Energy and AI 12 (2023) 100227

Contents lists available at ScienceDirect

Energy and AI

journal homepage: www.elsevier.com/locate/egyai

Safe reinforcement learning for multi-energy management systems with known constraint functions

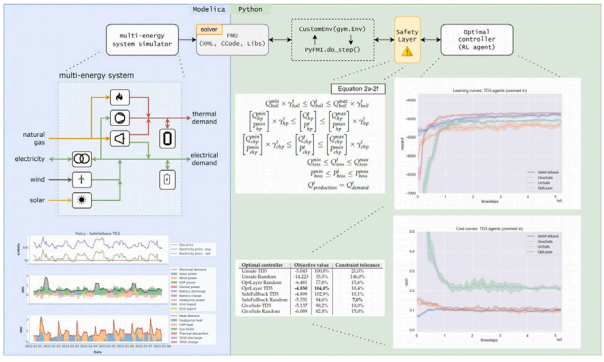
Glenn Ceusters [a,b,c,*], Luis Ramirez Camargo [b,d], Rüdiger Franke [a], Ann Nowé [c], Maarten Messagie [b]

[a] ABB, Hoge Wei 27, 1930 Zaventem, Belgium
[b] Vrije Universiteit Brussel (VUB), ETEC-MOBI, Pleinlaan 2, 1050 Brussels, Belgium
[c] Vrije Universiteit Brussel (VUB), AI-lab, Pleinlaan 2, 1050 Brussels, Belgium
[d] Copernicus Institute of Sustainable Development - Utrecht University, Princetonlaan 8a, 3584, CB Utrecht, Netherlands

HIGHLIGHTS

- A (near-to) optimal multi-energy management policy can be learned safely.
- Any reinforcement learning algorithm can be used safely.
- Hard-constraint guarantees without solving a mathematical program.
- Constraints can be formulated independently from the (optimal) control technique
- Better policies can be found starting with an initial safe fallback policy.

GRAPHICAL ABSTRACT



ARTICLE INFO

Keywords:
Reinforcement learning
Constraints
Multi-energy systems
Energy management system

ABSTRACT

Reinforcement learning (RL) is a promising optimal control technique for multi-energy management systems. It does not require a model *a priori* - reducing the upfront and ongoing project-specific engineering effort and is capable of learning better representations of the underlying system dynamics. However, *vanilla* RL does not provide constraint satisfaction guarantees — resulting in various potentially unsafe interactions within its environment. In this paper, we present two novel online model-free safe RL methods, namely SafeFallback and GiveSafe, where the safety constraint formulation is decoupled from the RL formulation. These provide hard-constraint satisfaction guarantees both during training and deployment of the (near) optimal policy. This is without the need of solving a mathematical program, resulting in less computational power requirements and more flexible constraint function formulations. In a simulated multi-energy systems case study we have shown that both methods start with a significantly higher utility compared to a *vanilla* RL benchmark and Optlayer benchmark (94,6% and 82,8% compared to 35,5% and 77,8%) and that the proposed SafeFallback method even can outperform the *vanilla* RL benchmark (102,9% to 100%). We conclude that both methods are viably safety constraint handling techniques applicable beyond RL, as demonstrated with random policies while still providing hard-constraint guarantees.

* Corresponding author at: Vrije Universiteit Brussel (VUB), ETEC-MOBI, Pleinlaan 2, 1050 Brussels, Belgium.
E-mail addresses: glenn.ceusters@be.abb.com, glenn.leo.ceusters@vub.be, gceusters@ai.vub.ac.be (G. Ceusters), l.e.ramirezcamargo@uu.nl (L.R. Camargo), ruediger.franke@de.abb.com (R. Franke), ann.nowe@ai.vub.ac.be (A. Nowé), maarten.messagie@vub.be (M. Messagie).

https://doi.org/10.1016/j.egyai.2022.100227
Received 10 October 2022; Received in revised form 15 December 2022; Accepted 22 December 2022
Available online 27 December 2022

1. Introduction

Energy systems continue to become increasingly interconnected with each other as the energy technologies that allow for this sector coupling are more mature and are being more widely implemented. This allows for an integrated control strategy that further can enhance the overall efficiency and performance of these so-called multi-energy, -carrier, -commodity or -utility systems. Furthermore, these multi-energy systems allow for the utilization of flexibility (i.e. storage, controllable loads) within and across all energy carriers. This integrated control strategy then typically [1] has an economic or environmental-oriented objective function or a combination thereof and therefore is multi-objective.

To ensure the optimum or Pareto optimum level of operation of such multi-energy systems, specific set-points are required to establish and maintain the desired objective (e.g. minimization of the energy costs or CO_2-equivalent emissions) while still fulfilling all system constraints [2]. As flexibility utilization exhibits dynamic behaviour and introduces a dependency between successive time steps, optimization across (or considering) numerous time steps is necessary. Additionally, multiple uncertainties (i.e. variation in demands, pricing and weather) need to be managed so that the stability of the multi-energy system is preserved.

Model-predictive control (MPC) and reinforcement learning (RL) have recently been benchmarked within such a multi-energy management system context [3]. Ceusters et al. [3] showed that RL-based energy management systems do not require a model *a priori* and that they can outperform linear MPC-based energy management systems after training. However, *vanilla* RL (see Fig. 1(a)) performs a large number of potentially unsafe interactions within its environment, which is unacceptable in many real-world applications. For example, in a multi-energy system, neglecting the crucial energy balance constraint could result in either under or over-production. For most power systems this imbalance could result in exceeding the maximum power capacity to or from the grid — especially relevant with the expected large-scale integration of electric vehicles. Moreover, this imbalance is particularly problematic for energy systems that are not connected to a larger power distribution grid or a district heating system and therefore lack a higher level of control. In this case, the imbalance could lead to loss of user comfort (e.g. power or heat outages).

Therefore, our goal is to ensure that *every* interaction with the underlying environment (a multi-energy system in our case study) satisfies a given set of safety constraints, *independently* of the used (optimal) control technique (see Fig. 1(b)). This, compared to formulating a specific safe RL algorithm which allows that future – presumably better – optimization algorithms can easily be used instead.

1.1. Contribution and outline

Our contributions can, to the best of our knowledge, be listed as the following:

- Online model-free safe RL method which provides hard-constraint, rather than soft- and chance-constraint, satisfaction guarantees that has a significantly higher initial utility;
- Decoupling architecture of safety constraint formulations from the RL formulation so that future – presumably better – optimization algorithms can easily be used instead;
- Hard-constraint satisfaction without the need of solving a mathematical program, resulting in less computational power requirements and a more flexible constraint function formulation (no derivative information is required);
- Demonstration of safe RL-based energy management on a detailed multi-energy system simulation environment.

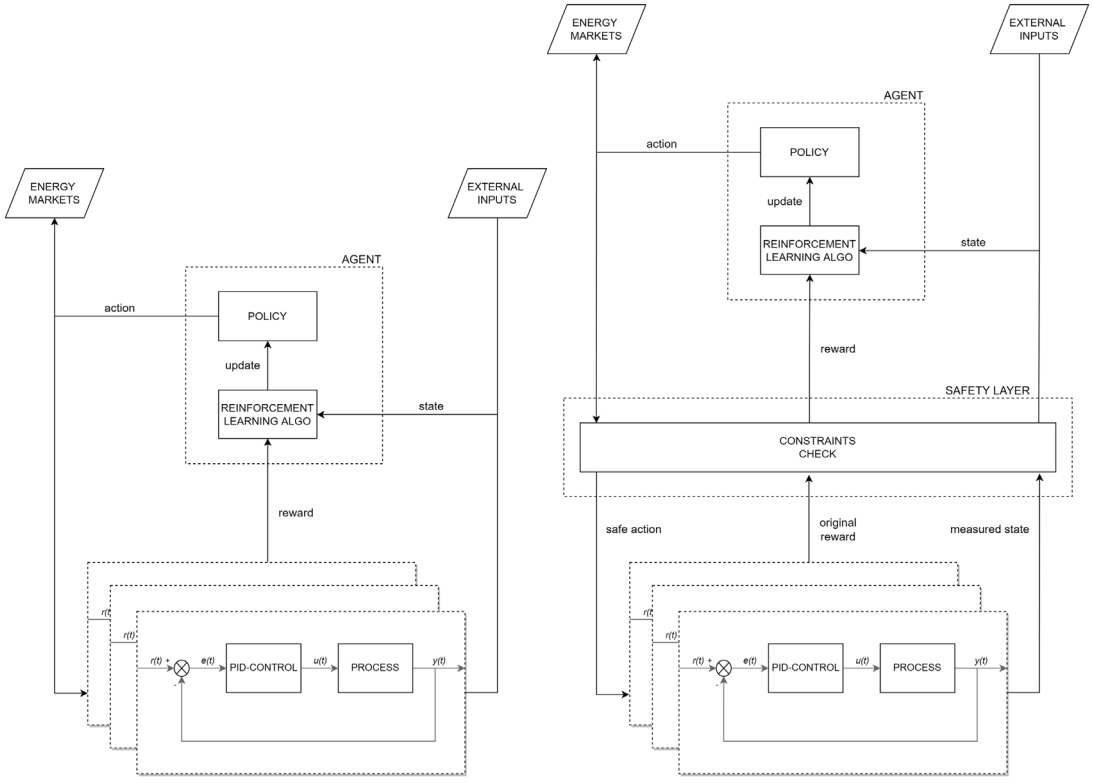
In Section 2 related work is discussed and our research question is formulated, Section 3 introduces the proposed methodologies, while in Section 4 the toolchain, the simulated multi-energy system environment, the safety layer, the RL agent and the evaluation procedure are presented. Furthermore, Section 5 discusses the results and provides directions for future work while Section 6 presents the conclusion. Finally, Appendix A shows time series visualizations of the different policies, Appendix B the full learning and cost curves for the assessed agents in the case study, Appendix C the pseudo-code of the specific RL agent (TD3) and Appendix D the run-time statistics.

2. Related work

2.1. Vanilla (unsafe) RL in multi-energy systems

In recent years, there have been numerous works that proposed RL for various applications within energy systems as reviewed by e.g. Cao et al. [4], Yang et al. [5,5] and Perera and Kamalaruban [6], and this even for the more specific (and arguably more challenging) multi-energy systems context by Zhou [7]. The majority of these applications can be classified under a broader energy management problem. RL-based energy management systems, which have been proposed and demonstrated in multi-energy systems include, for example, Rayati et al. [8] were some of the first to apply RL, specifically Q-learning [9], for the energy management of a simulated multi-energy residential building, which they later extended with demand-side management capabilities [10]. Posteriorly, Mbuwir et al. [11] successfully applied RL (Q-learning) for a battery energy management system within a simulated residential multi-energy system. They included a backup policy to overrule the actions of the RL agent in case of constraint violation. Furthermore, Wang et al. [12] used a path-tracking interior point method and a RL algorithm (Q-learning) for a bi-level interactive decision-making model with multiple agents in a regional multi-energy system. A multi-agent RL (Q-learning) approach was also proposed by Ahrarinouri et al. [13] and this for the energy management of a simulated residential multi-energy system. Around the same time, Ye et al. [14] proposed the usage of a deep RL algorithm, specifically a deep deterministic policy gradient (DDPG) [15] with a prioritized experience replay strategy, again within a simulated residential multi-energy system.

Moreover, Xu et al. [16] demonstrated an industrial multi-energy scheduling framework using a RL (Q-learning) based differential evolution approach that adaptively determines the optimal mutation strategy and its associated parameters. In the same line of developments, Zhu et al. [17] demonstrated a multi-agent deep RL energy management system, using multi-agent counterfactual soft actor-critic (mCSAC) [18], for a simulated multi-energy industrial park, and Zhou et al. [19] proposed deep reinforcement learning for the data-driven stochastic energy management of a multi-energy system, adding a prioritized experience relay to improve the training efficiency and therefore the convergence of the RL algorithm. Furthermore, Ceusters et al. [3] presented an on- and off-policy multi-objective model-free RL approach, using proximal policy optimization (PPO [20]) and twin delayed deep deterministic policy gradient (TD3 [21]) and they did benchmark this against a linear MPC — both derived from the general optimal control problem. They showed, on two separate simulated multi-energy systems, that the RL agents offer the potential to match and outperform the MPC. Whereas, Zhang et al. [22] presented a series of works [22–24] for the (near-to) optimal scheduling of an integrated energy system (a.k.a. multi-energy system) using deep reinforcement learning both for a single- and multi-objective and Zhang et al. [25] lather extended this for distributed multi-energy systems using multi-agent deep reinforcement learning.



(a) *vanilla* (unsafe) reinforcement learning **(b)** shielded (safe) reinforcement learning

Fig. 1. Block diagrams comparison: the feasibility of the RL agent's actions, being in a given state, are always checked against the *a priori* constraint functions acting as a safety layer — shielding the environment from unsafe (control) actions. Note that, we assume that the continuous unconstrained error handling (i.e. minimization of the difference between the desired set-point and a measured process variable) is performed by proportional–integral–derivative (PID) controllers.

2.2. Safe RL in multi-energy systems

RL inherently requires interaction with its environment. Adequate measures are required to avoid the violation of environmental-specific constraints both during online training and during pure policy execution (e.g. after training a policy offline). All the works in Section 2.1 have, knowingly (and thus reported as such) or non-knowingly, either neglected these specific environmental constraints or greatly simplified them — limiting the real-world use cases. This as, violating constraints in a real-world environment can result in undesirable specific losses (e.g., monetary, comfort) and, in extreme cases, in human harm. Safe RL, therefore, aims to: *"learn policies that maximize the expectation of the return in problems in which it is important to ensure reasonable system performance and/or respect safety constraints during the learning and/or deployment processes"* - as defined by García and Fernández [26].

In a comprehensive review García and Fernández [26] assess the use of safe RL beyond the energy systems management field and identified two main approaches: (1) modifying the optimality criteria with a safety factor; (2) modifying the exploration process by incorporating external knowledge or the guidance by a risk metric. More recently, Dulac-Arnold et al. [27] identified nine challenges that must be addressed to implement RL in real-world problems, including safety constraint violation — which was also underlined by Nweye et al. [28]. Furthermore, Brunke et al. [29] provided a broader safe learning review across both the control theory research space as well as the RL research space. More specifically, they showed (1) learning-based control approaches that start with an *a priori* model to safely improve the policy under the uncertain system dynamics, (2) safe RL approaches that do not need a model or even constraints in advance — but then also do not provide strict safety guarantees (yet encourages safety or robustness), and (3) approaches that provide safety certificates of any learned control policy.

In one of the first attempts to combine both hard-constraint satisfaction and RL in energy systems, Venayagamoorthy et al. [30] presented an intelligent dynamic energy management system for a smart microgrid using an action-dependent heuristic dynamic program, a type of adaptive critic design-based controller. They furthermore used an evolutionary algorithm to improve the dispatch solution over time and rejected candidate solutions that did not satisfy the critical load fulfilment constraint relying on power balancing rules and an initial decision-tree energy management system. Furthermore, Zhang et al. [31] proposed a bi-level power management system of networked microgrids in electric distribution systems. At the first level, a cooperative agent employs an adaptive model-free RL algorithm, to find the optimal retail price signals for the microgrids. While on the second level, each model-based microgrid controller solves a constrained mixed integer nonlinear program, based on the received price signal from the RL agent. Also, Zhao et al. [32] proposed a knowledge-assisted RL framework by combining a low-fidelity analytical model with a RL agent for a cooperative wind farm control problem. When the RL agent selects a naive action, a constraint action is calculated by solving an optimization problem using that analytical model. However, in all these cases it remains a heavy reliance on *a prior* physical models that are used in a separate optimization problem. Such developments contain a presumed transition function, a separated objective function (separated from the reward signal, which can introduce bias) and constraint functions — and thus not *only* constraint functions.

Nevertheless, recently and independent from this work, Park et al. [33] devised a novel RL algorithm, inspired by `OptLayer` [34], namely distance-based incentive/penalty Q-learning (DIP-QL) which also does not assume an *a prior* transition function and only uses constraint functions to provide hard-constraint guarantees as they demonstrated on a microgrid control problem. Yet, it uses a deep Q-learning algorithm as the backbone for their proposed method — where

we propose to decouple the constraint handling from the RL algorithm so that future – presumably better – optimization algorithms (as our framework is not limited to RL) can easily be used instead.

2.3. Conclusions from related work

The reviewed literature shows that RL is a promising and widely proposed approach for various applications in energy systems (and other domains, not discussed here), as well as for energy management systems specifically. It is also clear that the transition of RL towards real-world applications is not trivial and requires special attention concerning safety. Multiple safe (reinforcement) learning approaches exist, ranging in level of safety namely (from lower to higher level): (1) soft-constraint satisfaction, (2) chance-constraint satisfaction and (3) hard-constraint satisfaction. However, a model-free safe RL method of the following *combined* characteristics has – to the best of our knowledge – never been proposed and demonstrated yet: (i) providing hard-constraint, rather than soft- and chance-constraint, satisfaction guarantees with multiple constraints, (ii) which is decoupled from the RL (as a Markov Decision Process) formulation, (iii) both during training a (near) optimal policy (which involves exploratory and exploitative, i.e. greedy, steps) as well as during deployment of any policy (e.g. random agents or offline trained RL agents) and (iv) this without the need of solving a mathematical program, while (v) demonstrating for the energy management of multi-energy systems.

3. Proposed methodology

Following the standard RL formulation of the state-value function, yet extending this towards constraints subjection, the objective is to find a policy π, which is a mapping of states, $s \in S$, to actions, $a \in A(s)$, that maximizes an expected sum of discounted rewards and is subject to constraint sets X and U:

$$\max_{\pi}\left(E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^{k}(R-c)_{t+k+1} \right\} \right) \tag{1a}$$

$$s.t. \quad s_t = s \qquad t \in \mathbb{T}_0^{+\infty} = 0, \dots, +\infty \tag{1b}$$

$$s_t \in X \qquad t \in \mathbb{T}_0^{+\infty} = 0, \dots, +\infty \tag{1c}$$

$$a_t \in U \qquad t \in \mathbb{T}_0^{+\infty} = 0, \dots, +\infty \tag{1d}$$

where E_{π} is the expected value, following the policy π, of the rewards R and costs c (which is made explicit here for clarity reasons, i.e. a negative reward term for the cases when the *proposed* actions violate the constraints) - reduced with the discount factor γ over an infinite sum at any time step t. The specific algorithms are given in algorithms 1 and 2.

Note that Eq. (1a) is a discrete[1] time-invariant infinite-horizon stochastic optimal control problem, as also indicated by Ceusters et al. [3], yet differs from the standard formulation of RL due to the *a priori* constraint functions in the sets X and U. We acknowledge that methods without *a priori* constraint functions exist, yet this can – under the state-of-the-art – only reach safety level 2 at best (chance-constraint satisfaction), while we provide safety level 3 (hard-constraint satisfaction) as defined by Brunke et al. [29]. Rather than proposing a specific safe RL algorithm, we propose to decouple the *a priori* constraint function formulation from the (RL) agent so that any (new RL) algorithm can be used — while always guarantying the hard-constraint satisfaction. Although the proposed algorithms are conceptually simple, we will later show their effectiveness on a multi-energy system (which includes a non-grid connected thermal system).

[1] As the continuous error handling is performed by PID-controllers, see Fig. 1.

3.1. SafeFallback method

The first method we propose relies on an *a priori* safe fallback policy π^{safe}, which typically can be derived through classic control theory in the form of a set of hard-coded rules such as a simple rule-based policy (e.g. a priority-based energy management strategy – which is commonly available or easily constructible – see Section 4.4 for the safe fallback policy of the considered case study). As we furthermore assume that the constraint functions are given, we can simply check if the selected actions a while in state s satisfy the constraint conditions. When the constraints conditions are satisfied, the selected actions a are *considered* to be safe actions a^{safe} and are then executed in the environment so that the next state s', the reward r and done signal d are observed — which is the regular experience tuple (s, a^{safe}, r, s', d) for the RL agent. However, if the constraint conditions are violated, the selected action a is overruled by the safe action a^{safe} using the *a priori* safe fallback policy π^{safe}. Now not only the experience tuple (s, a^{safe}, r, s', d) is formed, but also the experience tuple $(s, a, r - c, s', d)$ containing the infeasible action and additional negative reward (i.e. cost, c). The pseudo-code is given in algorithm 1.

Algorithm 1: SafeFallback

```
Input: initialize RL algorithm, initialize constraint functions in
  sets X and U, initialize safe fallback policy π^safe
for k = 0, 1, 2, … do
    Observe state s and select action a
    if constraint check = True then
        keep selected action a as safe action a^safe
    else
        get safe action a^safe from safe fallback policy π^safe
    end
    Execute a^safe in the environment
    Observe next state s', reward r and done signal d to
     indicate whether s' is terminal
    Give experience tuple (s, a^safe, r, s', d) and if a^safe ≠ a :
     (s, a, r − c, s', d) with cost c
    If s' is terminal, reset environment state
end
```

3.2. GiveSafe method

Our second method does not require an *a priori* safe fallback policy, yet relies on the RL agent itself to pass safe actions a^{safe} - which can again be checked by the given constraint conditions. If the selected actions a while in state s passes the constraint check, the safe actions a^{safe} get executed in the environment and a regular experience tuple (s, a^{safe}, r, s', d) is received. However, if the constraints get violated the RL agent receives the experience tuple (s, a, c, s, d). Hence, the transition towards the next state is not observed (as the infeasible action is not executed) and a cost c (i.e. negative reward) is given. The RL agent then selects a new action a and a new constraint check is done. This is repeated until the constraint check gets passed and the selected action is considered to be a safe action a^{safe}. This safe action is then executed in the environment and a regular experience tuple is received. The pseudo-code is given in algorithm 2 and a graphical representation, in the form of a Markov Chain, in Fig. 2.

Note that the cost c, in this method, is preferably an informative function (i.e. a *shaped cost*), as it does not benefit from receiving a *shaped reward* when the constraint check is not passed.

4. Case study

4.1. Toolchain

A multi-energy systems simulation model, that was developed by Ceusters et al. [3], was used as it allowed for the verification of

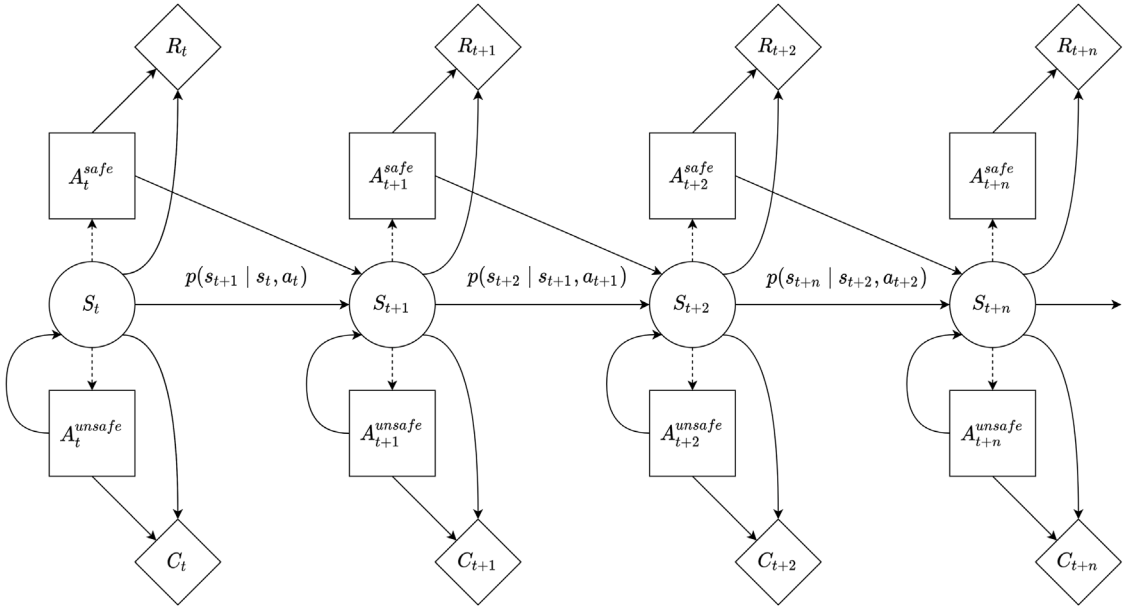


Fig. 2. Markov Chain of algorithm 2. When the selected action is infeasible (does not satisfy the constraints), that unsafe action A_t^{unsafe} is not executed in the environment so no transition to the next state S_{t+1} is observed and a cost C_t is given. When the selected actions are feasible (satisfies the constraints), those safe actions A_t^{safe} are executed in the environment so a transition to the next state S_{t+1} is observed with probability $p(s_{t+1} \mid s_t, a_t)$ and a reward R_t is given.

Algorithm 2: GiveSafe

```
Input: initialize RL algorithm, initialize constraint functions in
  sets X and U
for k = 0, 1, 2, ... do
    Observe state s and select action a
    if constraint check = True then
        keep selected action a as safe action a^safe
    else
        while constraint check = False do
            give experience tuple (s, a, c, s, d) with cost c
            agent selects new action a
            check constraints
        end
        return safe action a^safe
    end
    Execute a^safe in the environment
    Observe next state s', reward r and done signal d to
      indicate whether s' is terminal
    Give experience tuple (s, a^safe, r, s', d)
    If s' is terminal, reset environment state
end
```

the safety-critical operation (e.g. if all energy demands are fulfilled) of the multi-energy system without consequences (i.e. without the risk of violating real-life constraints and its associated harm). It is a `Modelica` [35] model, as it allowed for the convenient construction of the real-life *presumed* system dynamics using multi-physical first-principle equations and due to the available highly specialized libraries, elementary components and its object-oriented nature.

This `Modelica` model is then exported as a co-simulation *functional mock-up unit* (FMU), similar to [36], and wrapped into an `OpenAI` gym *environment* [37] in `Python`, similar to [3,38]. The architecture of the toolchain is shown in Fig. 3. Notice that the Differential Algebraic Equations solver is part of the co-simulation FMU and that the `do_step()` method in `PyFMI` [39] is used over `simulate()` - again as in [3] due to the significant run-time speed-up when initialized properly.

Table 1
Dimensions of the multi-energy system.

Energy asset	Input	Output	P_{nom}	P_{min}	E_{nom}
Transformer	elec	elec	$+\infty$	$-\infty$	
Wind turbine	wind	elec	0.8 MW_e	1.5%	
Solar PV	solar	elec	1.0 MW_e	0%	
Boiler	CH_4	heat	2.0 MW_{th}	10%	
Heat pump	elec	heat	1.0 MW_{th}	25%	
CHP	CH_4	heat	1.0 MW_{th}	50%	
	CH_4	elec	0.8 MW_e	50%	
TESS	heat	heat	+0.5 MW_{th}	−0.5 MW_{th}	3.5 MWh
BESS	elec	elec	+0.5 MW_e	−0.5 MW_e	2.0 MWh

4.2. Simulation model

The considered multi-energy system is similar to the one from Ceusters et al. [3], yet without the gas turbine (back-up genset),[2] and has the following structure (see Fig. 4):

It includes (from left to right, from top to bottom): an electric transformer, a wind turbine, a photovoltaic (PV) installation, a natural gas boiler, a heat pump (HP), a combined heat and power (CHP) unit, a thermal energy storage system (TESS) and a battery energy storage system (BESS). The dimensions of the considered multi-energy system are also from [3] and are summarized in Table 1.

While the dynamic simulation model is a detailed system of differential–algebraic equations (2.548 equations and thus an equal amount of variables [3]), it does not include any virtual control system (e.g. PID controllers, as shown in Fig. 1). In this case study, we make an abstraction of this control system. This means that the executed actions, in the simulation environment, will be kept constant over the considered control horizon (15 min) and that there is no continuous unconstrained error handling (i.e. minimization of the difference between the desired set-point and a measured process variable). While we believe this simplification is fair (and made for every considered energy management algorithm, so that the comparison is valid), it does

[2] As it is not required, nor does it provide additional value, to test the proposed methods.

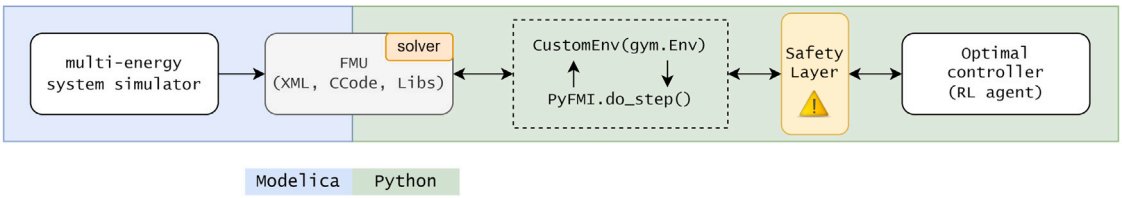


Fig. 3. Architecture of the tool-chain.

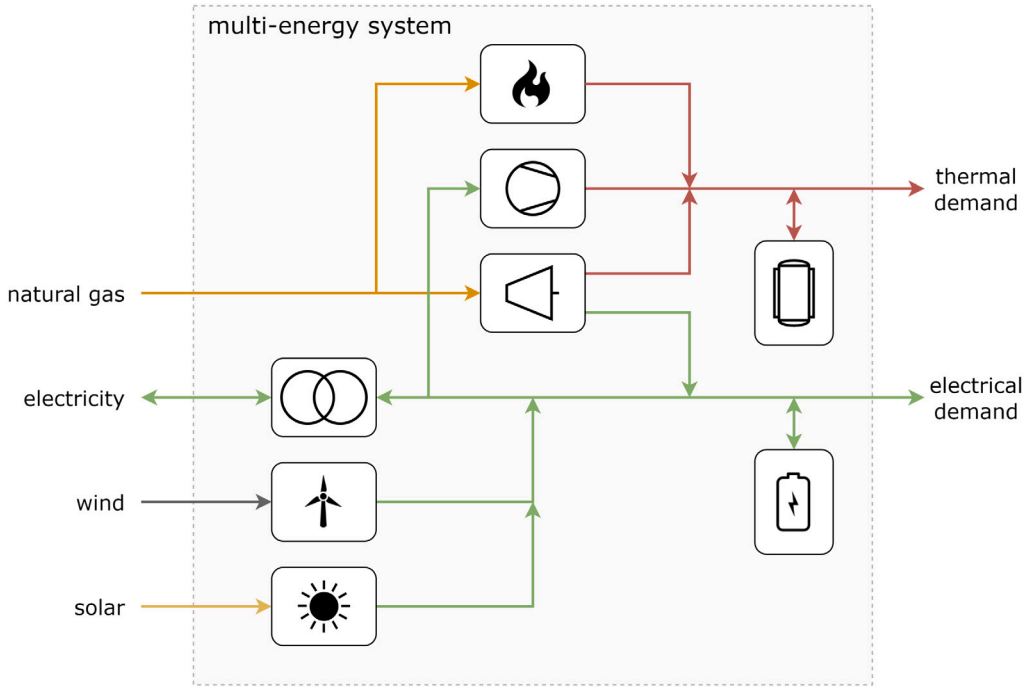


Fig. 4. Structure of the simulated multi-energy system.

result in a control error of approximately 5%. This control error was determined by separate simulations using a reduced control horizon (5 s).

4.3. Safety layer

The constraint functions, Eqs. (1c) and (1d), are in this case study specifically (see note of Eq. (3c) regarding modelling effort):

$$Q^{min}_{boil} \times \gamma^t_{boil} \leq Q^t_{boil} \leq Q^{max}_{boil} \times \gamma^t_{boil} \qquad \forall t, \quad \gamma^t_{boil} \in \{0,1\} \tag{2a}$$

$$\begin{bmatrix} Q^{min}_{hp} \\ P^{min}_{hp} \end{bmatrix} \times \gamma^t_{hp} \leq \begin{bmatrix} Q^t_{hp} \\ P^t_{hp} \end{bmatrix} \leq \begin{bmatrix} Q^{max}_{hp} \\ P^{max}_{hp} \end{bmatrix} \times \gamma^t_{hp} \qquad \forall t, \quad \gamma^t_{hp} \in \{0,1\} \tag{2b}$$

$$\begin{bmatrix} Q^{min}_{chp} \\ P^{min}_{chp} \end{bmatrix} \times \gamma^t_{chp} \leq \begin{bmatrix} Q^t_{chp} \\ P^t_{chp} \end{bmatrix} \leq \begin{bmatrix} Q^{max}_{chp} \\ P^{max}_{chp} \end{bmatrix} \times \gamma^t_{chp} \qquad \forall t, \quad \gamma^t_{chp} \in \{0,1\} \tag{2c}$$

$$Q^{min}_{tess} \leq Q^t_{tess} \leq Q^{max}_{tess} \qquad \forall t \tag{2d}$$

$$P^{min}_{bess} \leq P^t_{bess} \leq P^{max}_{bess} \qquad \forall t \tag{2e}$$

$$Q^t_{production} = Q^t_{demand} \qquad \forall t \tag{2f}$$

where Q^t_{boil}, Q^t_{hp}, Q^t_{chp} and Q^t_{tess} are the thermal powers of the natural gas boiler, the heat pump, the combined heat and power unit (CHP) and the thermal energy storage system (TESS) respectively while P^t_{hp}, P^t_{chp} and P^t_{bess} represent the electrical powers of the heat pump, CHP and battery energy storage system (BESS), all constraint by its associated minimal and maximal power (see Table 1). Furthermore, γ^t_{boil}, γ^t_{hp} and γ^t_{chp} are binary variables that turn on/off the given asset (i.e. as the minimal powers are not zero). Yet these constraints, i.e. Eq. (2a) till Eq. (2e), are easily handled by the dimensions of the (control) action space itself, i.e. Eq. (4b) till Eq. (4f), and therefore do not require a specific constraint check.

While the electrical energy balance is always fulfilled (given the assumption that the electrical grid connection is sufficiently large), the thermal energy balance (Eq. (2f)) does require special attention in order to achieve hard-constraint satisfaction. No additional constraints are considered in this case study (e.g. ramping rates, minimal run- and downtime) as energy balance equations are considered to be the most limiting constraints in energy management problems. Writing out the thermal energy balance in more detail and relaxing the equality constraint formulation (towards an inequality constraint) then becomes:

$$\left| Q^t_{boil} + Q^t_{hp} + Q^t_{chp} + Q^t_{tess} - Q^t_{demand} \right| \leq Q_{tol} \qquad \forall t \tag{3a}$$

$$\left| A^t_{boil} \cdot \eta_{boil} \cdot Q^{max}_{boil} + A^t_{hp} \cdot \frac{COP_{hp}}{COP^{max}_{hp}} \cdot Q^{max}_{hp} + A^t_{chp} \cdot \eta^{th}_{chp} \cdot Q^{max}_{chp} + A^t_{tess} \cdot f(SOC^t_{tess}) - Q^t_{demand} \right| \leq Q_{tol} \qquad \forall t \tag{3b}$$

$$\left| A^t_{boil} \cdot f(T^t_{boil}) \cdot Q^{max}_{boil} + A^t_{hp} \cdot \frac{f(T^t_{evap}, T^t_{cond})}{COP^{max}_{hp}} \cdot Q^{max}_{hp} + A^t_{chp} \cdot f(P^t_{chp}, Q^t_{chp}, T^t_{env}) \cdot Q^{max}_{chp} + A^t_{tess} \cdot f(\overline{T^t_{tess}}) - Q^t_{demand} \right| \leq Q_{tol} \qquad \forall t \tag{3c}$$

Table 2
Safety layer model metrics with `test_size` of 0.25. Mean Absolute Error (MAE), Normalized Mean Absolute Error (NMAE) by range, i.e. NMAE = MAE/range(actual values).

Energy asset	R2-score	MAE	NMAE
Boiler	99.92%	7.2 kW	0.34%
Heat pump	99.74%	6.1 kW	0.62%
CHP	99.86%	4.3 kW	0.38%
TESS	96.22%	12.9 kW	1.37%
BESS	99.43%	4.6 kW	0.46%

where A^t are the (control) actions, η the energy efficiencies, COP the coefficient of performance, SOC the state of charge and T various specific temperatures (i.e. T^t_{boil} the return temperature to the boiler, T^t_{evap} the evaporator temperature of the heat pump, T^t_{cond} the condenser temperature of the heat pump, T^t_{env} the environmental air temperature and $\overline{T^t_{tess}}$ the average temperature in the stratified hot water storage tank). We set Q_{tol} to be 15,0% of the total Q^t_{demand} in the evaluation period, which we acknowledge to be relatively high — yet is chosen to keep the computational complexity low (Appendix D). Note that the different functions $f(\cdot)$ from Eq. (3c) are typically not trivial to *model* accurately. Therefore we assume the availability of a historical dataset to supervisory learn (using a Random Forest Regression algorithm) the function between the thermal power and the action directly (i.e. $Q^t_{asset} = f(A^t_{asset}, \chi^t_{asset})$), with the possibility to include informative exogenous variables χ^t_{asset}. This dataset is generated from the simulation model, rather than using the underlying differential algebraic equations, to mimic a real-life case study.

4.4. Safe fallback policy

As presented in Section 3.1, algorithm 1 relies on an *a prior* safe fallback policy π^{safe} which can be *any* (non-optimal) policy that satisfies the constraints and can typically be provided by domain experts. In our case study, this is a simple priority rule:

Algorithm 3: safe fallback policy

```
if Q^t_demand < Q^min_chp then
    Q^t_chp = 0
    Q^t_boil = Q^t_demand
else
    if Q^t_demand < Q^max_chp then
        Q^t_chp = Q^t_demand
        Q^t_boil = 0
    else
        Q^t_chp = Q^max_chp
        Q^t_boil = Q^t_demand − Q^max_chp
    end
end
```

Note that, for clarity concerns, the policy has been written out in terms of thermal power outputs yet is still converted to actions A^t_{chp} and A^t_{boil} as going from Eq. (3a) to Eq. (3b).

4.5. Energy managing RL agent

The fully observable discrete-time Markov decision process (MDP) is formulated as the tuple $\langle S, A, P_a, R_a \rangle$ so that:

$$S^t = (E^t_{th},\ E^t_{el},\ P^t_{wind},\ P^t_{solar},\ X^t_{el},\ SOC^t_{tess},\ SOC^t_{bess},\ h^t,\ d^t) \qquad S^t \in S \tag{4a}$$

$$A^t_{boil} = (0,\ A^{min}_{boil} \rightarrow A^{max}_{boil}) \qquad A^t_{boil} \in A \tag{4b}$$

$$A^t_{hp} = (0,\ A^{min}_{hp} \rightarrow A^{max}_{hp}) \qquad A^t_{hp} \in A \tag{4c}$$

$$A^t_{chp} = (0,\ A^{min}_{chp} \rightarrow A^{max}_{chp}) \qquad A^t_{chp} \in A \tag{4d}$$

Table 3
TD3 hyper-parameters.

Hyper-parameters: TD3	Algorithm 1	Algorithm 2	Unsafe	Optsafe
Gamma	0.7	0.95	0.9	0.7
Learning_rate	0.000583	0.000119	0.0003833	0.000583
Batch_size	16	16	100	16
Buffer_size	1e6	1e5	1e5	1e6
Train_freq	1e0	1e1	2e3	1e0
Gradient_steps	1e0	1e1	2e3	1e0
Noise_type	normal	normal	normal	normal
Noise_std	0.183	0.791	0.329	0.183

$$A^t_{tess} = (A^{min}_{tess} \rightarrow A^{max}_{tess}) \qquad A^t_{tess} \in A \tag{4e}$$

$$A^t_{bess} = (A^{min}_{bess} \rightarrow A^{max}_{bess}) \qquad A^t_{bess} \in A \tag{4f}$$

$$R_a = -(a \times L^t_{cost} + b \times L^t_{comfort}) - c \tag{4g}$$

where E^t_{th} is the thermal demand, E^t_{el} the electrical demand, P^t_{wind} the electrical wind in-feed, P^t_{solar} the electrical solar in-feed, X^t_{el} the electrical price signal (i.e. day-ahead spot price), SOC^t_{tess} the state-of-charge (SOC) of the TESS, SOC^t_{bess} the SOC of the BESS, h^t the hour of the day and d^t the day of the week all at the tth step, which constitute the state-space S. The action-space A includes the control set-points from, A^t_{boil} the natural gas boiler, A^t_{hp} the heat pump, A^t_{chp} the CHP unit, A^t_{tess} the TESS and A^t_{bess} the BESS all between a minimum and maximum power rate as shown in Table 1.

The objective of the energy managing agent is given by the reward function R_a (i.e. where we try to maximize a negative function, and thus minimize the positive version of that function, in accordance with Eq. (1a)) and is the negative loss in energy costs L^t_{cost} and loss in comfort $L^t_{comfort}$ both at the tth time-step with scalarization weights a and b and with an additional cost c when the constraints are *expected* to be violated.[3] The loss in comfort is defined as $|E^t_{th} - Q^t|$, where Q^t is the thermal energy production. The electrical demand and natural gas consumption can always be fulfilled by (buying from) their respective *infinitely* large main grid connection, i.e. within the `Modelica` simulation model, it is assumed that the grid connections are sufficiently large. Note that the loss in comfort $L^t_{comfort}$ is bound by the tolerance of Eq. (3c). This term, in the reward function, therefore serves as a fine-tuning mechanism to further minimize the loss in comfort within that bound (to guide the RL agents towards safer actions) *and* to mitigate the modelling error of the constraints itself (see Table 2 for the quality of the constraint functions).

The energy costs L^t_{cost} is in EUR with scalarization weight $a = 1/10$, the loss in comfort $L^t_{comfort}$ is in Watt with scalarization weight $b = 1/5e5$ and cost $c = 1$ in algorithm 1 and $c = -50 + (10$ **if** $A^t_{chp} > 0.5$ **else** $0)$ in algorithm 2. The discrete-time control horizon is 15 min. The state-space S is normalized and all actions in the action-space A are scaled between $[+1, -1]$, while the scalarization weights are from Ceusters et al. [3].

Finally, we use a twin delayed deep deterministic policy gradient (TD3) agent, as it is considered one of the state-of-the-art RL algorithms, from the `stable baseline` [40] implementations and this with the following hyper-parameters (see Table 3, found after a hyper-parameter optimization study, similar to [3], for algorithms 1 and 2). The pseudo-code of the TD3 algorithm is given in Appendix C.

We would like to emphasize that the RL agents themselves do not need an "offline" pre-training step using an *a priori* dataset and that they are safely trained "online" (although simulated here). The RL agents, therefore, start without any prior knowledge of the environment, as their dataset (experience tuples) is generated while they

[3] As these unsafe actions are not executed in the environment — see algorithms 1 and 2.

Table 4
5-run average policy performance with a training budget of 15 years worth of time steps per run (i.e. 525.150 time steps per run).

EMS algorithm	Objective value		Constraint
	Absolute	Relative	tolerance
Unsafe TD3	−5.043	100,0%	21,0%
Unsafe Random	−14.223	35,5%	146,0%
OptLayer TD3	**−4.850**	**104,0%**	10,4%
OptLayer Random	−6.481	77,8%	15,6%
SafeFallback TD3	−4.899	102,9%	10,1%
SafeFallback Random	−5.331	94,6%	7,0%
SafeFallback (π^{safe})	−5.228	96,5%	**6,3%**
GiveSafe TD3	−5.137	98,2%	10,0%
GiveSafe Random	−6.089	82,8%	15,0%

interact with the environment (a multi-energy system in our case study).

4.6. Evaluation

The performance, in terms of energy cost minimization subject to the (thermal comfort) constraint fulfilment, of the proposed methods algorithms 1 and 2 is compared against an unconstrained (and therefore possibly unsafe) RL agent, the OptLayer RL agent proposed by Pham et al. [34] (identified as related work), as well as against safe and unsafe random agents. The random agents serve as minimal learning benchmarks for the associated algorithm (i.e., how much did the agent learn while safely interacting with its environment) and is also the performance of that algorithm before any training has occurred. These random agents are, therefore, not intended as viable optimal controllers. We use a year-long training environment, a 15-min control horizon (i.e. the energy managing RL agent can select new actions every simulated 15 min[4]) and a week-long evaluation environment while participating in a day-ahead electricity market. Any uncertainty (from e.g. demands, prices or wind and solar power generation) is inherently handled by the RL agent, as it is formulated as a discrete time-invariant infinite-horizon *stochastic* optimal control problem (see [3] for the derivation from a continuous time-varying stochastic system). The model-predictive controller from [3] is here not considered, as constraints can be formulated directly in the method.

5. Results and discussion

The simulation results of the objective values (i.e. rewards) are shown, in Table 4, both in absolute values as relative to the unconstrained RL benchmark. The *inequality* constraint tolerance, Eq. (3c), is shown relative to the total demand (0% would mean *equality* constraint satisfaction, i.e. all thermal demand is being fulfilled including any thermal energy storage charging actions).

These results (Table 4) show that algorithm 1 (SafeFallback: 102,9%) outperforms algorithm 2 (GiveSafe: 98,2%) and the *vanilla* unsafe TD3 benchmark (100%), yet is slightly worse then OptLayer (104,0%). This as, using the *a prior* safe fallback policy has the highest utility before training (94,6% compared to 82,8%, 35,5% and 77,8%), indicating the additionally given expert knowledge. The additional expert knowledge (of the safe fallback policy of algorithm 1 itself — see π^{safe}) is reflected in the initially higher constraint tolerance (7,0%) as well, yet reaches an acceptable 10,1% (below the maximum tolerance of 15%, as set in Eq. (3c)). Algorithm 2 and OptLayer have initially a higher constraint tolerance (15,0% and 15,6%), yet reaching approximately the same tolerances (10,0% and 10,4%). The SafeFallback policy itself (π^{safe}) has, as expected, the lowest constraint tolerance (6,3%). The higher tolerance of SafeFallback Random (and thus also SafeFallback TD3 — before training) can be explained by the fact that it can select safe actions by chance, i.e. random safe actions, which are bound by the maximum tolerance of 15%. Both of the proposed methods are, therefore, as intended, significantly safer than the unconstrained *vanilla* TD3 benchmark – which has an initial constraint tolerance of 146,0% and reaching only 21,0% (while training has been done unsafely, i.e. without hard-constraint satisfaction guarantees, which would not be allowed in a real environment – yet purely acts as the *vanilla* RL benchmark). Note that, OptLayer, initially violates the maximum tolerance of 15%, as set in Eq. (3c). This happens because OptLayer involves solving a mixed-integer quadratic problem to compute the nearest feasible actions. Moreover, in OptLayer linear analytical approximations are used instead of surrogate functions $f(\cdot)$ from Eq. (3c) with the metrics provided in Table 2, since there is no derivative information present.

The learning curves of the TD3 agents (using algorithms 1, 2 and OptLayer, as well as without any safety layer — indicated as UnSafe) are presented in Fig. 5, where the initial (at time step 0) and final (at time step 525.150) results are the figures reported in Table 4. We observe a steep initial learning rate, low variance, and a stable (slightly increasing) performance with an increasing number of interactions with its environment. We also observe that algorithm 1 (SafeFallback), algorithm 2 (GiveSafe) and OptLayer have a significantly higher initial performance (before any training has occurred, i.e. at time step 0) compared to its *vanilla* unsafe TD3 counterpart. This, again, is due to the *a priori* expert knowledge in the form of a known safe fallback policy and in the form of safety constraint equations. The unsafe RL agent only reaches the *initial* performance (−6.481) of OptLayer after ∼ 35.000 time steps (1 year) and of algorithm 2 (−6.089) after ∼ 50.000 time steps (1 year and 5 months) and of algorithm 1 (−5.331) after ∼ 85.000 time steps (2 years and 5 months). Furthermore, the initial performance of Optlayer is lower than algorithms 1 and 2, surpassing algorithm 2 after ∼ 18.575 time steps (6 months) and algorithm 1 after ∼ 30.000 time steps (10 months) and that the performance gap remains significant with algorithm 2 (5,8%) while algorithm 1 reaches a similar performance (1,1%).

The cost curves (constraint tolerance) of the TD3 agents are presented in Fig. 6. We observe a steep initial decrease of the constraint tolerance of the *vanilla* TD3 agent, yet never converging to the safety threshold of 15% as defined by Eq. (3c) - while algorithms 1 and 2 never exceed this threshold (i.e. proving the hard-constraint satisfaction during training) and OptLayer slightly exceeds this threshold as noted here before. The constraint tolerance convergence of all safe methods is less steep and reaches a stable performance (∼ 10%) after approximately 3 years (∼ $1e^5$ time steps). The constraint tolerances converge towards this limit (∼ 10%) as defined by the multi-objective reward function, Eq. (4g), and its associated scalarization weights a and b (i.e. energy cost minimization and energy demand fulfilment are conflicting objectives). Hence, without the $L^t_{comfort}$ term specifically, the constraint tolerance would converge to the maximum tolerance of 15%, as pure energy cost minimization would try to avoid consuming costly energy as much as possible (without either the constraints or the $L^t_{comfort}$ term — all controllable energy assets would be switched off when having positive energy prices). Therefore we observe both declining cost curves (starting above the converging limit) and an increasing cost curve (starting below the converging limit).

However, the performance in terms of both the utility (objective value) and cost (constraint tolerance) of algorithm 1 (SafeFallback), algorithm 2 (GiveSafe) and OptLayer relies on an accurate formulation of the actual constraints, i.e. the accurate formulation of Eq. (3c) in this case study. As presented in Section 4.3, this is not always trivial — especially for the TESS and the HP. When we replace the pre-trained TESS and HP functions from the safety layer, with simpler (linear) analytical equations we observe a reduction in performance (i.e. the equations also used in OptLayer). For example, for algorithm 1 by its initial objective value of −5.331 to −5.431 and its initial constraint

[4] Given the absence of unconstrained error handling, this does result in a control error of approx. 5%, as mentioned in Section 4.2.

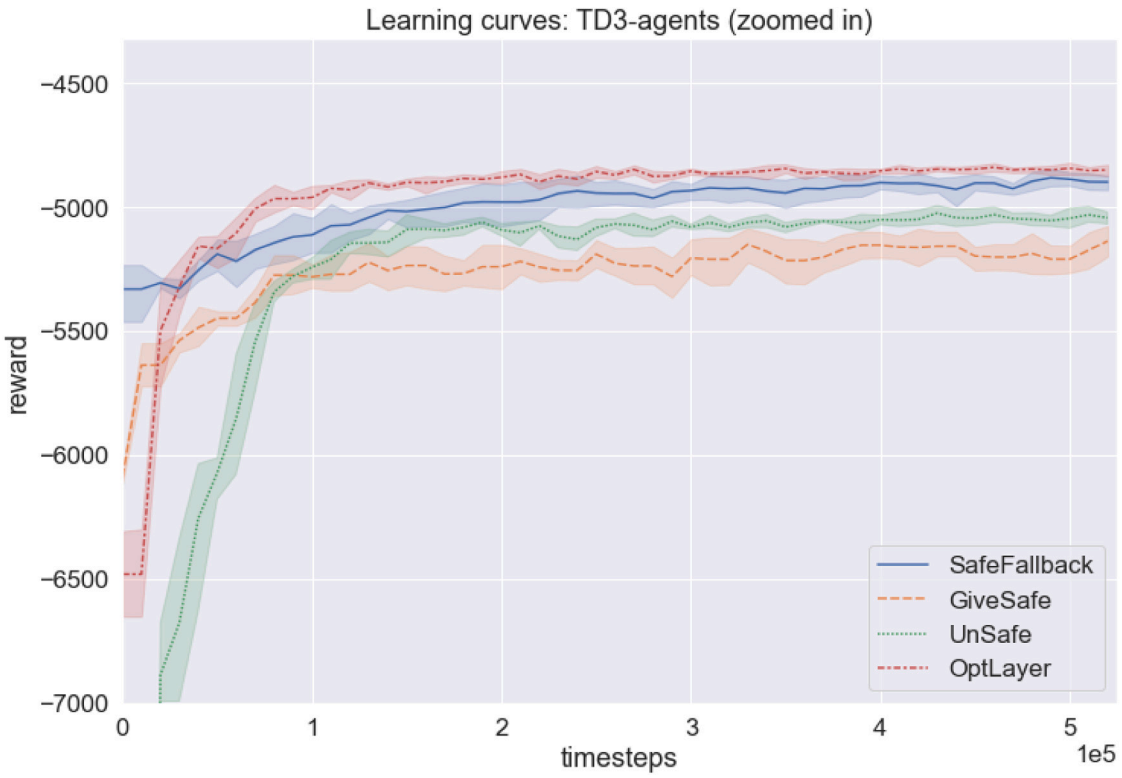


Fig. 5. 5-run average learning curves with a training budget of 15 years worth of time steps per run (i.e. 525.150 time steps per run). Note that the y-axis is zoomed in (see Fig. B.1 in Appendix B for the zoomed-out version).

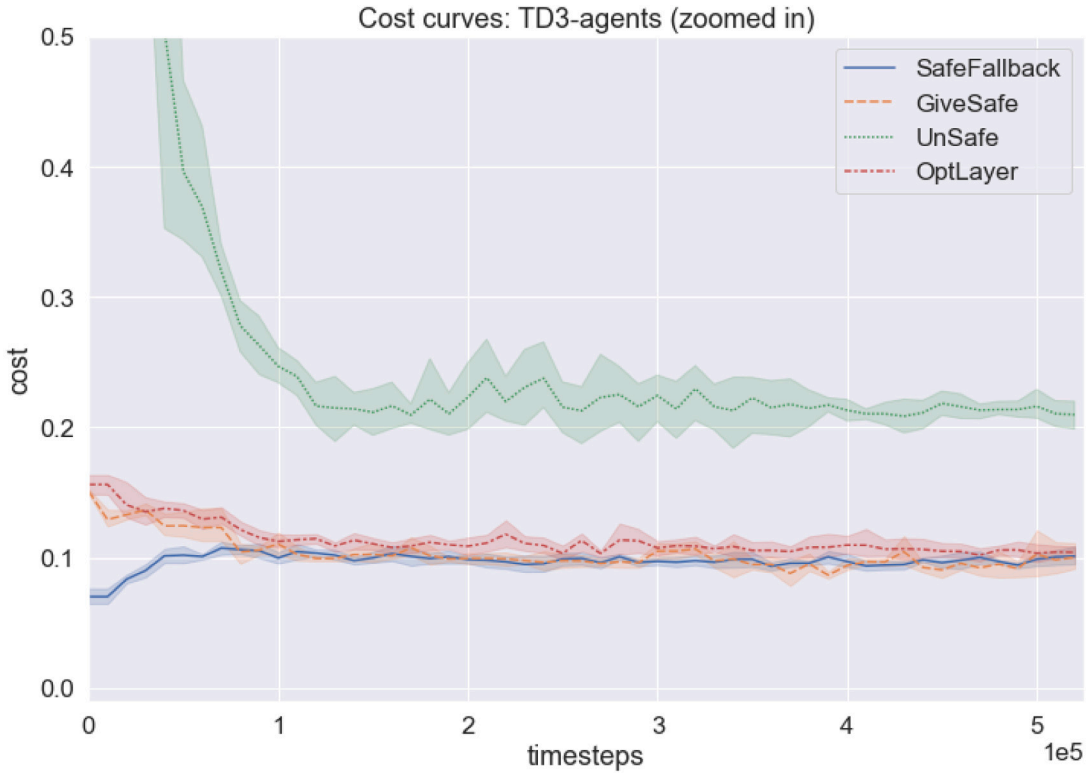


Fig. 6. 5-run average cost curve (i.e. constraint tolerance) with a training budget of 15 years worth of time steps per run (i.e. 525.150 time steps per run). Note that the y-axis is zoomed in (see Fig. B.2 in Appendix B for the zoomed-out version).

tolerance of 7,0% to 7,6%. This problem can be mitigated, however, by artificially lowering Q_{tol} from Eq. (3c).

Nevertheless, training a RL agent on a real safety-critical environment would only be possible with a sufficiently accurate safety layer (i.e. constraints) and using adequate *a priori* unknown hyper-parameters. Therefore, we propose the following directions for future work:

- Providing chance-constraint satisfaction guarantees for when no constraint functions are available *a priori* (but only limited, known to be safe, operational data) and to safely improve the *a prior* constraint functions as more data becomes available. Exploratory and exploitative steps will then never leave the safe region with high probability, by updating a statistical model (e.g. a Gaussian Process model).
- Further reducing the training budget (i.e. improving sample efficiency, e.g. by combining SafeFallback with OptLayer) and rolling out a fixed sequence of (robust[5]) control actions (e.g. using model-based RL agents) for day-ahead market *planning*.
- Robustness of the RL-based energy management systems under faulty and noisy measurements/observations and utilizing *online* hyper-parameter optimization methods (i.e. that the hyper-parameters are tuned during online training).

6. Conclusion

This paper presented two novel model-free safe reinforcement learning (RL) methods of the following *combined* characteristics: (i) providing hard-constraint, rather than soft- and chance-constraint, satisfaction guarantees with multiple constraints, (ii) which is decoupled from the RL (as a Markov Decision Process) formulation, (iii) both during training a (near) optimal policy (which involves exploratory and exploitative, i.e. greedy, steps) as well as during deployment of any policy (e.g. random agents or offline trained RL agents) and (iv) this without the need of solving a mathematical program. These methods were demonstrated in a multi-energy management systems context, where detailed simulation results are provided.

Both of the proposed methods are viable safety constraint handling techniques applicable beyond state-of-the-art RL, as demonstrated by random agents while still providing strict safety guarantees. Preferably, however, algorithm 1 (SafeFallback) is used as it showed good performance, does not require solving a mathematical program (e.g. a mixed-integer quadratic program in the case of OptLayer), and as the availability of a simple safe fallback policy is common or relatively easily constructible (i.e. in the form of a simple rule-based policy, e.g. a priority-based control strategy).

CRediT authorship contribution statement

Glenn Ceusters: Conceptualization, Methodology, Software, Validation, Formal analysis, Resources, Data curation, Writing – original draft, Visualization, Funding acquisition. **Luis Ramirez Camargo:** Conceptualization, Writing – review & editing, Supervision. **Rüdiger Franke:** Supervision. **Ann Nowé:** Writing – review & editing, Supervision. **Maarten Messagie:** Supervision.

Data availability

The data that has been used is confidential.

[5] The transition probability matrix can then also be used to generate a *robust* planning rather than a pure *most-likelihood* planning.

Acknowledgements

This work has been supported in part by ABB n.v., Belgium and Flemish Agency for Innovation and Entrepreneurship (VLAIO) grant HBC.2019.2613, Belgium.

Appendix A. Simulations visualization

In this section, we show a time series visualization sample (a week) of the found control policies. The first observation that can be made (in Fig. A.1) is the violently unsafe behaviour (146,0% of constraint tolerance, Table 4) of the TD3 agent before training, which at this stage acts as an unconstrained random agent. Specifically, at this stage, a large thermal overproduction is the cause of the thermal discomfort and thus the constraint violation (as the total thermal installed capacity, given in Table 1, is significantly higher than the thermal demand, e.g. due to the back-up boiler capacity — and given the random behaviour before training, the sum of the total thermal output is expected to be significantly high). The overproduction is avoided after training the TD3 agent (Fig. A.2). The policy itself has a high utility, yet now a significant thermal underproduction is observed (21,0% of constraint tolerance, Table 4). In practice, the natural gas boiler could be forced on to satisfy the thermal underproduction (yet this by itself would be an *a prior* "fallback" policy).

When analysing the SafeFallback (algorithm 1) policies, and comparing them against the *vanilla* unsafe TD3 policies, we observe safe behaviour. Before training, the constraint check mostly fails — using the safe fallback policy. Initially (Fig. A.3), when the constraint check passes, safe random actions are observed (e.g. thermal "overproduction" is properly stored in the TESS). After *safely* training the TD3 agent, a policy with a high utility and a low constraint tolerance is observed (Fig. A.4). Thermal underproduction is still present, yet within the set bound Q_{tol} from Eq. (3c).

When analysing the GiveSafe (algorithm 2) policies, and comparing them against the *vanilla* unsafe TD3 policies, we again observe safe behaviour. Before training (see Fig. A.5), *all* actions are random but safe (e.g. thermal demand is matched by the thermal production or any thermal "overproduction" is stored in the TESS) - resulting in both a lower initial utility and higher constraint tolerance as Fig. A.3. After *safely* training the TD3 agent, a policy with a high utility and a low constraint tolerance is observed (Fig. A.6) - yet with a lower utility as Fig. A.4. Thermal underproduction is again still present, yet within the set bound Q_{tol} from Eq. (3c).

Finally, when analysing the OptLayer policies, we again observe safe behaviour (even though the maximum tolerance of 15% is slightly violated, i.e. 15,6% as discussed before). Before training (Fig. A.7), all the actions proposed by the TD3 agents are random and are therefore corrected towards the closed feasible actions (see [34] for the details of this algorithm). Even though this resembles the policy from algorithm 2, these actions are then no longer completely random and almost always result in a distribution among actions (every continuous action all have some part in the feasible solution) and this results in a worse initial utility. After *safely* training the TD3 agent (Fig. A.8), a policy with a high utility and a low constraint tolerance is observed, yet again with some thermal underproduction (within the inequality bounds) as expected due to the conflicting objectives (the first term in Eq. (4g) minimizes the energy costs and therefore the production) (see Fig. A.5).

Appendix B. Learning and costs curves: zoomed out

This appendix shows the zoomed-out learning and costs curves of the TD3-agents (using algorithms 1, 2 and OptLayer, as well as without any safety layer — indicated as UnSafe) so that all curves are fully visible. Hence, the UnSafe curves are visible for all time steps.

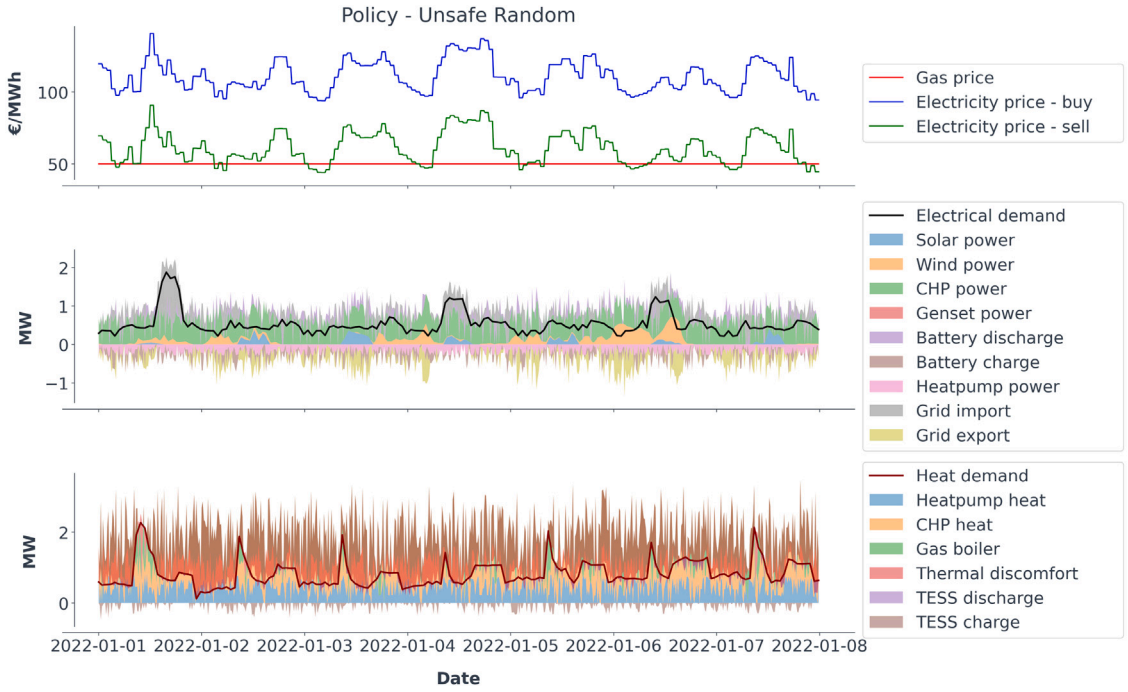


Fig. A.1. Policy visualization: unsafe random (or TD3 before training).

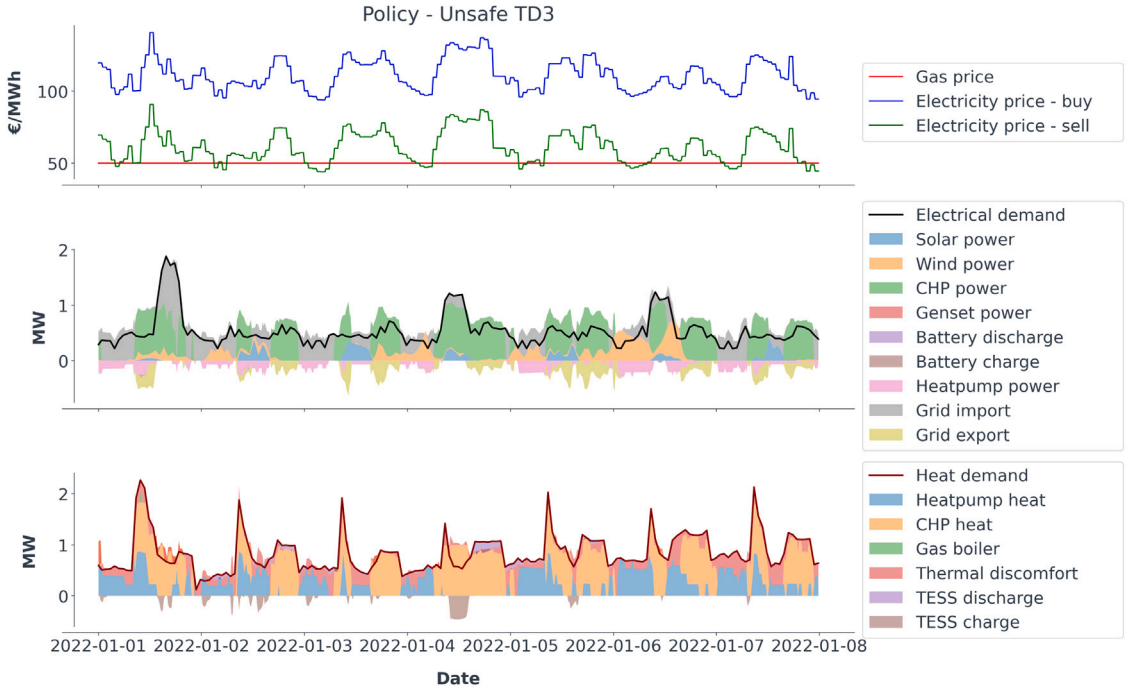


Fig. A.2. Policy visualization: unsafe TD3 (after unsafe training).

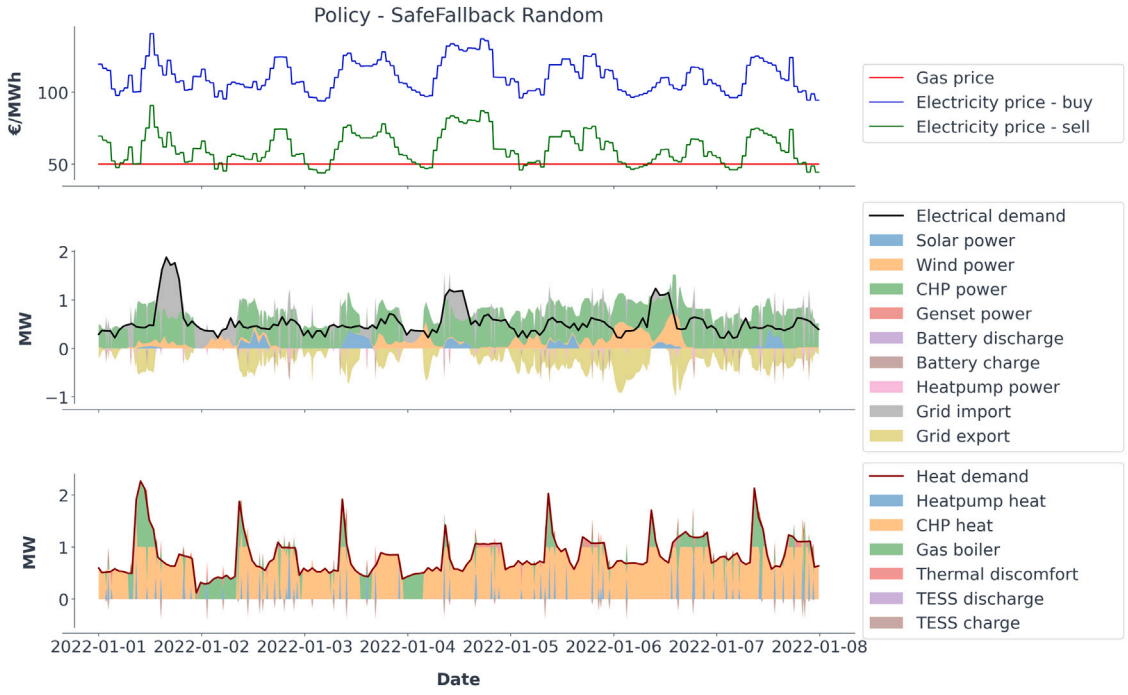


Fig. A.3. Policy visualization: SafeFallback random (or TD3 before training).

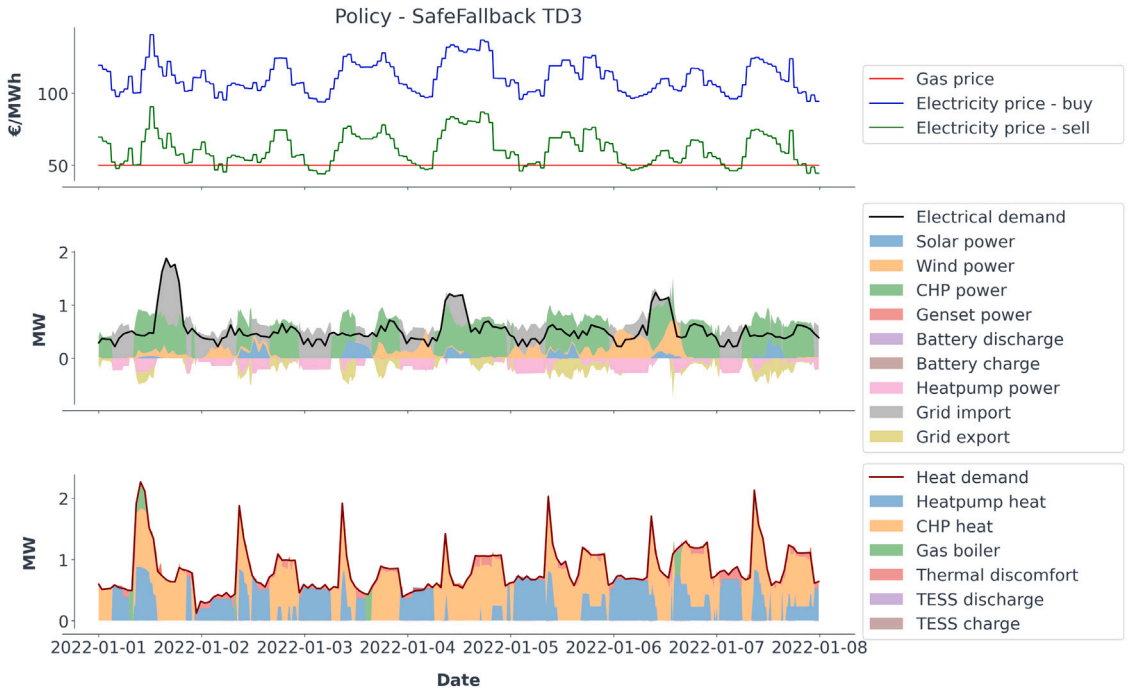


Fig. A.4. Policy visualization: SafeFallback TD3 (after safe training).

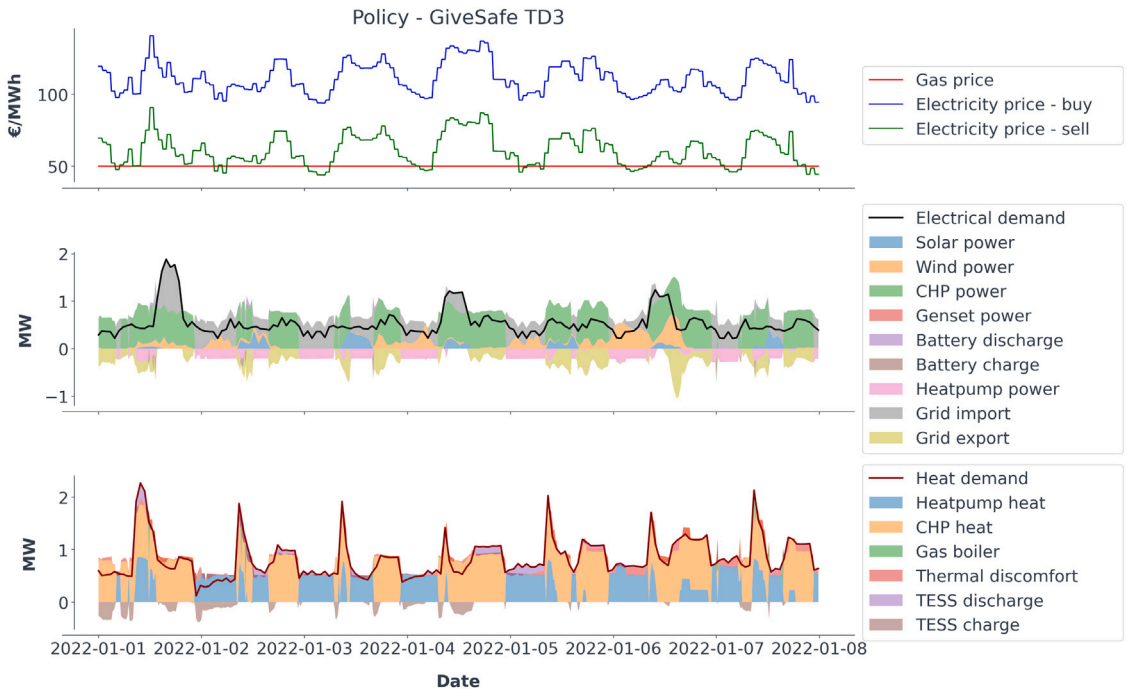


Fig. A.5. Policy visualization: GiveSafe random (or TD3 before training).

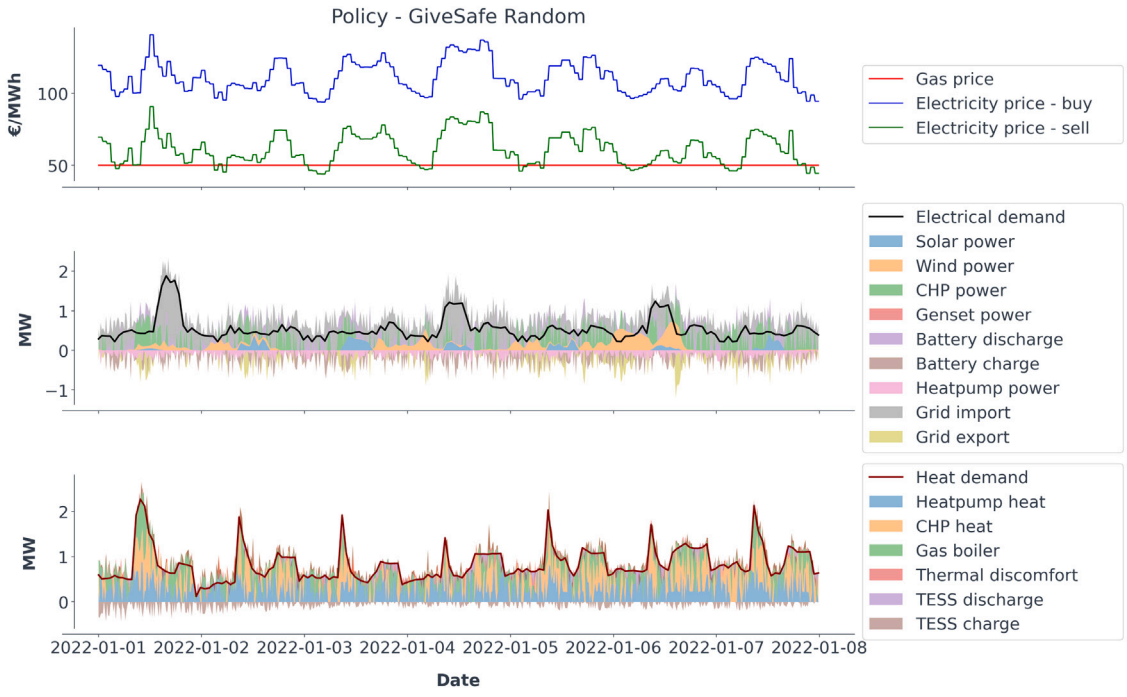


Fig. A.6. Policy visualization: GiveSafe TD3 (after safe training).

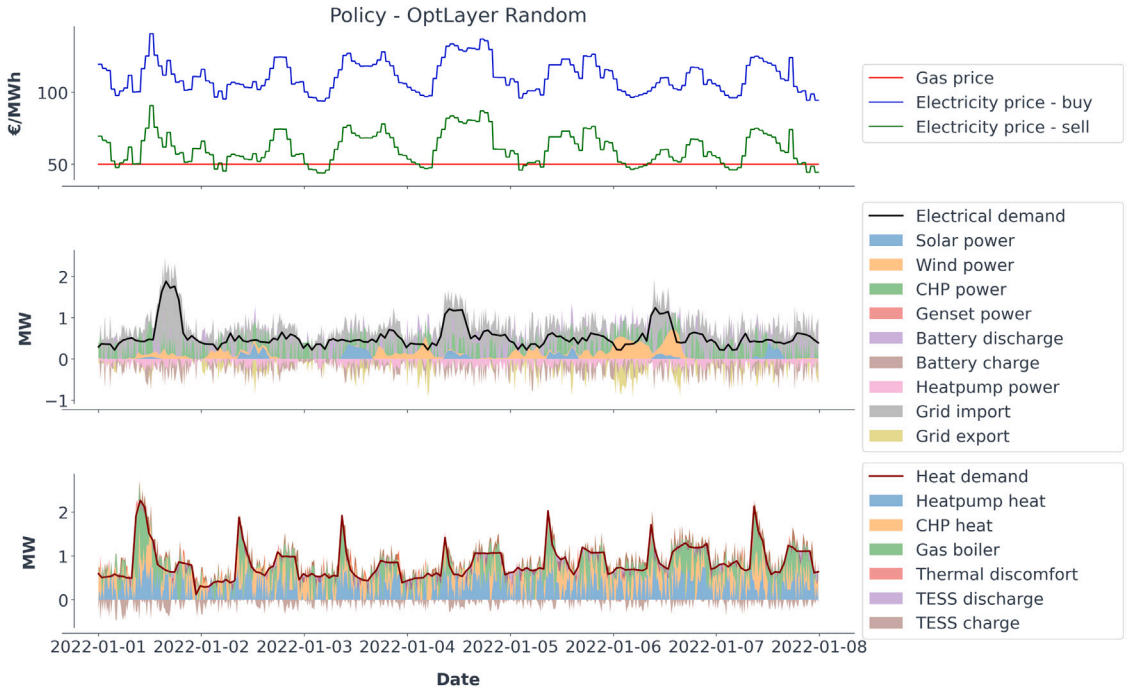


Fig. A.7. Policy visualization: OptSafe random (or TD3 before training).

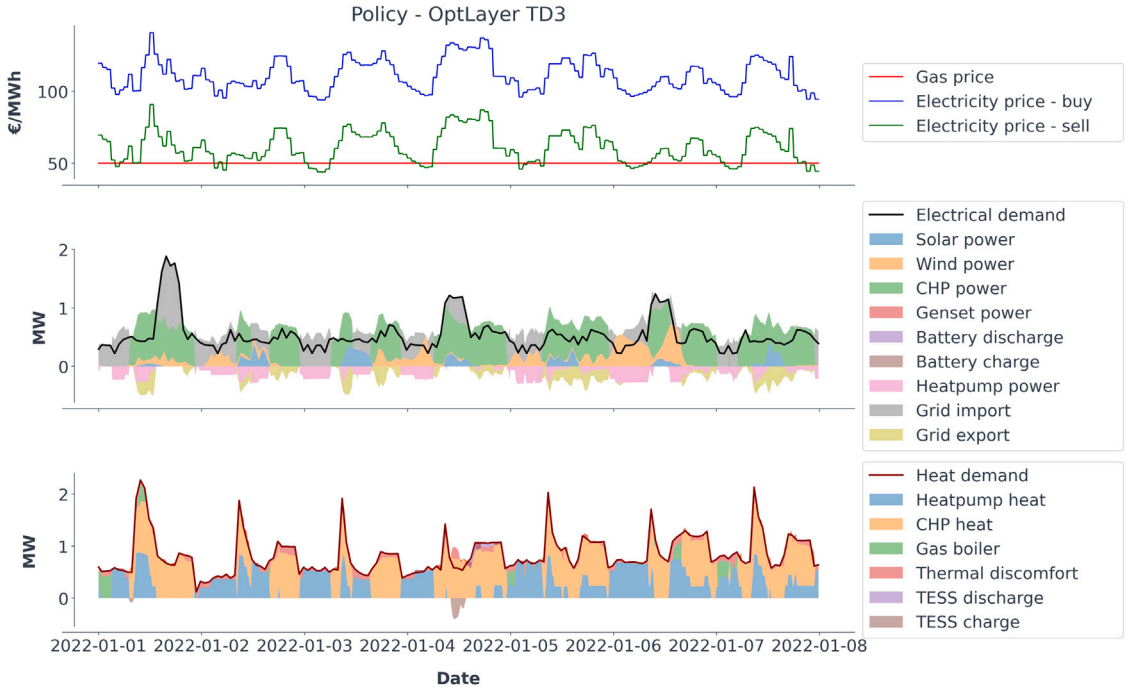


Fig. A.8. Policy visualization: OptLayer TD3 (after safe training).

Fig. B.1. 5-run average learning curves with a training budget of 15 years worth of time steps per run (i.e. 525.150 time steps per run).

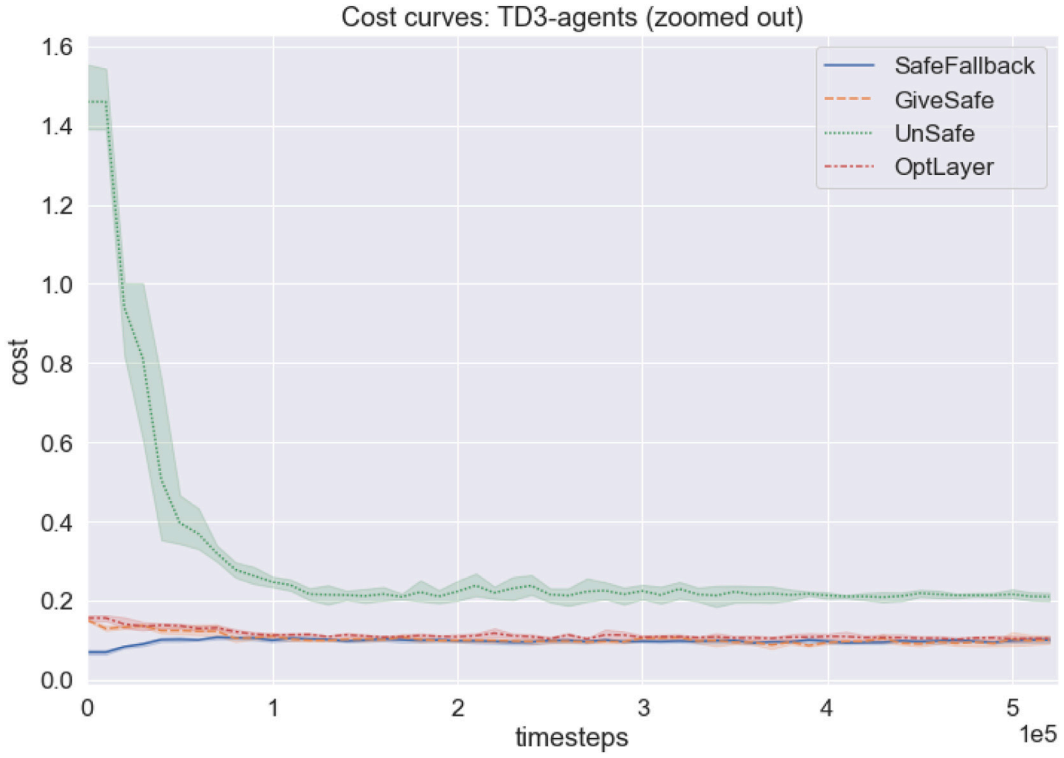


Fig. B.2. 5-run average cost curve (i.e. constraint tolerance) with a training budget of 15 years worth of time steps per run (i.e. 525.150 time steps per run).

Appendix C. Pseudo-code of TD3 [41]

Algorithm 4: Twin Delayed DDPG (TD3) [41]

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\theta_{targ,1} \leftarrow \theta_1$, $\theta_{targ,2} \leftarrow \theta_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state
  if it is time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\big(\mu_{\theta_{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\big), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{targ,i}}(s', a'(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2 \quad \text{for} \quad i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1 - \rho)\phi_i \qquad \text{for} \quad i = 1, 2$$
$$\theta_{targ} \leftarrow \rho \theta_{targ} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

Appendix D. Run-time statistics

The simulations are conducted on a local machine with an Intel® Core™ i5-8365U CPU @1.6 GHz, 16 GB of Ram and an SSD. Over a yearly simulation, the following run-time statistics per simulated time-step (with a control horizon of 15 min) are observed (see Table D.1).

The maximum run-time per time-step never exceeds the control horizon of 15 min, as this otherwise would be considered impractical with the given hardware. We observe that the unsafe agents have the fastest run-time, as they do not have the constraint check to compute. Yet, we have argued that using unsafe agents is not realistic in safety-critical environments and are here given for completeness only. Furthermore, we observe that the SafeFallback method itself (demonstrated by using random agents) is significantly faster than the GiveSafe method, as the GiveSafe method can require multiple additional "offline" time-steps for every "online" (i.e. real) time-step, and is significantly faster than OptLayer, as this involves solving a mixed-integer quadratic problem (MIQP) for every time step an infeasible action is selected by the TD3 agent. After the TD3 agents are trained though, the run-time is approximately the same — as the amount of unsafe actions proposed by the TD3 agent (and thus the need for additional "offline" training steps or MIQP solving) is greatly reduced.

Table D.1
Run-time statistics.

Optimal controller	Min	Mean	Std	Max	Total
Unsafe TD3	0,027 s	0,041 s	0,006 s	0,100 s	1.424 s
Unsafe Random	0,027 s	0,040 s	0,011 s	0,120 s	1.394 s
OptLayer TD3	0,044 s	0,069 s	0,031 s	0,398 s	2.418 s
OptLayer Random	0,041 s	0,231 s	0,084 s	6,395 s	8.091 s
SafeFallback TD3	0,042 s	0,058 s	0,008 s	0,250 s	2.047 s
SafeFallback Random	0,037 s	0,044 s	0,005 s	0,142 s	1.545 s
SafeFallback (π^{safe})	0,037 s	0,048 s	0,010 s	0,190 s	1.672 s
GiveSafe TD3	0,041 s	0,060 s	0,040 s	2,661 s	2.090 s
GiveSafe Random	0,041 s	2,272 s	3,760 s	52,390 s	79.615 s

Notice that these are the run-time statistics *after* training (i.e., pure policy execution, in the case of the TD3 agents). Including the online training run-time statistics (i.e. fitting the function approximation algorithm — which is a multi-layer perceptron in our case), the mean run-time would result in 0,058 s for the unsafe TD3 agent, 0,122 s for the OptLayer TD3 agent, 0,076 s for the SafeFallback TD3 agent and 2,229 s for the GiveSafe TD3 agent. These online training run-time statistics are still magnitudes faster than the control horizon of 15 min.

References

[1] Fabrizio E, Filippi M, Virgone J. Trade-off between environmental and economic objectives in the optimization of multi-energy systems. Build Simul 2009;2(1):29–40. http://dx.doi.org/10.1007/S12273-009-9202-4.
[2] Engell S. Feedback control for optimal process operation. J Process Control 2007;17(3):203–19. http://dx.doi.org/10.1016/J.JPROCONT.2006.10.011.
[3] Ceusters G, Rodríguez RC, García AB, Franke R, Deconinck G, Helsen L, et al. Model-predictive control and reinforcement learning in multi-energy system case studies. Appl Energy 2021;303:117634. http://dx.doi.org/10.1016/j.apenergy.2021.117634.
[4] Cao D, Hu W, Zhao J, Zhang G, Zhang B, Liu Z, et al. Reinforcement learning and its applications in modern power and energy systems: A review. J Mod Power Syst Clean Energy 2020;8(6):1029–42. http://dx.doi.org/10.35833/MPCE.2020.000552.
[5] Yang T, Zhao L, Li W, Zomaya AY. Reinforcement learning in sustainable energy and electric systems: a survey. Annu Rev Control 2020;49:145–63. http://dx.doi.org/10.1016/J.ARCONTROL.2020.03.001.
[6] Perera AT, Kamalaruban P. Applications of reinforcement learning in energy systems. Renew Sustain Energy Rev 2021;137:110618. http://dx.doi.org/10.1016/J.RSER.2020.110618.
[7] Zhou Y. Advances of machine learning in multi-energy district communities–mechanisms, applications and perspectives. Energy AI 2022;10:100187. http://dx.doi.org/10.1016/J.EGYAI.2022.100187.
[8] Rayati M, Sheikhi A, Ranjbar AM. Applying reinforcement learning method to optimize an energy hub operation in the smart grid. In: 2015 IEEE power and energy society innovative smart grid technologies conference. 2015, http://dx.doi.org/10.1109/ISGT.2015.7131906.
[9] Watkins CJCH. Learning from delayed rewards (Ph.D. thesis), King's College, Cambridge United Kingdom; 1989.
[10] Sheikhi A, Rayati M, Ranjbar AM. Demand side management for a residential customer in multi-energy systems. Sustainable Cities Soc 2016;22:63–77. http://dx.doi.org/10.1016/j.scs.2016.01.010.
[11] Mbuwir BV, Kaffash M, Deconinck G. Battery scheduling in a residential multi-carrier energy system using reinforcement learning. In: 2018 IEEE international conference on communications, control, and computing technologies for smart grids. Institute of Electrical and Electronics Engineers Inc. 2018, http://dx.doi.org/10.1109/SmartGridComm.2018.8587412.
[12] Wang X, Chen H, Wu J, DIng Y, Lou Q, Liu S. Bi-level multi-agents interactive decision-making model in regional integrated energy system. In: 2019 3rd IEEE conference on energy internet and energy system integration: ubiquitous energy network connecting everything. Institute of Electrical and Electronics Engineers Inc. 2019, p. 2103–8. http://dx.doi.org/10.1109/EI247390.2019.9061889.
[13] Ahrarinouri M, Rastegar M, Seifi AR. Multi-agent reinforcement learning for energy management in residential buildings. IEEE Trans Ind Inf 2020;1. http://dx.doi.org/10.1109/tii.2020.2977104.
[14] Ye Y, Ye Y, Qiu D, Wu X, Strbac G, Ward J. Model-free real-time autonomous control for a residential multi-energy system using deep reinforcement learning. IEEE Trans Smart Grid 2020;11(4):3068–82. http://dx.doi.org/10.1109/TSG.2020.2976771.
[15] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. In: 4th International conference on learning representations, ICLR 2016 - conference track proceedings. International Conference on Learning Representations, ICLR; 2015.

[16] Xu Z, Han G, Liu L, Martinez-Garcia M, Wang Z. Multi-energy scheduling of an industrial integrated energy system by reinforcement learning-based differential evolution. IEEE Trans Green Commun Netw 2021;5(3):1077–90. http://dx.doi.org/10.1109/TGCN.2021.3061789.

[17] Zhu D, Yang B, Liu Y, Wang Z, Ma K, Guan X. Energy management based on multi-agent deep reinforcement learning for a multi-energy industrial park. Appl Energy 2022;311:118636. http://dx.doi.org/10.1016/j.apenergy.2022.118636.

[18] Pu Y, Wang S, Yang R, Yao X, Li B. Decomposed soft actor-critic method for cooperative multi-agent reinforcement learning. 2021, arXiv.

[19] Zhou Y, Ma Z, Zhang J, Zou S. Data-driven stochastic energy management of multi energy system using deep reinforcement learning. Energy 2022;261:125187. http://dx.doi.org/10.1016/J.ENERGY.2022.125187.

[20] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv.

[21] Fujimoto S, Van Hoof H, Meger D. Addressing function approximation error in actor-critic methods. In: 35th International conference on machine learning. Vol. 4. International Machine Learning Society (IMLS); 2018, p. 2587–601.

[22] Zhang B, Hu W, Cao D, Li T, Zhang Z, Chen Z, et al. Soft actor-critic – based multi-objective optimized energy conversion and management strategy for integrated energy systems with renewable energy. Energy Convers Manage 2021;243:114381. http://dx.doi.org/10.1016/j.enconman.2021.114381.

[23] Zhang B, Hu W, Cao D, Huang Q, Chen Z, Blaabjerg F. Deep reinforcement learning–based approach for optimizing energy conversion in integrated electrical and heating system with renewable energy. Energy Convers Manage 2019;202:112199. http://dx.doi.org/10.1016/j.enconman.2019.112199.

[24] Zhang B, Hu W, Li J, Cao D, Huang R, Huang Q, et al. Dynamic energy conversion and management strategy for an integrated electricity and natural gas system with renewable energy: Deep reinforcement learning approach. Energy Convers Manage 2020;220:113063. http://dx.doi.org/10.1016/j.enconman.2020.113063.

[25] Zhang G, Hu W, Cao D, Zhang Z, Huang Q, Chen Z, 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 Convers Manage 2022;255:115340. http://dx.doi.org/10.1016/j.enconman.2022.115340.

[26] García J, Fernández F. A comprehensive survey on safe reinforcement learning. J Mach Learn Res 2015;16:1437–80.

[27] Dulac-Arnold G, Levine N, Mankowitz DJ, Li J, Paduraru C, Gowal S, et al. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Mach Learn 2021;110(9):2419–68. http://dx.doi.org/10.1007/S10994-021-05961-4/TABLES/11.

[28] Nweye K, Liu B, Stone P, Nagy Z. Real-world challenges for multi-agent reinforcement learning in grid-interactive buildings. Energy AI 2022;10:100202. http://dx.doi.org/10.1016/J.EGYAI.2022.100202.

[29] Brunke L, Greeff M, Hall AW, Yuan Z, Zhou S, Panerati J, et al. Safe learning in robotics: From learning-based control to safe reinforcement learning. Ann Rev Control Robot Auton Syst 2021;5(1). http://dx.doi.org/10.1146/annurev-control-042920-020211.

[30] Venayagamoorthy GK, Sharma RK, Gautam PK, Ahmadi A. Dynamic energy management system for a smart microgrid. IEEE Trans Neural Netw Learn Syst 2016;27(8):1643–56. http://dx.doi.org/10.1109/TNNLS.2016.2514358.

[31] Zhang Q, Dehghanpour K, Wang Z, Huang Q. A learning-based power management method for networked microgrids under incomplete information. IEEE Trans Smart Grid 2020;11(2):1193–204. http://dx.doi.org/10.1109/TSG.2019.2933502.

[32] Zhao H, Zhao J, Qiu J, Liang G, Dong ZY. Cooperative wind farm control with deep reinforcement learning and knowledge-assisted learning. IEEE Trans Ind Inf 2020;16(11):6912–21. http://dx.doi.org/10.1109/TII.2020.2974037.

[33] Park H, Min D, Ryu Jh, Choi DG. DIP-QL: A novel reinforcement learning method for constrained industrial systems. IEEE Trans Ind Inf 2022. http://dx.doi.org/10.1109/TII.2022.3159570.

[34] Pham TH, De Magistris G, Tachibana R. OptLayer - practical constrained optimization for deep reinforcement learning in the real world. In: Proceedings - IEEE international conference on robotics and automation. Institute of Electrical and Electronics Engineers Inc. 2018, p. 6236–43. http://dx.doi.org/10.1109/ICRA.2018.8460547.

[35] Mattsson SE, Elmqvist H, Otter M. Physical system modeling with Modelica. In: Control engineering practice. Vol. 6. Pergamon; 1998, p. 501–10. http://dx.doi.org/10.1016/S0967-0661(98)00047-1.

[36] Gräber M, Fritzsche J, Tegethoff W. From system model to optimal control - A tool chain for the efficient solution of optimal control problems. In: Proceedings of the 12th international modelica conference, Prague, Czech Republic, May 15-17, 2017. Vol. 132. Linköping University Electronic Press; 2017, p. 249–54. http://dx.doi.org/10.3384/ecp17132249.

[37] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. OpenAI Gym. 2016, arXiv.

[38] Lukianykhin O, Bogodorova T. ModelicaGym: Applying reinforcement learning to Modelica models. In: ACM international conference proceeding series. Association for Computing Machinery; 2019, p. 27–36. http://dx.doi.org/10.1145/3365984.3365985.

[39] Andersson C, Akesson J, Fuhrer C. PyFMI: A python package for simulation of coupled dynamic models with the functional mock-up interface. Technical Report in Mathematical Sciences, (2):Lund University; 2016, p. 1–40.

[40] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable reinforcement learning implementations. J Mach Learn Res 2021;22(268):1–8.

[41] OpenAI. Twin delayed DDPG — Spinning up documentation. 2020, Openai.Com.