model predictive shielding for provably safe reinforcement learning. (arXiv:2405.13863), Dec 2024. URL http://arxiv.org/abs/2405.13863. arXiv:2405.13863 [cs].

Osbert Bastani. Safe reinforcement learning with nonlinear dynamics via model predictive shielding, 2020. URL https://arxiv.org/abs/1905.10691.

Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

Roderick Bloem, Peter Gjøl Jensen, Bettina Könighofer, Kim Guldstrand Larsen, Florian Lorber, and Alexander Palmisano. It's time to play safe: Shield synthesis for timed systems. (arXiv:2006.16688), Jun 2020. DOI: 10.48550/arXiv.2006.16688. URL http://arxiv.org/abs/2006.16688. arXiv:2006.16688 [cs].

Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J. Kochenderfer, and Jana Tumova. Reinforcement learning with probabilistic guarantees for autonomous driving. (arXiv:1904.07189), May 2019. DOI: 10.48550/arXiv.1904.07189. URL http://arxiv.org/abs/1904.07189. arXiv:1904.07189 [cs].

Jun Cao, Dan Harrold, Zhong Fan, Thomas Morstyn, David Healey, and Kang Li. Deep reinforcement learning-based energy storage arbitrage with accurate lithium-ion battery degradation model. *IEEE Transactions on Smart Grid*, 11(5):4513–4521, September 2020. ISSN 1949-3053, 1949-3061. DOI: 10.1109/TSG.2020.2986333.

Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe reinforcement learning via shielding under partial observability. (arXiv:2204.00755), Aug 2022. DOI: 10.48550/arXiv.2204.00755. URL http://arxiv.org/abs/2204.00755. arXiv:2204.00755 [cs].

Charline David, Alexandre Blondin Massé, and Arnaud Zinflou. Fast short-term electrical load forecasting under high meteorological variability with a multiple equation time series approach. *International Journal of Electrical and Computer Engineering*, 17(10):234 – 241, 2023. ISSN eISSN: 1307-6892. URL https://publications.waset.org/vol/202.

Thomas G. Dietterich, George Trimponias, and Zhitang Chen. Discovering and removing exogenous state variables and rewards for reinforcement learning. *CoRR*, abs/1806.01584, 2018. URL http://arxiv.org/abs/1806.01584.

A. L. Dimeas and N. D. Hatziargyriou. Multi-agent reinforcement learning for microgrids. In *IEEE PES General Meeting*, pp. 1–8, July 2010. DOI: 10.1109/PES.2010.5589633.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

Michael Eichelbeck, Hannah Markgraf, and Matthias Althoff. Contingency-constrained economic dispatch with safe reinforcement learning. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 597–602, Dec 2022. DOI: 10.1109/ICMLA55696.2022.00103. URL http://arxiv.org/abs/2205.06212. arXiv:2205.06212 [cs, eess].

Ingy ElSayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. Safe multi-agent reinforcement learning via shielding. *arXiv preprint arXiv:2101.11196*, 2021.

E. Foruzan, L.-K. Soh, and S. Asgarpoor. Reinforcement learning approach for optimal distributed energy management in a microgrid. *IEEE Transactions on Power Systems*, 33(5):5749–5758, September 2018. DOI: 10.1109/TPWRS.2018.2823641.

Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(11), Apr 2018. ISSN 2374-3468. DOI: 10.1609/aaai.v32i1.12107. URL https://ojs.aaai.org/index.php/AAAI/article/view/12107.

Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Hilde Gerold and Sergio Lucia. Safe reinforcement learning via adaptive robust model predictive shielding. *Computers & Chemical Engineering*, 206:109521, 12 2025. DOI: 10.1016/j.compchemeng.2025.109521.

Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort, 2020. URL https://arxiv.org/abs/2002.08550.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

A. H. Hajimiragha and M. R. D. Zadeh. Research and development of a microgrid control and monitoring system for the remote community of bella coola: Challenges, solutions, achievements and lessons learned. In *2013 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, pp. 1–6, August 2013. DOI: 10.1109/SEGE.2013.6707898.

Rachid Hassani and Mathieu Lambert. Real-time battery optimization for remote microgrids. Les Cahiers du GERAD G-2024-49, Groupe d'études et de recherche en analyse des décisions, GERAD, Montréal QC H3T 2A7, Canada, August 2024.

J. Hirwa, O. Ogunmodede, A. Zolan, and A. M. Newman. Optimizing design and dispatch of a renewable energy system with combined heat and power. *Optimization and Engineering*, 23(3): 1–31, September 2022. DOI: 10.1007/s11081-021-09674-4.

Homayoun Honari, Amir Mehdi Soufi Enayati, Mehran Ghafarian Tamizi, and Homayoun Najjaran. Meta sac-lag: Towards deployable safe reinforcement learning via metagradient-based hyperparameter tuning, 2024. URL https://arxiv.org/abs/2408.07962.

Kai-Chieh Hsu, Haimin Hu, and Jaime Fernández Fisac. The safety filter: A unified view of safety-critical control in autonomous systems. (arXiv:2309.05837), Sep 2023. DOI: 10.48550/arXiv.2309.05837. URL http://arxiv.org/abs/2309.05837. arXiv:2309.05837 [cs, eess].

International Organization for Standardization (ISO). Reciprocating internal combustion engine driven alternating current generating sets — part 1: Application, ratings and performance, 2018. URL https://www.iso.org/standard/68539.html. Edition 3.

Nils Jansen, Bettina Könighofer, Sebastian Junges, and Roderick Bloem. Shielded decision-making in mdps. In *arXiv preprint arXiv:1807.06096*, 2018.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods in System Design*, 57:79–115, 2020.

Hanna Krasowski, Jakob Thumm, Marlon Müller, Lukas Schäfer, Xiao Wang, and Matthias Althoff. Provably safe reinforcement learning: Conceptual analysis, survey, and benchmarking. (arXiv:2205.06750), Nov 2023. DOI: 10.48550/arXiv.2205.06750. URL http://arxiv.org/abs/2205.06750. arXiv:2205.06750 [cs].

Kyung-Bin Kwon and Hao Zhu. Reinforcement learning-based optimal battery control under cycle-based degradation cost. *IEEE Transactions on Smart Grid*, 13(6):4909–4917, Nov 2022. ISSN 1949-3053, 1949-3061. DOI: 10.1109/TSG.2022.3180674.

M. Lambert and R. Hassani. Diesel genset optimization in remote microgrids. *Applied Energy*, 340: 121036, June 2023. DOI: 10.1016/j.apenergy.2023.121036.

Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4508–4515. International Joint Conferences on Artificial Intelligence Organization, 8 2021. DOI: 10.24963/ijcai.2021/614. URL https://doi.org/10.24963/ijcai.2021/614. Survey Track.

Zuxin Liu, Zhepeng Cen, Vladislav Isenbaev, Wei Liu, Zhiwei Steven Wu, Bo Li, and Ding Zhao. Constrained variational policy optimization for safe reinforcement learning, 2022. URL https://arxiv.org/abs/2201.11927.

Kaustubh Mani, Vincent Mai, Charlie Gauthier, Annie S Chen, Samer B. Nashed, and Liam Paull. Risk informed policy learning for safer exploration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gJG4IPwg6l.

Daniel Melcer, Christopher Amato, and Stavros Tripakis. Shield decomposition for safe reinforcement learning in general partially observable multi-agent environments. *Reinforcement Learning Journal*, 4:1965–1994, 2024.

Martin Obermayr, Christian Riess, and Jürgen Wilde. A novel online 4-point rainflow counting algorithm for power electronics. *Microelectronics Reliability*, 120:114112, May 2021. ISSN 00262714. DOI: 10.1016/j.microrel.2021.114112.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

J. Silvente, G. M. Kopanos, E. N. Pistikopoulos, and A. Espuña. A rolling horizon optimization framework for the simultaneous energy supply and demand planning in microgrids. *Applied Energy*, 155:485–501, October 2015. DOI: 10.1016/j.apenergy.2015.05.090.

Sean R. Sinclair, Felipe Frujeri, Ching-An Cheng, Luke Marshall, Hugo Barbalho, Jingling Li, Jennifer Neville, Ishai Menache, and Adith Swaminathan. Hindsight learning for mdps with exogenous inputs. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Payam Siyari, Bistra Dilkina, and Constantine Dovrolis. Emergence and evolution of hierarchical structure in complex systems. In *Dynamics On and Of Complex Networks III: Machine Learning and Statistical Physics Approaches 10*, pp. 23–62. Springer, 2019.

Akifumi Wachi, Xun Shen, and Yanan Sui. A survey of constraint formulations in safe reinforcement learning, 2024. URL https://arxiv.org/abs/2402.02025.

Masaki Waga, Ezequiel Castellano, Sasinee Pruekprasert, Stefan Klikovits, Toru Takisaka, and Ichiro Hasuo. Dynamic shielding for reinforcement learning in black-box environments, 2022. URL https://arxiv.org/abs/2207.13446.

Y. Wang, M. Xiao, Y. You, and H. V. Poor. Optimized energy dispatch for microgrids with distributed reinforcement learning. *IEEE Transactions on Smart Grid*, pp. 1–1, 2024. DOI: 10.1109/TSG.2023.3331467.

Bolun Xu, Alexandre Oudalov, Andreas Ulbig, Goran Andersson, and Daniel S. Kirschen. Modeling of lithium-ion battery degradation for cell life assessment. *IEEE Transactions on Smart Grid*, 9 (2):1131–1140, Mar 2018. ISSN 1949-3053, 1949-3061. DOI: 10.1109/TSG.2016.2578950.

T. Yang, L. Zhao, W. Li, and A. Y. Zomaya. Dynamic energy dispatch strategy for integrated energy system based on improved deep reinforcement learning. *Energy*, 235:121377, November 2021. DOI: 10.1016/j.energy.2021.121377.

Wen-Chi Yang, Giuseppe Marra, Gavin Rens, and Luc De Raedt. Safe reinforcement learning via probabilistic logic shields, 2023. URL https://arxiv.org/abs/2303.03226.

Yujian Ye, Hongru Wang, Peiling Chen, Tang Yi, and G. Strbac. Safe deep reinforcement learning for microgrid energy management in distribution networks with leveraged spatial–temporal perception. *IEEE Transactions on Smart Grid*, 14:3759–3775, 02 2023. DOI: 10.1109/TSG.2023.3243170.

Erol Özkan, Ibrahim Kok, and Suat Ã–zdemİr. Deeptwin: A deep reinforcement learning supported digital twin model for micro-grids. *IEEE Access*, PP:1–1, 01 2024. DOI: 10.1109/ACCESS.2024.3521124.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Agent architecture

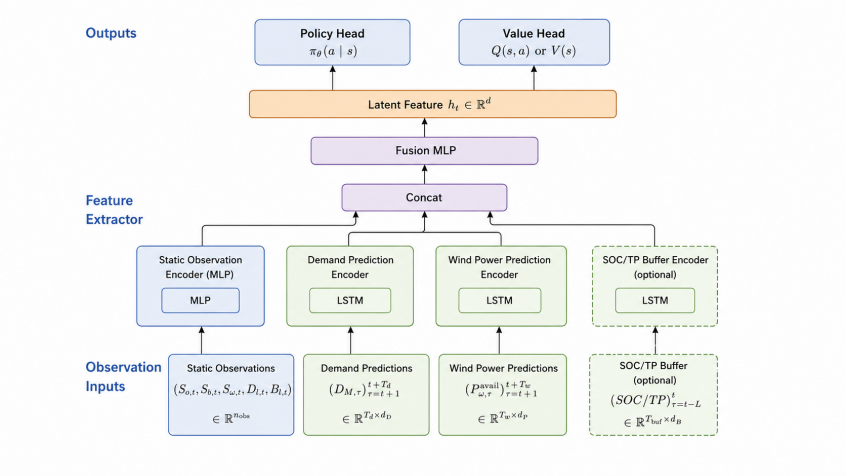


Figure 5: The agent architecture

All RL agents use SAC for decision making. Both the actor and the critic use the same architecture (See Figure 5). The state is formatted as a vector and fed into a multi-layer perceptron (MLP). To take into account the predicted future demand and wind power, we added two LSTM modules which takes the demand and wind power predictions and concatenate their hidden representations with the encoded representation of the state. The final representation is passed through another MLP to compute the action probabilities and state values.

B Learning hyper-parameters

In Table 1, we provide the values for the learning agent and the optimization hyper-parameters.

SAC agent		
Observation encoder	MLP	2 layers (128, 32)
Hidden dimension size	H	3×32
Optimization		
Learning Rate	lr	0.001, **0.0003**
Batch size	B	**32**, 64

Table 1: Learning Hyper-parameters. The final selected values are in bold.

C Microgrid optimization details

C.1 Single-line diagram of the microgrid

Figure 2 presents the microgrid through its *control* hierarchy (the SCU decomposition). For completeness, and to make the physical setup explicit to readers from the power-systems community, Figure 6 gives the corresponding *electrical* single-line diagram. All controllable devices and the village load connect to a common AC bus. The two dispatchable gensets and the wind turbine feed

the bus, the battery exchanges power through its power-conversion system (PCS), and the exogenous village load δ_t draws from it. Device ratings are those used in the simulator (Section 4.2); the italic gray labels recall the control setpoints the corresponding SCU receives, linking each electrical component to the agent's action. The zero-balance constraint enforced by the microgrid SCU is precisely the requirement that net injection at this bus equals the load δ_t at every time step.

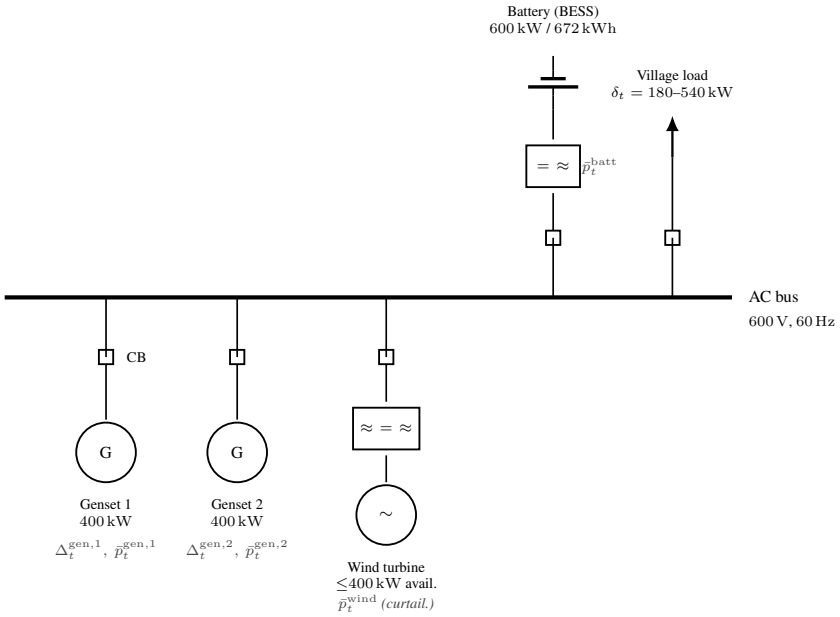


Figure 6: *Single-line diagram of the remote microgrid. Two fuel gensets and a variable-speed wind turbine (through an AC/DC/AC converter) feed a common AC bus; a battery energy-storage system (BESS) exchanges power via its PCS inverter; the village load draws the exogenous demand δ_t. Each feeder includes a circuit breaker (CB). Gray italic labels indicate the control setpoints dispatched to each device by its SCU.*

C.2 Formal definition of the microgrid MDP

This appendix gives a formal definition of the decision process the RL agent interacts with in Section 4, and states precisely which variables are observed, how the state evolves, and in what sense the process is Markov. Because demand and wind are uncontrolled inputs driven by an external generating process, we formalise the environment as a finite-horizon *Markov decision process with exogenous inputs* (Dietterich et al., 2018; Sinclair et al., 2023), i.e. a tuple

$$\mathcal{M} = \big(\mathcal{S},\ \mathcal{X},\ \mathcal{A},\ P^s,\ P^x,\ r,\ \gamma,\ \rho_0,\ H\big), \tag{1}$$

where $\mathcal{S}$ is the endogenous (controllable) state space, $\mathcal{X}$ the exogenous state space, $\mathcal{A}$ the action space, P^s the endogenous transition kernel, P^x the exogenous transition kernel, r the reward, $\gamma \in (0, 1]$ the discount factor, ρ_0 the initial-state distribution, and H the horizon. The environment is stepped at a fixed interval $\Delta t = 1\,\text{min}$, and each episode spans one day, $H = 1440$ steps.

Exogenous state and process. The exogenous state at step t is $x_t = (\delta_t,\ p_t^{\text{wind,avail}}) \in \mathcal{X} \subseteq \mathbb{R}^2_{\geq 0}$, the power demand and the available wind power. It evolves according to $x_{t+1} \sim P^x(\cdot \mid x_t)$, a process determined by the real weather and consumption data (Section 4.1); crucially, P^x does *not* depend on the agent's action, and its underlying generating state (meteorological conditions, consumption drivers) is *not* available to the agent. The agent instead receives, at each step, an L-point forecast at 15 min resolution ($L = 30$, covering the next 7.5 h),

$$D_t = (\hat{\delta}_{t+k})_{k=1}^{L}, \qquad P_t^{\text{avail}} = (\hat{p}_{t+k}^{\text{wind,avail}})_{k=1}^{L}, \tag{2}$$

produced by the industry-standard models of Section 4.1.

Endogenous state. The endogenous state aggregates the internal state of every device SCU:

$$s_t = \big(\underbrace{\mathrm{SoC}_t,\ R_t}_{\text{battery}},\ \underbrace{\{\, m_t^i,\ \tau_t^i,\ \rho_t^i,\ W_t^i,\ p_t^{\mathrm{gen},i} \,\}_{i\in\{1,2\}}}_{\text{gensets}} \big) \in \mathcal{S}, \tag{3}$$

where $\mathrm{SoC}_t \in [0,1]$ is the battery state of charge (as a fraction of the 672 kWh capacity C_{batt}); R_t is the rainflow switching-point buffer used to compute the degradation cost (Appendix C.4); $m_t^i \in \{\textsc{off}, \textsc{warmup}, \textsc{on}, \textsc{cooldown}\}$ is the operating mode of genset i; τ_t^i is its elapsed-routine timer (warm-up/cool-down); ρ_t^i is its minimum-runtime counter; W_t^i is the sliding window of power output over the last 48 h used to enforce the ISO 8528-1 average-power limit; and $p_t^{\mathrm{gen},i}$ is its current power. The wind turbine is memoryless — its output $p_t^{\mathrm{wind}} = \mathrm{clip}(\bar{p}_t^{\mathrm{wind}}, 0, p_t^{\mathrm{wind,avail}})$ depends only on the current compliant setpoint and the available wind — so it carries no persistent constrained state and contributes no variable to s_t; its available power $p_t^{\mathrm{wind,avail}}$ (exogenous, see below) and produced power p_t^{wind} are nonetheless observed by the agent. Because precise sensors are available, the digital-twin state estimator is an exact update (Section 4.2), so the SCU state estimates coincide with the true device states.

Action space and shielding. The agent action is two-dimensional,

$$a_t = (\Delta_t^{\mathrm{orch}},\ \bar{p}_t^{\mathrm{batt}}) \in \mathcal{A} = \{\textsc{start}, \textsc{stop}, \textsc{idle}\} \times [-\bar{P}^{\mathrm{batt}}, \bar{P}^{\mathrm{batt}}], \tag{4}$$

with $\bar{P}^{\mathrm{batt}} = 600\,\mathrm{kW}$ and the convention that $\bar{p}_t^{\mathrm{batt}} > 0$ discharges the battery. The action is not applied directly. The microgrid SCU applies the shielding map of Algorithm 4,

$$\Pi_{\mathrm{shield}} : (a_t, s_t, x_t) \longmapsto a_t^{\mathrm{comp}} = \big(\underbrace{\bar{p}_t^{\mathrm{comp,batt}}}_{\text{battery}},\ \underbrace{\bar{p}_t^{\mathrm{comp,wind}}}_{\text{wind}},\ \underbrace{(\Delta_t^{\mathrm{comp},i},\ \bar{p}_t^{\mathrm{comp},i})_{i\in\{1,2\}}}_{\text{gensets}} \big) \tag{5}$$

which returns the device-level compliant actions (genset status changes and power setpoints, battery setpoint, wind curtailment) that satisfy every operational constraint of Section 4, including recoverability. Applying a_t^{comp} to the devices induces the powers $p_t^{\mathrm{batt}}, p_t^{\mathrm{wind}}, p_t^{\mathrm{gen},i}$ and the next endogenous state.

Transition dynamics. Given (s_t, x_t, a_t), the compliant action $a_t^{\mathrm{comp}} = \Pi_{\mathrm{shield}}(a_t, s_t, x_t)$ drives the deterministic device updates (Section 4.2). The battery state of charge evolves as

$$\mathrm{SoC}_{t+1} = \mathrm{SoC}_t - \frac{\Delta t}{C_{\mathrm{batt}}}\,\phi\big(p_t^{\mathrm{batt}}\big), \qquad \phi(p) = \begin{cases} p/\eta & p \ge 0 \text{ (discharge)} \\ p\,\eta & p < 0 \text{ (charge)} \end{cases} \tag{6}$$

with round-trip efficiency $\eta = 0.95$ (5% loss per direction), and R_{t+1} is updated by the online rainflow algorithm (Algorithms 7–8). Each genset mode m^i follows the finite-state machine defined by its start/stop routine, minimum-runtime and power constraints (Section 4.2), with τ^i, ρ^i, W^i advanced accordingly. The endogenous transition is therefore $s_{t+1} \sim P^s\big(\cdot \mid s_t, x_t, a_t^{\mathrm{comp}}\big)$, and the exogenous transition $x_{t+1} \sim P^x(\cdot \mid x_t)$ is independent of the action. Substituting (5) defines the *shielded* transition kernel actually faced by the learning agent,

$$\widetilde{P}\big(s_{t+1} \mid s_t, x_t, a_t\big) \;=\; P^s\big(s_{t+1} \mid s_t, x_t, \Pi_{\mathrm{shield}}(a_t, s_t, x_t)\big). \tag{7}$$

Reward. The reward is the negative operating cost combining fuel consumption and battery degradation over the step,

$$r_t = -\Big(F_{o,t} + \alpha\, d_{b,t}\Big), \qquad F_{o,t} = \sum_{i\in\{1,2\}} \Big(0.25\, p_t^{\mathrm{gen},i} + 10 \cdot \mathbb{1}[m_t^i = \textsc{on}]\Big)\Delta t_h, \tag{8}$$

where $F_{o,t}$ is the genset fuel consumption in litres ($0.25\,\mathrm{L\,kW^{-1}\,h}$ proportional term plus a $10\,\mathrm{L\,h^{-1}}$ idling term, $\Delta t_h = \frac{1}{60}$ h), $d_{b,t}$ is the battery degradation cost of Appendix C.4, and $\alpha \ge 0$ is the utility-chosen tradeoff weight. Ties are broken in favour of policies using less battery/genset reserve. The agent maximises the expected return $\mathbb{E}\big[\sum_{t=0}^{H-1} \gamma^t r_t\big]$.

Observation and policy. The agent does not observe x_t's generating process; its observation is

$$o_t = \big(\, \hat{s}_t,\ p_t^{\mathrm{wind}},\ \delta_t,\ p_t^{\mathrm{wind,avail}},\ D_t,\ P_t^{\mathrm{avail}} \,\big), \tag{9}$$

the aggregated (exact) subcomponent state estimate $\hat{s}_t$, the current produced wind p_t^{wind}, the current exogenous values $(\delta_t, p_t^{\mathrm{wind,avail}})$, and the two forecast series. The policy $\pi_\theta(a_t \mid o_t)$ encodes $D_t, P_t^{\mathrm{avail}}$ with LSTM modules (Appendix A), so that o_t carries a summary of the anticipated exogenous evolution.

Markov property and partial observability. Two points deserve emphasis, as they bear directly on how the results should be read. First, the endogenous state s_t is deliberately constructed to be Markov: the rainflow buffer R_t and the 48 h windows W_t^i are carried in the state *precisely* so that the degradation cost $d_{b,t}$ and the ISO average-power constraint depend only on $(s_t, a_t^{\mathrm{comp}})$ rather than on the full history (Appendix C.4). Conditioned on the exogenous trajectory $x_{0:H}$, the tuple (s_t) is therefore Markov under $\widetilde{P}$. Second, the exogenous process is only *partially observed*: the agent sees the realised values x_t and a finite forecast $(D_t, P_t^{\mathrm{avail}})$, but not the latent state driving P^x. Strictly, the joint process (s_t, x_t) is Markov, whereas the information state o_t available to the agent is not a sufficient statistic for x_{t+1} unless the forecast is exact; the agent thus faces a POMDP over the exogenous component, which we address in the standard way by augmenting o_t with the forecasts and a recurrent encoder rather than by maintaining an explicit belief. This is why forecast quality, and not just the control policy, influences performance, and it is a property of the environment shared by *every* agent and baseline evaluated under the same o_t.

Remark on shielding and baseline comparison. Equation (7) makes explicit that the shield is folded into the transition kernel: from the agent's perspective the constrained problem is turned into an unconstrained MDP with the modified kernel $\widetilde{P}$ and unchanged observation o_t. Removing the recovery shield (the ablation of Section 5) therefore changes $\widetilde{P}$ — the environment no longer auto-corrects unrecoverable commands — while leaving the agent's information set o_t unchanged; it is not a change in the information available to the policy. This distinction is what the shielded-versus-unshielded and shielded-versus-SAC-Lag comparisons isolate: the effect of enforcing constraints *by construction* in $\widetilde{P}$ versus *by penalty* in the objective. We note that these two mechanisms do not in general induce identical safety–performance tradeoffs, and we make the shielded objective and kernel explicit here so that the comparison can be interpreted on this basis.

C.3 Microgrid-specific SCU algorithms

Algorithms 1, 2 and 3 are written for a general SCU hierarchy. Here we instantiate them for the remote microgrid of Section 4, whose SCU decomposition is shown in Figure 2. The hierarchy has depth 3: the microgrid SCU U^{mg} (level 3, interfacing with the agent) has three children — the battery SCU U^{batt}, the wind-turbine SCU U^{wind} and the genset-orchestrator SCU U^{orch} (all level 2) — and the orchestrator itself has two genset children $U^{\mathrm{g1}}, U^{\mathrm{g2}}$ (level 1). The battery, wind turbine and gensets are devices; the microgrid and the orchestrator are systems. Demand δ_t and available wind power $p_t^{\mathrm{wind,avail}}$ are exogenous observations feeding the digital twins, not controlled children.

The instantiation only substitutes the ad-hoc functions (SHIELD, STATEESTIM, OBS/AGG/DISAGG, CONTROLLERSTATE) with their microgrid implementations and unrolls the recursion over this fixed tree. The agent action is $a_t = (\Delta_t^{\mathrm{orch}}, \bar{p}_t^{\mathrm{batt}})$. Because precise sensors are available (Section 4.2), the device STATEESTIM reduces to an exact update, $\mathrm{StateEstim}(s_t^{\mathrm{DT}}, o_{t+1}) = o_{t+1}$.

Algorithm 4 Microgrid SCU step function

// LEVEL 3 — MICROGRID SCU (TOP LEVEL, CALLED BY THE RL AGENT). Shield Δ_t^{orch} for recoverability, // dispatch the setpoints to enforce 0-balance, execute the three child SCUs, then update the controller.

function STEP(U^{mg}, $a_t = (\Delta_t^{\text{orch}}, \bar{p}_t^{\text{batt}})$)
  $\tilde{\Delta}_t^{\text{orch}} \leftarrow$ RECOVERYSHIELD($U^{\text{mg·DT}}, \Delta_t^{\text{orch}}$) — roll out warm-up+cool-down horizon; least-different recoverable Δ
  // Balance dispatch: solve $p_t^{\text{batt}} + p_t^{\text{wind}} + p_t^{\text{orch}} = \delta_t$ via one-step DT predictions, wind before fuel.
  $\bar{p}_t^{\text{wind}} \leftarrow$ WINDSETPOINT($U^{\text{mg·DT}}, \bar{p}_t^{\text{batt}}, \tilde{\Delta}_t^{\text{orch}}, \delta_t$) — wind first (greedy under wind-priority)
  $\bar{p}_t^{\text{orch}} \leftarrow$ GENSETSETPOINT($U^{\text{mg·DT}}, \bar{p}_t^{\text{batt}}, \bar{p}_t^{\text{wind}}, \tilde{\Delta}_t^{\text{orch}}, \delta_t$; reserve=OFF) — gensets cover residual, no overload
  $\bar{p}_t^{\text{batt}\star} \leftarrow$ ADJUSTBATTERY($U^{\text{mg·DT}}, \bar{p}_t^{\text{batt}}, \bar{p}_t^{\text{wind}}, \bar{p}_t^{\text{orch}}, \delta_t$) — shield: correct battery if balance $\neq 0$ (reserve allowed)
  $\bar{p}_t^{\text{orch}} \leftarrow$ GENSETSETPOINT($U^{\text{mg·DT}}, \bar{p}_t^{\text{batt}\star}, \bar{p}_t^{\text{wind}}, \tilde{\Delta}_t^{\text{orch}}, \delta_t$; reserve=ON) — re-solve gensets, overload reserve allowed
  $s_{t+1}^{\text{batt}} \leftarrow$ STEP($U^{\text{batt}}, \bar{p}_t^{\text{batt}\star}$) — recurse into each child SCU
  $s_{t+1}^{\text{wind}} \leftarrow$ STEP($U^{\text{wind}}, \bar{p}_t^{\text{wind}}$)
  $s_{t+1}^{\text{orch}} \leftarrow$ STEP($U^{\text{orch}}, (\tilde{\Delta}_t^{\text{orch}}, \bar{p}_t^{\text{orch}})$)
  $o_{t+1} \leftarrow$ OBS(AGG($s_{t+1}^{\text{batt}}, s_{t+1}^{\text{wind}}, s_{t+1}^{\text{orch}}$), $\delta_{t+1}, p_{t+1}^{\text{wind,avail}}, D_{t+1}, P_{t+1}^{\text{avail}}$) — + exogenous obs & forecasts
  $s_{t+1} \leftarrow$ UPDATECONTROLLER($U^{\text{mg·SC}}, o_{t+1}$)
  return s_{t+1}^{DT}
end function

// LEVEL 2 — BATTERY SCU (DEVICE). Clip the setpoint to safe SoC/power limits, execute, update.

function STEP($U^{\text{batt}}, \bar{p}_t^{\text{batt}}$)
  $\bar{p}_t^{\text{comp}} \leftarrow$ CLIP($U^{\text{batt·DT}}, \bar{p}_t^{\text{batt}}$) — shield: SoC$\in [10, 90]\%$ (+5% emerg.), $|\cdot| \leq 600$kW
  $o_{t+1} \leftarrow$ STEP($U^{\text{batt,dev}}, \bar{p}_t^{\text{comp}}$) — execute on real device: SoC, p^{batt}, rainflow buffer R
  return UPDATECONTROLLER($U^{\text{batt·SC}}, o_{t+1}$)
end function

// LEVEL 2 — WIND-TURBINE SCU (DEVICE). No constraint: relay the setpoint unchanged.

function STEP($U^{\text{wind}}, \bar{p}_t^{\text{wind}}$)
  $o_{t+1} \leftarrow$ STEP($U^{\text{wind,dev}}, \bar{p}_t^{\text{wind}}$) — execute on real device: $p^{\text{wind}} = \text{clip}(\bar{p}^{\text{wind}}, 0, p^{\text{wind,avail}})$
  return UPDATECONTROLLER($U^{\text{wind·SC}}, o_{t+1}$)
end function

// LEVEL 2 — GENSET-ORCHESTRATOR SCU (SYSTEM). Split one orchestrator command into two // per-genset commands respecting priority & equal-fraction, then recurse into both gensets.

function STEP($U^{\text{orch}}, (\Delta_t^{\text{orch}}, \bar{p}_t^{\text{orch}})$)
  $(\Delta_t^{\text{g1}}, \Delta_t^{\text{g2}}) \leftarrow$ PRIORITYSM($U^{\text{orch·DT}}, \Delta_t^{\text{orch}}$) — shield 1: one on/off per step, priority order
  $(\bar{p}_t^{\text{g1}}, \bar{p}_t^{\text{g2}}) \leftarrow$ EQUALFRACTION($U^{\text{orch·DT}}, \bar{p}_t^{\text{orch}}$) — shield 2: equal % of nominal; 70%/48h
  $s_{t+1}^{\text{g1}} \leftarrow$ STEP($U^{\text{g1}}, (\Delta_t^{\text{g1}}, \bar{p}_t^{\text{g1}})$) — recurse into each genset SCU
  $s_{t+1}^{\text{g2}} \leftarrow$ STEP($U^{\text{g2}}, (\Delta_t^{\text{g2}}, \bar{p}_t^{\text{g2}})$)
  $o_{t+1} \leftarrow$ OBS(AGG($s_{t+1}^{\text{g1}}, s_{t+1}^{\text{g2}}$))
  return UPDATECONTROLLER($U^{\text{orch·SC}}, o_{t+1}$)
end function

// LEVEL 1 — GENSET SCU (DEVICE). Enforce single-genset operational limits, execute, update.

function STEP($U^{\text{gen}}, (\Delta_t^{\text{gen}}, \bar{p}_t^{\text{gen}})$)
  $(\Delta_t^{\text{comp}}, \bar{p}_t^{\text{comp}}) \leftarrow$ GENSETSHIELD($U^{\text{gen·DT}}, \Delta_t^{\text{gen}}, \bar{p}_t^{\text{gen}}$) — shield: routines, min-runtime, 120–400kW (+40), ISO 70%/48h
  $o_{t+1} \leftarrow$ STEP($U^{\text{gen,dev}}, (\Delta_t^{\text{comp}}, \bar{p}_t^{\text{comp}})$) — execute on real device: status, p^{gen}, fuel F_o, runtime, 48h avg
  return UPDATECONTROLLER($U^{\text{gen·SC}}, o_{t+1}$)
end function

At the microgrid level the shield is the only *composite* one. The predictive RECOVERYSHIELD acts on the discrete orchestrator command Δ_t^{orch}: it rolls out the digital twin over the combined warm-up and cool-down horizon (the longest genset routine that could block a later start) under a worst-case exogenous scenario (demand rising and available wind falling at their historical rates, with reserve) and a no-reserve constant scenario; if either predicts a negative balance, the command is nudged one step toward safety (STOP $\rightarrow$ IDLE $\rightarrow$ START). The remaining setpoints are then dispatched to enforce the 0-balance constraint $p_t^{\text{batt}} + p_t^{\text{wind}} + p_t^{\text{orch}} = \delta_t$ through the four-step sequence above: wind is set first (greedily under the wind-over-fuel priority), the gensets cover the residual demand without overload, the battery setpoint is corrected if a nonzero balance remains (now allowing reserve), and the genset setpoint is finally re-solved with the overload reserve available. Surplus wind that neither the demand nor the battery can absorb is curtailed. Each of these steps queries the digital twin for the one-step power each device would produce, so the dispatch respects every lower-level device

constraint. All other shields are per-SCU clips or state machines on known dynamics, which is precisely what the decomposition buys.

Algorithm 5 Microgrid digital twin update function

function UPDATEDT($U^{\text{mg·DT}}, o_{t+1}$)
  $(o^{\text{batt}}_{t+1}, o^{\text{wind}}_{t+1}, o^{\text{orch}}_{t+1}) \leftarrow \text{Disagg}(o_{t+1})$
  $s^{\text{batt·DT}}_{t+1} \leftarrow o^{\text{batt}}_{t+1}$ — exact estimation: SoC, p^{batt}, R
  $s^{\text{wind·DT}}_{t+1} \leftarrow o^{\text{wind}}_{t+1}$ — exact estimation: p^{wind}, $p^{\text{wind,avail}}$
  $(o^{\text{g1}}_{t+1}, o^{\text{g2}}_{t+1}) \leftarrow \text{Disagg}(o^{\text{orch}}_{t+1})$
  for $j \in \{\text{g1}, \text{g2}\}$ **do**
    $s^{j\cdot\text{DT}}_{t+1} \leftarrow o^{j}_{t+1}$ — status, runtime, 48h-avg window
    $s^{j}_{t+1} \leftarrow \text{UpdateController}(U^{j\cdot\text{SC}}, o^{j}_{t+1})$
  end for
  $s^{\text{orch·DT}}_{t+1} \leftarrow \text{Agg}(s^{\text{g1}}_{t+1}, s^{\text{g2}}_{t+1})$
  $s^{\text{DT}}_{t+1} \leftarrow \text{Agg}(s^{\text{batt·DT}}_{t+1}, s^{\text{wind·DT}}_{t+1}, s^{\text{orch·DT}}_{t+1})$
  return s^{DT}_{t+1}
end function

The genset devices are the only components whose CONTROLLERSTATE carries non-trivial internal state — elapsed warm-up/cool-down timers, the minimum-runtime counter and the 48h moving-average window — which the orchestrator digital twin reads to enforce the coupled equal-fraction and ISO 70%/48h constraints across both gensets.

Algorithm 6 Microgrid controller update function

function UPDATECONTROLLER($U^{\text{mg·SC}}, o_{t+1}$)
  $s^{\text{SC}}_{t+1} \leftarrow \text{ControllerState}(U^{\text{mg·SC}}, o_{t+1})$ — = agent observation (Sec. C.2)
  $= [\,\textit{SoC}, p^{\text{batt}}, \textit{R}; p^{\text{wind}}, p^{\text{wind,avail}}; \textit{per-genset status/runtime/48h-avg}, p^{\text{orch}}; \delta_{t+1}, p^{\text{wind,avail}}_{t+1}, D_{t+1}, P^{\text{avail}}_{t+1}\,]$
  $s^{\text{SC·DT}}_{t+1} \leftarrow \text{UpdateDT}(U^{\text{mg·SC·DT}}, o_{t+1})$ — internal twin of {batt, wind, orch→ {g1,g2}}
  $s_{t+1} = [s^{\text{SC}}_{t+1}, s^{\text{SC·DT}}_{t+1}]$
  return s_{t+1}
end function

The controller state s^{SC}_{t+1} is exactly the observation the RL agent receives (Section C.2): the subcomponent state estimations together with the two exogenous forecast series. The rainflow buffer R is included so that the battery-degradation cost $d_{b,t}$ remains Markovian. The reward $r_t = -(F_{o,t} + \alpha\, d_{b,t})$ is read directly off o_{t+1} — fuel $F_{o,t}$ from the genset devices and $d_{b,t}$ from the battery buffer R — and is therefore external to these three functions.

C.4 Battery Degradation

Different factors can degrade a battery and reduce its storage capacity. Time and temperature affect its capacity, however, these are not under the agent's control. More relevant here, cycling, i.e. repeated charging and discharging, significantly impacts the battery's degradation (Cao et al., 2020). Cycle-based battery degradation is often estimated through the rainflow analysis of the battery's state of charge (SoC) (Xu et al., 2018), which accounts for the number and amplitude of cycles along time. However, this process is usually done offline, given a history of data. In RL however, a degradation cost $d_{b,t}$ at every time step is needed that reflects the impact of the agent's action while, when summed over a trajectory, being equal to the result of the offline process.

A commonly used approach for online battery degradation estimation is to use a linear approximation, as presented by Cao et al. (2020). The linearized degradation cost is calculated using a simplified linear model, where the cost is directly proportional to the absolute value of the battery SoC change, $\text{b}_t = \text{soc}_t - \text{soc}_{t-1}$. The linearized degradation cost is thus computed as $d_{b,t} = \alpha_d \cdot |\text{b}_t|$ where α_d is the degradation coefficient. This linear approach provides a straightforward estimation

of the degradation cost, making it easier to implement and understand while still capturing the essential impact of battery usage on its lifespan.

In this paper, we aimed at representing the industrial environment as precisely as possible, and as such, to model the battery degradation more accurately. Kwon & Zhu (2022) have proposed an approach to online cycle-based degradation cost estimation for RL. In battery fatigue modeling, the impact of a cycle on degradation is exponential to the cycle amplitude. An interesting insight from Kwon & Zhu (2022) is their decomposition of the cost for each time step as

$$d_{b,t} = \alpha_d \cdot \exp(\beta \cdot |\text{soc}_t + b_t - R[-1]|) - \alpha_d \cdot \exp(\beta \cdot |\text{soc}_t - R[-1]|)$$

where α_d represents the degradation rate, β indicates the sensitivity to SoC changes, and $R[-1]$ is the most recent switching point in the SoC history.

The difficulty of the rainflow algorithm is however to find the right value of $R[-1]$, as cycles can close, and at the next time step, the SoC evolution can pertain to another cycle open further in history. An important problem is that, in theory, the necessary memory to correctly assign each time step to the correct value of $R[-1]$ can be as long as the complete SoC history. Kwon & Zhu (2022) propose an algorithm to keep only three points, however, it does not fully respect rainflow analysis. Instead, we based our approach on Obermayr et al. (2021), where a 4-point online rainflow algorithm allows to identify the last switching point while memory requirement is hard-capped thanks to a discretization process. Our slightly adapted algorithm is summarized in Algorithms 7 and 8. The resulting buffer R is used to compute the reward, and is given as an observation to the agent as part of the battery state to respect the Markovian assumption for the environment.

Algorithm 7 Rainflow 4-point algorithm

```
// 4-point rainflow condition check
function RAINFLOW4P(R)
    cycleFound ← False
    if min(R[-4], R[-1]) ≤ min(R[-3], R[-2]) and then
      max(R[-3], R[-2]) ≤ max(R[-4], R[-1])
        cycleFound ← True
    end if
    return cycleFound
end function
// Update switching point buffer R
function UPDATESWITCHINGPOINTS(x, F, R)
    F[2] ← Discretize(x, d_load)
    F, tpFound ← HysteresisFilter(F)
    if tpFound then
        Append(R, Discretize(F[0], d_load))
    end if
    Append(R, Discretize(x, d_load))
    while Length(R) ≥ 4 and Rainflow4P(R) do
        R[−3] ← R[−1]
        Delete(R[-2:])
    end while
    Delete(R[-1])
    return F, R
end function
```

Algorithm 8 Hysteresis Filter algorithm

```
function HYSTERESISFILTER(F)
    // Apply hysteresis filter to signal
    // Define helper functions
    function SHIFT(F)
        F[0] ← F[1]
        F[1] ← F[2]
        return F
    end function
    function SKIP(F)
        F[1] ← F[2]
        return F
    end function
    tpFound ← False
    if F[2] < F[1] then
        if F[0] ≥ F[1] then
            F ← Skip(F)
        else
            tpFound ← True
            F ← Shift(F)
        end if
    else if F[2] > F[1] then
        if F[0] ≤ F[1] then
            F ← Skip(F)
        else
            tpFound ← True
            F ← Shift(F)
        end if
    else
        if F[0] > F[1] then
            F[1] ← min(F[1], F[2])
        else
            F[1] ← max(F[1], F[2])
        end if
    end if
    return F, tpFound
end function
```

Note that the discretization can cause artifacts, such as the calculated degradation cost $d_{b,t}$ being negative close to switching points. In this case, we instead approximate it for this time step based on the discretization window w: $d_{b,t} = \alpha_d \cdot (\exp(\beta \cdot |w|) - 1)/w$.

A final question regards the value of α_d and β, so that $d_{b,t}$ represents a realistic capacity loss for the battery. We chose $\beta = 1$ and $\alpha_d = 5$ as in (Cao et al., 2020). However, these values should be calibrated with a real battery if to be deployed in the real world. This explains our choice to describe the battery degradation units are arbitrary in the paper, and keep the reward function flexible with different values of weight α.

This approach ensures that the degradation cost accurately reflects the impact of battery usage on its lifespan, considering both the magnitude and frequency of usage changes.