Future Generation Computer Systems 167 (2025) 107736

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A Web of Things approach for learning on the Edge–Cloud Continuum

Luca Bedogni [a,*], Federico Chiariotti [b]

[a] University of Modena and Reggio Emilia, Modena, 41125, Italy
[b] University of Padova, Padua, 35131, Italy

ARTICLE INFO

Keywords:
Web of Things
Digital twins
Deep Reinforcement Learning
Model exchange

ABSTRACT

Internet of Things (IoT) devices provide constant, contextual data that can be leveraged to automatically reconfigure and optimize smart environments. Artificial Intelligence (AI) and deep learning techniques are tools of increasing importance for this, as Deep Reinforcement Learning (DRL) can provide a general solution to this problem. However, the heterogeneity of scenarios in which DRL models may be deployed is vast, making the design of universal plug-and-play models extremely difficult. Moreover, the real deployment of DRL models on the Edge, and in the IoT in particular, is limited by two factors: firstly, the computational complexity of the training procedure, and secondly, the need for a relatively long exploration phase, during which the agent proceeds by trial and error. A natural solution to both these issues is to use simulated environments by creating a Digital Twin (DT) of the environment, which can replicate physical entities in the digital domain, providing a standardized interface to the application layer. DTs allow for simulation and testing of models and services in a simulated environment, which may be hosted on more powerful Cloud servers without the need to exchange all the data generated by the real devices. In this paper, we present a novel architecture based on the emerging Web of Things (WoT) standard, which provides a DT of a smart environment and applies DRL techniques on real time data. We discuss the theoretical properties of DRL training using DTs, showcasing our system in an existing real deployment, comparing its performance with a legacy system. Our findings show that the implementation of a DT, specifically for DRL models, allows for faster convergence and finer tuning, as well as reducing the computational and communication demands on the Edge network. The use of multiple DTs with different complexities and data requirements can also help accelerate the training, progressing by steps.

1. Introduction

The Internet of Things (IoT) has transformed several aspects of our daily lives, as modern, intelligent devices are now available in a plethora of different use cases. This large amount of information, together with the capability of IoT devices to seamlessly interact with the real world, opens up several possibilities for the automatic reconfiguration and control of systems based on Artificial Intelligence (AI) and Deep Learning (DL), without the need of a human intervention in the loop [1]. The Deep Reinforcement Learning (DRL) revolution, started in 2015 with the first Deep Q-Network (DQN) model [2] to reach human-level performance in several complex tasks, has led to a flourishing of new solutions in various fields, including the IoT. In the years since, DRL models have proven their value in a number of applications, from complex board games to cellular network optimization. However, some inherent challenges still limit their use in practical scenarios [3]: while these models often outperform traditional approaches after convergence, the training phase is often computationally expensive, taxing Edge devices with limited capabilities and energy constraints [4].

On the other hand, the benefits of local learning exploited by the Edge Learning paradigm [5] should not be underestimated: transmitting all data to the Cloud to allow for remote training of DRL agents might require significant communication resources. The solution to this conundrum is given by *model-based* DRL training [6]: in this paradigm, domain knowledge about the task of the learning agent is used to create a model of the environment, which can be used as a simulation environment for offline virtual training or directly integrated into the agent [7]. The DRL agent can then be safely trained on the Cloud, while the Edge node needs to generate a model of the environment that can be used to evaluate counterfactual scenarios and different control policies.

This can be accomplished by exploiting another paradigm that is rapidly attracting interest in the engineering community, the *Digital Twin (DT)* [8]. DTs include sensory, simulation, and management aspects of objects and processes, effectively bridging the physical and digital worlds [9]: a twin can be used to monitor the state of its physical counterpart, by connecting it directly to sensors, or to simulate the effects of choices and actions, using it as a dynamic model disconnected

* Corresponding author.
E-mail address: luca.bedogni@unimore.it (L. Bedogni).

https://doi.org/10.1016/j.future.2025.107736
Received 19 September 2024; Received in revised form 18 December 2024; Accepted 23 January 2025
Available online 8 February 2025

from physical reality. In fact, a DT can provide the virtual environment for model-based DRL, acting as a sandbox to train the learning agent in without affecting the real environment. Finally, DTs represent a much more scalable solution than transmitting all sensory data to the Cloud, as even neural network models have a limited number of parameters, and can be compressed further [10].

At the same time, the DT can be refined with updated data from the real sensors, more accurately matching the physical system it represents; this operation can be designed to be computationally sustainable, adapting the type of model to the capabilities of the Edge device. This can allow system operators to scale up to highly complex DRL agents, as they can easily be retrained on the DT, incorporating the new model with the experience on the actual physical system while exploiting the full computational power of the Cloud.

Nowadays, DTs are used in several different scenarios, since they offer unique advantages both from an architectural and a performance optimization perspective. In this work, we leverage both advantages: the former is useful to have external applications to always connect to the same software entity, without the need to know the physical details of the sensors and the actuators. This improves the interoperability of components, as applications can interact through a standardized interface through which DT may also change the specific devices without the need to notify external applications, which will continue to communicate with the DT as they were doing before. The latter advantage is leveraged, as we noted, to speed up the training process of the system.

The speed of the learning process is another issue that DTs can help tackle in another sense: training even a modest sized DQN often needs millions of steps, and more advanced architectures such as actor–critic and policy gradient approaches may take even longer. Many domotic and industrial IoT applications [11] have relatively slow control loops, and training a DRL agent in real time may require weeks or even months of highly suboptimal control due to the need for exploring the state and action spaces. On the other hand, purely passive training (i.e., training the model on experience samples obtained using a policy not controlled by the learning agent) has known convergence issues [12]: if a traditional control policy is used during training, avoiding highly damaging outcomes, the DQN might never learn a policy better than the one it is trained on.

Using a DT can allow the agent to make mistakes freely and learn from experience in a safe simulated environment, without actually causing any damage in the real world, and the model can be sped up to make the duration of the training process independent from the timing of the task, as the only limit to the virtual environment is the available processing speed (which, as we noted, is not an issue in the Cloud). Moreover, several DRL architectures and models may be a viable solution for the same task, and finding the one which fits the best according to the objective becomes easier when multiple agents can be trained in parallel. Naturally, the model itself is not reality, and the agent might need an online training phase in the real world after reaching convergence in the simulated environment, but the difficulty of transfer learning from the virtual environment to the real one is much lower than learning the policy from scratch, and more importantly, extremely risky and damaging actions are already associated to very low values, and a directed exploration policy will never select them, achieving a much smaller optimality gap during the online training phase.

This work deals primarily with the final piece of the puzzle, designing an architecture that can flexibly and seamlessly interconnect the Edge and Cloud, along with the sensors and actuators, exploiting the Web of Things (WoT) standard. The combination of WoT, DTs, and DRL enables the possibility for the development of intelligent systems which can interact with the physical world in a more efficient and interoperable manner. The contributions in this work are the following:

- We define a general-purpose WoT architecture that can seamlessly support the creation and twinning of DTs and interface directly with sensors and actuators through the WoT standard protocols. Our architecture can be easily integrated into other WoT systems thanks to the standardization of the interfaces, making the components interoperable and exchangeable, and can be run over an Edge node;
- We define a middleware layer that can integrate legacy systems and proprietary IoT frameworks, such as most domotic systems, into the architecture. This enables our architecture to also leverage and act on proprietary systems which may be already present in the smart environment;
- We define automated procedures for the model-based training of DRL agents in a virtual environment composed by one or more DTs on the Cloud, which have the objective of optimizing the system given the available sensors and actuators in the environment;
- We provide a smart home use case, in which DRL controls the heating system of a room, considering the number of people occupying it as well as associated energy efficiency, to showcase the potential of the proposed architecture. The DT and DRL agent are connected to a real deployment, using data from the installed sensors and providing notifications to the user with advice on thermostat settings. As more data becomes available over time, the DT for each room is automatically updated, exploiting more complex models when their training becomes possible, and the deployed DRL agent is retrained using the new models, providing a self-improving temperature control with a seamless occupant experience.

The rest of this paper is organized as follows: at first, we provide a review of the relevant literature in the domain of this work in Section 2. We then describe our architecture, the split between Edge and Cloud tasks, and the model-based DRL training procedure in Section 3, and we present our real smart home use case in Section 5. Finally, Section 6 concludes the paper and presents some possible avenues of future work.

2. Related work

The IoT has quickly become pervasive in both domestic and industrial scenarios, but it has been plagued by interoperability issues, with incompatible devices made by different vendors and adopting different communication protocols [1]. The WoT standard provides a homogeneous representation of devices and of operations which can be performed on them [13,14], while also offering Quality of Service (QoS) levels depending on the application requirements, and security mechanisms to protect the data and the devices. Nevertheless, there are still many open questions on practical interoperability issues in the WoT specifically for communication and data semantics.

However, even being able to seamlessly interconnect devices does not fully unleash the possibilities offered by the IoT. What is still missing is the possibility for devices to automatically understand what services can spark from the devices and the data which is available in their networks. In other words, what is missing is the interpretation of the data, which goes beyond pure classification [15], to go towards automatic service composition built by the device from the data itself [16]. This would open up exciting avenues for new research, with truly self-organizing systems that discover, leverage, and create new services automatically. There have been many contributions in this domain, most of which focused on a specific aspect of the issue, such as how to discover devices using the same protocol and exchanging similar information [17], and how to leverage ontologies to provide semantics to the data, hence to be able to provide a shared meaning to updates from different devices in the environment [18,19].

There are fewer works in the literature that address the autonomous creation of services from the devices, and more generally how different

devices can form novel services without human intervention [20]. Aside from understanding the nature of the data, devices also need to be able to perform different actions on the network to change its behavior and recognize what actions may bring the system towards a better state [21]. In other words, the network has to automatically understand the effects of actions on specific variables of interest, which requires to perform several experiments and possibly have a long history of data, which is not always feasible in an IoT scenario. There have been some advances in this domain [22], but there are still no organic solutions that can also deal with all the challenges that deployment in real scenarios brings [23]. Hence on the one hand there is the need for a high amount of quality data to find patterns and train models, but at the same time it is difficult to do so due to the lack of interoperability between different environments, which makes data collection and interpretation challenging.

During the past decade, another revolution in computing and AI took place, in parallel to the development of IoT systems: DRL, whose practical applications had been mostly limited to board games and parametric optimization of hand-designed algorithms, was successfully integrated with deep neural networks [2], taking a significant leap in terms of performance and capabilities. Over the past few years, DRL has become a staple solution in many fields, including robotics, automated factory management, and communication networks. However, DRL solutions for real systems have two significant problems, which stem from the model-free nature of the learning procedure: firstly, they require a long training process, which must include samples of highly suboptimal actions in order to fully explore the solution space, and secondly, counterfactuals and alternative options are hard to evaluate [24], problems similar to those we have already discussed.

Domotics, and the control of slow processes such as building heating and climate control, are an application field that is particularly vulnerable to long training times [11]: while training a DRL agent in faster environments, with sub-second time steps, can take a few hours, training time for agents on processes whose actions are separated by minutes or hours can be measured in months. Furthermore, the robustness of the learned solution needs to be verified thoroughly, as the consequences of failures may range from annoying to dire. In these cases, virtual pre-training before deployment is a common technique [25]. Complex simulators or simple black-box models can provide a safe environment for the DRL agent to make mistakes [26], transferring the acquired knowledge to the real world. However, most works using virtual environments for training designed them *ad hoc* [27], requiring extensive human intervention, and the quality of the simulation environment is crucial for effective transfer learning [28]. With respect to the challenges we have described, *ad hoc* solutions, although customized on the reference scenario, may have difficulties to provide and obtain data to and from other devices outside of their environment, hence requiring manual intervention by users. Thus, it is crucial to design a standardized open solution which can accommodate a variety of different scenarios, maintaining a common structure while using different models.

Causal reasoning and the creation of models of the environment, either in advance or during training and operation, are two possible ways to mitigate these issues: modeling the effects of actions, and whether the available actions affect control performance [29], can significantly improve DRL training. The basic concept of a DT is to create a probabilistic model of a complex system [30], which can be run as a standalone simulation tool or synchronized with its real counterpart and used for decision-making [31]. A common feature of DTs is the automatic evolution of the model, which is continuously trained and improved using the feedback from the real system, which in turn is optimized using its virtual twin. In this way, DTs can be used as an additional building block in an already existing system, to leverage the advantages they bring without the need to substitute the running system.

In its simplest form, the DT paradigm can aid the virtual environment design process [32], providing the basic building blocks for automated data-driven training [33]. However, more advanced applications and a deeper integration of the paradigm emphasize the importance of modeling: if the DT is integrated in the decision-making and planning for a system, using the wrong model can significantly downgrade performance, as the mismatch between the digital and physical worlds results in suboptimal strategies [34]. On the other hand, using a DT can help connect different levels of abstraction in controlling a system [35], resulting in a better integration between system-level and granular control. DTs are a key component of the Industry 4.0 paradigm [36], and have been integrated in the IoT framework [37], also considering the use of Cloud processing [38]. The use of DTs is also common in agent-based design and simulation [39] for applications such as Smart Cities [40], e-health [41], and agriculture [42].

Another significant issue that the deployment of AI in the IoT faces is the communication cost of transmitting large volumes of data to the Cloud: the Edge Learning paradigm allows devices at the network edge to learn patterns from data without the need for Cloud infrastructure, with a large number of small devices with limited capabilities cooperating to learn in a distributed way [43]. It is also obvious that there is a trade-off in doing so, and a recent work [44] analyzes what part of the learning process can be performed on the Edge and what should instead be offloaded to a centralized server. This has benefits in terms of bandwidth and utilized resources, since less data needs to be transferred to a central entity for further processing. This has also been studied in Reinforcement Learning (RL) [45], analyzing the incentives needed for Edge nodes to participate in the learning process.

This work combines ideas from the DT and WoT literature in a single architecture, which can easily integrate multiple DTs from different types of sensors and transmit those models to the Cloud. This allows for the training of complex DRL agents in a virtual environment generated by a selection of DTs, exploiting the full resources of the Cloud, while leaving the pattern extraction work to the Edge nodes to reduce communication costs. The modularity of the architecture and the possibility for further expansion to the Federated Learning (FL) paradigm make it a flexible and general solution for deploying DRL on the IoT, which is applicable in several different use cases, keeping the same structure and interactions but training specific models depending on the specific deployment scenario.

3. Architecture

In this section, we describe the building blocks of our framework. We leverage the WoT model, which enables a standardized interface and communication process for smart devices belonging to a network. We focus our efforts on a Smart Home scenario, but our architecture can be easily extended and adapted to other environments as well. This can be done thanks to the WoT standard, which enables the interaction between different software components easier, thanks to a standardized description of any device's capabilities.

Section 3.1 presents our framework, which is based on a WoT network through which devices are able to communicate. The training procedure for model-based DRL agents is then described in Section 4, including the twinning of the digital twin model to the selected sensory set and the training of the agent in the virtual environment provided by the twin. We then describe the inclusion of legacy components in Section 3.2, to extend our vision and encompass other devices which may be available through other networks or systems.

3.1. WoT network

Our architecture extends the simpler version presented in [16] and mainly builds on the WoT paradigm, which enables devices to provide their representation in terms of a Thing Description (TD), which advertises their own capabilities to other devices willing to interact with

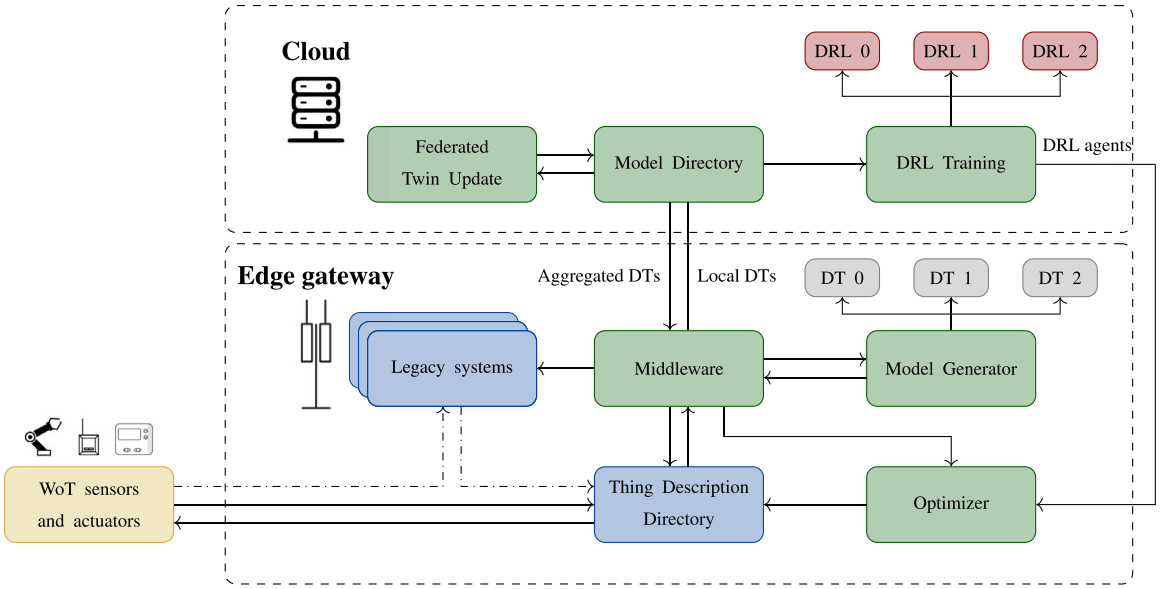


Fig. 1. Our proposed Web of Things architecture.

them. Fig. 1 highlights the different components of our architecture and their interactions.

On the lower left corner of the figure, we see sensors and actuators, which are built following the WoT standard: they provide a minimalist HTTP server which serves the requests, and also provide their TD whenever their root URL is visited. The system is then implemented on the Edge–Cloud continuum, with two main nodes at either end. We also remark that the modular nature of our architecture allows the system to be run as a monolith if enough computational power is available at the Edge, as well as making it easy to move components from the Edge to the Cloud, at the cost of a significantly higher communication burden.

3.1.1. Edge gateway

Upon joining the network, devices are requested to join a Thing Description Directory (TDD), which stores all the TDs from devices in the network. This TDD needs to be implemented on the Edge gateway to avoid an excessive load between the Edge and the Cloud; sensor updates are stored locally in order to avoid overloading the backhaul link and preserve privacy.

Using a TDD has numerous advantages, such as the possibility to perform queries on it to retrieve specific groups of devices, or to gather all devices providing the same kind of data and so on. Moreover, it allows to keep track of all the devices which are available in the system. The TDD also provides an interface to notify other software components of devices joining the network. We consider three additional components that need to be implemented on the Edge gateway: the Middleware, the Model Generator, and the Optimizer.

The Middleware is in charge of recognizing devices joining and leaving the network and managing the interaction between the TDD and other system components by performing three main operations:

- It monitors the TDD for devices which have joined the network analyzes their TD and informs the Model Generator and Optimizer;
- It connects to legacy systems which may be already present in the smart environment, and provides a WoT interface to them. These newly created WoT devices will register to the TDD, so that other devices in the system can leverage their data and can interact with them;
- It transmits the DTs of the subsystems registered in the TDD to the Cloud, allowing for the remote replication of the observed environment and for FL.

Overall, the Middleware is in charge of maintaining the modules and DT components updated according to the devices active in the system, as well as transmitting the updated DTs to the Cloud to maintain the twinning. Therefore, it also periodically checks whether registered devices are still alive and removes the models or DTs whenever such devices left the network. Finally, it needs to manage the connection to the Cloud, handling the transmission and reception of DTs.

The second component is the Model Generator, which creates and updates the DTs of individual sensors or more complex subsystems, incorporating data from the relevant sensors into the models. The DTs can be implemented with classical, physics-driven parametric models or deep learning-based data-driven models. In both cases, the local models can be refined using the FL approach.

Finally, the Optimizer controls the environment by considering at all the Things registered in the TDD and their capabilities, as well as monitoring the agent rewards.

3.1.2. Cloud

The Cloud part of the architecture involves three components: the Model Directory, the Federated Twin Update, and the DRL Training. The Model Directory is an equivalent of the TDD, as it stores all the local DTs for the environments it controls. A single Cloud instance might control several Edge gateways, each with its own local models.

Applying the FL paradigm, the Federated Twin Update can take DTs from several similar sensors and aggregate the models, generating a more precise DT to distribute back to the Edge gateways. In this way, the models can be updated with data from multiple environments, ensuring a better training and more efficient use of the data.

The third component, i.e., the DRL Training, will be described in the following. Its function is to perform the training of complex DRL model by fully exploiting the computational power of the Cloud, using one or multiple DTs as a virtual environment to train safely and quickly, without needing to wait for the real-time consequences of actions.

3.2. Legacy components

In order to integrate our architecture into an existing system, the Middleware also includes the possibility to interact with legacy systems already available in the smart environment. To accomplish this, it is also possible to design specific components directly in the Middleware to interconnect to other systems exposing data or services.

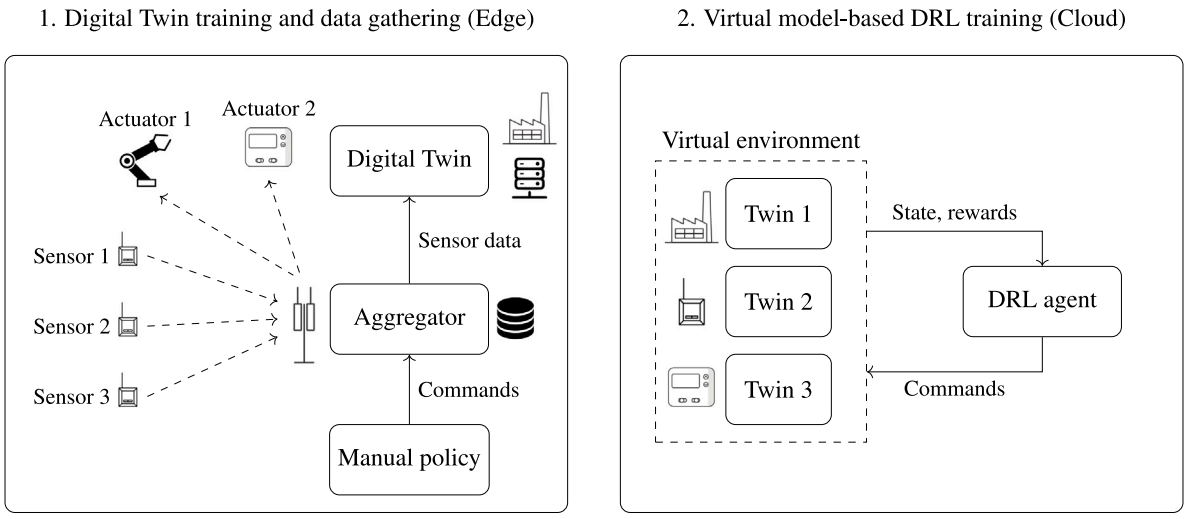


Fig. 2. Diagram of the two stages of a model-based DRL training in the WoT platform.

A note on this is that those specific components, developed to connect to legacy systems, have to be also deployed as a Thing themselves. This allows for them to be registered in the TDD upon connecting to the legacy system, so that other devices can leverage the data exposed by them, or connect to the offered services. Clearly, the precise definition of one or more legacy components has to be made according to the scenario and to already existing deployments, as it enables our system to integrate and cooperate with the others. The definition of each software component as a separate Thing allows for better scalability, since it is rather easy to add a component into the system without the need to reconfigure it from scratch, and multiple legacy systems can co-exists between each other.

3.3. Network considerations

From the point of view of the IoT devices, i.e., the sensors and actuators, our architecture is entirely transparent: devices, whether they natively implement the WoT standard or they are managed by the Middleware as legacy components, transmit observations and receive commands from the Edge gateway in exactly the same way as in a legacy system. The proposed architecture introduces no additional signaling over the IoT wireless access segment, which is the most critical in terms of energy consumption and bandwidth limitations.

On the other hand, our system significantly reduces the need for constant communications: offloading DRL training to the Cloud without a DT would mean constantly transmitting sensor readings from the Edge to the Cloud, maintaining a coherent state that the DRL agent can then make decisions on. For the same reason, commands would need to be transmitted with strict latency requirements. The proposed architecture reduces this significantly: instead of having to constantly transmit sensor data, the Edge gateway periodically transmits an updated DT model, which is significantly more compact (in particular, it might require only a few parameters if it is based on a classical dynamic model instead of deep learning). In the same way, the computationally expensive training of the DRL agent can be carried out in the Cloud, only transmitting the trained models to the Edge. The additional computational cost represented by the training of the DT and the exploitation of the trained DRL agents is well within the current capabilities of Edge gateways.

The flexibility of the architecture, which can easily support different models and neural network compression techniques, since the training of the DTs and DRL agents is carried out asynchronously, can even fit smaller, less powerful gateways with significant computational constraints: in this case, the models will need to be simplified, but the overall mechanism is preserved.

4. Model-based training

The proposed architecture includes a two-step procedure for training model-based DRL agents, as shown in Fig. 2: the basic concept is to exploit the DT paradigm to provide a virtual environment that is safe for an agent to explore, without any impacts on the actual physical system.

4.1. Digital twin training and data gathering

Firstly, the Edge gateway needs to gather sensory observation and create a DT of the observed environment. In our architecture, the TDD may include different sensors and actuators, belonging to different logical and physical environments and tasks: for example, two temperature sensors might be in different parts of a building or plant, and two sensors in the same room might have different logical functions, which make them functionally independent (e.g., light and temperature sensors). The system administrator then needs to select the set of sensors to gather in a single DT by querying the TDD. This may also be done automatically, but it involves a deeper understanding of the environment and how its subcomponents interact, requiring further study. Once the model is instantiated, the Middleware layer automatically subscribes it to the sensors fitting the specified TD, starting the twinning process. The platform is entirely agnostic to the training method used for the model, which could use either statistical or learning-based methods: since the architecture operates over standardized queries and responses, the Model Generator block can be implemented as a black box, even using different programming languages. During this stage, the actuators in the environment must be operated using a legacy policy, which may be defined by an algorithm or directly by the user, as shown in Fig. 2. The objective of this phase is to create a simulated environment which is as close as possible to the real, physical system it is twinned to.

4.2. Virtual model-based DRL training

After the DTs have reached the required accuracy, a DRL agent can be trained in a virtual environment without affecting the real system. The twins can be used in a predictive fashion, applying the results of the agent's choices and simulating the real system. The virtual environment may include more than one twin, representing different subsystems that are affected by the agent's actions and concur in determining the system reward. Naturally, the choice of which DTs must be included, as well as which actuators can be controlled by the DRL agent, is made

by the system administrator. The state observed by the DRL agent is then the union of the states of the twins, which mimic the real system dynamics. The agent learns to solve a separate problem, i.e., to control the virtual environment, and then applies transfer learning to adapt the policy to the real world. This virtual training procedure can be performed entirely in the Cloud, running the model faster than real time and quickly reaching convergence thanks to the higher available computing power.

Let us consider a theoretical representation of the target environment: DRL controls a Markov Decision Process (MDP) operating over discrete time steps. At each time step t, the agent observes a state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$, which influences the next state s_{t+1}. We denote the state transition probability as $P(s_{t+1}|s_t, a_t)$, and the policy mapping states to actions as $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The objective of the agent is encoded through a reward signal r_t, which it tries to maximize over the long term. The long-term reward is defined as

$$G_\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \lambda^t r_t | \pi\right], \tag{1}$$

where $\lambda \in [0, 1)$ is a discount factor that ensures convergence. A higher value of λ leads to a more foresighted agent, which considers future rewards more, while agents with a lower λ will tend to prefer short-term rewards over longer-term consequences. At the beginning of the training, the agent does not know the transition matrix $\mathbf{P}$ or the reward function r, which it indirectly learns by trying to adapt the policy to maximize G_π.

In order to train the agent virtually, we need to know both $\mathbf{P}$ and r: in other words, if the DT perfectly matches the real environment, the DRL agent will learn the same policy that it would if it were trained online [46, Theorem 7]. We can extend this to the concept of ε-bisimulation [47, Definition 9]: if two MDPs have a similar behavior, up to a maximum difference ε, the optimal solution of one will provide near-optimal performance in the other. A similar result is provided by the stricter notion of ε-homogeneity [48]: if the difference between the effect of any control action in two MDPs is smaller than ε, optimal solutions for one are approximately optimal for the other.

We can then consider the simulated environment generated by a DT as another MDP, characterized by the tuple $\langle \mathcal{S}', \mathcal{A}, \mathbf{P}', r' \rangle$: while the action space is the same, all other components may be different. Firstly, the internal representation of the state of the DT s' does not necessarily correspond to the physical state s: the former may have internal model variables that do not correspond to the real system, while the latter may have hidden components that affect the observability of the process, so that $\mathcal{S} \neq \mathcal{S}'$ in the general case. Additionally, there will be significant differences over the transition probabilities: accurately capturing the dynamics of the environment in a DT often requires a large amount of data, and training a DRL agent when $\mathbf{P}'$ is significantly different from $\mathbf{P}$ after translating the equivalent states will lead to highly suboptimal policies.

In any case, a short online training phase in the real environment is still possible, with a small cost in terms of system efficiency. This latter phase will need to be performed at the Edge, with an additional cost in terms of computing resources, but is much shorter than training an agent from scratch. During this phase, a directed exploration algorithm will still avoid any catastrophic failure modes, as the agent will have learned that they lead to significant penalties in the virtual environment. On the other hand, from the WoT perspective, the Optimizer just needs to connect the agent to the proper components: during the virtual training, observations of the environment are provided by the DTs, while during real operation, observations from the sensors are pushed directly to the agent. In the same way, its commands are only directed to the DTs during the virtual training, but are routed to the real actuators in the operational phase.

4.3. The bias-complexity trade-off in the virtual environment

As we discussed above, any DT represents an approximation of the virtual environment, as does modeling any real use case as an MDP. We can consider the most accurate formulation of the problem as $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, r \rangle$: in this case, the state space $\mathcal{S}$ will include all relevant variables, i.e., the ones that can affect the performance of the system. On the other hand, a formulation $\langle \mathcal{S}', \mathcal{A}, \mathbf{P}', r' \rangle$ using a DT involves two approximations: firstly, the state space is usually reduced, as the DT is a simpler model of the environment, which may elide some variables and group some states. This is an inherent limit of the DT model, which may not be overcome even with unlimited training data: the difference between the two models may cause an unavoidable approximation error, which grows as the DT becomes simpler and less adherent to reality.

On the other hand, the second source of error depends on the quantity of available training data: bootstrapping a complex DT with a few data points may cause a large estimation error. If we consider the extreme case in which $\mathcal{S}' = \mathcal{S}$, i.e., the DT has a one-to-one correspondence with the real system, the estimation of $\mathbf{P}$ and r may require large datasets. This is a well-known trade-off in machine learning [49], and may not even have a single minimum. Additionally, a biased DT may still return a robust policy, resulting in good performance in the real system.

While this trade-off is complex and highly dependent on the specific learning problem, our architecture provides a flexible way to adapt the DRL agent as new data comes in: since its training is performed in the Cloud over a virtual environment, it can be repeated over different DT models, selecting the one that provides the closest approximation of the real environment in each training iteration. We will see an example of this in the following section: if we consider a few days of data, simpler models that only consider a few internal variables outperform more complex ones, and training a DRL agent on them provides a better final performance. On the other hand, more complex models become more accurate as more data comes in, leading us to retrain the DRL agent and get a better final performance. This procedure is empirical, but still avoids the major disadvantages of training DRL agents directly on the real environment, namely, the long training time and the initial drop in performance due to the necessity of exploration.

5. Use case: Heating system DRL control

In this section, we describe a prototype implementation of our system, and we detail how we implemented all the software components described in Section 3. We consider a smart home scenario, in which sensors can provide environmental data in order to schedule activities and optimize the planning. We focus our efforts on deploying our system aside a legacy system which manages different sensors and actuators around a house, which are not compliant to the WoT standard. This scenario can be found in many modern homes, which leverage data from off-the-shelf devices which also provide proprietary software solutions to manage them. We base our analysis on the climate operations of the house, which means that our aim is to optimize the temperature of different rooms accounting for the room occupancy and for the inside and outside temperature. Specifically, we deploy our system in a real house deployment with 2 floors, three rooms and four inhabitants which provide heterogeneity in objectives and room occupancy, which are:

- A living room open space, which also includes the kitchen, on the ground floor of the house;
- A bathroom on the first floor;
- A master bedroom shared by two people on the first floor.

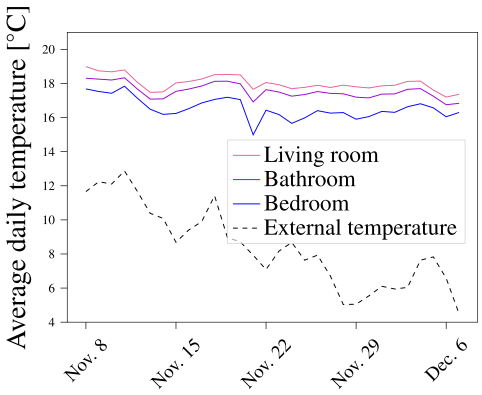


Fig. 3. Temperature dataset used for our experiments.

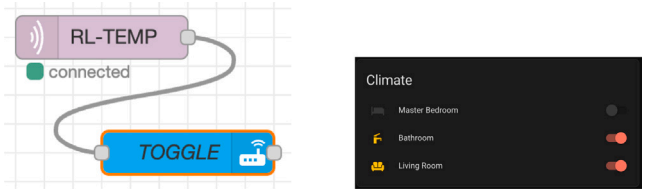
We note that there are also other rooms in the house: another bedroom (shared by the other inhabitants), another bathroom and a studio. However, these rooms have not been used as part of the evaluation of this study since they do not provide temperature readings.

The living room is generally occupied during the morning, at lunchtime and during the evenings, but it may also get sporadic occupation throughout the entire day in case people work from home or at lunch and dinner times. A bathroom typically has a flat occupancy during the day, meaning that people can use it at different times without a specific routine. On the other hand, the bedroom instead has zero occupancy during the day, and full occupancy during the night, although bedtime and wake ups can have relatively small variations on different days of the week. We will detail the parameters we took into account below.

In Fig. 3, we also show the temperature dataset we have collected over 40 days. This data has been obtained through the legacy system sensors which were already deployed in the house in November–December 2022. The black line represents the outdoor temperature and shows a significant drop, from roughly 12 °C to around 5 °C. The other three lines show the temperature inside the three rooms considered as part of this study. While the absolute values differ, their trends are similar and also reflect the current configuration of the already existing heating system. The heating in the three rooms is controlled by a classical thermostat system, in which the inhabitants set the desired temperature in different part of the day and the heating system is switched on to match that temperature. The schedule varies greatly for the three rooms: the master bedroom is only heated at night, the living room only during the day, while the bathroom is heated in the morning and in the evening. Nevertheless, they show a similar trend as heat spreads through the whole house. We can also notice that the bedroom has the lowest temperature among the rooms considered, as it is the only room which faces North West, and which therefore is not directly exposed to sunlight during the winter months.

5.1. Implementation

The house was already running a legacy system provided by Home Assistant,[1] which is a personal smart home hub providing integration with several off-the-shelf devices. Home Assistant allows users to interconnect components offered by different vendors, and provides a standardized domain and naming schemes to obtain data and perform actions. Through it is possible to interconnect thermostats, smart plugs and smart lights, external data and many other things thanks to a plugin architecture. We describe at first the implementation of the Legacy



(a) Node-RED integration into Home Assistant

(b) Dashboard visualization and state.

Fig. 4. Integration of the legacy component output inside the Home Assistant instance.

component connecting the middleware to Home Assistant, and we then highlight how it is integrated into our architecture.

Home Assistant provides data with different APIs and different methodologies, and it is also possible to install external software and services into it, to extend it and tailor it depending on the needs of the user.

For our purpose, we performed the integration through the data exposed in InfluxDB, a popular time-series database which is integrated within Home Assistant and which stores data from the devices managed by Home Assistant. Since our aim is to be WoT compliant to make devices able to register in the TDD, we developed our software component as a Thing, which then register itself in the system and provides its capabilities in the same way as the other components of it.

Specifically the Thing we developed performs the following:

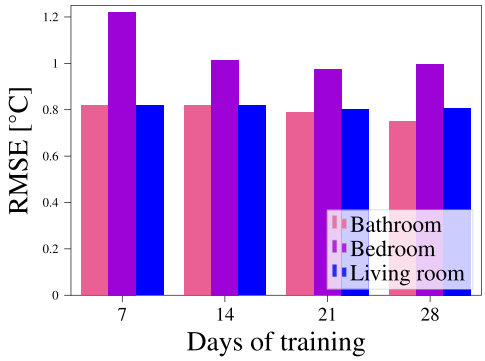
- It connects to InfluxDB and reads all the devices providing a variable of interest. In our case, since we are focusing on optimizing the temperature, the corresponding domain is `climate`, which is the group of devices in Home Assistant which can provide temperature, humidity and also perform operations such as switching on a cooling or heating system.
- For each discovered device, it provides a specific GET operation advertised in its thing description. For each of these methods in the TD, whenever they are requested it will translate the query to an appropriate InfluxDB query to retrieve the data and eventually provide it to the client requesting it.

This software component is then registered in the TDD, so that other devices have seamless access to its data without requiring specific knowledge on the technical details of the connection, but leveraging the standard HTTP connection which all Things must provide.

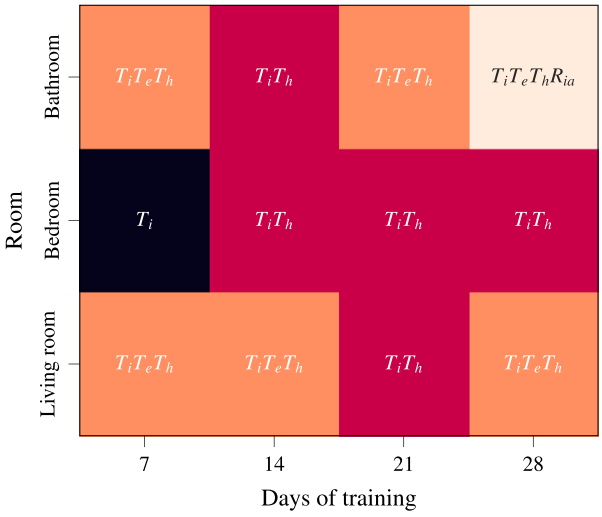
The developed software component also provides bidirectional communication, meaning that it can also feedback to Home Assistant. This is realized by exposing a PUT method in the TD of the software, which allows clients to send data to the Thing, which then routes it back to Home Assistant. To realize this part, without loss of generality we have adopted the MQTT[2] protocol, by publishing data on the Home Assistant MQTT broker. Clearly, other protocols can be adopted depending on the scenario and on the specific needs. We show an example of this behavior in Fig. 4: Fig. 4(a) shows the MQTT subscribe (RL-TEMP) node inside Node-RED running in Home Assistant. Fig. 4(b) shows instead the Home Assistant dashboard, with the three heating systems managed by our models and with their current statuses. This specific part of our system also closes the loop between the sensor data, used to train the DRL inside a DT, which then feedbacks on the real system advising the user what to switch on and off, thanks to an extended training outside of the real system. The DT and DRL training was performed using Python code, which was run over a laptop running GNU-Linux Fedora 40 over an Intel i7-1255U 12th generation CPU with 40 GB of RAM. Both the DT twinning and the DRL training only take a few

[1] https://www.home-assistant.io/.

[2] MQTT originally stood for Message Queuing Telemetry Transport, but the acronym has officially not stood for anything since 2013.



(a) MSE for the best model over 28 days.



(b) Best models for the TT.

Fig. 5. Performance of the TT models as a function of the amount of available data.

seconds for each model, even when considering several weeks of data, over the considered hardware and software. As such, these operations would not constitute a significant burden for a Cloud server, and might even be performed on the network edge if necessary.

5.2. Digital twin evaluation

In this section, we show the performance evaluation of the DT model in the real deployment described above, leveraging a Thermal Twin (TT) which twins the temperature dynamics accounting for outside and inside temperatures, and a Room Twin (RT) which instead accounts for the rooms occupancy.

The TT was adapted from gray box RC-equivalent models in the relevant literature [50]. We considered two measured parameters, the room's internal temperature T_i and the ambient temperature outside the building T_a, which are available from the smart home sensors, along with a set of hidden variables which can help in tracking the thermal evolution of a room. The T_iT_e model includes the temperature of the envelope (i.e., the building's external walls), and the T_iT_h model includes the temperature of the heating system. The $T_iT_eT_h$ model includes both, and the $T_iT_eT_hR_{ia}$ includes a variable thermal resistance between the air in the room and the exterior of the building. Naturally, more complex models are more accurate, but require more data to avoid overfitting.

Fig. 5(a) shows the Root Mean Square Error (RMSE) of the internal temperature predicted by the TT for the three rooms we have considered in our study. As expected, the RMSE generally decreases as the available data become richer, but the three rooms all have different trends: while 7 days of data are enough to train the DT of the living room, the master bedroom needs significantly more data and has a far higher error. This can be explained by the fact that the master bedroom is the room which is least occupied during the day: the model is then able to learn about the heating system dynamics only during the night, and the nighttime temperature is kept at a low value of 16 °C. This does not allow it to obtain a sufficiently wide variety of samples to learn from; therefore, the model does not learn as fast as in the other rooms, which show a similar behavior.

Nevertheless, all three rooms show large improvements when trained with a longer dataset: the average error for the master bedroom decreases from more than 1.8 °C with only 1 day of training to less than 1.2 °C when trained with a month of data, with roughly a 30% reduction in the total error. A similar comment can be also made for the other two rooms, where at the beginning of the training the error is around 1.2 °C and it is reduced to roughly 0.7 °C. We can also consider which model performs best in each room for different amounts of training data: the best model for each considered case is shown in Fig. 5(b), which confirms our analysis. More complex models are usually better with more data, and the evolution of the best model selection follows this trend, with some oscillations.

There are three main takeaways from this analysis: *(i)*, personalized models have to be created and trained for each room, even in the same house, as the heating and temperature dynamics vary greatly between rooms, depending on their size, isolation, exposure to sunlight, and heating units. This can be clearly seen from the chart, as each room has a different performance thus demand a specific model; *(ii)*, the larger the training set, the better the results, as it is possible to observe a wider variety of samples. At the same time, we also experience a slow convergence to a stable result, which may indicate that it exists a quantity of data ideal for each room; *(iii)*, a DT can effectively help to understand when a model is ready to be deployed, since it can be trained on a portion of data and tested with fresh data, and then used whenever the error falls under a scenario-specific threshold, and *(iv)*, the DT can include different types of models with different levels of complexity, which can be hot-swapped as new data become available and richer models can be trained effectively. This also includes the possibility to have different models for different period of the years, or for different events and scenario which may happen within the smart environment.

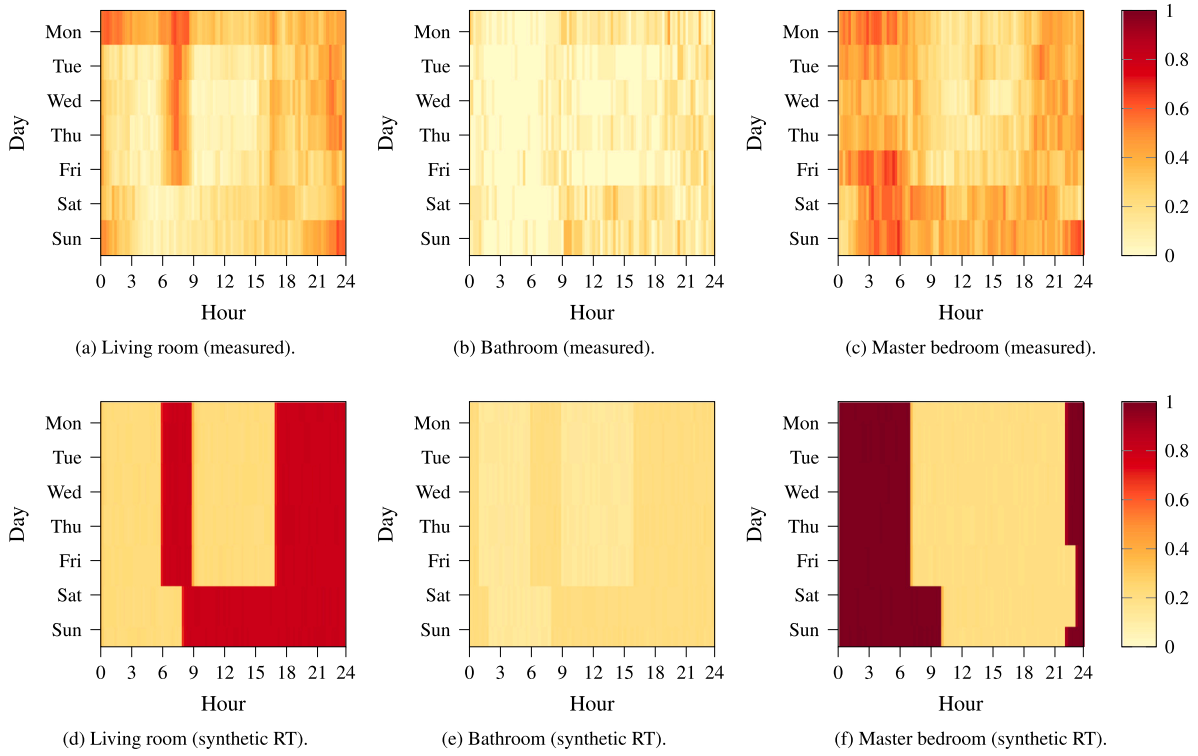
We also performed an additional study, based on 3 additional months of data on temperatures between December 2023 and February 2024 and comparing all models' RMSE in the last month, when training for a different number of days. The full results are shown in Table 1. We can see that the T_iT_e model is the quickest to converge to a good result in the living room data, but its quality does not improve. On the other hand, other models completely miss the actual trends when trained over a single week. The most complex model, $T_iT_eT_hR_{ia}$, reaches the best performance after 2 weeks, while other, simpler models have a much slower convergence. The same trend is visible in the master bedroom data, with a higher overall error: the T_iT_e model is the only one to get an acceptable result with a single week of data, but after a few weeks, there are enough data to allow the more complete $T_iT_eT_hR_{ia}$ model to outperform all others.

For a wider review of RC-equivalent model parameters and training, we refer the reader to [50]. We also remind the reader that the objective of the TT is to capture the dynamics of the temperature in each room, so as to provide a digital environment in which a DRL agent can be trained much faster than real time: in this case, the probability $\hat{P}(s'_{t+1}|s'_t, a_t)$ corresponds to the solution to the RC-equivalent model.

On the other hand, the smart home deployment did not provide room-level occupant data, so we assumed a simple Markovian model. For each room, we considered a maximum number of occupants $N_{\max}$, each of whom behaves independently. The occupant is in state 1 if they are inside the room, and the evolution of their state is driven by the

Table 1
TT model performance, measured using the RMSE, as a function of the training time, tested over a 6 week period. Missing values indicate that no model converged.

TT model	Test RMSE (living room)						Test RMSE (master bedroom)					
	1 wk.	2 wk.	3 wk.	4 wk.	5 wk.	6 wk.	1 wk.	2 wk.	3 wk.	4 wk.	5 wk.	6 wk.
T_i	7.6	4.05	2.26	1.24	1.06	0.88	33.68	**4.26**	4.39	4.63	3.71	3.38
T_iT_e	**0.76**	0.78	0.9	0.9	1.2	1.02	**4.59**	4.73	3.84	3.38	3.46	2.6
T_iT_h	7.62	4.05	2.23	1.26	1.06	0.88	34.82	7.12	4.39	4.63	3.71	3.38
$T_iT_eT_hR_{ia}$	3.17	**0.61**	**0.66**	**0.64**	**0.61**	**0.64**	27.25	4.58	**2.44**	**2.61**	**2.73**	**1.68**



(a) Living room (measured). (b) Bathroom (measured). (c) Master bedroom (measured).

(d) Living room (synthetic RT). (e) Bathroom (synthetic RT). (f) Master bedroom (synthetic RT).

Fig. 6. Heatmap of the probability for each room to be occupied over time in the dataset from [51] and synthetic RT model.

matrix $\mathbf{P}(d,h)$, which is a function of the day of the week and of the current time. Fig. 6 shows the average probability of each room being occupied over the whole week, considering a real dataset from [51] and our synthetic RT model: the bedroom is occupied during the night, with a slightly shifted timeframe in the weekend, while the bathroom and living room are usually occupied during the day and evening, with different transition probabilities. Naturally, the actual probability of a room being occupied at a given time will depend on the state, but we can see that the bedroom is more predictable, while, as expected, the bathroom has a much more random pattern throughout the day. The assumed probability for someone to be in the bathroom is low but constant throughout the day, while the occupants remain in the living room for longer times during the morning and evening, resulting in a higher probability of the room being occupied. This is consistent with the patterns in the real dataset in the top row, although the real occupancy data tends to be noisier, due to less predictable patterns and sensor errors. Additionally, the living room tends to be less occupied on weekends, while the master bedroom is more occupied throughout the day.

The three rooms also have different numbers of occupants: we assume that the whole family of 4 may be present in the living room, while only 2 people use the master bedroom, and only one person at a time can be in the bathroom (in this case, one person leaving the bathroom and another entering would count as one person remaining in the room). The use this an artificial model for the room occupancy reduces its accuracy in the real deployment, but the architecture can easily accommodate new sensors such as presences and movement sensors, and the DTs can automatically retrain the DRL agent after updates to the underlying models. In fact, since a time-dependent RT would need to be trained over multiple weeks to figure out the family's habits, starting from a relatively accurate prior would significantly reduce the DT training time. Additionally, as we remarked above, the general trends are extremely similar, and we can consider the RT as a solid approximation of real occupancy patterns.

5.3. Agent evaluation

In this section, we evaluate the performance of the agent, which leverages the RT and TT to forecast actions which improve the overall status of the system.

The temperature control MDP is relatively simple: the state of the system is represented by the states of the two twins, which include information about the thermal state of the room and heating system, as well as the date and current occupancy of the room. Depending on the specific thermal twin model, the state space S' might include any combination of T_e, T_i, T_h and R_{ia}, as well as the ambient temperature T_a: the DRL agent sees the state of the thermal DT as part of the overall state, along with the number of people currently in the room and the day and hour. However, the internal room temperature T_i is always part of the state, and is used in the reward. The possible actions available to

Table 2
MDP parameters.

Parameter	Symbol	Value
Episode duration (steps)	L	1000
Step duration (minutes)	t	15
Comfort temperature	T_0	18 °C
Threshold temperature	T_{lim}	3 °C
Energy cost	α	0.25
Swing penalty	β	0.2
Discount factor	λ	0.95

the agent are A levels of heating system power, which are normalized from 0 (the heating system is off) to 1, corresponding to the maximum available heating power.

The reward function for taking action a when there are o people in the room, resulting in an internal temperature T_i, is defined as follows:

$$r(T_i, o, a) = \mathbb{1}(T_i - T_0)\mathbb{1}(o-1) - \alpha a \\ -\beta \mathbb{1}(|T_i - T_0| - T_{\text{lim}})\,(1 - 2\mathbb{1}(a))\,\text{sign}(T_i - T_0), \tag{2}$$

where T_0 and T_{lim} are user-defined threshold temperatures, $\mathbb{1}(\cdot)$ is the stepwise function, equal to 1 if the argument is greater than 0 and 0 otherwise, and α and β are penalty parameters representing the cost of energy and a generic penalty for. In other words, the agent receives a reward 1 if the temperature is above the comfort level T_0, but only if there are people in the room; on the other hand, using energy to heat always has a cost. The second penalty term is used to avoid excessive temperature swings: if the room is too warm or too cold by more than T_{lim} degrees, actions that do not lead the system back towards equilibrium (i.e., not turning on the heat if it is too cold, or turning it on when the room is already too hot) are penalized. The values we selected for the parameters are listed in Table 2. A policy π is then a function mapping states to actions, and the objective of the DRL agent is then to find the policy π^* that maximizes the long-term discounted reward G_π. A possible solution to the maximization problem which we just presented can be found analytically by solving the following Bellman equation:

$$Q(s,a) = \mathbb{E}[r(s,a)] + \sum_{s' \in S} \lambda P(s'|s,a) Q(s', \pi(s')). \tag{3}$$

We then implement a DQN model, with an architecture that depends on the DT models, and is defined in Table 3, along with the main training parameters: the RT only needs $N_{\text{RT}} = 3$ parameters (the day of the week, time, and number of current occupants of the room), while the TT can have a variable N_{TT} depending on the RC-equivalent model. The T_i model only has a single parameter, while the more complex $T_i T_e T_h R_{ia}$ model has 4. The final input value is the ambient temperature T_a outside the room. The neural network uses a Softmax exploration policy, whose temperature decreases logarithmically from 1 to 10^{-6}, and the Adam optimizer, with ReLU activation. We did not perform a significant parameter search over the model architecture, as the main issue is not the sample efficiency of the training, but rather the adherence of the DT model to the real-world behavior of the Smart Home: improving the training would reduce the computational burden on the Cloud, but since the virtual environment can generate as many samples as needed, this does not affect the optimality of the solution after convergence. However, as DTs trained on a limited amount of data may lead to a virtual environment that does not effectively simulate the real environment, the policies learned in the former may be distorted and have a much worse performance when translated in the latter.

Fig. 7 shows the performance of the DRL agent, as measured by the reward, when it is trained using a TT with a limited amount of data. In all cases, the RT is assumed to be perfect. We can immediately note that the bedroom needs at least 3 weeks of data to control the room temperature properly, while the bathroom can achieve good performance with a single week. This is consistent with the RMSE performance of the models: while the bathroom dynamics are easy to learn, and the model predictions are similar to reality, the significant errors in the bedroom mean that the virtual training leads the DRL agent to the wrong policy, with catastrophic results. As the TT improves, so does the performance of the agents, gradually reaching the optimum. We also highlight that the optimal performance for a room depends on how much it is occupied: the dashed lines in the figure show the performance of an ideal system with perfect room occupancy prediction and free energy, i.e., the upper bound to real performance for each room. The upper bound is extremely hard to reach in the bigger rooms, as the real agent has to deal with winter temperatures outside, and consequently frequently uses energy to power the heating system. On the other hand, the bathroom does not directly face any walls, and maintaining a comfortable temperature is relatively easy, so it is possible to achieve an almost perfect performance. We also compare the DRL solution with a legacy thermostat policy, which tries to maintain the desired temperature during the hours when the room is usually occupied, following a predefined schedule (as most home thermostats actually do today). The legacy performance is significantly worse than for the DRL, as it does not have the concept of an RT: the great advantage of DRL in this case is that it can consider and predict the actual occupancy of the room, which may be different from day to day, and adapt its actions to save energy when the room is actually empty. We also want to remark once more how the DRL system greatly benefits from an extensive period of training: as it can be seen, with only a week of training the performance are not satisfactory, however when the amount of data increases the DRL improves its ability to forecast the occupancy and predict the correct times in which the heating system has to be switched on or off.

To provide a better understanding of how the model works, we also show an example sequence of actions performed by the DRL agent over the course of a day for the master bedroom in Fig. 8. The room is only

Table 3
DQN architecture and training parameters.

Layer	Symbol	Parameters
Input	L_0	$N_{\text{TT}} + N_{\text{RT}} + 1$
Input	L_1	$10L_0$
Input	L_2	$5L_1$
Output	O	A
Hyperparameter	**Symbol**	**Value**
Activation function	ϕ_a	ReLU
Learning rate	ℓ	10^{-5}
Dropout probability	p_d	0.1
Epochs	E	25
Episodes per epoch	e	20

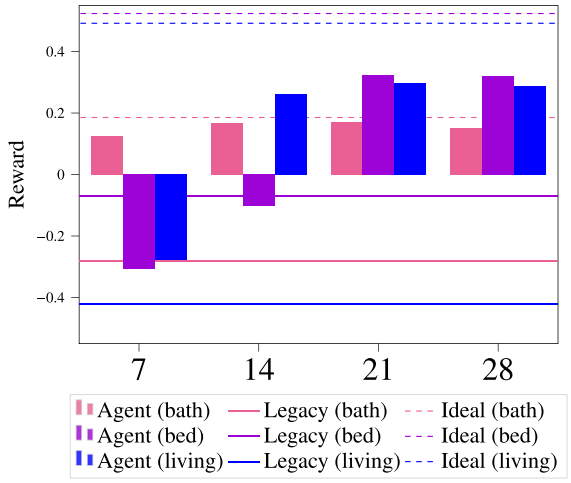


Fig. 7. DRL agent performance at convergence as a function of the number of days of available data.

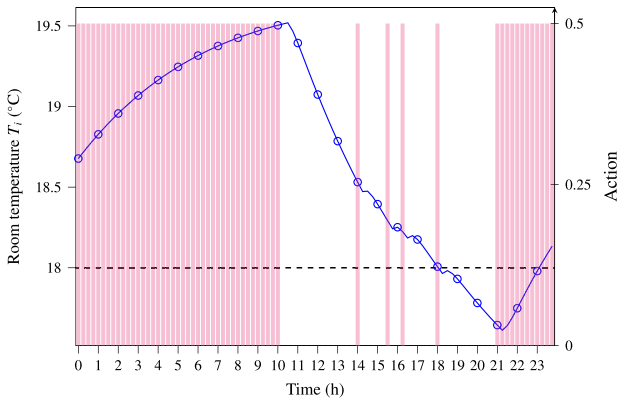


Fig. 8. Temperature evolution and control actions in the master bedroom over the course of a day.

occupied at night, although during the day has a non-zero probability of having occupants. During the night (i.e., leftmost part of the plot), the agent heats up the room beyond the minimum desired temperature of 18 °C, with the heating system at half power. In the morning, the probability of having at least one occupant significantly drops, hence the DRL agent stops to heat the room. Sporadic adjustments are made through the day, since the DRL agent wants to keep the room above the minimum temperature to avoid any reward losses. In the evening the DRL agent starts to heat the room again some time before the usual bedtime, raising the temperature above the minimum threshold in time for them to go to bed. Obviously, having a perfect representation of the occupancy would avoid the DRL agent to sporadically heat the room when the probability of having occupants is low. Still, Fig. 8 confirms that the system actually learns a non-obvious policy from the two digital twins, and improves the system overall.

Naturally, the reward depends on a specific energy cost, represented by the parameter α, and on the required comfort temperature: a more efficient heating system, or fragile inhabitants who might suffer serious health consequences from the cold, would shift the trade-off towards more aggressive heating, while factors such as increases in gas prices could increase α, leading the agent to act more conservatively. Using the DT virtual training, we can simply retrain the agent with the new reward function in the virtual environment when these shifts happen, achieving a flexibility that manually programmed thermostats cannot match.

5.4. Automatic agent definition

In the use case we defined, the DRL agent is still partially hand-designed: the selection of the DT models was performed manually, as well as the architecture and hyperparameter search for the training. However, the WoT architecture we define provides significant room for expansion and for automation, as agents, models, sensors, and actuators are all controlled by the same platform.

We can foresee a use case in which multiple twins (e.g., including multiple rooms, sensors, and activities) can be automatically mixed and matched to tasks (as represented by actuators). The full potential of the proposed architecture is to automate DQN training as well, trying different state definitions and combinations of actions, as well as automatically optimizing hyperparameters for each agent. In this case, the only hand-designed part of the system would be the reward function, which would be ideal in a domotic scenario: the inhabitants only need to give the smart home a simple list of priorities, which they can adjust at any time, and the architecture will figure out which agents it will need to deploy, which twins they will need to accomplish their tasks, and which tasks are functionally independent and can be separated to simplify the action space.

5.5. The cloud–edge continuum

The framework we described and the smart home proof of concept show that creating DTs of an environment locally and using them to train DRL agents in the Cloud is a viable strategy for the IoT. The proposed framework is fully modular, and both the Model Generator block and the DRL training block can be safely shifted from the Edge to the Cloud and vice versa. It is also possible to include multiple models, as well as learning simultaneously from several separate environments through the Federated Twin Update module.

Replacing a DT with another one that leverages different sensors or provides more accurate predictions is also relatively easy, since the interface will be the same as before: the other components are unaffected, and the DT can be swapped online as more data become available, as we did in the proof of concept when switching between models. This is made possible by the use of the WoT standard and by implementing all components as Things, which allows an easier interoperability between the different parts of the system.

With the same methodology, the DRL agents can also be improved or substituted completely as the DT evolves: it is also possible to deploy a DRL training module locally, with a simple DQN that does not overtax the Edge node's processing power, and perform the training without Cloud assistance, then replacing the agent with a more complex and optimized one when the Cloud's processing power becomes available. This enables the possibility to have more personalized models in different environments, maintaining the same architecture but specializing its components according to the objectives, the data availability, and the connectivity and processing costs.

6. Conclusions

In this paper, we have a presented a novel framework to create and exchange DTs and DRL agents between the Edge and Cloud based on the WoT standard. Our architecture allows to build customized DTs for complex scenarios, leveraging data from custom-made sensors or legacy ones, by integrating with already-existing platforms, as well as allowing for FL on the Cloud, collecting data from multiple related environments. Our results indicate the benefits of using DTs for model-based reinforcement control, as they allow to test different models and deploy the one which fits the scenario the best, as well as relieving the Edge node of the computational burden of training a DQN. This has been confirmed by deploying our platform in a real scenario on which a legacy system was already running, integrating our platform with the existing software through the WoT, highlighting the importance of having components which can be exchanged and seamlessly deployed.

Specifically, we have shown how the WoT standard can help in managing heterogeneous scenarios, by deploying software components with standardized interfaces which can interact together in the same environment. Moreover, we have also shown how the DTs are used as architectural components in our system, allowing for seamless sensor or actuator replacement without the need to notify or modify external applications, and as a simulation sandbox, which in our case was exploited to train the DRL remotely on the Cloud.

Future work on this topic will be focused on the extension of our framework to other sensor types in a smart home, and extended results on other house types and rooms, which will better generalize our findings. Furthermore, the automation of DT design, agent selection, and MDP design is an interesting future direction, which can improve the resilience of the system and reduce the need for system administrator intervention.

CRediT authorship contribution statement

Luca Bedogni: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Federico Chiariotti:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Chiariotti Federico reports financial support was provided by European Commission. Chiariotti Federico reports a relationship with European Commission that includes: funding grants. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Federico Chiariotti's work was funded in part by the European Union, as part of the Italian NRRP under the Young Researchers grant "REDIAL."

Data availability

Data will be made available on request.

References

[1] F. Montori, L. Bedogni, M. Di Felice, L. Bononi, Machine-to-machine wireless communication technologies for the Internet of Things: Taxonomy, comparison and open issues, Pervasive Mob. Comput. 50 (2018) 56–81, http://dx.doi.org/10.1016/j.pmcj.2018.08.002.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nat. 518 (7540) (2015) 529–533.

[3] G. Dulac-Arnold, N. Levine, D.J. Mankowitz, J. Li, C. Paduraru, S. Gowal, T. Hester, Challenges of real-world reinforcement learning: definitions, benchmarks and analysis, Mach. Learn. 110 (9) (2021) 2419–2468.

[4] P. Kang, J. Jo, Benchmarking modern Edge devices for AI applications, IEICE Trans. Inf. Syst. 104 (3) (2021) 394–403.

[5] J. Zhang, Z. Qu, C. Chen, H. Wang, Y. Zhan, B. Ye, S. Guo, Edge learning: The enabling technology for distributed big data analytics in the edge, ACM Comput. Surv. 54 (7) (2021) 1–36.

[6] N. Yang, S. Wang, M. Chen, C.G. Brinton, C. Yin, W. Saad, S. Cui, Model-based reinforcement learning for quantized federated learning performance optimization, in: Global Communications Conference, GLOBECOM, IEEE, 2022, pp. 5063–5068.

[7] R. Kidambi, A. Rajeswaran, P. Netrapalli, T. Joachims, Morel: Model-based offline reinforcement learning, in: 34th Conference on Neural Information Processing Systems, NeurIPS, Vol. 33, 2020, pp. 21810–21823.

[8] S. Boschert, R. Rosen, Digital twin—the simulation aspect, in: Mechatronic Futures, Springer Cham, New York City, USA, 2016, pp. 59–74.

[9] 5G Alliance for Connected Industries and Automation (5G-ACIA), Using digital twins to integrate 5G into production networks, 2021, URL https://5g-acia.org/wp-content/uploads/2021/05/5G-ACIA_Using_Digital_Twins_to_Integrate_5G_into_Production_Networks-.pdf. (Accessed 5 December 2022).

[10] Y. Chen, C. Li, L. Gong, X. Wen, Y. Zhang, W. Shi, A deep neural network compression algorithm based on knowledge transfer for Edge devices, Comput. Commun. 163 (2020) 186–194.

[11] Z. Wang, T. Hong, Reinforcement learning for building controls: The opportunities and challenges, Appl. Energy 269 (2020) 115036.

[12] G. Ostrovski, P.S. Castro, W. Dabney, The difficulty of passive learning in deep reinforcement learning, in: 35th Conference on Neural Information Processing Systems, NeurIPS, Vol. 34, 2021, pp. 23283–23295.

[13] D. Raggett, The Web of Things: Challenges and opportunities, Comput. 48 (5) (2015) 26–32, http://dx.doi.org/10.1109/MC.2015.149.

[14] D. Zeng, S. Guo, Z. Cheng, The web of things: A survey, J. Commun. 6 (6) (2011) 424–438, http://dx.doi.org/10.4304/jcm.6.6.424-438.

[15] F. Montori, K. Liao, P.P. Jayaraman, L. Bononi, T. Sellis, D. Georgakopoulos, Classification and annotation of open internet of things datastreams, in: 19th International Conference on Web Information Systems Engineering, WISE, Springer, 2018, pp. 209–224.

[16] L. Bedogni, F. Poggi, A web of things context-aware IoT system leveraging q-learning, in: 19th Annual Consumer Communications & Networking Conference, CCNC, IEEE, 2022, pp. 405–410.

[17] D. Georgakopoulos, P.P. Jayaraman, M. Zhang, R. Ranjan, Discovery-driven service oriented IoT architecture, in: Conference on Collaboration and Internet Computing, CIC, IEEE, 2015, pp. 142–149, http://dx.doi.org/10.1109/CIC.2015.34.

[18] R. Agarwal, D.G. Fernandez, T. Elsaleh, A. Gyrard, J. Lanza, L. Sanchez, N. Georgantas, V. Issarny, Unified IoT ontology to enable interoperability and federation of testbeds, in: 3rd World Forum on Internet of Things, WF-IoT, IEEE, 2016, pp. 70–75, http://dx.doi.org/10.1109/WF-IoT.2016.7845470.

[19] N. Seydoux, K. Drira, N. Hernandez, T. Monteil, IoT-O, a core-domain IoT ontology to represent connected devices networks, in: European Knowledge Acquisition Workshop, EKAW, Springer, 2016, pp. 561–576, http://dx.doi.org/10.1007/978-3-319-49004-5_36.

[20] D. Mønster, R. Fusaroli, K. Tylén, A. Roepstorff, J.F. Sherson, Causal inference from noisy time-series data–testing the convergent cross-mapping algorithm in the presence of noise and external influence, Future Gener. Comput. Syst. 73 (2017) 52–62.

[21] B.M. Lake, T.D. Ullman, J.B. Tenenbaum, S.J. Gershman, Building machines that learn and think like people, Behav. Brain Sci. 40 (2017) e253.

[22] M. Lippi, S. Mariani, F. Zambonelli, Developing a "sense of agency" in IoT systems: Preliminary experiments in a smart home scenario, in: International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops, IEEE, 2021, pp. 44–49.

[23] I. Agadakos, G.F. Ciocarlie, B. Copos, T.L. Lepoint, U. Lindqvist, M. Locasto, Butterfly effect: Causality from chaos in the IoT, in: 1st International Workshop on Security and Privacy for the Internet-of-Things, IoT S&P, ACM, 2018, pp. 26–30.

[24] P. Madumal, T. Miller, L. Sonenberg, F. Vetere, Explainable reinforcement learning through a causal lens, in: Conference on Artificial Intelligence, Vol. 34, AAAI, 2020, pp. 2493–2500.

[25] Y. Lei, S. Zhan, E. Ono, Y. Peng, Z. Zhang, T. Hasama, A. Chong, A practical deep reinforcement learning framework for multivariate occupant-centric control in buildings, Appl. Energy 324 (2022) 119742.

[26] L. Di Natale, B. Svetozarevic, P. Heer, C. Jones, Deep reinforcement learning for room temperature control: a black-box pipeline from data to policies, in: Journal of Physics: Conference Series - CISBAT 2021 Special Issue, Vol. 2042, 2021, 012004.

[27] G. Pinto, Z. Wang, A. Roy, T. Hong, A. Capozzoli, Transfer learning for smart buildings: A critical review of algorithms, applications, and future perspectives, Adv. Appl. Energy 5 (2022) 100084.

[28] T. Schreiber, C. Netsch, M. Baranski, D. Müller, Monitoring data-driven reinforcement learning controller training: A comparative study of different training strategies for a real-world energy system, Energy Build. 239 (2021) 110856.

[29] M. Seitzer, B. Schölkopf, G. Martius, Causal influence detection for improving efficiency in reinforcement learning, in: 35th Conference on Neural Information Processing Systems, NeurIPS, Vol. 34, 2021, pp. 22905–22918.

[30] A. Fuller, Z. Fan, C. Day, C. Barlow, Digital twin: Enabling technologies, challenges and open research, IEEE Access 8 (2020) 108952–108971.

[31] L. Sciullo, A. De Marchi, A. Trotta, F. Montori, L. Bononi, M. Di Felice, Relativistic digital twin: bringing the iot to the future, Future Generation Computer Systems 153 (2024) 521–536, http://dx.doi.org/10.1016/j.future.2023.12.016.

[32] M. Matulis, C. Harvey, A robot arm digital twin utilising reinforcement learning, Comput. Graph. (2021).

[33] K. Xia, C. Sacco, M. Kirkpatrick, C. Saidy, L. Nguyen, A. Kircaliali, R. Harik, A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence, J. Manuf. Syst. 58 (2021) 210–230.

[34] L. Sciullo, A. Trotta, F. Montori, L. Bononi, M. Di Felice, Wotwins: automatic digital twin generator for the web of things, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2022, pp. 607–612, http://dx.doi.org/10.1109/WoWMoM54355.2022.00095.

[35] R. Minerva, G.M. Lee, N. Crespi, Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models, Proc. IEEE 108 (10) (2020) 1785–1824.

[36] E. Negri, L. Fumagalli, M. Macchi, A review of the roles of digital twin in CPS-based production systems, Procedia Manuf. 11 (2017) 939–948.

[37] Y. He, J. Guo, X. Zheng, From surveillance to digital twin: Challenges and recent advances of signal processing for industrial internet of things, IEEE Signal Process. Mag. 35 (5) (2018) 120–129.

[38] K.M. Alam, A. El Saddik, C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems, IEEE Access 5 (2017) 2050–2062.

[39] A.M. Madni, C.C. Madni, S.D. Lucero, Leveraging digital twin technology in model-based systems engineering, Syst. 7 (1) (2019) 7.

[40] G. White, A. Zink, L. Codecá, S. Clarke, A digital twin smart city for citizen feedback, Cities 110 (2021) 103064.

[41] H. Elayan, M. Aloqaily, M. Guizani, Digital twin for intelligent context-aware IoT healthcare systems, IEEE Internet Things J. 8 (23) (2021) 16749–16757.

[42] C. Pylianidis, S. Osinga, I.N. Athanasiadis, Introducing digital twins to agriculture, Comput. Electron. Agric. 184 (2021) 105942.

[43] J. Song, M. Kountouris, Wireless distributed edge learning: How many edge devices do we need? IEEE J. Sel. Areas Commun. 39 (7) (2021) 2120–2134, http://dx.doi.org/10.1109/JSAC.2020.3041379.

[44] S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, K. Chan, When edge meets learning: Adaptive control for resource-constrained distributed machine learning, in: Conference on Computer Communications, INFOCOM, IEEE, 2018, pp. 63–71, http://dx.doi.org/10.1109/INFOCOM.2018.8486403.

[45] Y. Zhan, J. Zhang, An incentive mechanism design for efficient edge learning by deep reinforcement learning approach, in: Conference on Computer Communications, INFOCOM, IEEE, 2020, pp. 2489–2498, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155268.

[46] R. Givan, T. Dean, M. Greig, Equivalence notions and model minimization in Markov decision processes, Artificial Intelligence 147 (1–2) (2003) 163–223.

[47] J. Desharnais, F. Laviolette, M. Tracol, Approximate analysis of probabilistic processes: Logic, simulation and games, in: 5th International Conference on Quantitative Evaluation of Systems, QEST, IEEE, 2008, pp. 264–273, http://dx.doi.org/10.1109/QEST.2008.42.

[48] T. Dean, R. Givan, S. Leach, Model reduction techniques for computing approximately optimal solutions for Markov decision processes, in: 13th Conference on Uncertainty in Artificial Intelligence, CUAI, 1997, pp. 124–131.

[49] Z. Yang, Y. Yu, C. You, J. Steinhardt, Y. Ma, Rethinking bias-variance trade-off for generalization of neural networks, in: 37th International Conference on Machine Learning, ICML, Vol. 119, PMLR, 2020, pp. 10767–10777.

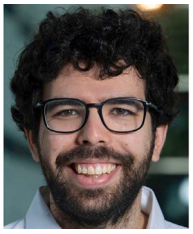
[50] J. Leprince, H. Madsen, C. Miller, J.P. Real, R. van der Vlist, K. Basu, W. Zeiler, Fifty shades of grey: Automated stochastic model identification of building heat dynamics, Energy Build. 266 (2022) 112095.

[51] B. Dong, Z. Li, G. Mcfadden, An investigation on energy-related occupancy behavior for low-income residential buildings, Sci. Technol. Built Environ. 21 (6) (2015) 892–901.

Luca Bedogni is Associate Professor at the Department of Physics, Informatics and Mathematics at the University of Modena and Reggio Emilia, Italy. He received his Bachelor Degree and Master Degree (Summa Cum Laude) in Computer Science from the University of Bologna, Italy. His Master thesis developed a MAC Protocol for Vehicular Ad Hoc Networks (VANETs). In 2015, he received the PhD degree from the University of Bologna, with the additional title of "Doctor Europeaus". In 2013, he was a visiting researcher at the RWTH Aachen university and in 2017, he was a visiting scholar at the University of California, Irvine.

He won the best paper award at MOBIWAC 2012 for his paper "DySCO: A DYnamic Spectrum and COntention Control Framework for Enhanced Broadcast Communication in Vehicular Networks". He won the best paper award at Med-Hoc-Net 2013 for his paper "Re-establishing Network Connectivity in Post-Disaster Scenarios Through Mobile Cognitive Radio Networks". He won the best paper award at CANDAR 2014 for his paper ""A Collision-Free Contention Protocol Based on Pulse/Tone Signals".

He published more than 100 papers on topics such as the Internet of Things, Communication protocols and Edge computing, in International conferences and journals. His current research interest span from the Internet of Things for heterogeneous devices, to context aware computing.



Federico Chiariotti is currently an assistant professor at the Department of Information Engineering, University of Padova, Italy, where he also received his Ph.D. in 2019. Between 2020 and 2022, he worked as a post-doctoral researcher and as an assistant professor at the Department of Electronic Systems, Aalborg University, Denmark. He has authored over 90 published papers on semantic communication, Age of Information, Smart Cities, transport layer protocols, and the application of reinforcement learning to networking and wireless communications problems. He was a recipient of the Best Paper Award at several conferences, including the IEEE INFOCOM 2020 WCNEE Workshop. His current research interests include network applications of machine learning, transport layer protocols, Smart Cities, bike sharing system optimization, and adaptive video streaming. He is currently an Associate Editor of the IEEE Networking Letters and the IEEE Transactions on Wireless Communications.