h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

Fachbereiche Mathematik und Naturwissenschaften
& Informatik

Deep Reinforcement Learning for Heat Pump Control

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

im Studiengang Data Science

vorgelegt von

Tobias Rohrer

Referentin	:	Prof. Dr. Elke Hergenröther
Korreferent	:	Prof. Dr. Andreas Weinmann
Externe Betreuerin	:	Dr. Lilli Frison

Ausgabedatum:	01.01.2022
Abgabedatum:	31.05.2022

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kentlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Freiburg im Breisgau, den 31.05.2022 .....
Unterschrift

Abstract

Heating in private households accounted for 26% of total energy consumed in Germany in 2020, which is a major contributor to the emissions generated today [1]. Heat pumps are a promising alternative for heat generation and are a key technology in achieving our goals of the German energy transformation which includes the reduction of gas emissions by 55% until 2030, compared to 1990 [2, 3]. Today, the majority of heat pumps in the field are controlled by a simple heating curve [4], which is a naive mapping of the current outdoor temperature to a control action. An alternative approach is Model Predictive Control (MPC) which was applied in multiple research works to heat pump control. However, MPC is heavily dependent on the building model, which has several disadvantages. Motivated by this and by recent breakthroughs in the field, this work applies deep reinforcement learning (DRL) to heat pump control in a simulated environment. Through a comparison to MPC, it could be shown that it is possible to apply deep reinforcement learning to archive MPC-like performance while having reduced model dependency. This work extends other works which have already applied DRL to building heating operation by performing an in-depth analysis of the learned control strategies and by giving a detailed comparison of the two state-of-the-art control methods.

Zusammenfassung

Das Heizen in privaten Haushalten hat im Jahr 2020 26% des gesamten Energieverbrauchs in Deutschland ausgemacht [1]. Dies stellt einen wesentlichen Beitrag zu den heute erzeugten Emissionen dar. Wärmepumpen sind eine vielversprechende Alternative für die Wärmeerzeugung und dadurch eine Schlüsseltechnologie zur Erreichung der Ziele der deutschen Energiewende, die eine Reduzierung der Treibhausgasemissionen um 55% bis 2030 im Vergleich zu 1990 vorsieht [2, 3]. Ein Großteil der heute im Einsatz befindlichen Wärmepumpen wird über eine einfache Heizkurve gesteuert [4], die eine einfache Abbildung der aktuellen Außentemperatur auf die einzuhaltende Vorlauftemperatur darstellt. Ein alternativer Ansatz ist die modellprädiktive Regelung (Model Predictive Control, MPC), die in zahlreichen Forschungsarbeiten zur Steuerung von Wärmepumpen eingesetzt wurde. MPC ist jedoch stark von einem Gebäudemodell abhängig, was mehrere Nachteile mit sich bringt. Aus diesem Grund und aufgrund der jüngsten Durchbrüche auf dem Gebiet, wird in dieser Arbeit Deep Reinforcement Learning (DRL) auf die Wärmepumpensteuerung in einer simulierten Umgebung angewendet. Durch einen Vergleich mit MPC konnte gezeigt werden, dass es möglich ist, mit Deep Reinforcement Learning eine MPC-ähnliche Leistung zu erzielen und gleichzeitig die Modellabhängigkeit zu verringern. Diese Arbeit leistet einen Beitrag zu anderen Arbeiten, die DRL bereits auf den Heizungsbetrieb von Gebäuden angewendet haben, indem eine eingehende Analyse der erlernten Regelungsstrategien und ein detaillierter Vergleich der beiden Regelungsmethoden durchführt wird.

Contents

List of Figures

List of Tables

Nomenclature

General

$\mathbb{E}_*[\cdot]$	expectation when acting optimally
$\mathbb{E}_\pi[\cdot]$	expectation when acting according to policy π

MDP

$\mathcal{A}(s)$	set of possible actions when in state s
$\mathcal{S}$	set of possible environment states in the MDP
a	realization of an action
a'	realization of successor action of s after incrementing t
A_t	any action at timestep t
G_t	the return, which defines the sum of discounted future rewards from the current time step t on
r	realization of a reward
R_{t+1}	any reward at timestep $t+1$, after taking an action at timestep t
s	realization of a state
s'	realization of successor state of s after incrementing t
S_0	any initial state
S_{t+1}	any state at timestep $t+1$, after taking any action at timestep t
S_T	any terminal state
t	discrete timestep $\in \mathbb{N}_0$, which gets incremented after the agent takes an action

Policies and Value Functions

π	policy which is used to select actions

$\pi(a|s)$ policy written as probability distribution over actions given the state

π^* a policy which acts optimal

$Q_*(s, a)$ optimal action-value function, which determines the expected future return by being in state s and taking action a and acting optimally from there on

$Q_\pi(s, a)$ action-value function, when acting according to policy π after taking action a

$V_*(s, a)$ optimal state-value function, which determines the expected future return by being in state s and acting optimally from there

$V_\pi(s, a)$ state-value function when acting according to π

Deep Reinforcement Learning

$\mathcal{L}()$ loss function to be minimized during training of a neural network

$J()$ objective function to be maximized during trainig

α learning rate

θ trainable parameters of a neural network (weights and biases)

$\nabla_\theta J(\theta)$ gradient of the objective function with respect to the parameters of the neural network

It must be noted that the notations and therefore this nomenclature is based on [5].

1 Introduction

One goal of the German energy transformation is to reduce greenhouse gas emissions by 55% by 2030, compared to 1990 [2]. In 2020, room heating in private households accounted for 26% of total energy consumed in Germany. Unfortunately, as of 2020, 68% of this heating energy was generated by heating systems that rely on fossil fuels such as oil or gas, making heat generation in private households a major contributor to the emissions generated today [1].

Heat pumps, on the other hand, are an attractive alternative to be used for room heating in private households, as they exploit heat from natural energy sources such as ambient air or groundwater and bring it to a higher temperature level using electrical energy, so that it can then be used to supply heat to buildings [6]. Therefore, heat pumps can be used for emission-free room heating if the electricity used to operate the heat pump comes from renewable energy sources. This makes heat pumps one of the key technologies for achieving the goals of the ongoing energy transition [3].

While the penetration of heat pumps in Germany is steadily increasing [7], most heat pumps in the field are controlled by a simple heating curve [4]. The heating curve presents a static mapping from the current outdoor temperature to a control action of the heat pump. While this approach is easy to implement and maintain, it has potential for improvement as other factors, like a forecast of the outdoor temperature, do not have any influence on a heat pump controlled by a heating curve but could improve the control strategy in terms of energy efficiency and comfort [4]. Additionally, due to the increasing share of renewable energy sources, the electricity supply is becoming more dependent on external factors such as the weather [8]. Therefore, it is becoming more and more important to be able to balance the supply and demand of electricity. This can be done by time-based electrical prices, which serve as an incentive to regulate the electricity demand, thus the name *demand response* (DR) [9]. As of today, time-based varying electrical prices were not applied to residential customers on a larger scale in Germany [9]. Anyhow, it is expected and is reasonable that varying electricity prices will be available for residential customers in the future [8, 9]. More advanced control strategies are needed to exploit the variable prices by shifting the heating load to low price periods, while also incorporating other factors such as the weather forecasts.

As a result, model predictive control (MPC) has been applied to heat pump control in the past years. The basic idea of MPC is to make use of a simplified building model to predict the effect of control actions. As promising as this sounds, this also introduces a strong dependency on the

model which is used by MPC. This dependency entails disadvantages that are described in more detail in section 3.2.2. Overall, in recent years, the work on MPC used for heat pump control has increased. However, so far it has not gained acceptance in practice [10–12].

Motivated by the recent breakthroughs achieved by deep reinforcement learning, the goal of this work was to apply deep reinforcement learning to heat pump control. The concepts were implemented and evaluated solely in simulation, which was provided by the Fraunhofer Institute for Solar Energy Systems (ISE). Based on this, the following three research questions were to be answered:

1. Is it possible to apply deep reinforcement learning to learn efficient heat pump control strategies in the simulation provided?

2. How well is it working compared to MPC and the heating curve?

3. Can it be extended to a demand response scenario?

This work was carried out in collaboration with the Fraunhofer ISE and extends other works which have already applied deep reinforcement learning to heat pump control mainly by (1) providing another working example, (2) performing a detailed evaluation of the learned heating strategies, and (3) performing a comparison between MPC and deep reinforcement learning.

The rest of the work at hand is structured as follows: Chapters 2 and 3 provide the fundamentals of deep reinforcement learning and heat pumps. Chapter 4 lists related works which have applied deep reinforcement learning to control heat pumps or related heating systems. Chapter 5 describes the simulation framework which was used in this work and specifies the research questions in more detail. Chapter 6 describes the technical concept of how deep reinforcement learning was applied. Chapter 7 provides high level implementation details of the presented solution. Chapter 8 describes experiments that were conducted in order to evaluate the functionality of the proposed solution. A discussion and final conclusion are made in chapters 9 and 10.

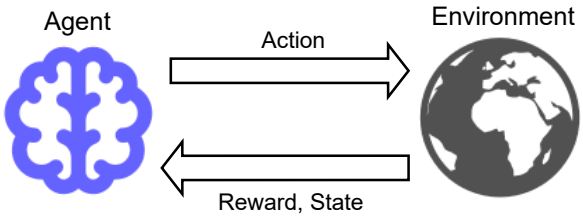

Figure 1: Basic functionality of reinforcement learning. The agent interacts with the environment by choosing actions. As a result it receives a reward and state information of the environment. The image is derived from [13,14].

2 Fundamentals of Reinforcement Learning

The following chapter provides the necessary fundamentals to understand the methods used in this thesis. In 2.1 the reader is given an overview of the basic functionality of reinforcement learning. Section 2.2 describes how the concepts of deep learning are applied to reinforcement learning, resulting in *deep* reinforcement learning (DRL). Thereby, it is assumed that the reader already has knowledge of techniques of supervised learning, especially about deep neural networks. Section 2.3 categorizes different approaches of reinforcement learning. Finally, in section 2.4 the reader is given an overview of recent advantages in the field of reinforcement learning.

It must be noted that the intention of this chapter is to explain the fundamental concepts of reinforcement learning as well as to give a basic picture by explaining different approaches. This involves explanations of dynamic programming 2.1.2 and temporal difference learning 2.1.3, which are no longer state of the art to be used purely in practice. However, they are described as their core ideas are still fundamental for more modern algorithms. The reader who is primarily interested in the methods used in this work is referred to sections 2.1.1, 2.2.2 and from there onwards.

2.1 Reinforcement Learning

Deep reinforcement learning targets the learning of strategies through interactions between an *agent* and an *environment* [15]. Figure 1 shows this basic relationship. As described in [13, 14, 16], the agent interacts continually over discrete time steps $t \in \mathbb{N}_0$ with the environment by choosing an *action* $A_t \in \mathcal{A}(s)$. As a response, the agent gets information about the new environment *state* $S_{t+1} \in \mathcal{S}$ and a scalar *reward* $R_{t+1} \in \mathbb{R}$. The reward serves as feedback on the quality of a certain action in a given state. The goal of the

agent is to maximize the sum of rewards over time. [13, 14, 16]

This fundamental concept will be explained in more detail in this section. Please note that the explanations, definitions, derivations and notations from this section and its subsections are if they are not explicitly cited otherwise, based on Sutton and Barto [13].

2.1.1 Markov Decision Process

The Markov decision process (MDP) [17] serves as a mathematical formalism of sequential decision making problems. It is used to formally define environments in reinforcement learning. The terms MDP and environment are often used interchangeably in the literature. In this work, the term MDP is used when describing the formal properties and the term environment when referring to implementation aspects of reinforcement learning. In the following, the components which are used to define the MDP are listed and described in more detail.

State Space The state space $\mathcal{S}$ defines the possible states $S_t \in \mathcal{S}$ that the MDP can evolve into. A state is used to describe the MDP at a given time step. A fundamental concept of states in MDPs is the Markov property, which defines that "The future is independent of the past given the present", as cited in [14, p. 4]

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2..., S_t]. \tag{1}$$

Therefore, in order for the Markov Property to not be violated, a state at time step t must "include information about all aspects of the past agent–environment interaction that make a difference for the future" [13, p. 49]. Only episodic MDPs are considered in this work. Unlike continuing MDPs, episodic MDPs contain terminal states S_T with a final time step T. Any initial state is denoted as S_0. An interaction between environment and agent which stretches from $S_0, A_0, S_1, A_1..., S_T, A_T$ is called an *episode*.

Action Space The action space $\mathcal{A}(s)$ defines the possible Actions $A_t \in \mathcal{A}(s)$ the agent can take in a given state. Taking an action triggers a *state transition* and results in a new MDP state S_{t+1}. Action spaces can be categorized by their *action type*, which can be either discrete or continuous. In continuous action spaces, an action can take any real number in a specified interval. On the other hand, the actions in discrete action spaces can only take discrete values.

Reward Function After taking an action, the agent not only receives information about the new MDP state but also a scalar reward $R_{t+1} \in \mathbb{R}$. The calculation of the reward given can be expressed as a function $r(s, a, s')$, which depends on the current state s, the successor state s' as well as the action a taken to transfer from the current to the successor state. The *return* G_t defines the sum of discounted future rewards from the current time step t on:

$$G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t+1} R_i = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{2}$$

$$= R_{t+1} + \gamma G_{t+1}. \tag{3}$$

The sum is discounted by factor $\gamma \in [0, 1]$ which is used to define how much impact future rewards have compared to immediate rewards. The higher γ, the more impact future rewards have on the return. In the case where $\gamma = 1$, rewards in the far future would have the same impact on the return as immediate rewards [14]. Equation 3 shows the recursive relationship of G_t which will be later used to derive the Bellman optimality equation in section 2.1.2.

Transition probability As mentioned before, taking an action causes the MDP to transfer into a new state. In *stochastic environments*, random dynamics are involved during state transitions. Therefore, the MDP can transition into a set of different possible successor states $s' \in \mathcal{S}$ and experience a different reward r, given the same state s and action a. The transition probability

$$p(s', r|a, s) = \mathbb{P}[S_{t+1} = s', R_t = r|A_t = a, S_t = s] \tag{4}$$

defines the probability, that the MDP evolves into the successor state s' and experiences reward r, if the agent chooses action a in state s. In contrast, in *deterministic environments* where no random dynamics during state transitions are involved, the successor state s' can be calculated based on s and a deterministically.

The components described above define an environment for reinforcement learning. The agent chooses actions based on a *policy* π to interact with this environment. More formally, a policy π specifies the behaviour of an agent by defining a probability distribution of actions based on the current state [14]:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{5}$$

The goal of the agent is to find a policy, which maximizes the return (2). A policy, which acts optimally for a given MDP is denoted as *optimal policy* π^*. To be able to evaluate and find policies that are optimal, value functions can be used to assess "how good it is for the agent to be in a given state" [13, p. 58]. The *state-value function* $V_\pi(s)$ determines the expected future return by being in state s and following policy π from there on:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]. \tag{6}$$

Likewise, the *optimal* state-value function $V_*(s)$ determines the expected future return by being in state s and by acting *optimally* and not according to a certain policy π from there.

Derived from the state-value function, the *action-value* function $Q_\pi(s, a)$ estimates the return when taking an action in a particular state and following policy π afterwards:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{7}$$

The action-values are commonly named Q-values. If assumed to act optimally after taking action a, the action-value function is denoted as $Q_*(s, a)$.

In case the optimal action-value function $Q_*(s, a)$ is known, acting optimally can be done by simply following a greedy policy, which chooses the action which maximizes $Q_*(s, a)$ in each time step:

$$\pi^* = \arg\max_a Q_*(s, a). \tag{8}$$

Or in simple words, acting optimally can be done by choosing the best possible action in a given state. [13, 14, 17]

2.1.2 Dynamic Programming

According to (8) an optimal policy can be easily obtained once the optimal action-value function $Q_*(s,a)$ is known. Given the strong limitation that the environment is fully known, dynamic programming can be applied to calculate $Q_*(s,a)$ by utilizing the *Bellman optimality equation* [18], which was derived in [13, p. 63] as

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}_*[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_*[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma V_*(S_{t+1})|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a')|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} Q_*(s', a')). \end{aligned} \tag{9}$$

The Bellmann optimality equation exploits the recursive definition of the return G_t (3), as well as the relationship between the optimal state-value and action-value functions, defined by

$$V_*(s) = \max_a Q_*(s,a). \tag{10}$$

The optimal state-value equals the optimal action-value of the best action. This allows defining a recursive relationship of the optimal action-value function $Q_*(s,a)$. Therefore, $Q_*(s,a)$ can be calculated for each state-action pair iteratively, which is according to the concept of dynamic programming. The results of the calculations are stored in a tabular manner in *Q-tables* and updated iteratively until convergence. It is important that a strong restriction on the full knowledge of the MDP must be given in order to apply dynamic programming. This involves the knowledge about all transition probabilities, the complete state, and action space, as well as the reward function. [13]

2.1.3 Temporal-Difference Learning

In reality, it is rare that the properties of the MDP are fully known. Therefore the transition probabilities and reward function cannot be directly substituted into the Bellman optimality equation (9) to calculate $Q_*(s,a)$. In this case, Temporal-Difference (TD) learning can be used. In TD learning, value functions like $Q_*(s,a)$ are estimated iteratively based on experiences, which are collected by the agent due to interaction with the environment[1]. It is

[1]The same applies for other value functions like $V_*(s), V_\pi(s)$ and $Q_\pi(s)$.

important to note, that the updates of the estimated value functions take place at every time step and not only after finishing one episode, which differentiates TD learning from Monte Carlo learning. As a result, reinforcement learning can be applied without prior knowledge of the environment's reward function and transition probabilities as they are estimated by interaction. A more detailed idea of TD, the concept of exploration as well as the distinction between on and off-policy algorithms will be explained based on the following two TD learning algorithms:

SARSA Based on the main idea of TD learning, SARSA iteratively estimates $Q_\pi(s, a)$ (7) by interaction with the environment. $Q_\pi(s, a)$ is represented in form of a Q-table, which stores the estimated Q-values of each state-action pair. The estimated Q-values are iteratively updated using the following update rule:

$$Q_{i+1}(S_t, A_t) \leftarrow Q_i(S_t, A_t) + \alpha \underbrace{\left[\underbrace{R_{t+1} + \gamma Q_i(S_{t+1}, A_{t+1})}_{\text{TDtarget}} - Q_i(S_t, A_t) \right]}_{\text{TDerror}}. \quad (11)$$

The parameters used in the update $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ are the reason for the name SARSA. The subscript i indicates the incremental update. The subscript π is omitted in the above equation to prevent overloading of notations. The update is parametrized by a step size or learning rate $\alpha \in (0, 1]$. It defines the rate of impact every sampled experience has on the approximated Q-value. Since the concept of a SARSA TD update is central for the understanding of other RL algorithms, a more detailed explanation follows: By interacting one step with the environment, a tuple of the parameters $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ is collected, which makes one *experience*. Based on the experience and by applying the idea derived from the Bellman equation (9), a more accurate approximation of $Q_\pi(S_t, A_t)$ can be made by the TD target (see equation 11). The ultimate goal is to minimize the TD error (11). Finally, the estimation of $Q_\pi(S_t, A_t)$ is updated relative to the TD error and the learning rate α.

When estimating Q-values with SARSA using a purely deterministic greedy policy (8), there is a high risk that many state-action pairs will not be visited. This leads to biased estimations of the Q-values as some parts of the environment remain unexplored. Therefore, to

ensure exploration, a *ϵ-greedy policy* can be used, which takes a random action with probability ϵ at every time step. [13]

Q-learning Another example for a TD algorithm is Q-learning [19]. The update rule in Q-learning is defined as follows:

$$Q_{i+1}(S_t, A_t) \leftarrow Q_i(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q_i(S_{t+1}, a) - Q_i(S_t, A_t)]. \quad (12)$$

The update rule seems similar to that of the SARSA algorithm (11). But is different, as it approximates the optimal action-value function $Q_*(s, a)$ by always using the maximum Q-value (12) as TD target, even though another policy might be followed by the agent. Because of this, the optimal policy used to estimate the Q-values differs from the one used by the agent to interact with the environment, which must include an exploration element and is therefore not optimal. Considering that, Q-learning is an example of an *off-policy* algorithm. In contrast, SARSA uses the same policy for interacting with the environment as well as for estimating the Q-values. Therefore, SARSA is an example for an *on-policy* algorithm. [13]

2.2 Deep Reinforcement Learning

The algorithms in 2.1.2 and 2.1.3 can be classified as *tabular solution methods*, as they use tables to represent $Q(a, s)$ [20]. As a result, those methods suffer from the "curse of dimensionality" [18], as their computational and storage requirements grow exponentially with the size of the action and state space [15]. In many scenarios of practical interest, the state and action space quickly exceed the limit of what is manageable by using tabular methods [20]. As an example, the game of backgammon has over 10^{20} different states [15], and even a higher number of state-action pairs which had to be maintained by a Q-table and thus stored in memory. Even if it would be possible to maintain a table large enough for all state-action pairs to fit in, the time required for the agent to experience every possible state-action pair in order to estimate its Q-value would be immense [15]. Deep neural networks, on the other hand, are known for their generalization capabilities. Therefore, the problems just listed above can be solved by the use of deep neural networks to approximate the value functions [20]. As value functions are used, those methods are also called *value-based* methods. Deep Q-learning [21] was the first algorithm according to this concept and is presented in section 2.2.1. Alternatively, instead of approximating the value functions, deep neural networks can be also used to approximate an optimal policy directly [20].

This approach is called *policy gradient methods* and is described in section 2.2.2. Lastly, *Actor-critic* methods combine the ideas of both approaches and are additionally presented in section 2.2.2. In all cases, deep neural networks are used in combination with reinforcement learning and hence the name deep reinforcement learning [20].

2.2.1 Deep Q-Learning

In deep Q-learning [21], a deep neural network with parameters θ, denoted as $Q(s, a; \theta)$, is used as approximation of the optimal action-value function. To be able to train $Q(s, a; \theta)$, the concept of a TD update (11) is applied. Therefore, the loss function, which is to be minimized during training of the deep neural network can be defined as follows [21]:

$$\mathcal{L}(\theta_i) = \mathbb{E}[(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_i) - Q(S_t, A_t; \theta_i))^2]. \tag{13}$$

Derived from the main ideas of supervised learning, backpropagation is used to compute the gradient of $\mathcal{L}(\theta_i)$ with respect to the parameter vector θ_i, which is denoted as $\nabla_\theta \mathcal{L}(\theta_i)$.[2] By taking a stochastic gradient decent step $\theta_{i+1} = \theta_i - \alpha \nabla_\theta \mathcal{L}(\theta_i)$, the parameters θ are updated in the direction which minimizes $\mathcal{L}(\theta_i)$ [21].

Additionally, the deep Q-learning algorithm introduced in [21] made use of an *experience replay memory* [22]. The experience replay memory stores experiences that are collected by the agent through interaction with the environment. Instead of training $Q(s, a; \theta)$ based on online experiences while interacting with the environment, the network is trained based on *mini-batches* drawn at random from the experience replay memory. This introduced two main advantages, which stabilizes the training of $Q(s, a; \theta)$ [21]. Firstly, the correlation between training samples in one training batch is reduced. Secondly, experiences can be used multiple times for training [21].

2.2.2 Policy Gradient Methods

Value-based methods presented in section 2.2.1 aim to learn a value function and use a policy that selects actions based on the approximated Q-values. In contrast, policy gradient methods learn a parametrized policy directly [20]. Similar to (5) the policy to be learned is a probability distribution over actions given the state and parameters θ, defined as [20]:

$$\pi(a|s, \theta) = \mathbb{P}[A_t = a_t | S_t = s_t, \theta_t = \theta]. \tag{14}$$

[2] Analogue to (11), the subscript i denotes a step count which is incremented after every update.

For sake of readability, in the following, an action taken based on a state by a policy is simply named *decision*. Like in section 2.2.1, it is assumed that a deep neural network is used as an approximator and θ denotes its parameters. During training, the parameters of the network are updated with the goal to maximize expected rewards. This is quantified by an *objective function*, which serves as a scalar performance measure, which is defined as [20]:

$$J(\theta) = V_{\pi_\theta}(S_0) = \mathbb{E}[G_0]. \tag{15}$$

$J(\theta)$ utilizes (6) but differs, as it only considers cases when starting from a start state [3] [20]. For simplification, discounting (3) is neglected by setting $\gamma = 1$. In case the gradient of the objective function $\nabla_\theta J(\theta_t)$ with respect to θ is known, maximizing the objective function and thus the expected reward can be done by utilizing gradient ascent[4] [23]:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\theta_i). \tag{16}$$

The goal and at the same time biggest challenge of policy gradient methods is to find estimates for $\nabla_\theta J(\theta)$ [24]. The difficulty here is that the expected return of an episode and therefore $\nabla_\theta J(\theta)$ depend not only on the policy but also on the environment dynamics (4), which are unknown [23]. Anyhow, based on ideas presented in [25], the gradient can be estimated independent of the environment dynamics in a *model free* manner by[5]

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t, \theta) G_0]. \tag{17}$$

In order to estimate the gradient according to (17), information are collected by interaction with the environment and averaged over N episodes [26]:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t, \theta) G_0. \tag{18}$$

The term G_0 represents the observed return, and therefore the goodness of an episode. Intuitively, the log probabilities of decisions made by π in the collected episode are increased relative to the goodness of the episode [24]. Interacting for multiple episodes and by performing multiple gradient ascent

[3] In literature the terms rollout or trajectories are commonly used and donated with τ.

[4] Gradient ascent works analogue to gradient descent, but differs in the direction of the update.

[5] A detailed derivation of the gradient estimation (17) can be found in [20, 23, 24]

steps using the estimated gradients, increases the probability of taking decisions that maximize the objective function in the long run [24].

Complementary to definition (17), a baseline b can be subtracted from the return in order to reduce the variance of the gradient estimation [24, 25]. Additionally, the causality principle is applied which states that rewards experienced at any time step $t^* < t$ do not depend on actions taken from time step t on [23, 27]. This leads to the following estimator of the policy gradient [24]:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t, \theta)(G_t - b(S_t))] \\ &= \mathbb{E}[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t, \theta) \operatorname{Adv}_t]. \end{aligned} \tag{19}$$

Any function can be used as b, but choosing the state-value function $V_\pi(s)$ (6) has shown to be effective [24, 28]. As $V_\pi(s)$ is an estimate of the expected future return and G_t is the observed future return, their difference can be interpreted as how much better a decision was than expected [24]. Thus, the gradient updates are weighted by an *advantage* denoted as Adv_t, which gives positive weighting to decisions that are better than expectation and a negative weighting to decisions that are worse [24].

The methods presented above estimate the gradient necessary to maximize the objective function $J(\theta)$ independent of the environment dynamics. The main ideas presented in this section are derived from [25, 27] and are known as *vanilla policy gradient method*. Even though the ideas have been presented some time ago, they still build the theoretical foundation of more advanced reinforcement learning algorithms [24].

Actor-Critic The vanilla policy gradient method (19) requires the return of an episode to estimate the advantage and therefore the gradient. Thus, it must complete a full episode before the gradient can be estimated [6]. Actor-Critic methods enable advantage estimation and thus learning at every time step [20]. Based on [27] and the idea of the TD error (section 2.1.3), the advantage can be estimated by

$$\operatorname{Adv}_t = R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t). \tag{20}$$

Analogue to the policy $\pi(a|s, \theta)$, the value function $V_\pi(s, w)$ can be approximated by a neural network with parameters w, which is used

[6] As vanilla policy gradient requires a full episode, it is a Monte Carlo method.

to estimate the advantage according to (20) [20]. As explained before, the advantage suggests the direction for the gradient update when *acting* according to $\pi(a|s,\theta)$, thus its name *critic*. A neural network for $\pi(a|s,\theta)$ is often called *policy network*, whereas a neural network for $V_\pi(s,w)$ is often called *value network*. [20, 23]

PPO Proximal policy optimization (PPO) [29] is a state-of-the-art policy gradient method that has already been applied for complex problems. For example, OpenAI five builds on PPO and was the first AI that was able to defeat the world champion in an esports game [30]. PPO was designed with the goal to balance complexity and simplicity [29], which makes it robust to the choice of hyperparameters such as the learning rate. PPO builds on the basic ideas of trust region policy optimization (TRPO) [24], which was designed with the aim of providing stability during training by constraining the size of the policy updates [29, 31]. The primary goal is to prevent destructively large policy updates [29]. PPO also aims to limit policy updates in a way to ensure training stability, but unlike TRPO it does not add hard constraints, which makes PPO easier to implement [32]. In addition to this central idea, PPO uses estimates of the advantage for the policy update in an actor-critic style, like it was presented above [29]. Lastly, unlike standard policy gradient methods which use each experience only once for gradient estimation, PPO enables the use of multiple training epochs on the collected experiences. This is done by alternating between interaction with the environment, where experiences are collected and optimization, where the experiences are used for policy updates [29]. Intuition about the training procedure of PPO is given by the pseudocode 1. This pseudocode is based on [29].

Algorithm 1 PPO, Actor-Critic and Single Actor Style

for iteration=1,2, ... **do**
  Run policy $\pi_{\theta_{old}}$in the environment for T time steps
  Compute advantage estimates $\text{Adv}_1, \text{Adv}_2, ..., \text{Adv}_T$
  Optimize with K epochs
  $\theta_{old} \leftarrow \theta$
end for

2.3 Taxonomy

As the previous chapters have already made clear, reinforcement learning consists of a wide variety of algorithms and approaches. The goal of this section is to categorize and thus provide an overview. In addition, the section is intended to help to place the concepts used in this work into the big picture of (deep) reinforcement learning. Figure 2 places (deep) reinforcement learning in the field of machine learning and shows a classification of different approaches. Thereby, it is once again made clear that the concepts used in this thesis belong to the class of model free deep reinforcement learning. Model free methods treat the environment as a black box and solely learn by experiences collected by interaction. Model based methods, on the other hand, use or learn a model to reason about the future and therefore *plan* their actions [33]. As promising as this sounds, model free methods are easier in implementation and more widely used [34]. Besides the branching points shown in 2, (deep) reinforcement learning algorithms can be further subdivided by the following properties:

On-policy or off-policy The distinction of either learning on-policy or off-policy was already explained in section 2.1.3, by taking SARSA and Q-learning as example algorithms. On-policy algorithms optimize their policy based on data that was collected by interacting with the environment according to the very same policy [34]. Off-policy algorithms use different policies for interaction and optimization. An example of an off-policy algorithm is deep Q-learning, as it stores its experiences in a replay buffer [34]. During learning, experiences are drawn at random from this buffer, thus a mini-batch may include experiences that were collected using older versions of the policy. With exceptions, policy gradient methods work mostly on-policy, whereas value based methods usually work off-policy [34].

Action type Lastly, as already mentioned in section 2.1.1, (deep) reinforcement learning algorithms are also classified according to their action type, as they support either discrete, continuous, or both action types. This is an important distinction since some problems are either discrete or continuous in nature, which limits the choice of algorithm.

Online or offline learning The concepts presented so far belong to the class of online reinforcement learning. Even if an algorithm works off-policy, it learns online, by interacting iteratively with an environment. Offline reinforcement learning on the other hand aims to learn strategies

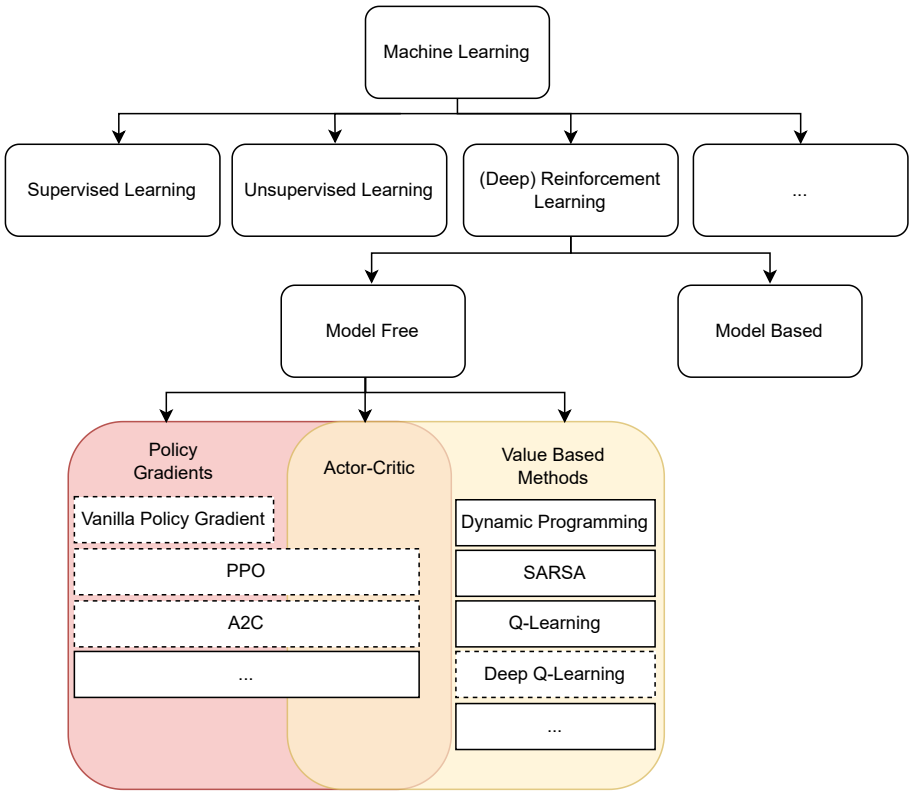

Figure 2: A Non-exhaustive taxonomy of (Deep) Reinforcement Learning algorithms, derived by [34] and extended here. The dashed lines indicate algorithms which rely on neural networks as function approximators and thus belong to the category of *deep* reinforcement learning.

based on previously collected data, without any online interaction with an environment [35, 36]. The concept is also called full batch, fully off-policy, or data-driven reinforcement learning.

2.4 State of the Art

Deep learning has refined the state-of-the-art capabilities in a wide range of areas like natural language processing, image processing or anomaly detection [37]. Similarly, deep learning has also advanced reinforcement learning, which resulted in deep reinforcement learning. Two works, in particular, demonstrated the possibilities of DRL. At first, Mnih et al. [21] proposed the concept of deep Q-learning and showcased its functionality by

training agents to autonomously play Atari 2600 games. With the same algorithm and hyperparameters, they were able to set new benchmarks compared to previous algorithms for every single game played. This was the first time an artificial agent was able to excel at a diversity of different complex tasks [21]. Second, Silver et al. [38] managed to defeat the European champion in the game of Go, which has long been considered the most challenging classic game for artificial intelligence to play [38]. Besides the application to games, which is mostly taken for demonstration or development purposes [30, 37–39], DRL has been applied to areas such as robotics [40–43], economics [43, 44], natural language processing [45, 46], autonomous driving [47–49], energy system management [50] and many more [43, 51].

Especially in the field of model free reinforcement learning, a lot of research was done which has led to the development of a large number of different algorithms [29, 31, 37, 52–59]. A rough overview of how those algorithms can be classified is provided in section 2.3. A more detailed evaluation and description of some of those algorithms was carried out by [60]. Anyhow, a complete performance comparison of the different algorithms is complex as their performance often depends on implementation details, the problem at hand, or preset random seeds [50, 61].

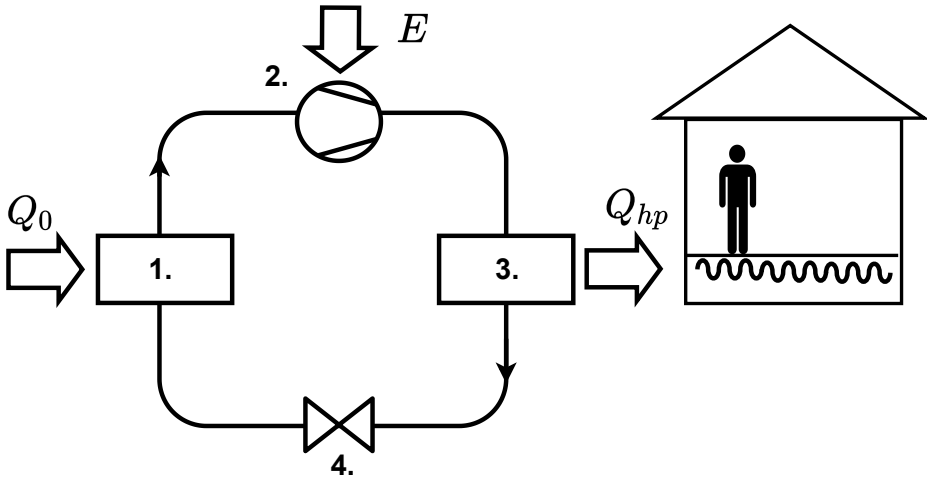

Figure 3: Basic components of a heat pump supplying heat to an underfloor heating system: (1) evaporator absorbs heat from surrounding source, (2) compressor rises temperature, (3) condenser extracts heat which can be used for heating and (4) expansion valve returns cooled refrigerant. Derived from [62, 63]

3 Fundamentals of Heat Pumps

The goal of this chapter is to provide basic knowledge about heat pumps. Additionally, two control approaches for heat pumps are presented and discussed.

3.1 Heat Pumps

The basic concept of heat pumps is to absorb heat from surrounding low-temperature sources and to bring it to a higher temperature level, so it can be used for space heating or for domestic hot water supply [6]. Examples of common low-temperature sources are ambient air, soil or underground water [63]. The majority of heat pumps in the field are electrically driven [64] and consist of four elements: A compressor driven by an electric motor, an expansion valve, an evaporator, and a condenser. The components are connected and form a circuit through which a refrigerant circulates [6].

This circle is illustrated by figure 3 and is explained based on [6, 62, 63] in more detail below: (1) The evaporator allows the refrigerant to absorb heat from the surrounding source Q_0, which has a higher temperature. This causes the refrigerant evaporate to gas. In the work at hand, an air source

heat pump is used, which absorbs heat from ambient air. (2) Subsequently the gaseous refrigerant is compressed to higher pressure, driven by electrical energy E, which causes a rise in temperature. (3) Heat is extracted for room heating, thus the refrigerant condenses back into liquid. The recovered heat Q_{hp} is used to heat water which is used, for example by radiators or underfloor heating for space heating. The temperature of the heated water is referred to as *supply temperature* T_{sup}. (4) Lastly, in the expansion valve, the refrigerant drops to lower pressure and cools down further before it returns to the evaporator at low-temperature level ready to absorb heat from the surrounding temperature source again to start the process from the beginning. [6, 62, 63]

The efficiency of a heat pump can be measured by the coefficient of performance (COP), which defines the ratio between the supplied energy W necessary to operate the heat pump and the heat Q_{hp} gained from it [63]:

$$COP = \frac{Q_{hp}}{E}. \tag{21}$$

In addition to the efficiency of the heat pump itself, the used energy E and thus the COP is highly influenced by the difference between the temperature of the low-temperature source and the supply temperature of the heating medium [6]. Therefore, low-temperature heating systems like under-floor and in-wall heating systems are more suitable for heat pump operation than traditional radiators, which need higher supply temperatures caused by their small surface [6]. Furthermore, as already mentioned, air source heat pumps use the ambient air as a low-temperature source. This makes the efficiency of the heat pump dependent on the outside air temperature.

As the compressor is driven by electricity, heat can be generated solely on the use of electric energy without directly burning fossil fuels such as oil or gas. Heat pumps can be seen as climate neutral energy supplier, if the consumed energy is provided by regenerative sources. This makes heat pumps one of the key technologies for achieving the goals of the ongoing energy transition [3].

3.2 Heat Pump Control

In the following, some of the basic heating control techniques for heat pumps are explained. For a more complete overview of heating control techniques, the reader is referred to [4, 6, 11].

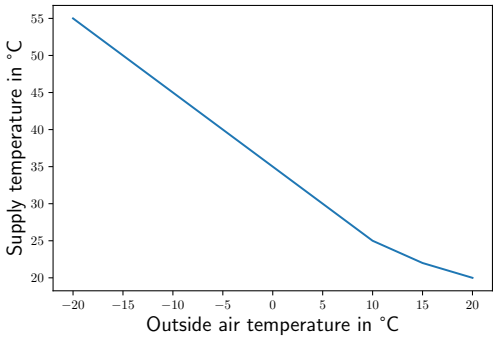

Figure 4: Exemplary representation of a heating curve, which defines the supply temperature based on the outside air temperature. Based on [4].

3.2.1 Heating Curve

Today, most of the heat pumps in the field are controlled by a heating curve [4]. As illustrated in figure 4, the heating curve defines the supply temperature to be maintained by the heat pump depending on the outside temperature. In addition, thermostats are often used to adjust the heating for individual temperature zones. This is done by a thermostatic valve which regulates the mass flow $\dot{m}$ (in kg/s) of the heated water [6]. Nevertheless, as described in section 3.1, a high supply temperature leads to an inefficient control of the heat pump. The heating curve can be classified as a feed-forward control system as the control of the supply temperature does not depend on any feedback, like the current indoor temperature [6]. Because of ease of implementation and maintenance, this concept is the most common in the field [4].

3.2.2 Model Predictive Control

The heating curve based control strategy can be categorized as non-predictive [4], as it only reacts based on the current outside temperature. Other parameters like the the weather forecast, occupancy profiles, or potential time varying price signals are not taken into account. However, the inclusion of these parameters could significantly improve the control strategy in terms of energy efficiency, comfort and operational costs.

As a result, model predictive control (MPC) has been applied to heat pump control. As described in various works [6,11,65], the basic idea of MPC is to make use of a simplified building model to predict the effect of control

actions. An optimal control action is obtained by defining an optimization problem that minimizes a cost function. This cost function can be used to encode multiple objectives like minimization of electricity usage and comfort. External factors like weather data and the electricity price can be included in order to find an optimal control strategy. After an optimal action is determined by solving the optimization problem, the action is executed. This causes the controlled system to evolve into a new state. Information about the new system state, like the new indoor temperature, is used to define a subsequent optimization problem for the next time step. [6, 11, 65]

Since MPC reacts to the new system information when planning, it can be categorized, unlike the heating curve, as a feedback control system. The work on MPC in the context of optimal heat pump operation goes back to 1988 [65, 66]. Today, many works have been made in this area, of which summaries can be found in [10, 11, 67, 68].

As promising as this sounds, the MPC approach has not yet gained acceptance for heat pump control in practice [10, 11]. This is due, among other reasons, to drawbacks of the method. On the one hand, the performance of MPC highly depends on the accuracy of the building model [4, 10, 12]. Additionally, the computational effort required by MPC to solve the optimization problem at every time step is relatively high compared to other methods. This results in additional hardware requirements on the controller level [4, 10, 11]. This means, the model must be relatively simple in order to achieve reasonable run times, but at the same time it must be accurate enough to make it useful for MPC [10, 69, 70]. This leads to the fact, that in practice often two models are necessary. One simplified model which can be used by MPC to optimize and one more complex and accurate model which is used as test environment to simulate the performance of the MPC controller [10, 67]. Lastly, every building is different and therefore has different thermodynamic properties. This leads to a high customization effort as the models used by MPC must be customized to the building at hand [10].

4 Related Works on Reinforcement Learning for Heating Systems

As described in section 3.2, controlling a heat pump using MPC or the traditional heat curve presents challenges. As a result of this, combined with breakthroughs in the field of deep reinforcement learning, work has already been done on controlling heat pumps using deep reinforcement learning. As described in section 2, the basic idea of deep reinforcement learning is to solve a task by learning from interactions with an environment. So theoretically, deep reinforcement learning could be directly applied to heat pump control in the real world, making the laborious process of model creation obsolete. However, this approach is impractical as (1) a big amount of interactions are required for learning, and (2) exploration during training causes heating strategies with random actions which could cause uncomfortable thermal conditions. Therefore, most of the works listed in this section used a model as an environment for reinforcement learning to train control strategies in simulation, which can then be applied to the real world. Thus, as with MPC, a model is also necessary in order to apply deep reinforcement learning in a model free and online manner (which is mostly used today [34]). Nevertheless, the intertwining with the model is lower as with MPC, as the simulation and therefore the model is only used in training. This results in lower requirements for a model with DRL than with MPC, which brings two main advantages:

1. More complex and thus more realistic building models can be used with DRL.

2. As shown by [70, 71], building models from widely-adopted building simulation programs like EnergyPlus[7] can be used for DRL directly.

Unfortunately, there are only a few works, also in other research areas then heating control, that perform a comparison between MPC and DRL [72].

The rest of this section is organized as follows: The works listed in section 4.1 focus primarily on the usage of deep reinforcement learning to find optimal control mechanisms for *heat pumps*. A heat pump belongs to the superordinate category of heating, ventilation, and air conditioning (HVAC) systems. The micro control mechanisms of different HVAC systems may differ, but the research work on control methods is related and is therefore listed in section 4.2. Additionally, works using deep reinforcement learning

[7] `https://energyplus.net/`

for the coordination of multiple HVAC subsystems or even the coordination between different systems in a building will be explained in section 4.3.

A summary of related work is given by table 1. For sake of completeness, table 1 also lists older works that do not use neural networks as function approximators for reinforcement learning. However, due to the breakthrough made by the usage of neural networks in the context of deep reinforcement learning, these works can be considered outdated and are not described in more detail.

Table 1: Summary of Related Work

Reference	Algorithm	DRL	DR	Multiple Components	Heating System	Year
[73]	Q-learning [19]	No	No	No	HVAC	2014
[74]	Fitted Q-iteration [75]	No	Yes	No	Heat Pump	2015
[76]	Fitted Q-iteration [75]	No	Yes	No	Heat Pump	2015
[77]	Q-learning [19]	No	No	No	HVAC	2015
[70]	Deep Q-learning [21]	Yes	Yes	No	HVAC	2017
[71]	Actor Critic	Yes	No	No	HVAC	2017
[78]	NFQ [79]	Yes	Yes	No	Heat Pump	2017
[80]	NFQ [79]	Yes	Yes	No	Heat Pump	2018
[81]	A3C [52]	Yes	No	No	HVAC	2018
[82]	NFQ [79]	Yes	Yes	No	Heat Pump	2018
[83]	Deep Q-learning [21]	Yes	Yes	No	Heat Pump	2018
[84]	BDQN [55]	Yes	No	Yes	HVAC	2019
[85]	DDPG [53]	Yes	No	No	HVAC	2019
[86]	PPO [29]	Yes	No	No	HVAC	2019
[87]	DDPG [53]	Yes	No	Yes	HVAC	2020
[88]	Deep Q-learning [21]	Yes	No	Yes	HVAC	2020
[89]	MAAC [54]	Yes	Yes	No	HVAC	2020
[90]	Deep Q-learning [21]	Yes	Yes	No	HVAC	2020
[91]	DDPG [53]	Yes	Yes	No	HVAC	2021
[92]	Deep Q-learning [21]	Yes	No	No	Heat Pump	2021
[93]	PPO [29]	Yes	No	No	Heat Pump	2021

Note: Summary of related works where (deep) reinforcement learning was applied to HVAC control. The column DRL indicates the works that used neural networks, thus *deep* reinforcement learning. The column DR specifies the works, which have taken the demand response aspect into account by including price signals. The column Multiple Components shows works, where more than one system was controlled. The column Heating System specifies works which used a heat pump or any other HVAC system.

4.1 Deep Reinforcement Learning Applied to Heat Pump Control

In this section, the related works, which applied deep reinforcement learning to heat pump control are first described per se. Subsequently, the related works are summarized and differentiated from the present work.

In a previous master thesis at Fraunhofer ISE, the control of a heat pump by deep reinforcement learning was treated as a subtopic [94]. Anyhow, the focus of their thesis was on imitation learning and a working deep reinforcement learning implementation could not be established. In addition, this work can be distinguished as follows: (1) They used another simulation framework, which included a forecast of the heat demand by the inhabitants which was to be fulfilled by controlling the heat pump. (2) The focus of their work was more on implementing deep reinforcement learning from scratch, which could not be done fully functional. (3) Their focus was on the scenario with variable electricity prices and not on the scenario of efficient operation per se.

Peirelinck et al. [80] used neural fitted Q-iteration (NFQ) [79][8]. They were able to learn strategies that could indirectly control a heat pump, by using a binary action state that either sets a low or high target room temperature. Therefore, compared to the work at hand, the problem can be considered simplified, since it has targeted optimization only in terms of minimizing energy consumption and not in terms of comfort deviations. Still they could report promising cost savings by including energy prices in their problem statement. The works of [74, 76, 78, 82] are related to [80] as the authors overlap and all works apply fitted Q-iteration to a similar simulation framework.

Heidari et al. [92] applied deep Q-learning [21] to control a simulated heat pump for hot drinking water usage. While there are also requirements for comfort in this problem statement, their focus can be considered different as their goal was to learn control strategies that are hygiene-aware. They use a binary action space to control the heat pump. By including occupants' warm water usage behaviour, Heidari et al. could enable their agent to learn control strategies that consider occupants' behaviour. They could report savings in energy usage by 24,5% compared to a rule-based controller which they used as a baseline.

In [93], Ghane et al. applied deep reinforcement learning by using PPO [29] to control a simulated heat pump. Their problem statement differs

[8] Similar to deep Q-learning, the central idea of NFQ is to learn a neural network which represents a Q-function [79].

as their goal is to find optimal control strategies for a central heat pump, which provides heat to a heating network consisting of multiple houses. Similar to the work at hand their goal is to minimize electricity usage while meeting the comfort requirements of the occupants. They compared their results against a heating curve based controller. They could report 16.03% of reduction in energy demand compared to their baseline. They used both discrete and continuous actions to validate their solution.

In [83], Nagy et al. applied deep Q-learning [21] to a simulated air source heat pump. By defining 6 discrete control actions, they managed to learn control strategies that stays in a comfort temperature range while reducing run cost based on an electricity price. They have used a dual price signal with two different fixed prices during the day and night. A baseline comparison using rule based and MPC-based control strategies was conducted. They reported savings of 5-10% compared to a rule based controller. Like in the work at hand, MPC was used as upper performance limit, as it used the simulation as a model for planning and had thus full knowledge of the target environment. The work from Nagy et al. [83] can be considered as the most similar to the work at hand.

The works listed in this section can be summarized and differentiated from the presented work, as:

1. Like the work at hand, all the works presented in this section, with the exception of [74, 76] which used a randomized tree ensemble in combination of reinforcement learning, used *deep* reinforcement learning to control heat pumps.

2. Like the work at hand, none of the works applied heat pump control outside of a simulation to the real world.

3. The work at hand treats heat pump control as a continuous problem statement, as in reality heat pump control is continuous and accuracies are lost due to discretization. All works except [93] treated heat pump control as discrete or even binarized problem.

4. The work at hand conducted an extensive baseline comparison against control strategies by the heat curve and MPC. While most of the works listed in this section conducted a baseline comparison against a heat curve or an arbitrary rule based strategy, only Nagy et al. [83] conducted a baseline comparison including MPC. Also in other research fields, only few works compare MPC and DRL [72].

5. None of the works from this section could report learned heat pump control strategies that exploit heat storage capabilities of a building in order to operate the heat pump more efficiently. As will be described later in section 8.1.2, learning of control strategies that do exactly that could be reported in the work at hand.

4.2 Deep Reinforcement Learning Applied to HVAC Control

In this section, the related works which applied deep reinforcement learning to other HVAC systems than heat pumps are described.

In [70] they used Deep Q-learning to learn control strategies for an HVAC system in a simulated environment. Through a discrete action space, they could learn a control strategy that was able to hold the indoor room temperature in the desired range. By the inclusion of varying electricity price data, they could report savings of 20%-70%, compared to a rule based controller. In [71] they applied a policy gradient method by implementing an actor-critic architecture. This allowed them to control a HVAC cooling system in simulation. Even though they used a policy gradient method that supports continuous actions, they learned a control strategy by using 26 discrete actions, which represent the target indoor temperature. In [81] they learn control strategies for a HVAC system in a simulated office building using the asynchronous advantage actor-critic algorithm [52]. In their work, they focused on the description of the process of creating and calibrating the model used for training the reinforcement learning agent. They also describe the process of deployment of the agent to the real building. Anyhow, they have not yet applied their learned agent to the real building, which they have planned for future work.

Some other works have already addressed the problem of transferring the reinforcement learning agent which was trained in simulation to a real environment [86,90]. In [86] they pretrained their deep reinforcement learning agent using imitation learning on data that they collected from an MPC controlled HVAC system. After pretraining they deployed their agent to simulation and to a real-world conference room. They could report energy savings compared to existing control strategies which were based on a fixed schedule. In [90] they discussed challenges when transferring a learned agent into the real world in detail. Additionally, they propose a solution of how to mitigate these problems. They validated their concepts on a real building, where they reported cost reductions of 21%.

The works listed in this section can be differentiated from the presented work, as they applied deep reinforcement learning to HVAC systems other

than heat pumps. As mentioned in section 3.1, the efficiency of an air source heat pump, which is considered in this work, depends on factors such as the outdoor air temperature. Therefore, the efficient control strategies for heat pumps might differ from those for other HVAC systems. The works of [86, 90] stand out as they applied their solution to the real world.

4.3 Deep Reinforcement Learning Applied to Coordinative Control

This section presents works that have used deep reinforcement learning to coordinate control of HVAC systems with other building systems. These works differ because their focus is on the coordination of multiple subsystems not the optimization of control for a single heating system. Nevertheless, some works are presented to give a more complete picture of the works that applied deep reinforcement learning to heating systems.

In [84] they took a holistic approach to smart building control. By using DRL with a branching duelling Q-network (BDQN) [55], they learned a control strategy that considers multiple subsystems in a smart building, including HVAC, lighting, blinds and window systems. By the joint control of the named subsystems, they were able to report energy savings while maintaining comfort in a simulated environment. In [89] they applied multi agent deep reinforcement learning by using Actor-Attention-Critic [54]. They coordinated the damper positions of the air handling unit and air supply rate in each temperature zone of the building. By doing so, they were able to minimize the operation cost of an HVAC system in a commercial building with multiple temperature zones while maintaining thermal and air quality comfort. In [87] they use deep reinforcement learning by using deep deterministic policy gradients (DDPG) to learn a continuous control policy for an HVAC system that minimizes electricity usage. The policy controls the temperature and humidity set points of the HVAC system. Based on the set points, the simulated HVAC system derives the control actions to adjust the temperature and humidity in the building. The key point of their work is that the occupants' comfort is estimated based on room temperature and humidity and not assumed to be static as in other works. A more detailed and complete overview of works in this area is provided by Yu et al. [50].

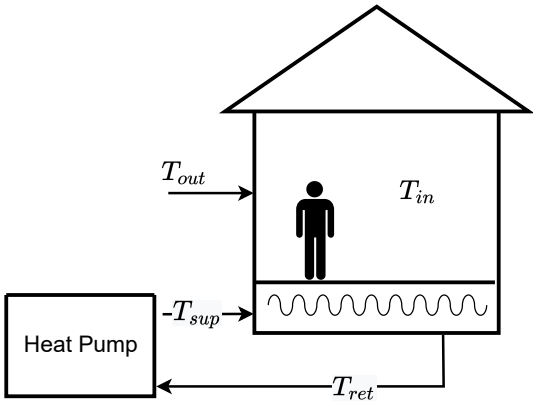

Figure 5: Overview of the simulation framework

5 Research Environment and Task Definition

As mentioned earlier, this work focuses on applying DRL to learn efficient control strategies for heat pump control in *simulation*. Therefore, in section 5.1 the simulation used in this work is described in detail. Compared to reality, the use of the simulation results in simplifications and assumptions, which are described in section 5.2. Finally, the research questions are defined in more detail in section 5.3.

5.1 Simulation

The simulation framework used in this work was provided by the Fraunhofer ISE. As illustrated in figure 5, the simulation environment mimics a simplified building with a single room, which is heated by a floor heating system supplied with heat which is generated by a heat pump. The framework can be used to simulate a wide variety of buildings by setting different parameters. Those parameters include the building floor area, room height, as well as the position and size of windows. Additionally, the heat capacity, which defines the amount of heat that can be stored by the building and the building's heat loss through windows or walls can be configured. The framework contains pre-defined parameter sets of existing buildings. The buildings used to evaluate the functionality of the proposed methods will be described in section 8.1.

The core functionality of the simulation framework is shown in figures 5 and 6. Using the simulation, one can model the effect of the outside temperature T_{out} and heat pump supply temperature T_{sup}, which is the

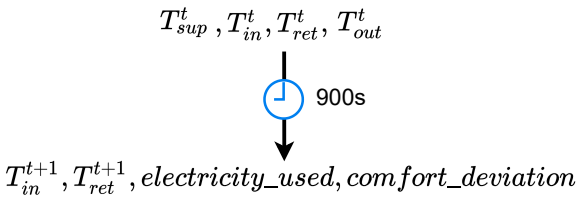

Figure 6: Variables required and returned by a simulation step

temperature of the water heated by the heat pump, on the building indoor temperature T_{in}. A simulation step is modelled as if 900 seconds would pass in real-time. This means that the simulation calculates the indoor temperature as if the outside temperature and supply temperature are kept constant for 900s. Besides this basic functionality of calculating the new indoor temperature T^{t+1}_{in}, the following values are additionally calculated and returned by performing one simulation step (see figure 6):

Return Temperature The *return temperature* T_{ret} of the water after performing a simulation step. In contrast to the supply temperature T_{sup} which describes the temperature of the water entering the floor heating system, the return temperature describes the temperature of the water coming back from the floor heating system which is to be heated again by the heat pump. The return temperature returned at simulation step t serves as input for the upcoming simulation step $t + 1$.

Electricity Used The amount of electrical energy which was consumed by the heat pump in order to heat the water from the return T_{ret} to the supply temperature T_{sup} during the simulation step. This value is to be minimized by an efficient control strategy.

Comfort Deviation The comfort temperature range is defined between 21°C and 25°C. If the indoor temperature is outside of this interval during a simulation step, the difference will be reported as comfort deviation in °C. This value is to be minimized by an efficient control strategy.

Figure 6 summarizes the values needed as inputs as well as the resulting values of a single simulation step. By chaining multiple subsequent simulation steps, one can model heat pump operation over a period of time. The value which needs to be controlled at every simulation step in order to operate the heat pump is the supply temperature T_{sup}. The other input variables

of the simulation step (see figure 6) can be treated as given, as the outdoor temperature T_{out} can be taken from a weather profile[9] and the values of the indoor temperature T_{in} and return temperature T_{ret} are calculated by a previous time step by the simulation itself. However, a requirement by Fraunhofer ISE was to control the heat pump by controlling its amount of thermal power $\dot{Q}_{hp}$ which should be supplied during a time step, instead of the resulting supply temperature T_{sup} which is required by the simulation framework. Practically, both values are in a direct relationship and can therefore be easily converted. This relationship is defined as

$$\dot{Q}_{hp} = (T_{sup} - T_{ret}) * \dot{m} * c. \tag{22}$$

This makes it possible to use $\dot{Q}_{hp}$ as the control action and calculate the resulting supply temperature required by the simulation according to (22). The parameter $\dot{m}$ defines the mass flow which will be further described in 5.2 and c defines the specific heat capacity of the water flowing through the floor heating system. Both values are static over the course of the simulation.

Finally, it should be noted that the simulation is implemented as a Python function and could be therefore easily integrated into the implementation of the proposed solution.

5.2 Simplifications and Assumptions

Compared to reality, the following simplifications and assumptions are made in simulation:

Internal Gains Internal gains refer to the heat produced by inhabitants, pets or any devices, except the heating, that produce heat. This includes for example electrical devices, artificial lightning or body heat [95]. In reality, internal gains depend on behaviour of inhabitants and other factors. In the simulation, the internal gains are assumed to be static at $5W$ per square meter of heated living space, which is according to DIN4108-6 [95].

Solar Gains Solar gains represent heat gains from solar irradiation through windows. In simulation, the solar gains are not considered and therefore assumed to be $0W$. It must be noted, that the selection of the data used in the work at hand and the implementation of the presented method can be easily extended for the inclusion of the solar gains.

[9] In the work at hand, the outdoor temperature is taken from existing weather profiles as it is described in section 6.1

Single Temperature Zone The simulation framework only supports building models with a single temperature zone which assumes equal temperatures in the whole building.

Mass Flow and Thermostats The mass flow specifies the amount of heated water which flows through the heating circle per unit of time. In reality, thermostats regulate the mass flow individually per room and therefore control its temperature. Due to the fact that only a single temperature zone is considered by the simulation framework, a static mass flow without the use of any thermostats for the whole building is assumed. This means the indoor temperature is not regulated by any thermostats but solely by the supply temperature (which itself is regulated by controlling the thermal power).

Random Actions Temperature drops caused by opening windows or doors are not taken into account.

Perfect Forecast It is assumed that a perfect forecast of the outside temperature is available. However, in an experiment, the proposed solution is tested on noisy forecasts.

5.3 Research Task Definition

Although the simplifications and assumptions listed in 5.2 represent a gap from reality, they make it possible to concentrate on the control strategies. In the context of this work, a control strategy is one that controls the heat pump by selecting the thermal power $\dot{Q}_{hp}$ to be generated in every simulation step in order to control the indoor temperature. Hereby an *efficient control strategy* is one that minimizes electrical energy usage and comfort deviations at the same time. Finding efficient control strategies presents a challenging task, as not only the control action to be selected at every simulation step, but also the outside temperature has an impact on the indoor temperature. Furthermore, the selected control actions have an effect only with a delay, since the heat transfer does not take place instantaneously.

As described in 3.2.2, the usage of MPC in order to apply efficient heat pump control has shown to be challenging in practice and the usage of the heating curve offers room for improvement. Deep reinforcement learning has shown breakthrough successes in many application areas. Therefore, the core research task of this work consists of designing, implementing, and evaluating efficient control strategies for heat pumps by using deep

reinforcement learning. Based on this core research task, the following research questions are to be addressed:

- First, can deep reinforcement learning be applied in the given simulation framework, with the aim to learn efficient control strategies that minimize energy usage and comfort deviations at the same time?
- Second, in this setup, MPC uses the simulation framework itself as the model to plan and execute the control actions and is thereby given full transparency. This leads to MPC finding the optimal solutions, and thus can be considered the gold standard in this problem setup. However, the use of MPC in practice has some disadvantages which were discussed in section 3.2.2. The research question to be answered is if MPC-like performance can be achieved by deep reinforcement learning without its disadvantages. Additionally, control by the heating curve is to be included for this comparison, since this is the most widespread control strategy in the field today [4].
- Third, can the proposed solution be extended to a demand response scenario, where control happens with respect to a varying price signal?

6 Concept

This chapter describes the technical concept which was applied to implement deep reinforcement learning for efficient heat pump control, which is the main research goal of this thesis. The conceptual ideas surrounding the evaluation of this concept in order to answer the research questions formulated in section 5.3 will be described in chapter 8.

The main approach of the presented concept is to use the simulation framework described in section 5.1 as environment for reinforcement learning. On the agent side, a deep reinforcement learning framework is used to support design and implementation. The interaction of these two components makes it possible for the agent to learn heat pump control strategies in a trial and error manner, which is in line with the presented approach of reinforcement learning in general which was presented in section 2.

The rest of this chapter describes this main approach in more detail and is structured as follows: Section 6.1 describes the data which was used by the environment in order to simulate different weather profiles. Section 6.2 describes how the simulation framework is wrapped as environment for reinforcement learning. Section 6.3 describes the concept of the deep reinforcement learning agent. Finally, section 6.4 describes how the agent interacts with the environment to learn efficient control strategies.

6.1 Outside Air Temperature Data

Weather profiles containing data of the outside air temperature are required in order to simulate the heating process of a building as described in 5.1. Therefore, weather data from the photovoltaic geographical information system of the European Commission[10] was obtained. The data contains information about the weather in Freiburg, Germany between 2010 and 2016. The resolution of the weather data is hourly. Besides information about the outside air temperature which is required for the work at hand, the obtained data also includes information about the solar radiation and wind. Even though those factors are excluded from this work, they may be of interest in future works and are thus already made available.

It is important to note, that like in [94], data during months where heating is usually not necessary are excluded from the datasets. This leaves data from January to March and October to December of each year. Figure 7 illustrates the outside air temperature data used in this work.

[10] `https://re.jrc.ec.europa.eu/`

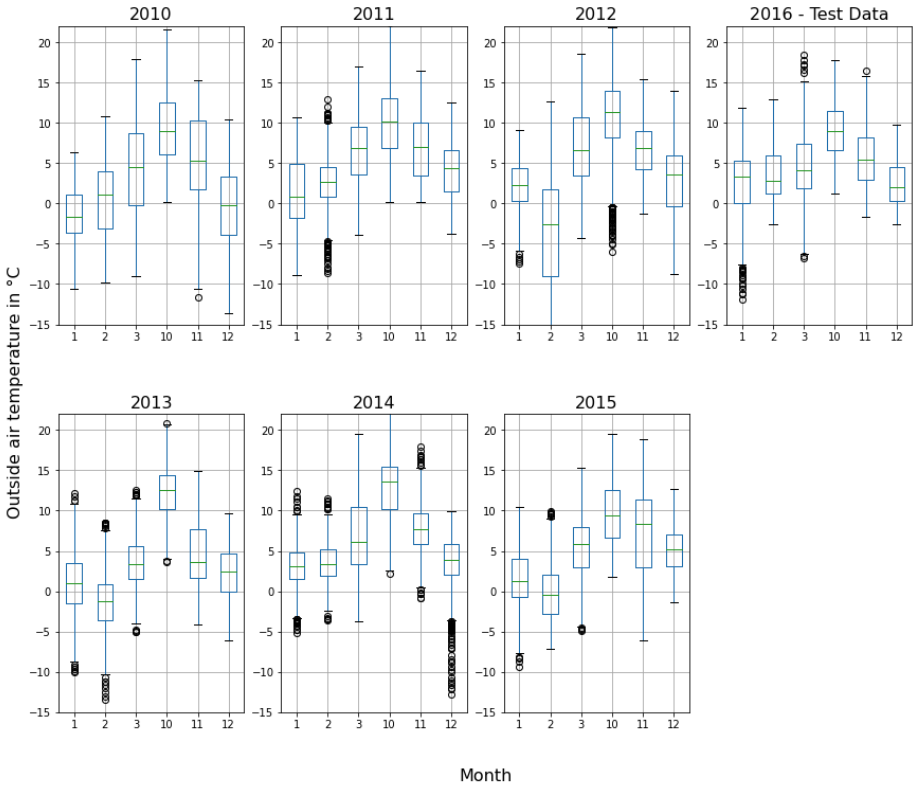

Figure 7: Distribution of the outside air temperature data used in this work. The boxes indicate the 0.25 and 0.75 quantiles of the data per month. The median of the data is shown by the green line in the box. The points past the whiskers are outliers. A detailed definition of the position of the whiskers and outliers can be found in the pandas documentation [11]

Besides the use of weather profiles containing the outside air temperature which is required by the simulation, no further external data is required as the reinforcement learning agent learns its control strategies by interaction with the simulation framework.

6.1.1 Data Preprocessing

The outside air temperature data obtained from the photovoltaic geographical information system of the European Commission was available at a resolution

[11] `https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.boxplot.html`

of one hour. However, as described in section 5.1, one step of the simulation framework corresponds to 900 seconds of real-time. Therefore, upsampling of the outside air temperature data was performed to gain information at a frequency of 900 seconds. Linear interpolation was used in order to fill the values between full hours. No further preprocessing was applied to the outside air temperature data.

6.1.2 Data Split

It is important to evaluate the presented method on outside air temperature profiles which were not used in training. This is necessary to evaluate how well the method generalizes to unseen weather profiles. Therefore, the outside air temperature data was split into different data sets. The *training* data set is used by the simulation while training the reinforcement learning agent. It contains weather data of the years 2010 to 2015. The *test* data is used solely for the evaluation of the presented method and is not used during training, nor considered for any design decisions or hyperparameter tuning. In contains weather data of the year 2016. Additionally, a *validation* data set was used in order to monitor the training process as it will be described in section 6.4.

6.2 Definition of the Environment

The definition of the environment is central to deep reinforcement learning because the agent and the resulting control strategies are learned solely through interaction with the environment. Therefore, the definition of the environment and its MDP structure, especially the definition of the reward function are important for the functionality of reinforcement learning. A more detailed description of the fundamentals of environments in reinforcement learning in general, can be found in section 2.1.1.

The basic idea of the environment used in this work is to wrap the simulation framework which is described in 5.1. This basic idea is illustrated in figure 8. One time step of the environment corresponds to one step of the simulation framework (see figure 6). The environment passes the outside temperature of the current time step to the simulation framework in order to simulate the influence of outdoor weather profiles. Above all, the environment defines the state, reward, and actions which are used for the interaction with the reinforcement learning agent. In the following those components are explained in more detail:

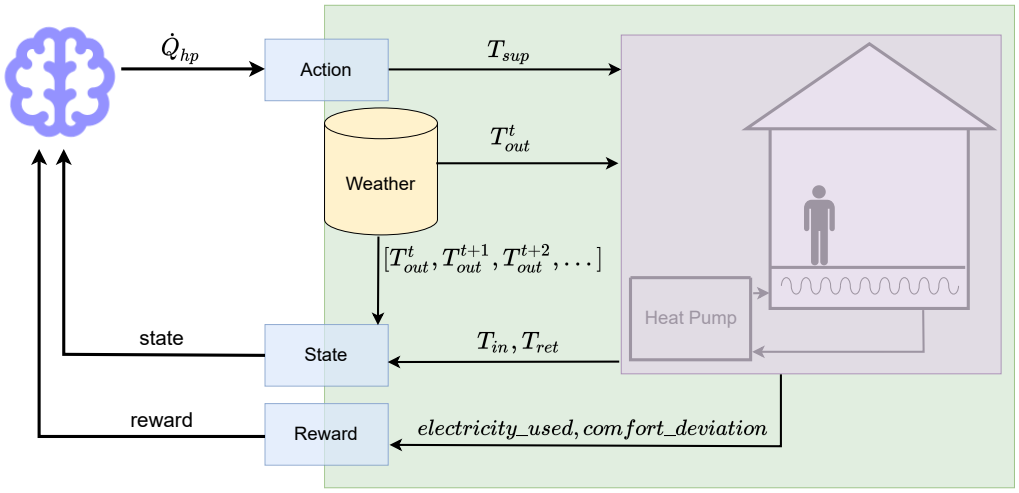

Figure 8: Overview of how the reinforcement learning environment wraps the simulation framework and supplies the outdoor air temperature data. Values returned by the simulation framework build the state and reward of the reinforcement learning environment. The purple area represents the simulation framework. The green area represents the environment for reinforcement learning which wraps the simulation framework. The interfaces for interaction with the agent (state, reward and action) are denoted in blue.

6.2.1 State

Each state, which describes the environment at a specific time step, is composed of a *system state* and a *forecast state*. Both parts are simply put together to form the environment state, which looks like

$[T_{in}^t, T_{ret}^t, T_{out}^t, T_{out}^{t+1}, T_{out}^{t+2}, ..., T_{out}^{t+n}]$, where n is the number of forecast steps included[12]. The system state contains information about the current indoor temperature T_{in} and the current return temperature T_{ret}. Both values are obtained directly by the simulation framework. The forecast state contains information about the outside temperature during the current and upcoming time steps. The length of the forecast included in the state is a hyperparameter, which was set empirically by experiments. The reader is again referred to figure 8, which illustrates the composition of the state. It is important to mention again, that a perfect weather forecast is assumed and therefore the agent is provided with the values of the outdoor temperatures of the upcoming time steps. By designing the state as described, the Markov property is followed, which is crucial for the functionality of reinforcement learning. This allows the agent to decide on actions based on a single state. The idea of merging system state information and forecast state information into a single state which can be used for DRL is in line with the related works from section 4.1 and 4.2, which have mostly taken a similar approach.

It is important to note that the individual features contained in the state are on different scales. This faces a problem, as the state is used as input for the DRL agent's neural network. Therefore, standardization of the state is required. This is done by calculating the mean μ and standard deviation σ of each of the features contained in the state individually and by applying the following formula:

$$Z = \frac{X - \mu_p}{\sigma_p}. \tag{23}$$

This results in a state, which contains features that are all on the same scale with $\mu = 0$ and $\sigma = 1$, which improves stability of training the neural network. Note that normally a data set is required in order to calculate μ_p and σ_p for every feature p contained in the state. This could be done for features concerning the current and future outside air temperature as this data is available beforehand. Anyhow, the distribution of T_{in} and T_{ret} is not known beforehand, and thus their μ and σ cannot be calculated. Therefore, μ_p and σ_p of all features are estimated by updating their values continuously during training. This concept is called moving or running average normalization

[12] As later shown in section 8.1, the number of forecast steps included in the step depends on the building to be heated.

and was applied by following the implementation of the deep reinforcement learning framework Stable Baselines3 which was used in this work [96].

In order to test the demand response capability of the proposed solution, an experiment was performed where a forecast about a varying electricity price was included in the state. This approach will be described in more detail in section 8.3.

6.2.2 Action

Based on the state, the reinforcement learning agent executes an action with the goal to maximize the return. The action represents the thermal power $\dot{Q}_{hp}$ which should be generated by the heat pump during the upcoming time step. It represents the control variable that is used to control the heat pump. The action space is defined to be continuous and thereby applies $\dot{Q}_{hp} \in [0, 12000W]$. The upper limit of the action space interval represents the maximum power of thermal heat which can be served by the heat pump which is defined by the simulation framework. As proposed by [97], the action space is internally rescaled, so the actions taken by the agent lie in the symmetric interval $[-1, 1]$. It must be noted, that the simulation framework works with the supply temperature T_{flow} as the control variable for the heat pump. Therefore, the control variable is transformed by equation (22) through the environment.

6.2.3 Reward

The reward encodes the goal of the research task at hand. Therefore, the reward must balance the minimization of electricity usage and comfort deviations at the same time. A reward is calculated at every time step according to the following formula:

$$r_t = -1 * (\beta * electricity\_used_t + comfort\_deviation_t). \tag{24}$$

Since the targets of minimizing electricity usage and minimizing comfort deviations are in conflict with each other, they are balanced on the basis of a trade-off parameter β. A high β will lead to a reward definition that focuses more on the minimization of the electricity, whereas a low β focusses on the minimization of comfort deviations. The value for β was empirically set to 0.45. Since the goal in reinforcement learning is always to *maximize* the reward, the reward given is multiplied by the factor -1. Intuitively, the reinforcement learning agent is punished for every Wh of electrical energy used and every K of comfort deviation. Note that most of the related works

listed in section 4.1 and 4.2 balance the minimization of comfort and some kind of cost and therefore have a related reward definition.

6.3 Deep Reinforcement Learning Agent

As described in 2.3, a number of algorithms exist which define different approaches of how DRL can be applied. In this work, PPO is used because of the following reasons:

1. PPO supports continuous action spaces.
2. Recent works were able to apply deep reinforcement learning to complex problems by using PPO [30, 98].
3. These works have reported stability in regards to the hyperparameters used.

Especially the last point was a key requirement. Since the performance of DRL often depends on different hyperparemeter settings [99, 100], an algorithm should be used that has been reported stable in this respect.

Implementing PPO from scratch would be error-prone and time consuming, but most importantly would lead to results that are difficult to interpret and reproduce by others. This is due to the fact that the functionality of reinforcement learning algorithms depends on the implementation details [99–101]. Therefore, the deep reinforcement learning library Stable Baselines3 [96] was used to apply PPO in this work. The library has abstracted away the implementation of the deep reinforcement learning agent and the calculation and application of the gradients used for training. Most design decisions related to PPO were taken away by using PPO mostly as it was defined in the default settings in Stable Baselines3. Settings that differ from the default will be explained in section 7.2. Still, the main structure of PPO as it is defined by Stable Baselines3 is described below, with the goal to provide an overview of how PPO is used in the work at hand:

As PPO is a policy gradient method, the deep reinforcement learning agent is represented by a neural network that serves as a policy to estimate optimal actions based on a state. The policy used in this work is represented by a fully connected neural network with two hidden layers containing 64 neurons each[13].

Since PPO is based on the actor-critic principle, an additional neural network, which represents the critic is used. As explained in section 2.2.2,

[13]The usage of alternative neural network types like convolutional neural networks (CNN) or recurrent neural networks (RNN) for the policy is left open for future work.

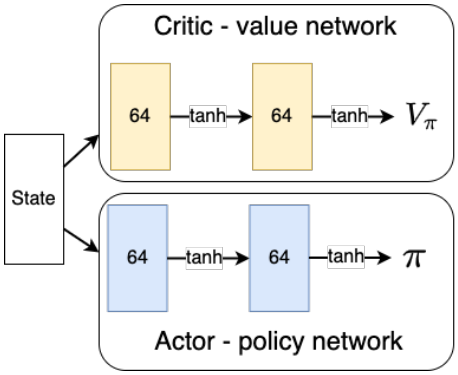

Figure 9: The architectures of the neural networks. The actor-critic network presentation in this image is based on [102].

this is used to estimate the advantage, which provides a baseline when estimating the gradients during training. The advantage estimation is based on estimations of the value function, thus the additional neural network is referred to as *value network*. Figure 9 shows the architectures of both neural networks, which represent the deep reinforcement learning agent used in this work. Both, the value and the policy network have the same architecture. In addition to the input and output layers, both networks have two fully connected hidden layers with 64 neurons each. This network can be considered small, compared to most neural networks in other areas like computer vision or natural language processing. The hyperbolic tangent (tanh) activation function is applied after each fully connected layer.

6.4 Training

During training, the weights of the policy and value network are updated in order to select actions that maximize the return. This process is completely carried out in simulation, by interacting with the environment defined in section 6.2. An intuition of the gradient computation which is used to update the parameters of the neural network during training is given in section 2.2.2. Anyhow, these low-level training details are abstracted away by using the deep reinforcement learning library Stable Baselines3. Although thereby training is implemented at an algorithmic low-level perspective, this section describes the overall high-level concept of the training process.

6.4.1 General Procedure

The training process was designed to be done in multiple episodes. Each episode contains 2880 interactions between the agent and the environment. This corresponds to approximately one month, as one time step represents 900 seconds in real time. The training is finished after 348 episodes were executed. In total, the agent is therefore trained over 1,000,000 time steps.

At the beginning of each episode, the environment is reset by setting it to an initial state. This involves resetting the environment indoor temperature to 21°C and the return temperature to 23°C. In addition, a new weather profile containing information about the outdoor temperatures is chosen at the beginning of each episode. This is done by drawing one month of a continuous weather profile at random from the pool of training data. This is an important step, as it prevents the learned strategies from overfitting to certain weather profiles and therefore enables the agent to generalize to different weather scenarios.

An alternative approach would have been to use one long continuous training episode containing all the available weather data from the train data set. Anyhow, the process of regularly resetting the environment at the beginning of each episode into a sane state introduced stability in the training process.

6.4.2 Periodical Validation

After every seventh episode of training, the DRL agent is being *validated*. This includes executing the current state of the agent for three episodes. Here, too, one episode contains 2880 time steps and therefore 2880 actions taken by the agent, which corresponds to approximately one month of heat pump control. The weather data used during those episodes are contained in the validation data set. During the validation, the training is paused, meaning no updates to the parameters of the neural networks are taking place. The results of the validation are stored and serve as statistics to judge about the quality of the training process afterwards.

Additionally, the regular validation during training is used to determine the best performing agent during training. This is done by storing the current parameters of the neural network if the agent performed better than all previous agents of this training run. The mean reward of the validation episodes is taken as the criterion for determining the best model. This step is necessary as the reward and thus the performance of the model does not always improve monotonically during training. Or to put it in other words,

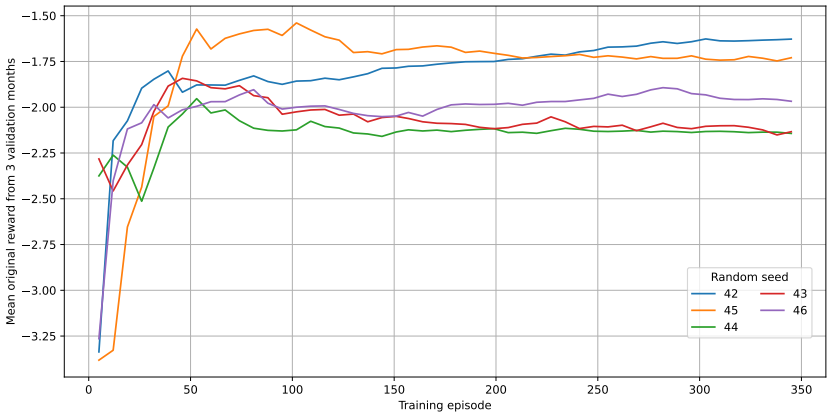

Figure 10: Exemplary progression of mean rewards over the training period of 348 episodes. Although the five training runs differ only in their preset random seed, their progression is different. The reward does not increase monotonically over the course of training. The best reward and thus model in the training run with seed 45 was obtained already at approximately episode 100.

the model obtained after the last training step is not necessarily the best. This is illustrated in figure 10, which shows an exemplary training run.

The decision on the validation frequency and the number of episodes used when validating trades off model performance and training time. The more frequent the validations take place and the more data is used in the validation, the higher the probability to obtain the best model which occurred any time during training. On the other hand, validating too often and using too much validation data causes impracticable training times. By conducting extensive experiments it turned out that validating every seventh train episode for three episodes showed to be a good trade-off to achieve manageable training times while having a good chance to obtain a good model.

6.4.3 Multiple Seeds

Lastly, in [100] it was shown that the results of deep reinforcement learning often depend on the random initialization of the parameters in the neural net. Therefore, it is necessary to run an experiment multiple times with different random seeds which cause different random initializations of the parameters. A single training run has only little informative value about the

performance of the method due to the potential variability in performance. Or in other words, good results can only have occurred by chance. Figure 10 illustrates this issue. Measured by the mean reward, the results of the runs differ largely, which is solely caused by different preset random seeds. Please note that this plot shows an extreme case, which was discarded afterwards. To obtain more meaningful and interpretable results, each training run is executed independently with five different pre-set random seeds.

The presented concept of how training is executed is summarized by algorithm 2.

Algorithm 2 Training

```
for seed in [42,43,44,45,46] do
    set random seed
    initialize PPO agent
    initialize environment
    while n_episode < 348 do
        T_amb ← 21
        T_ret ← 23
        get weather data from random start date from train dataset
        interact and train one episode
        n_episode += 1
        if n_episode % 7 == 0 then
            evaluate PPO agent
            if best model so far then
                save PPO agent
                save environment statistics
            end if
        end if
    end while
    save statistics
end for
```

7 Implementation Details

The purpose of this chapter is to give an overview of the most important implementation details of the work at hand. Note that the code used in this work is publicly available on GitHub[14].

7.1 Environment

The environment as described in 6.2 was implemented with the help of OpenAI Gym[15], which serves as a toolkit for implementing environments for reinforcement learning. It provides an interface that defines the interaction between agent and environment [103]. Therefore, OpenAI Gym provides a Python interface, which requires the implementation of two methods:

`step()` The `step()` method is the core element of the environments implementation. It defines one time step of interaction between the agent and the environment by taking an `action` as argument and by returning the tuple `(state, reward, done, info)` [103]. As the tuples variable names suggest, the `state` defines the new environments state and `reward` defines the reward given to the agent. Both, the state and the reward returned to the agent were implemented according to the concepts presented in section 6.2. The returned variable `done` is a `boolean` and defines, weather the episode ended with the last interaction or not. As described in 6.4.1, one episode ends when 2880 interactions took place, which corresponds to approximately one month in real time. The variable `info` was used for reporting and debugging purposes and contained information about the heat pump operation. This, for example, has enabled the creation of plots of the control strategies which will be presented in chapter 8.

`reset()` The `reset()` method resets the environment to an initial state. As described in 6.4.1, this includes resetting the current indoor and return temperature, as well as setting a new start date for the outdoor temperature profile. The `reset()` method must be called before the first time step can be executed. Additionally, it is called at the end of an episode.

Many deep reinforcement learning libraries, like Stable Baselines3 which was used in this work, support environments that are implemented according

[14] `https://github.com/tobirohrer/reinforcement-learning-heat-pump`
[15] https://gym.openai.com/

to the OpenAI Gym interface. This provides modularity to the overall architecture, as the reinforcement learning agent can be easily exchanged[16].

7.2 Agent

As already mentioned in section 6.3, the use of deep reinforcement learning libraries offers advantages compared to implementations from scratch. In this work, the deep reinforcement learning library Stable Baselines3 [96] was used. It provides open-source implementations of state-of-the-art deep reinforcement learning algorithms [96]. The algorithms are implemented with Python using the machine learning framework PyTorch [104]. The details on the implementation of PPO in Stable Baselines3 are summarized by [105]. As the performance and thus the reproducibility of results in deep reinforcement learning depends on implementation details [100, 101], it must be noted that the PPO implementation from the official git repository[17] with commit `a6f5049` was used. The following describes the hyperparameters that were set differently from the default settings of the implementation just described:

Learning Rate As shown in (16), the size of a gradient update during training depends on the learning rate α. Setting an appropriate learning rate is crucial for the functionality of deep reinforcement learning. Like in supervised deep learning, a learning rate set too high might cause the learning process to fluctuate around an optimum. A learning rate set too low requires more training iterations and thus more wall clock time for convergence. The learning rate for the work at hand was determined empirically by running multiple training runs and set to $0.5 * 10^{-4}$. The following two criteria were used to select the learning rate: First, the maximum reward that is achieved on the validation data during training. Second, the *evolution* of the mean reward on the validation data during training. An example of the evolution of the mean reward on validation data during training is shown in figure 11. The hypothesis is, that a learning rate set too low would cause a linear shaped evolution of the mean reward. A learning rate set too high would cause fluctuations in the evolution, as the training would overshoot the optimum frequently. This hypothesis leans on

[16] In early phases of the work at hand, experiments were conducted with the deep reinforcement learning framework ChainerRL. Thanks to the modular architecture, it was possible to switch to Stable Baselines3 with little effort.

[17] https://github.com/DLR-RM/stable-baselines3

the choice of the learning rate for supervised learning, which is among other criteria chosen on the same thoughts but looking at the evolution of the loss instead of the mean reward during training [106].

Discount Factor As described in section 2.1.1, the discount factor γ weights the influence of time-distant rewards on the return. The set value of γ will be discussed in section 8.1.2, as the experiments have shown that the learned strategy and therefore a good choice of γ depends on the building.

A complete list of the hyperparameters used for PPO can be found in appendix A.

Table 2: Summary of Simulated Buildings

	Old	Efficient	Efficient Enhanced
Floor Size in m^2	136	393	393
Heat Capacity in $Wh/m^2/K$	45	65.9	65.9
Transmission Losses in W/K	396	281.7	281.7
Energy-Efficiency Class[19]	F	A	A
Year of Construction	1984	2020	2020
Heat Storage Capabilities	No	No	Yes

Note: The Energy-efficiency class according to the German building energy act was determined by measuring the energy required per m^2 of heated living area when heating with the heating curve implementation provided by Fraunhofer ISE.

8 Experiments and Results

This chapter describes the experiments which were conducted in order to evaluate the concepts and implementations from this work and to answer the research questions defined in section 5.3. A total of four experiments were carried out for this purpose.

All experiments were conducted on a Debian 10 Linux server with an NVIDIA A100 GPU, 256GiB RAM and an AMD EPYC 7502P 32 core processor, which was provided by the Fraunhofer ISE. CUDA[18] version 11.5 was available on the server in order to support GPU acceleration while performing training of the DRL agent. The duration of the training depends on the experiment and is therefore reported in the experiments section. All four experiments are described by the same structure: (1) the setup, (2) the results, and (3) a summary.

8.1 Experiment 1: Efficient Control Strategies

In this experiment, the primary research objective was evaluated, which was to learn efficient control strategies for heat pumps using deep reinforcement learning in simulation. In this context, an efficient strategy is one that minimizes electrical energy usage and comfort deviations at the same time.

Note that this experiment evaluates the learned control strategies in a qualitative manner. Additionally, the strategies are evaluated quantitatively

[18] https://developer.nvidia.com/cuda-toolkit

[19] According to the German buidling energy act `https://www.bmwsb.bund.de/Webs/BMWSB/DE/themen/bauen/energieeffizientes-bauen-sanieren/gebaeudeenergiegesetz/gebaeudeenergiegesetz-artikel.html`

in terms of comfort deviations which were to be minimized. A quantitative evaluation of the energy efficiency of the learned strategies was done by experiment 8.2, as baselines are needed for comparison.

8.1.1 Setup Experiment 1: Efficient Control Strategies

The construction methods and therefore also the thermal properties of buildings differ largely. For this reason, three different buildings were used for the evaluation, each intended to represent a separate scenario. The three scenarios are: (1) an *old* building which is considered energy inefficient; (2) an *efficient* building, which as the name suggests can be considered efficient in terms of its thermal properties; and (3) an *efficient enhanced* building, which is the same as the *efficient* building but was extended by enhancing its heat storage capabilities [20]. The buildings are summarized in table 2 and were simulated using the framework described in section 5.1. The parameters of the *old* building were obtained from the online tool TABULA[21]. The parameters represent a single-family house that was built in 1984. The houses *efficient* and *efficient enhanced* represent the same existing house located in Freiburg. Its parameters were measured and obtained in the course of another project at Fraunhofer ISE. The heat pump which was to be controlled in all of the three houses corresponds to a Dimplex LA 6TU air source heat pump with maximum heating power set to 12kW. The selection of the houses and the parametrization of the simulation were carried out in consultation with the Fraunhofer ISE.

It is important to note, that the training of the DRL agents as described in 6.4 was carried out individually for each building, which led to three independent trained agents. Except for the change in the parametrization of the simulation framework, the training and testing procedure is the same for all three buildings.

8.1.2 Results Experiment 1: Efficient Control Strategies

Figure 11 shows the progression of the mean reward during training for each of the three buildings. As described in section 6.4.3, running the training multiple times with different preset random seeds is an elementary step when applying deep reinforcement learning in order to rule out the possibility that

[20] To be precise, the water volume in the heating circuit was increased to simulate the effect of a water tank for storing hot water. Unfortunately, the simulation framework did not support the usage of a water tank per se. However, the thermodynamic processes are similar and therefore sufficient to evaluate the proposed method.

[21] `https://webtool.building-typology.eu/`

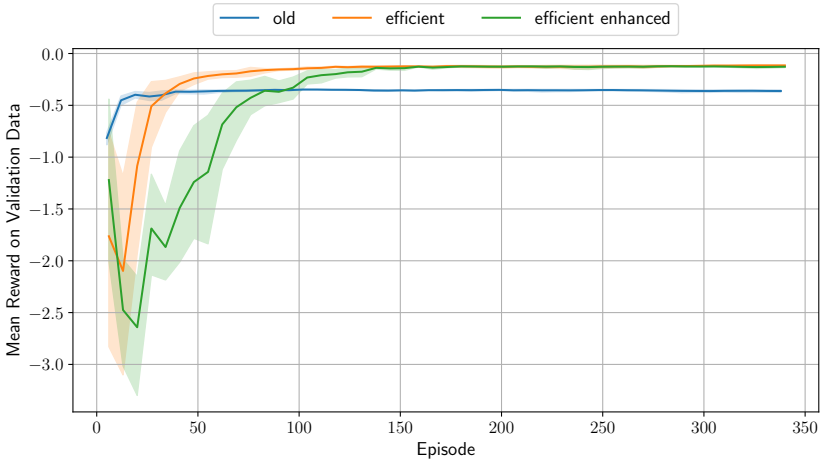

Figure 11: Evolution of the mean reward on the validation data per time step during training for each of the three buildings. For each building, the training was executed five times using different preset random seeds. The shaded areas show the variance of the progression of the mean reward caused by the different preset random seeds. The line indicates the mean progression.

good results were only achieved by chance. The shaded areas of the curves shown in figure 11 show that the variability of the mean rewards from the different seeds decreases as the training progresses. At the end of the training where the best rewards occur, the variability is minimal. Therefore, it can be excluded that good results were found only by chance and the performance of the proposed methods is independent of preset random seeds which can be interpreted as an indicator of stability. Since the variability is so low, the results are simply reported on the run that has achieved the highest reward. The training of the agents for each of the buildings containing the five runs with the different preset random seeds took 6 hours and 45 minutes on the server described in 8.

Subsequent to training, the agents were used to control the heat pumps of the three simulated buildings. To evaluate the functionality, the weather profiles contained in the test dataset were used. Therefore, heat pump control is evaluated in the months from January to March and from October to December in 2016.

The learned control strategies are demonstrated in figure 12. For a better overview, only the month of March is shown from the test data. The control during the other months from the test dataset can be seen in the appendix

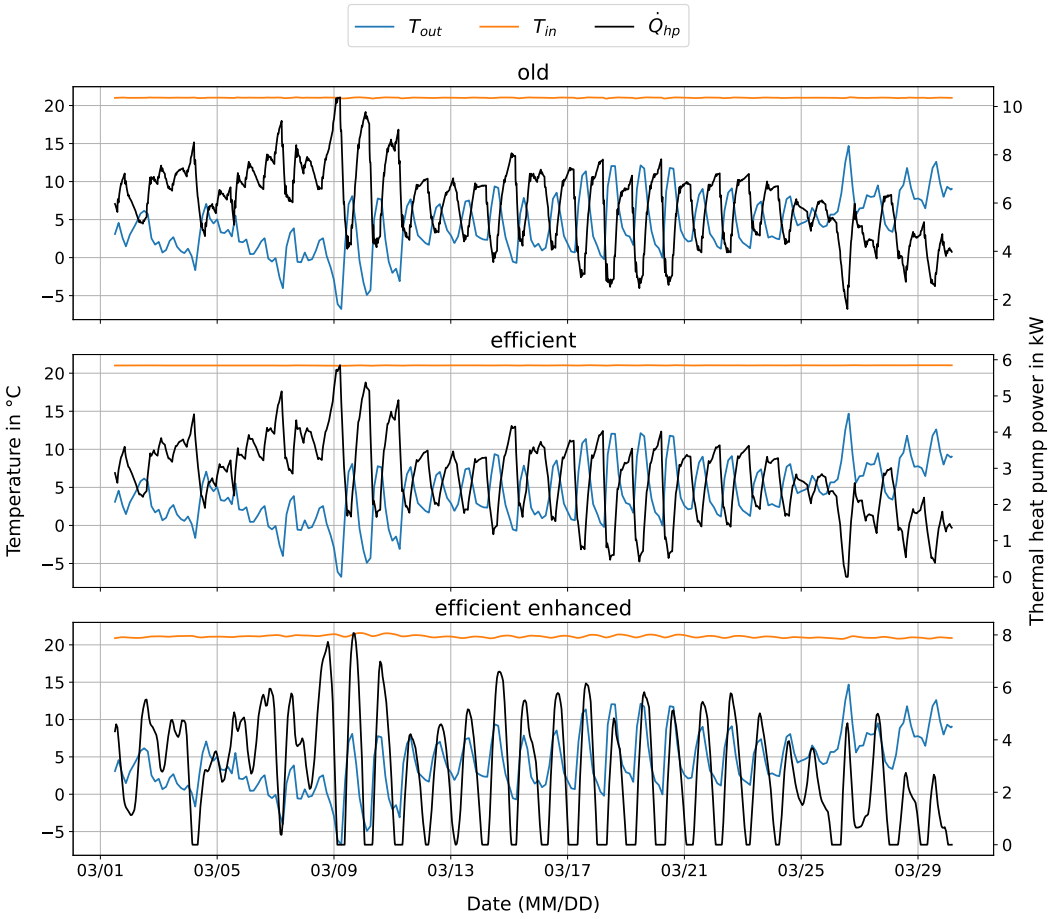

Figure 12: Heat pump control strategies for the three different buildings learned by the DRL agents. All three plots show the same weather profile, which represents the month of march from the test data. The control actions chosen by the agents are shown in black. The resulting indoor temperature is shown in orange. The outdoor temperature is shown in blue. Note that the strategy for the efficient enhanced building differs from the other two.

B. It can be seen, that for all three buildings, the agents learned to select control actions in order to keep the inside temperature in the comfort range, which was defined to be between 21°C and 25°C. If we look at the control actions chosen as a function of the outside temperature, we can see that the learned strategies of the old and efficient building are similar, but differ from the efficient enhanced building. Both learned control approaches are described below:

Strategy of Old and Efficient Building The learned strategies for the old and efficient building have the same approach and regulate the indoor temperature almost constantly at 21°C. Most of the heating is taking place when it is cold outside, and only a little when it is warm. Intuitively, this makes sense, as 21°C marks the lower comfort bound. Heating further would not increase any comfort by our definition, but lead to an increase in electricity usage. The strategy itself does not rely on predictions of the outside temperature that go far into the future. Experiments were conducted to determine the effect of different forecast lengths. The best results, which are reported here, were achieved with a prediction length of only 8 time steps which corresponds to two hours. Analogously, a relatively low γ of 0.96 was chosen, to minimize the impact of time distant rewards.

Strategy of Efficient Enhanced Building The strategy picked up by the agent controlling the heat pump in the efficient enhanced building is different. The majority of heating is taking place while the outside temperature is relatively high. This makes sense because as explained in section 3.1, the efficiency of the heat pump increases with the temperature of the low-temperature source, which in our case is the outdoor air temperature. The enhanced heat storage capability thus makes it possible to shift the heat generation to periods, where the heat pump can be operated efficiently. To obtain the results as given, a forecast length of 48 time steps which corresponds to 12 hours was used. Analogously, a relatively high γ of 0.99 was chosen, as actions at a current time step have a relatively high impact on future rewards.

Table 3 quantifies the results of the learned control strategies for all months contained in the test dataset. It can be seen, that for all three buildings, the agents managed to control the heat pump without any major comfort deviations. The increased standard deviation of the indoor temperature of the efficient enhanced building compared to the old and efficient building underline the two different strategic approaches presented above.

Table 3: Quantitative Results of Learned Heating Strategies

		Jan	Feb	Mar	Oct	Nov	Dec
Old	Mean Comfort Deviation	0	0	0	0	0	0
	Max Comfort Deviation	0.18	0.03	0.09	0.08	0.07	0.05
	Mean Indoor Temp	21.02	21.03	21.02	21.03	21.02	21.03
	Standard Deviation Indoor Temp	0.02	0.02	0.03	0.07	0.02	0.03
Efficient	Mean Comfort Deviation	0	0	0	0	0	0
	Max Comfort Deviation	0.07	0	0.03	0	0	0.02
	Mean Indoor Temp	21.02	21.02	21.02	21.05	21.03	21.02
	Standard Deviation Indoor Temp	0.02	0.01	0.01	0.06	0.01	0.01
Efficient Enhanced	Mean Comfort Deviation	0.02	0.01	0.02	0.04	0.04	0.01
	Max Comfort Deviation	0.16	0.2	0.21	0.19	0.19	0.21
	Mean Indoor Temp	21.12	21.1	21.1	21.07	21.02	21.13
	Standard Deviation Indoor Temp	0.17	0.11	0.15	0.22	0.13	0.11

Note: All units are in °C and rounded to two decimal places. The mean and standard deviation were calculated over one month and are to be considered per time step, which simulates 900s in real time. The Max denotes the maximum comfort deviation which occurred any time during the control period of that month.

Note that the evaluation of the consumed energy will be described in section 8.2, as baselines are necessary for interpretation.

8.1.3 Summary Experiment 1: Efficient Control Strategies

The key findings of this experiment are summarized as follows:

1. Based on the low variability in performance caused by different preset random seeds, the training of the method can be considered stable.

2. Just by learning through trial and error, all three trained agents picked up strategies that enabled them to control the heat pump in order to keep the indoor temperature in the comfort bound almost perfectly.

3. As shown by the use of different buildings, the learned strategic approaches adapt to the thermal properties of the building. If the building provides the capacity to store heat, the learned strategies take advantage of this to operate the heat pump efficiently by shifting heating loads. If the building does not offer enough capacity to store heat efficiently, the learned strategies simply regulate the indoor temperature at the lower comfort bound.

8.2 Experiment 2: Baseline Comparison

In order to be able to better classify the functionality of the trained DRL agents, a comparison against heating curve and MPC-based control was conducted. This is particularly important in order to be able to evaluate the energy consumption of the proposed method. This experiment targets to answer the second research question defined in section 5.3. The functionality of both baseline methods was described in section 3.2.

It must be noted, that MPC uses the simulation framework in order to plan and execute the control actions. As explained in section 3.2.2, this would mostly not be applicable in reality. Usually, MPC uses a simplified model to plan and a more complex model or the reality to execute the planned control actions [10, 67]. However, in our scenario, providing MPC with the same model for planning and execution can be considered an advantage, as it makes MPC the gold standard which finds the optimal control strategy and can thus be considered as upper performance limit.

8.2.1 Setup Experiment 2: Baseline Comparison

The implementations of the heating curve and MPC control, which served as baselines, were provided by the Fraunhofer ISE. As described in section 3.2.2, MPC plans based on the building model and external factors like the ambient temperature. Therefore, like the presented DRL method, MPC is given a perfect forecast of the outdoor temperature at every planning step. The length of the forecast provided to MPC was set to 96 time steps into the future. The forecast length was determined empirically by experiments. Longer forecasts did not improve the control quality of MPC further. Both methods were applied to control the heat pumps on the same weather data as the DRL agents. The same three buildings which were described in 8.1.1 were used.

8.2.2 Results Experiment 2: Baseline Comparison

Figure 13 contrasts the control strategies taken by the different methods for the three buildings qualitatively. The strategies learned by the DRL agents were already discussed in section 8.1.2. In the following, the strategies taken by MPC and the heating curve are discussed and compared to the strategies learned by DRL:

Heating Curve When looking at the strategy of the heating curve, it becomes clear that it is only a mapping of the current outside temperature to the control actions. This has two main disadvantages:

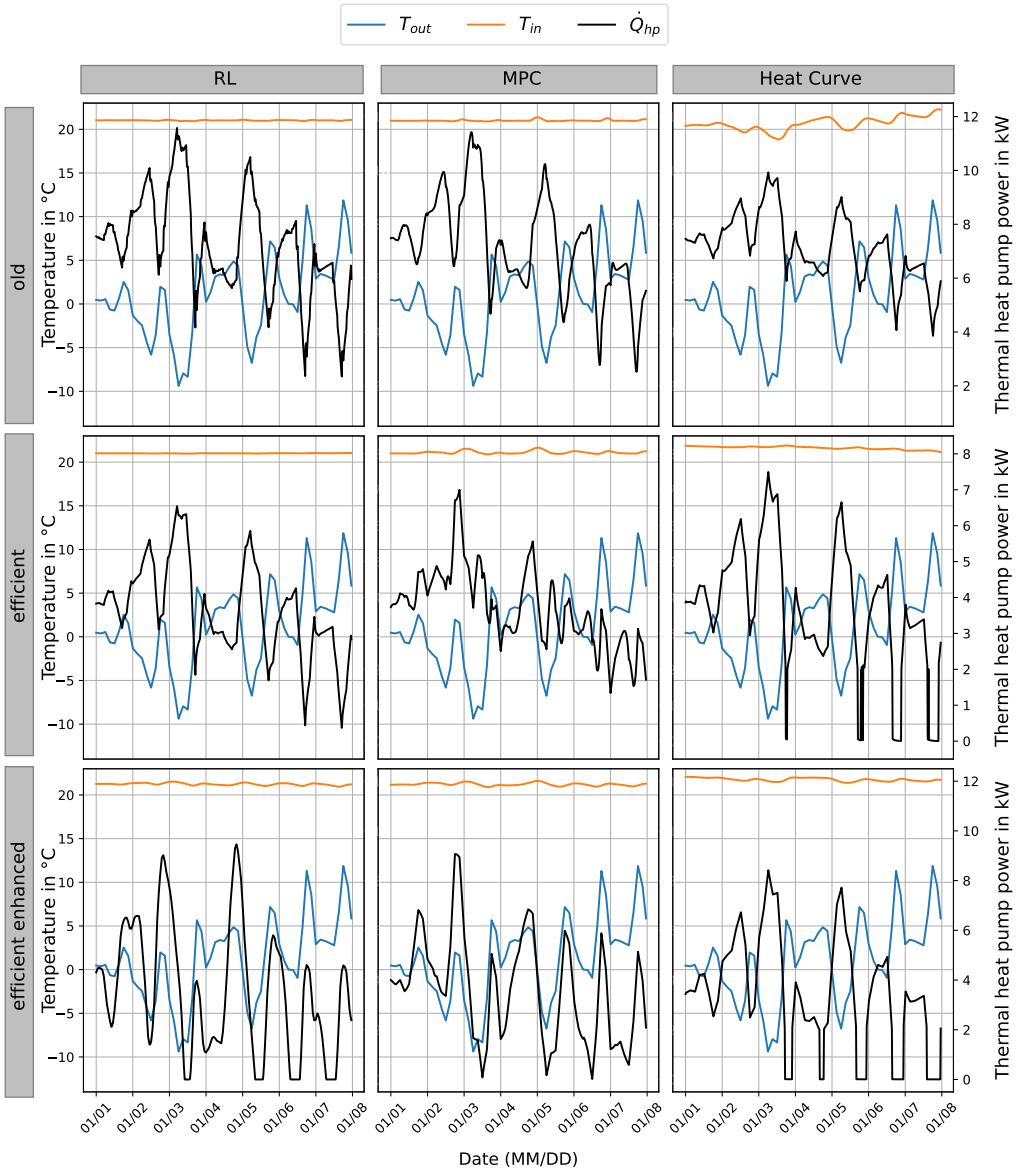

Figure 13: Comparison of strategies taken by the different control methods when heating the different buildings. In all cases, the same weather profile from the month of January from the test data was used. The control actions chosen are shown in black. The resulting indoor temperature is shown in orange. The outdoor temperature is shown in blue.

1. Comfort deviations occur, as the heating curve is just a feed-forward control system without any feedback about the current indoor temperature
2. Unlike the other two approaches, the heating curve cannot shift heating to periods when the heat pump can be operated efficiently, i.e. when it is already relatively warm outside.

It must be noted, that the first listed disadvantage is less relevant in practice. As described in section 3.2.1, thermostats are used in practice to counteract this problem to a certain degree. Thermostats were not supported by the simulation framework and therefore this problem occurs more in the setting at hand.

MPC The heating strategies applied by MPC differ from building to building. In the old building, the strategy is similar to the one taken by the DRL agent. Due to the limited thermal capacity of the old building, the majority of heating is done when its relatively cold. In the case of the efficient building, the strategies of MPC and DRL differ. Whereas DRL still takes the approach of heating most when it is cold outside, MPC already starts to shift the heating operation to periods when the outside temperature is relatively warm and thus the operation is more efficient. As can be seen in Table 4, MPC's strategy is marginally (approx. 1%) more efficient in terms of energy consumption. The hypothesis is, that this little difference in efficiency is not enough for DRL to pick up the more complex strategy while learning through trial and error. Anyhow, it can be seen that in the case of the efficient enhanced building, which offers more potential for shifting the heating operations, the strategies taken by DRL and MPC are quite similar again.

Table 4 quantifies the results of the strategies discussed above. The values refer in each case to the mean or the maximum of the entire test dataset. The mean values represent the average per time step which simulates 900s in real-time. It can be seen that DRL consumes only minimally more energy for all 3 buildings compared to MPC, which as explained above can be considered as optimal solution in this scenario. Additionally, it can be seen that DRL outperforms the heating curve in all three buildings in terms of energy usage and comfort.

The execution times indicate how long the execution took for all 6 months in the test data set. This includes the execution of a total of 17,262 simulation steps. Since the heating curve is only a static mapping from the outside temperature to a control action, its execution time can be roughly considered

Table 4: Quantitative Baseline Comparison

		DRL	MPC	Heating Curve
Old	Electricity Mean in Wh	405.15	403.23	419.25
	Comfort Deviation Mean in °C	0	0.02	0.11
	Comfort Deviation Max in °C	0.18	0.15	2.68
Efficient	Electricity Mean in Wh	138.08	136.67	142.42
	Comfort Deviation Mean in °C	0	0.01	0.1
	Comfort Deviation Max in °C	0.07	0.1	1
Efficient Enhanced	Electricity Mean in Wh	137.92	137.65	145.53
	Comfort Deviation Mean in °C	0.02	0	0.08
	Comfort Deviation Max in °C	0.21	0.20	0.74
	Execution Time in Seconds	38	1679	27

Note: Values were rounded to two decimal places. The mean values were calculated over the whole test dataset are to be considered per time step, which simulates 900s in real time. The Max denotes the maximum comfort deviation which occurred any time during the control period of the whole test dataset.

as the calculation time required by the simulation itself. The difference between this time and the times taken by MPC and DRL can be interpreted as the time taken for planning the actions of the respective methods. In order to create equal conditions during the measurement of the run times, the GPU was switched off for DRL. Although the simulation framework used can be considered a relatively simple model, it can be seen that the run times of MPC are many times longer than those of DRL.

8.2.3 Summary Experiment 2: Baseline Comparison

The key findings of this experiment are summarized as follows:

1. DRL performs better in terms of energy usage and comfort than the heating curve in all three buildings.

2. DRL only uses slightly more energy than MPC, which however can be considered the optimal solution in this scenario.

3. The run times of DRL are many times faster than those of MPC.

8.3 Experiment 3: Demand Response Capability

As explained in chapter 1, due to the increasing share of renewable energy sources, the electricity supply is becoming more dependent on external factors like the weather. To balance demand and supply of electricity on

the consumer side, it is expected that time-based varying electricity prices will be available to residential customers in the future [8, 9]. This is called demand response and therefore electrical devices should be able to regulate their load based on the varying price signals in order to save costs. The heat pump as an electric heater accounts for a large proportion of the energy usage of residential buildings, which is why it offers potential savings when controlled with respect to the varying price.

The aim of this experiment was to investigate the last research question which was defined in section 5.3. This has defined the question of whether the presented solution can be extended to a demand response scenario, where the electric load of the heat pump should take place with respect to the electricity price. Therefore, a varying price signal was included and the objective was to minimize the operational cost of the heat pump while still maintaining comfort. The operational cost result from the amount of energy consumed and the electricity price at that time.

8.3.1 Setup Experiment 3: Demand Response Capability

In order to train an agent with this new objective, the definition of the MDP from 6.2 was changed accordingly:

Reward As just mentioned, the new objective of the agent is to operate the heat pump with minimal operational cost while minimizing comfort deviations. Operational costs are defined as $electricity\_used_t * price_t$. This objective is encoded in the reward to be maximized, which is now defined as:

$$r_t = -1 * (\beta * electricity\_used_t * price_t + comfort\_deviation_t) \quad (25)$$

Note that the new reward definition is analogue to the one from (24), but differs as the price is included. Here too, a trade-off parameter β is used to balance comfort and costs. The search for a suitable β depends on the main objective. If the main objective is to keep comfort at all costs, β is to be set relatively low. If the main objective is to save cost and to accept comfort deviations, β is to be set relatively high. In this setup, a β of 0.07 has shown to provide a reasonable trade-off between comfort and costs.

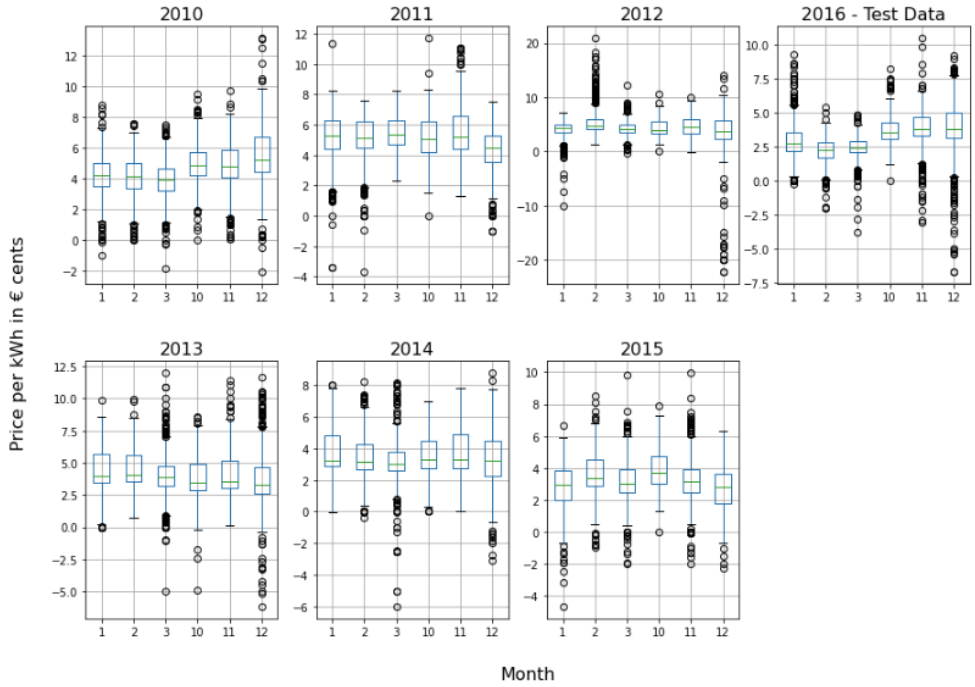

Figure 14: Distribution of the price data used in the experiment. A description of the boxplot itself can be found in figure 7.

State In order to give the agent the possibility to make decisions depending on the price, a price signal is given in the state in addition to the weather forecast. In this scenario, a state has the following form: $[T_{in}^t, T_{ret}^t, T_{out}^t, price^t, T_{out}^{t+1}, price^{t+1}, T_{out}^{t+2}, price^{t+2}, ...]$. It should be noted that, as with the weather, a perfect forecast is assumed. A forecast length of 32 time steps, which corresponds to 8 hours of the price and outside temperature has shown to perform best and is given in the state.

Besides changing the MDP as just described, data containing time-based varying electrical prices was required to conduct the experiment. With the exception of a few pilot projects, there are still no variable electricity prices for residential customers in the field in Europe. Accordingly, there is no data yet on these variable electricity prices. Therefore, like in [94], day-ahead electricity prices from the European power exchange (EPEX) spot marked for Germany were used to be able to evaluate the demand response capability of the proposed solution. These prices are used to balance supply and demand

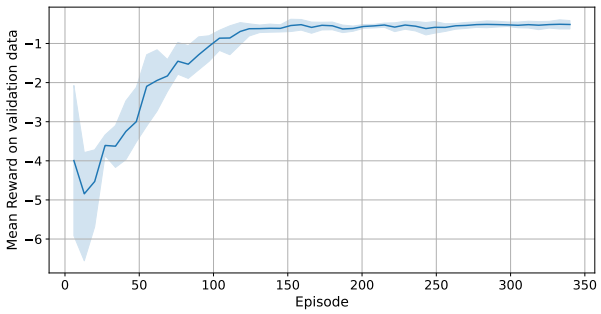

Figure 15: Evolution of mean reward during training. Please refer to the caption of figure 11 for a more detailed description of the plot.

between electricity producers and distributors. It must be noted, that the used EPEX spot prices differ from residential customer electricity prices as they do not include taxes and costs for grid usage. This becomes clear when looking at the median electricity prices shown in figure 14, which are cheap in comparison to real end residential customer electricity prices. Furthermore, figure 14 shows the presence of negative electricity price data. This is a realistic scenario since the storage of electricity is difficult and at times of high energy production from renewable sources, there is an abundance of electricity that needs to be consumed in order to ensure grid stability. The price data used in this experiment originate from the same time period as the weather data and were provided by the Fraunhofer ISE[22].

The experiment was conducted on the efficient enhanced building, as it has the highest heat storage capacity, which allows to efficiently shift the heating from high to low price periods. The training of the DRL agents was performed as described in 6.4. The training including 5 separate training runs using different preset random seeds took 7 hours to complete.

8.3.2 Results Experiment 3: Demand Response Capability

Figure 15 shows the progression of the mean reward during training. Analogue to figure 11, which was discussed in section 8.1.2, it shows the progression of the mean reward on the validation data during training. Here too, it can be seen, that the variability in the mean rewards caused by different preset random seeds minimizes as the training progresses. The run which achieved the highest mean reward during training was used to report the results in this section.

[22] https://energy-charts.info/

Table 5: Results of the Demand Response Scenario

	DRL	MPC	DRL Baseline
Cost Mean in € Cent	0.32	0.31	0.47
Comfort Deviation Mean °C	0.2	0.22	0.02
Comfort Deviation Max °C	1.08	0.85	0.21
Execution Time in Seconds	38	1677	38

Notes: Values were rounded to two decimal places and the statistics are calculated over the whole test dataset. The description of table 4 provides more details on how the mean and Max were calculated.

Subsequent to training, the agent was executed to control the heat pump. To evaluate the functionality, the weather and price profiles contained in the test dataset were used. Therefore, heat pump control is tested in the months from January to March and from October to December in 2016, using the strategies learned.

Figure 16 demonstrates the functionality of the trained agent for the efficient enhanced building qualitatively. It can be seen that the agent learned to heat whenever the price is relatively low. Therefore, the energy consumption is shifted to low price periods. It can be also observed, that in periods of negative prices, maximal heating is taking place.

Table 5 quantifies the results of this experiment. The cost and comfort deviation mean values denote the mean per time step and are calculated using all months contained in the test data. Based on the costs, it can be seen that the proposed solution is almost as effective as MPC in the demand response context, which can be considered as the gold standard. Additionally, the DRL method from section 8.1.2, which controls independent of the price signal is listed in the table and serves as an additional baseline. Although the DRL method described in this section causes more comfort deviations, operational costs could be reduced by almost a third compared to the DRL baseline. It must be noted, that the quantification of the saving potential depends on the price signal used and must be therefore interpreted with caution.

8.3.3 Summary Experiment 3: Demand Response Capability

The key findings of this experiment are summarized as follows:

1. The proposed method can be easily extended in order to be used in a demand response scenario, where control is to be done with respect to a time-based varying price signal.

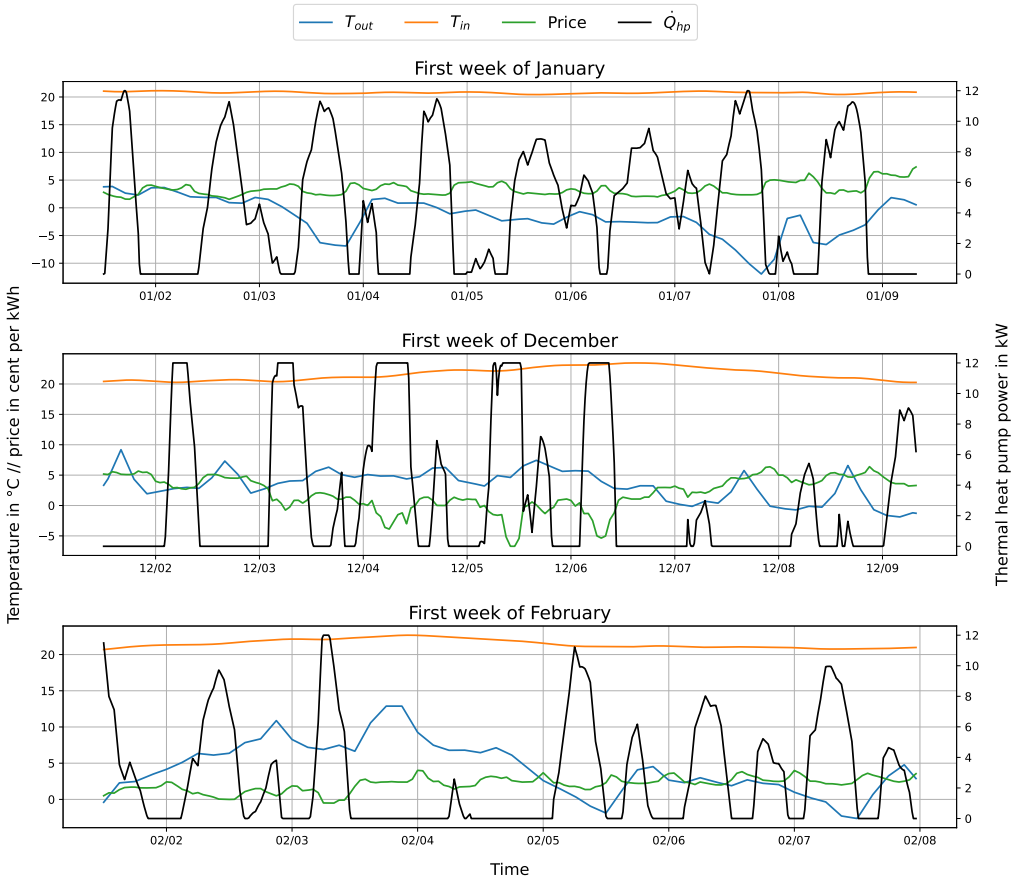

Figure 16: Heat pump control in a demand response scenario using the trained DRL agent for the efficient enhanced building. The three different subplots show three different weeks of heat pump control by the same agent in the same building. The indoor temperature is shown in orange. The control action is shown in black. The ambient temperature is shown in blue. The electricity price in green.

2. Here too, DRL achieves almost optimal results measured by MPC which serves as the gold standard.

8.4 Experiment 4: Robustness

As described in section 5.2, the simulation framework used in this work is a simplification of the problem statement of controlling a heat pump in the real world. However, the overall research goal in the field is to find efficient control strategies that are also applicable in the real world. Even though, the application of the presented methods to the real world was out of scope of this work, an experiment was conducted in order to evaluate the robustness of noisy sensor measurements and noisy weather forecasts which occur in practice. The goal of this experiment is to get a first idea of how the DRL solution reacts to noise. A more intensive investigation of random effects and noise is necessary as soon as the solution is to be transferred to a real heat pump, which is left open for future work. In addition, MPC was tested under the same noisy conditions, which allowed a further comparison of the two methods.

8.4.1 Setup Experiment 4: Robustness

The trained agents from experiment 1, which was described in section 8.1 were used. But with the difference that during execution on the test data, noisy sensors and forecasts were simulated. This means, the agents were trained without noise, but tested with noise. This was done by adding Gaussian noise[23] $X \sim \mathcal{N}(0,\, 0.5^2), Y \sim \mathcal{N}(0,\, 0.1^2), Z_n \sim \mathcal{N}(0,\, \sigma_{Z_n}^2)$ at every time step to the state, which thereby looks like this: $[T_{in}^t + X, T_{ret}^t + Y, T_{out}^t + Z_0, T_{out}^{t+1} + Z_1, T_{out}^{t+2} + Z_2, T_{out}^{t+n} + Z_n, ..., T_{out}^N + Z_N]$. Note that N denotes the length of the forecast and the standard deviation of Z at forecast step n is calculated by $\sigma_{Z_n} = 1.1 - 0.9^n$. This models the increasing uncertainty with the length of the forecast. Similarly, the same noise distribution was assumed and applied to MPC.

8.4.2 Result Experiment 4: Robustness

The qualitative results of this experiment for the DRL agents are illustrated by figure 17. The added noise causes the control strategy of the old building to fluctuate quickly. The same applies, but minimized, to the control strategy

[23] An alternative noise schema would be the Perlin noise [107] which can be evaluated in this setup in future works.

Table 6: Effects of Noise on the Performance

		DRL Baseline	**RL Noise**	MPC Baseline	**MPC Noise**
Standard	Electricity Mean in Wh	405.15	447.15	403.23	429.62
	Comfort Mean in °C	0	0.07	0.02	0.02
	Comfort Max in °C	0.18	0.88	0.15	0.72
Efficient	Electricity Mean in Wh	138.08	156.25	136.67	158.73
	Comfort Mean in °C	0	0	0.01	0
	Comfort Max in °C	0.07	0.2	0.1	0.06
Efficient Enhanced	Electricity Mean in Wh	137.92	139.35	137.65	153.63
	Comfort Mean in °C	0.02	0.02	0	0
	Comfort Max in °C	0.21	0.21	0.2	0.03

Note: The Baseline columns show the result as they were reported in the previous sections, without adding noise. The Noise columns shown the results with added noise during execution. The values were rounded to two decimal places and the statistics are calculated over the whole test dataset. The description of table 4 provides more details on how the mean and Max were calculated.

for the efficient building. For the efficient enhanced building, the impact of the added noise is not as big. The different influence of noise on the different strategies makes sense. As described in 8.1.2, the learned strategic approach of the old and efficient building is to regulate the indoor temperature almost constantly at 21°C. That makes the control strategies of these buildings mostly dependent on the measured indoor temperature T_{in}. However, that value was relatively strongly noisy as Gaussian noise with standard deviation of 0.5 was added, which explains the strong fluctuations. The learned strategy of the efficient enhanced building on the other hand is more complex as it plans to shift the heat loads to periods where heat pump operation is effective. Therefore, the dependency on the measurement of T_{in} is minimized, which is why the added noise does not have such a big influence on the control strategy.

This hypothesis is quantitatively supported by table 6, which compares the performance of the control strategies when tested with and without noise. It shows that the added noise does only have a small impact of performance on the control strategy of the efficient enhanced building. Additionally, table 6 lists the effect of equally distributed noise added to MPC during execution. Here it can be shown, that in the efficient and efficient enhanced scenario, the DRL methods outperform MPC in terms of energy consumption. However, for a final evaluated comparison, more extensive robustness experiments have to be performed.

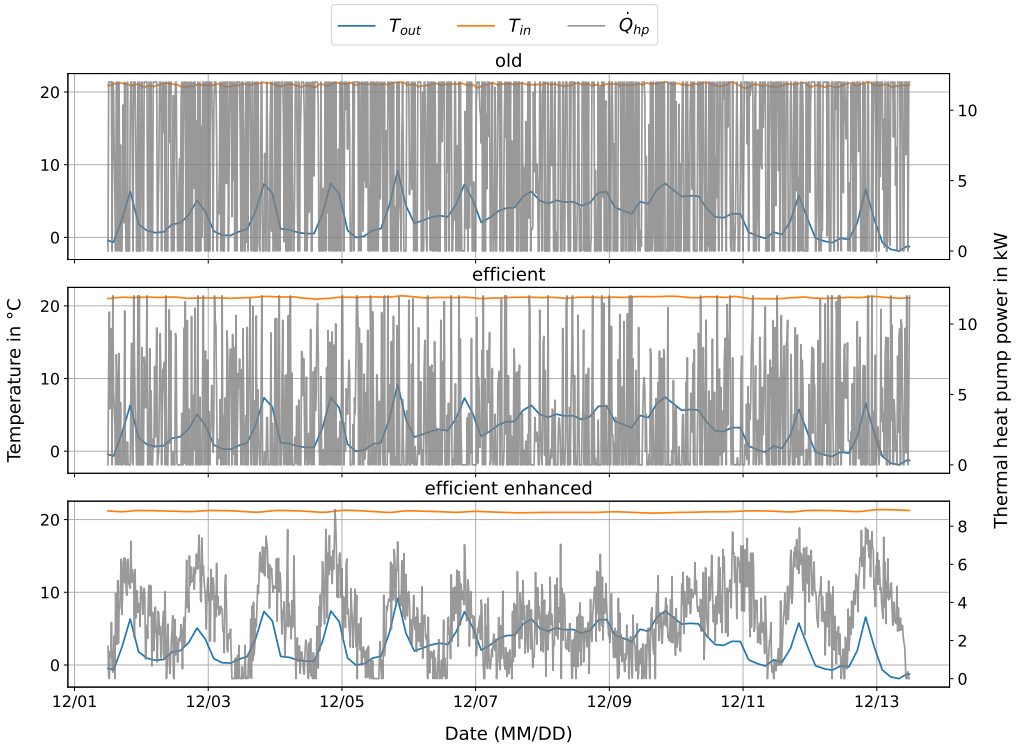

Figure 17: Effect of noise on the heat pump control strategies of the three buildings. Unlike the other plots, the control action is shown here in gray.

In an additional experiment based on these results, it was investigated how the large influence of noise can be counteracted, given the efficient building as an example. Therefore, the DRL agent of the efficient building was retrained by including the noise as described in 8.4.1 into *training*. To be clear, the agent was trained with noise and tested with noise. Additionally, domain randomization was implemented by randomizing the physical properties of the simulated building in each training episode. This is done by multiplying the floor size, heat capacity and transmission losses by a factor X individually drawn per parameter and episode from a normal distribution $X \sim \mathcal{N}(0,\, 0.2^2)$. Analogue approaches were taken by recent works [98, 108, 109] and reported good results by applying domain randomization. As illustrated by figure 18, by applying domain randomization and by including noise in training the agent was able to learn to control almost smoothly, even though the state was noisy. Anyhow, it is important to note that the noise added during training and testing was drawn from the same distribution with the same parameters. Further experiments in which the noise distribution from training and testing differ are left open for future work.

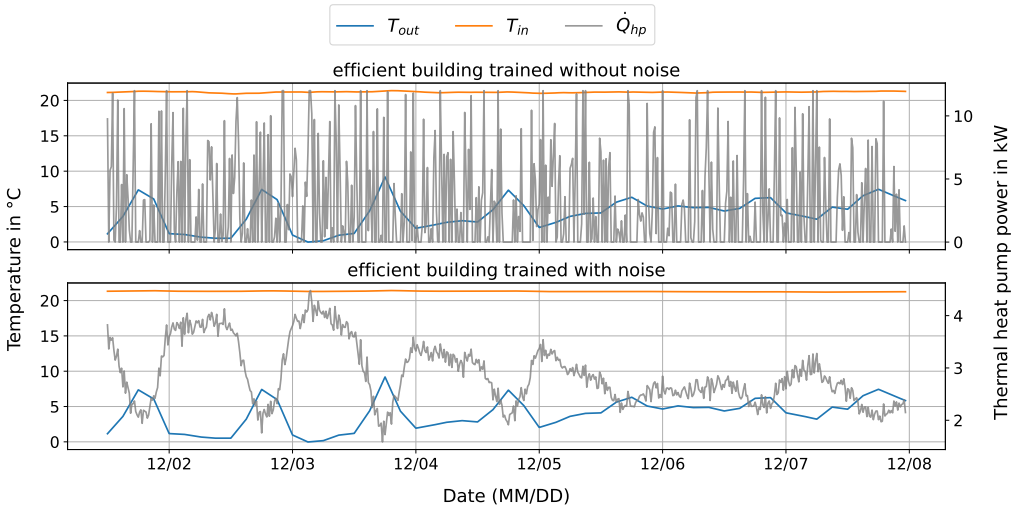

Figure 18: The effect of noise can be mitigated when training the agent with noise. The top image shows heat pump control when training the DRL agent without noise, but testing with noise. The bottom image shows heat pump control when training with noise and testing with noise. Note that the efficient building was used in this experiment. Unlike the other plots, the control action is shown here in gray.

9 Discussion

The main objective of the work at hand was to apply deep reinforcement learning to heat pump control. This overall objective was split into three research questions. Subsequent, these research questions are discussed in detail based on the experiments and their results which were reported in chapter 8:

Is it possible to apply DRL to heat pump control in the simulation provided ? The results from experiment 1 (see section 8.1) show that deep reinforcement learning can be applied to learn strategies to control a heat pump in the simulated environment. Thereby, it could be shown that heat pump control strategies can be learned in a trial and error manner by simply interacting with the simulation. The learned strategies can be considered efficient, as they did not cause major comfort deviations and have used only a little more electrical energy as MPC, which can be considered as the optimum solution in this setup. The results of the experiments were obtained by using weather profiles from the test dataset, which was not used during training. Thereby it could be shown that the solution generalizes to unknown weather profiles and does not overfit the train data.

By using three different buildings in the experiment, it could be shown that the learned control strategies depend on the building used. In the case of the two buildings with limited heat storage capabilities, the learned control strategy aimed to regulate the indoor temperature almost constantly by reacting on the current outside temperature. In the case of the building with enhanced storage capabilities, the learned control strategy aimed to exploit the heat storage to shift the heating loads to periods where the heat pump could be operated most efficiently.

These findings are in line with the works which were listed in section 4.1, which could also report a working deep reinforcement learning solution for heat pump control in simulation. Anyhow, the results extends those works by performing a detailed evaluation of the learned strategies. Especially, in respect to the fact that the learned strategies depend on the building and that learning of complex strategies which exploit the heat storage for heat load shifting can be learned in a trial and error manner. Additionally, by applying PPO out of the box, without the need for any handcrafted changes on the algorithm side, this work contributes by presenting another working example for the wide applicability of state-of-the-art deep reinforcement algorithms.

However, the solution was evaluated in simulation only. Therefore, the

problem statement at hand can be considered simplified, compared to the task of heat pump control in the real world. The simplifications and assumptions made by using the simulation were described in section 5.2. By those simplifications and assumptions, the used building simulation framework can be considered relatively simple. However, those simplifications made it possible to focus on learning efficient strategies in a trial and error manner by deep reinforcement learning and to perform a comparison to MPC.

Experiment 4 from section 8.4 has shown the influence of noisy sensors and forecasts on the solution's performance, but should only be considered as first robustness test in order to get a solution that can be efficiently transferred to the real world.

How well is the solution working compared to baselines ? The results from experiment 2 (see section 8.2) indicate that MPC-like performance can be achieved by deep reinforcement learning. This is in line with the findings from [83]. As described in section 3.2.2, the usage of MPC introduces some disadvantages due to its strong dependency on the model used. Applying deep reinforcement learning directly to the real world would make the time consuming process of model creation obsolete but is unpractical due to the number of interactions needed to learn useful control policies. Therefore, when deep reinforcement learning is implemented like in the work at hand by a model free and online manner, a building model, like with MPC is nevertheless necessary to either pretrain strategies which will be continually adapted after deployment to the real world, or to train strategies which are robust enough for direct deployment. However, the dependency on the model in the case of DRL is not as big, as the model is only needed during training and not during execution. This advantage has been demonstrated by the execution times of the two methods. Even though the building model used in this work can be considered relatively simple, the execution times of MPC in the setup at hand are more than 40 times longer than those of DRL. This alone would pose challenges if a more complex model was used, which would be closer to reality.

Additionally, it is speculated that the usage of DRL could be better in handling random events and uncertainties which are expected to occur when applying heat pump control to the real world. The reasons for this are: (1) DRL contains randomness by design, (2) neural networks have shown to be robust function approximators in related research areas, (3) DRL offers the possibility of continuous learning which can be applied in order to adapt to the circumstances in the real world and finally, (4) research on DRL is

rapidly increasing, leading to frameworks and concepts that facilitate implementations. Anyhow, this is a hypothesis that is left open for future work. Through experiment 4 (see section 8.4), it has already been suggested that DRL may do better with noisy sensor measurements and forecasts. However, for a final evaluated comparison, more extensive robustness experiments have to be performed. In general, this work contributes to the field by providing another comparison of MPC and DRL which is rare to find in other research works [72].

Finally, to be fair, it must be noted that the MPC implementation initially provided by the Fraunhofer institute was used as given and not extended or improved. It is, therefore, possible that with expert knowledge in the field of MPC, its implementation can be further improved, leading to other results, especially with regard to the long execution times and the results of the robustness experiment (see section 8.4). In summary, it can be said that the results show, that in the simulation provided, DRL is an alternative with similar performance to MPC. Anyhow, more work in this field, conducted by experts of both methods, is needed to finally answer the question: Which of these two methods is better suited for the application of heat pump control in the real world?

Besides the comparison to MPC, it could be shown that the learned control strategies are more effective in terms of energy usage and comfort than the heating curve. However, it should be noted that the heating curve which served as baseline in this setup was implemented as open loop control without any feedback loop[24]. Therefore, the comparison to the heating curve in the setup at hand can be considered biased. A fairer comparison between deep reinforcement learning and heating curve should be conducted as soon as the simulation framework supports thermostats which serve as regulators for the indoor temperature.

Can the described DRL method be extended to a demand response scenario ? The results from experiment 3 (section 8.3.2) show that an efficient control strategy with respect to a varying price signal can be learned by deep reinforcement learning. It could be shown that the agent learned to heat whenever the electricity price is relatively low, which was the expected strategy. Here too, a comparison to MPC was conducted and it could be shown, that in this setup, the performance was only by a margin

[24] As explained in section 3: In practice, thermostats are used to regulate the mass flow of the heated water in order to mitigate the non-existing feedback loop of this control strategy.

worse than the optimum provided by MPC. The conversion to the scenario with variable strategy prices was realized with only a few changes in the MDP definition, which shows the flexibility of the proposed solution.

It should be noted that the price data used to represent stock marked prices which are used for trading between electricity producers and distributors. Data on variable electricity prices for end consumers in Germany are not yet available.

10 Conclusion

This thesis aimed to apply deep reinforcement learning for heat pump control in a simulated environment. The results obtained indicate that heat pump control by deep reinforcement learning is a feasible alternative to control by MPC or the heat curve. Additional findings show that the presented solution can be easily extended to heat pump control with respect to variable electricity prices.

To summarize, the work contributes to the field by: (1) showcasing the wide applicability of deep reinforcement learning by providing another working example, (2) performing an in-depth evaluation of the learned control strategies, (3) providing a functioning DRL solution for heat pump control to the Fraunhofer ISE, which can build on it in future work and most importantly, (4) comparing deep reinforcement learning and MPC on the task of heat pump control, which has only been included in the work from [83].

However, it is important to note that the presented solution was applied in simulation and not to the real world. Therefore, the work at hand can be considered as proof of concept for heat pump control by deep reinforcement learning. The transfer of the solution to the real world remains open for future works. Based on the experience gathered from the work at hand, the following research recommendations for future works are given:

1. Based on the simplifications and assumptions of the building simulation framework used in this work (see section 5.2), it can be considered relatively simple. Therefore, the simulation framework used in this work should be extended, among other things, by supporting multiple rooms, introducing thermostats, and applying random inhabitant effects. The extended simulation framework can then be used to get results that are closer to reality.

2. The comparison of MPC and deep reinforcement learning should be extended to different building simulations and a larger number of simulated buildings to gather more insights and more significant results from the comparison.

3. Although the code of this master thesis has been published publicly on GitHub, the used simulation framework itself is not intended for open source use. To ensure comparability to other methods and works, it is necessary that simulations are published to serve as benchmarks.

4. The process of transferring the learned control strategies to the real world was not considered in this work. Concepts of continuously learning strategies to adapt to changing circumstances in the real world need to be tested. Additionally, concepts of making the strategies learned in simulation more robust and hence improve the chance that they will work in the real world need to be evaluated in this context.

5. In the work at hand, one deep reinforcement learning agent was trained per building. Another approach, which is worth evaluating would be to train a single agent with the goal, of achieving a generalized strategy, which can be used for a wide variety of different buildings. This strategy could include a step where the thermal properties of the building are first detected in order to adapt the strategy accordingly.

6. The concept of offline reinforcement learning seems to be promising for the problem of heat pump control in the real world. It would make the process of model creation obsolete, as control strategies could be learned from previously collected sensor data.

11 References

[1] AG Energiebilanzen e.V., "Anwendungsbilanzen zur Energiebilanz Deutschland." URL: https://ag-energiebilanzen.de/wp-content/uploads/2020/10/ageb_20v_v1.pdf. (accessed: Apr. 2, 2022).

[2] Bundesministerium für Wirtschaft und Klimaschutz, "Ziele der Energiewende." URL: https://www.bmwi.de/Redaktion/DE/Infografiken/Energie/ziele-der-energiewende.html. (accessed: Apr. 2, 2022).

[3] P. Sterchele, J. Brandes, J. Heilig, D. Wrede, C. Kost, T. Schlegl, A. Bett, and H. Henning, "Wege zu einem klimaneutralen Energiesystem. Die deutsche Energiewende im Kontext gesellschaftlicher Verhaltensweisen." Fraunhofer ISE, URL: https://www.ise.fraunhofer.de/content/dam/ise/de/documents/publications/studies/Fraunhofer-ISE-Studie-Wege-zu-einem-klimaneutralen-Energiesystem-Update-Klimaneutralitaet-2045.pdf, 2021. (accessed: May 23, 2022).

[4] D. Rolando and M. Hatef, "Smart control strategies for heat pump systems." KTH Royal Institute of Technology, URL: https://varmtochkallt.se/wp-content/uploads/Projekt/EffsysExpand/P18_Project_Report_final_reviewed.pdf, 2018. (accessed: Mar. 14, 2022).

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, ch. Summary of Notation. MIT press, second ed., 2018.

[6] K. Huchtemann, *Supply temperature control concepts in heat pump heating systems*. PhD thesis, Techn. Hochsch., Aachen, 2015.

[7] Bundesverband Wärmepumpe e.V. URL: https://www.waermepumpe.de/presse/zahlen-daten/. (accessed: Apr. 21, 2022).

[8] European Commission, "Incorporing demand side flexibility, in particular demand response, in electricity markets," *Commission staff working document*, 2013. (accessed: Apr. 4, 2022).

[9] C. Eid, E. Koliou, M. Valles, J. Reneses, and R. Hakvoort, "Time-based pricing and electricity demand response: Existing barriers and next steps," *Utilities Policy*, vol. 40, pp. 15–25, 2016.

[10] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini, and A. Bemporad, "Model predictive control (mpc) for enhancing building and hvac system energy efficiency: Problem formulation, applications and opportunities," *Energies*, vol. 11, no. 3, 2018.

[11] D. Fischer and H. Madani, "On heat pumps in smart grids: A review," *Renewable and Sustainable Energy Reviews*, vol. 70, pp. 342–357, 2017.

[12] J. Cígler, D. Gyalistras, J. Široky, V. Tiet, and L. Ferkl, "Beyond theory: the challenge of implementing model predictive control in buildings," in *Proceedings of 11th Rehva world congress, Clima*, vol. 250, 2013.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, ch. Part 1 Tabular Solution Methods, pp. 23–189. MIT press, second ed., 2018.

[14] D. Silver, "Lecture 1: Introduction to reinforcement learning." URL: `https://www.davidsilver.uk/teaching/`, 2015. (accessed: Feb. 23, 2022).

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, ch. Chapter 1 Introduction, pp. 1–22. MIT press, second ed., 2018.

[16] R. Güldenring, "Applying deep reinforcement learning in the navigation of mobile robots in static and dynamic environments," Master's thesis, Univeresität Hamburg, 2019.

[17] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[18] R. Bellman, "Dynamic programming," *Princeton University Press*, 1957.

[19] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, ch. Part 2 Approximate Solution Methods, pp. 195–337. MIT press, second ed., 2018.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through

deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[22] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992.

[23] J. Vitay, "Policy gradient methods." URL: `https://julien-vitay.net/deeprl/PolicyGradient.html#sec:policy-gradient-methods`. (accessed: Jan. 31, 2022).

[24] J. Schulman, *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2016.

[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[26] S. Levine, "Deep reinforcement learning, lecture 5: Policy gradients." URL: `https://rail.eecs.berkeley.edu/deeprlcourse/`, University of California, Berkeley, 2021. (accessed: Mar. 10, 2022).

[27] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[28] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning.," *Journal of Machine Learning Research*, vol. 5, no. 9, 2004.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] OpenAI et al., "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[31] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[32] "Proximal policy optimization." https://spinningup.openai.com/en/latest/algorithms/ppo.html. (accessed: Mar. 30, 2022).

[33] J. Vitay, "Model-based rl." URL: https://julien-vitay.net/deeprl/ModelBased.html#sec:model-based-rl. (accessed: Mar. 16, 2022).

[34] "Kinds of rl algorithms." https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. (accessed: Mar. 15, 2022).

[35] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[36] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement learning*, pp. 45–73, Springer, 2012.

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[38] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[39] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[40] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, pp. 698–721, jan 2021.

[41] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[42] O. Kroemer, S. Niekum, and G. D. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, vol. 22, no. 30, 2021.

[43] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[44] A. Mosavi, Y. Faghan, P. Ghamisi, P. Duan, S. F. Ardabili, E. Salwana, and S. S. Band, "Comprehensive review of deep reinforcement learning methods and applications in economics," *Mathematics*, vol. 8, no. 10, p. 1640, 2020.

[45] A. R. Sharma and P. Kaushik, "Literature survey of statistical, deep and reinforcement learning in natural language processing," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 350–354, IEEE, 2017.

[46] V. Uc-Cetina, N. Navarro-Guerrero, A. Martin-Gonzalez, C. Weber, and S. Wermter, "Survey on reinforcement learning for language processing," *arXiv preprint arXiv:2104.05565*, 2021.

[47] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[48] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[49] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[50] L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang, and X. Guan, "A review of deep reinforcement learning for smart building energy management," *IEEE Internet of Things Journal*, vol. PP, 05 2021.

[51] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[52] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, PMLR, 20–22 Jun 2016.

[53] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[54] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International Conference on Machine Learning*, pp. 2961–2970, PMLR, 2019.

[55] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[56] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[57] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[58] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*, pp. 449–458, PMLR, 2017.

[59] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[60] A. Lazaridis, A. Fachantidis, and I. Vlahavas, "Deep reinforcement learning: A state-of-the-art walkthrough," *Journal of Artificial Intelligence Research*, vol. 69, pp. 1421–1471, 2020.

[61] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, *et al.*, "What matters for on-policy deep actor-critic methods? a large-scale study," in *International conference on learning representations*, 2020.

[62] C. Bongs, D. Günther, S. Helmling, T. Kramer, M. Miara, T. Oltersdorf, and J. Wapler, *Wärmepumpen*, pp. 17–21. Fraunhofer IRB Verlag, 2015.

[63] S. Medved, S. Domjan, and C. Arkar, *Sustainable Technologies for Nearly Zero Energy Buildings*, pp. 143–150. Springer-Verlag, 2019.

[64] M. Platt, S. Exner, and R. Bracke, "Analyse des deutschen Wärmepumpenmarktes–Bestandsaufnahme und Trends." GeothermieZentrum Bochum, URL: `https://www.erneuerbare-energien.de/EE/Redaktion/DE/Downloads/Studien/analyse-waermepumpenmarkt.pdf?__blob=publicationFile&v=2`, 2010. (accessed: May 17, 2022).

[65] D. Fischer, J. Bernhardt, H. Madani, and C. Wittwer, "Comparison of control approaches for variable speed air source heat pumps considering time variable electricity prices and pv," *Applied Energy*, vol. 204, pp. 93–105, 10 2017.

[66] R. E. Rink, V. Gourishankar, and M. Zaheeruddin, "Optimal control of heat-pump/heat-storage systems with time-of-day energy price incentive," *Journal of Optimization Theory and Applications*, vol. 58, no. 1, p. 93–108, 1988.

[67] A. Afram and F. Janabi-Sharifi, "Theory and applications of hvac control systems – a review of model predictive control (mpc)," *Building and Environment*, vol. 72, pp. 343–355, 2014.

[68] Y. Yao and D. K. Shekhar, "State of the art review on model predictive control (mpc) in heating ventilation and air-conditioning (hvac) field," *Building and Environment*, vol. 200, p. 107952, 2021.

[69] S. Privara, J. Cigler, Z. Váňa, F. Oldewurtel, C. Sagerschnig, and E. Žáčeková, "Building modeling as a crucial part for building predictive control," *Energy and Buildings*, vol. 56, pp. 8–22, 2013.

[70] T. Wei, Y. Wang, and Q. Zhu, "Deep reinforcement learning for building hvac control," in *Proceedings of the 54th annual design automation conference 2017*, pp. 1–6, 2017.

[71] Y. Wang, K. Velswamy, and B. Huang, "A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems," *Processes*, vol. 5, no. 3, 2017.

[72] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 221–231, 2021.

[73] P. Fazenda, K. Veeramachaneni, P. Lima, and U.-M. O'Reilly, "Using reinforcement learning to optimize occupant comfort and energy usage in hvac systems," *Journal of Ambient Intelligence and Smart Environments*, vol. 6, pp. 675–690, 01 2014.

[74] F. Ruelens, B. Claessens, S. Vandael, B. De Schutter, R. Babuska, and R. Belmans, "Residential demand response applications using batch reinforcement learning," *arXiv preprint arXiv:1504.02125*, 2015.

[75] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning.," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 04 2005.

[76] F. Ruelens, S. Iacovella, B. J. Claessens, and R. Belmans, "Learning agent for a heat-pump thermostat with a set-back strategy using model-free reinforcement learning," *Energies*, vol. 8, no. 8, pp. 8300–8318, 2015.

[77] E. Barrett and S. Linder, "Autonomous hvac control, a reinforcement learning approach," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 3–19, Springer, 2015.

[78] F. Ruelens, B. Claessens, P. Vrancx, F. Spiessens, and G. Deconinck, "Direct load control of thermostatically controlled loads based on sparse observations using deep reinforcement learning," *CSEE Journal of Power and Energy Systems*, vol. 5, 07 2017.

[79] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, (Berlin, Heidelberg), p. 317–328, Springer-Verlag, 2005.

[80] T. Peirelinck, F. Ruelens, and G. Decnoninck, "Using reinforcement learning for optimizing heat pump control in a building model in model-

ica," in *2018 IEEE International Energy Conference (ENERGYCON)*, pp. 1–6, 2018.

[81] Z. Zhang, A. Chong, Y. Pan, C. Zhang, S. Lu, and K. P. Lam, "A deep reinforcement learning approach to using whole building energy model for hvac optimal control," in *2018 Building Performance Analysis Conference and SimBuild*, vol. 3, pp. 22–23, 2018.

[82] C. Patyn, F. Ruelens, and G. Deconinck, "Comparing neural architectures for demand response through model-free reinforcement learning for heat pump control," in *2018 IEEE International Energy Conference (ENERGYCON)*, pp. 1–6, 2018.

[83] A. Nagy, H. Kazmi, F. Cheaib, and J. Driesen, "Deep reinforcement learning for optimal control of space heating," *arXiv preprint arXiv:1805.03777*, 2018.

[84] X. Ding, W. Du, and A. Cerpa, "Octopus: Deep reinforcement learning for holistic smart building control," in *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*, pp. 326–335, 2019.

[85] A. Naug, I. Ahmed, and G. Biswas, "Online energy management in commercial buildings using deep reinforcement learning," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 249–257, 2019.

[86] B. Chen, Z. Cai, and M. Bergés, "Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy," in *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*, pp. 316–325, 2019.

[87] G. Gao, J. Li, and Y. Wen, "Deepcomfort: Energy-efficient thermal comfort control in buildings via reinforcement learning," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8472–8484, 2020.

[88] S. Nagarathinam, V. Menon, A. Vasan, and A. Sivasubramaniam, "Marco - multi-agent reinforcement learning based control of building hvac systems," in *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, e-Energy '20, p. 57–67, Association for Computing Machinery, 2020.

[89] L. Yu, Y. Sun, Z. Xu, C. Shen, D. Yue, T. Jiang, and X. Guan, "Multi-agent deep reinforcement learning for hvac control in commercial buildings," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 407–419, 2020.

[90] K. Kurte, J. Munk, O. Kotevska, K. Amasyali, R. Smith, E. Mckee, Y. Du, B. Cui, T. Kuruganti, and H. Zandi, "Evaluating the adaptability of reinforcement learning based hvac control for residential houses," *Sustainability (Basel)*, vol. 12, 9 2020.

[91] H. Zhao, J. Zhao, T. Shu, and Z. Pan, "Hybrid-model-based deep reinforcement learning for heating, ventilation, and air-conditioning control," *Frontiers in Energy Research*, vol. 8, p. 412, 2021.

[92] A. Heidari, F. Marechal, and D. Khovalyg, "An adaptive control framework based on reinforcement learning to balance energy, comfort and hygiene in heat pump water heating systems," *Journal of Physics: Conference Series*, vol. 2042, p. 012006, nov 2021.

[93] S. Ghane, S. Jacobs, W. Casteels, C. Brembilla, S. Mercelis, S. Latré, I. Verhaert, and P. Hellinckx, "Supply temperature control of a heating network with reinforcement learning," in *2021 IEEE International Smart Cities Conference (ISC2)*, pp. 1–7, 2021.

[94] S. Paul, "Learning to control heat pumps using supervised and reinforcement learning methods based on neural networks," Master's thesis, Albert-Ludwigs-University Freiburg, 2019.

[95] DIN V 4108-6:2003-06, "Thermal protection and energy economy in buildings - part 6: Calculation of annual heat and energy use."

[96] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[97] A. Raffin, "Reinforcement learning tips and tricks." `https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html`. (accessed: Mar. 15, 2022).

[98] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, 2018.

[99] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Conference on robot learning*, pp. 561–591, PMLR, 2018.

[100] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[101] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," *arXiv preprint arXiv:2005.12729*, 2020.

[102] L. Graesser and W. L. Keng, *Foundations of deep reinforcement learning: theory and practice in Python*, ch. Chapter 6: Advantage Actor-Critic (A2C), p. 148. Addison-Wesley Professional, 2019.

[103] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[104] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[105] Stable Baselines3, "PPO." `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`. (accessed: Feb. 5, 2022).

[106] F.-F. Li, "Cs231n: Deep learning for computer vision." URL: `https://cs231n.github.io/neural-networks-3/`, Stanford University, 2022. (accessed: Apr. 12, 2022).

[107] K. Perlin, "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, (New York, NY, USA), p. 287–296, Association for Computing Machinery, 1985.

[108] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[109] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810, IEEE, 2018.

Appendices

A PPO Parameters

The table lists the parameters set for PPO. Except for the learning rate and γ, the default parameters from Stable Baselines3[25] were used.

Table 7: Summary of Parameters

Parameter	Value
Algorithm	PPO
Learning Rate	$0.5 * 10^{-4}$
γ	0.96 or 0.99 (depending on the building)
Batch Size	64
Number of Epochs when Optimizing	10
Number of Interactions per Update	2048
Generalized Advantage Estimator Bias Variance Trade-Off Factor	0.95
Entropy Coefficient	0
Clip Parameter	0.2

[25] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

B Learned Strategies

Section 8.1.2 only illustrated the control strategies during the month of March. In the following the control strategies during all of the months contained in the test data are shown. All three subplots always show the same weather profile. The control actions chosen by the agent are shown in black. The indoor temperature is shown in orange. The outdoor temperature is shown in blue. Note that the strategy from the efficient enhanced building differs from the other two.

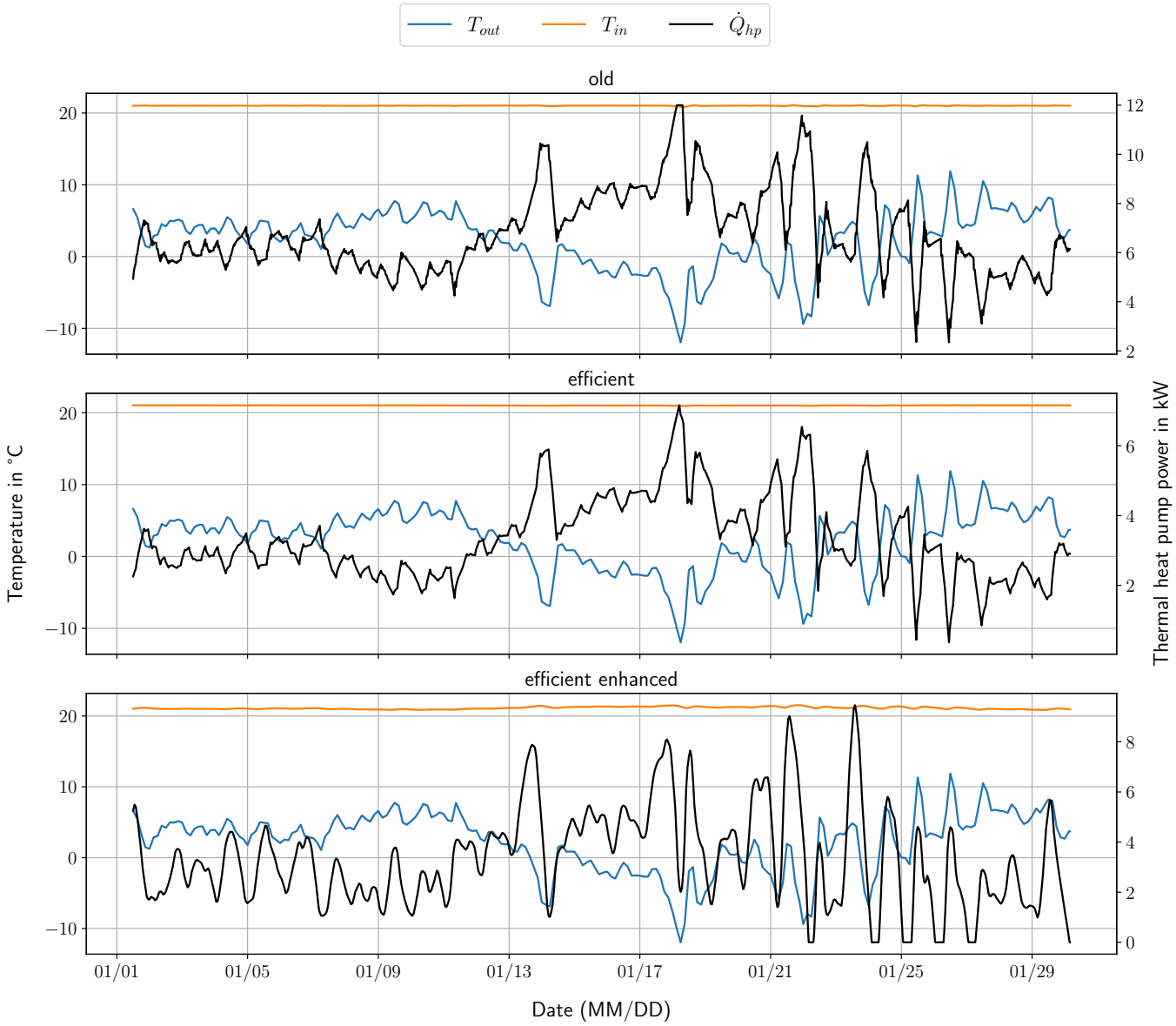

Figure 19: Control Strategies for January

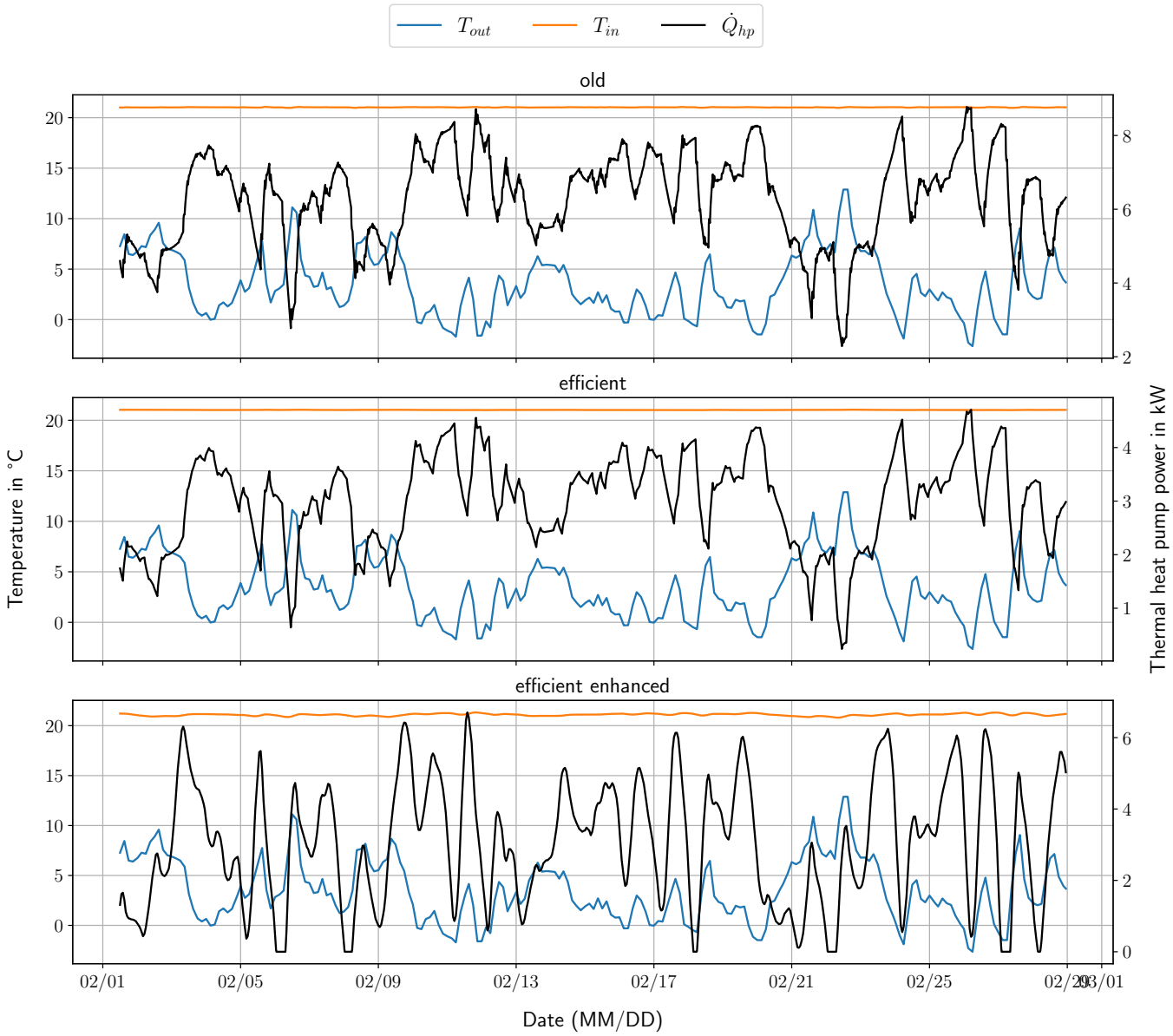

Figure 20: Control Strategies for February

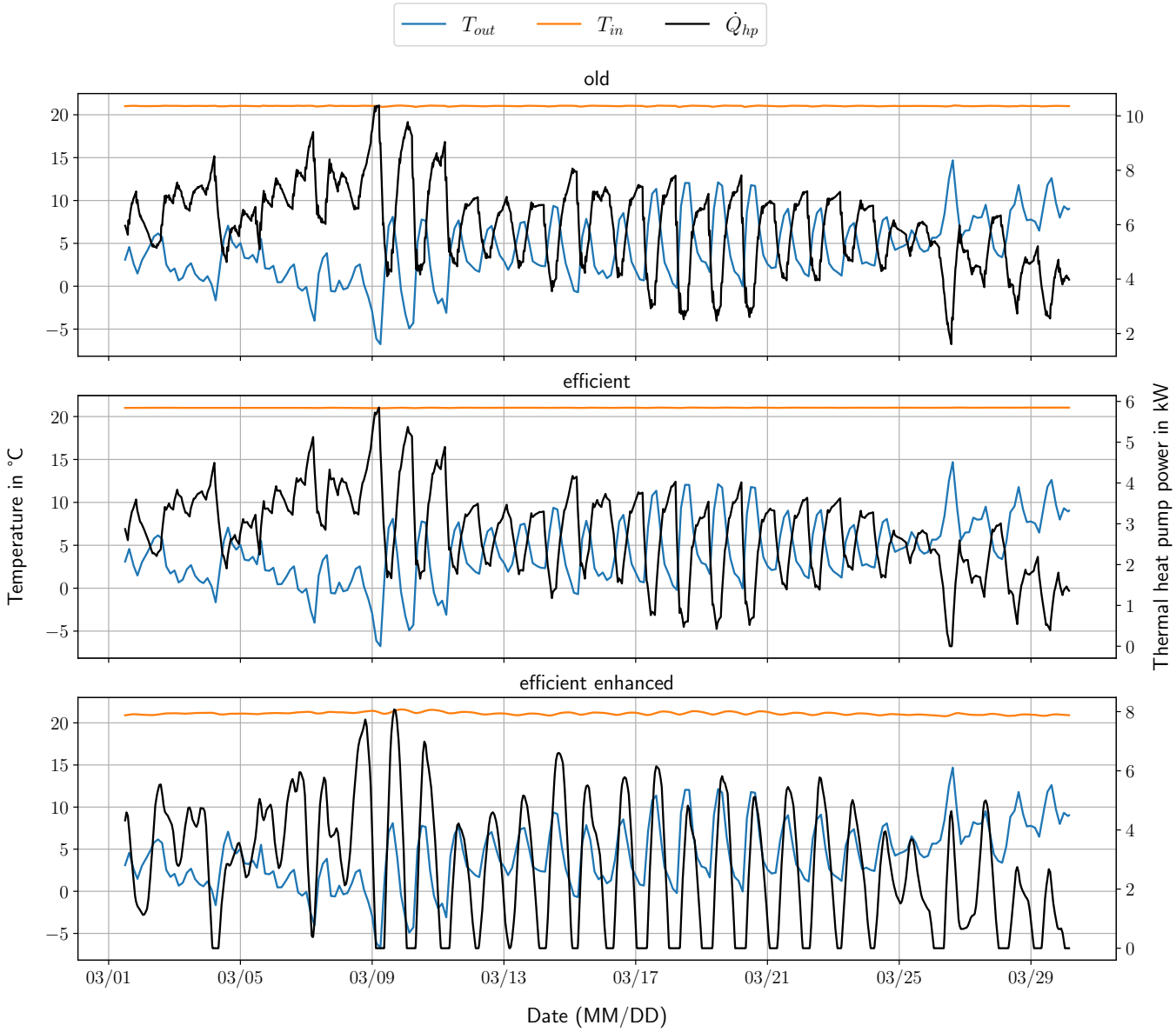

Figure 21: Control Strategies for March

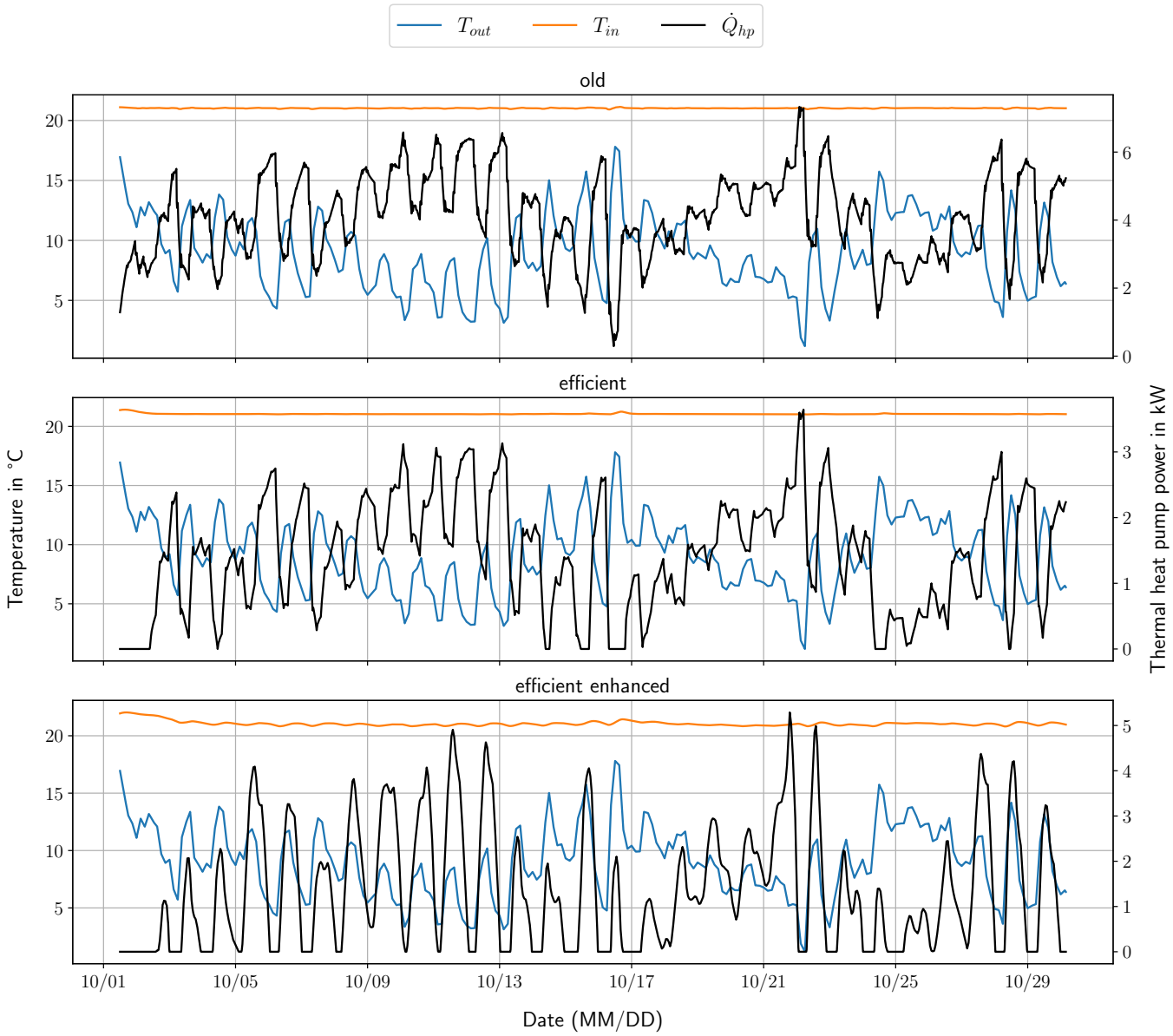

Figure 22: Control Strategies for October

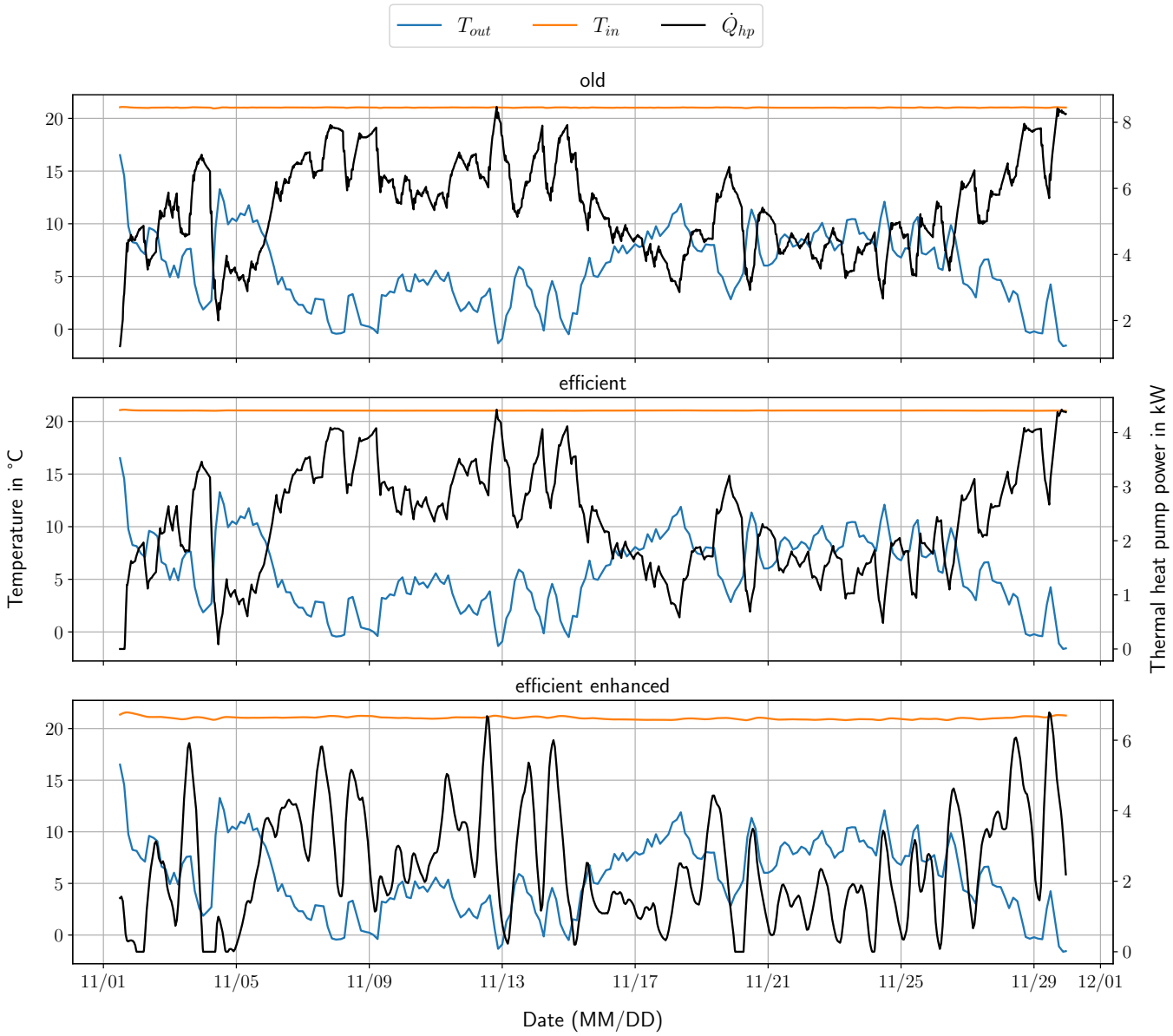

Figure 23: Control Strategies for November

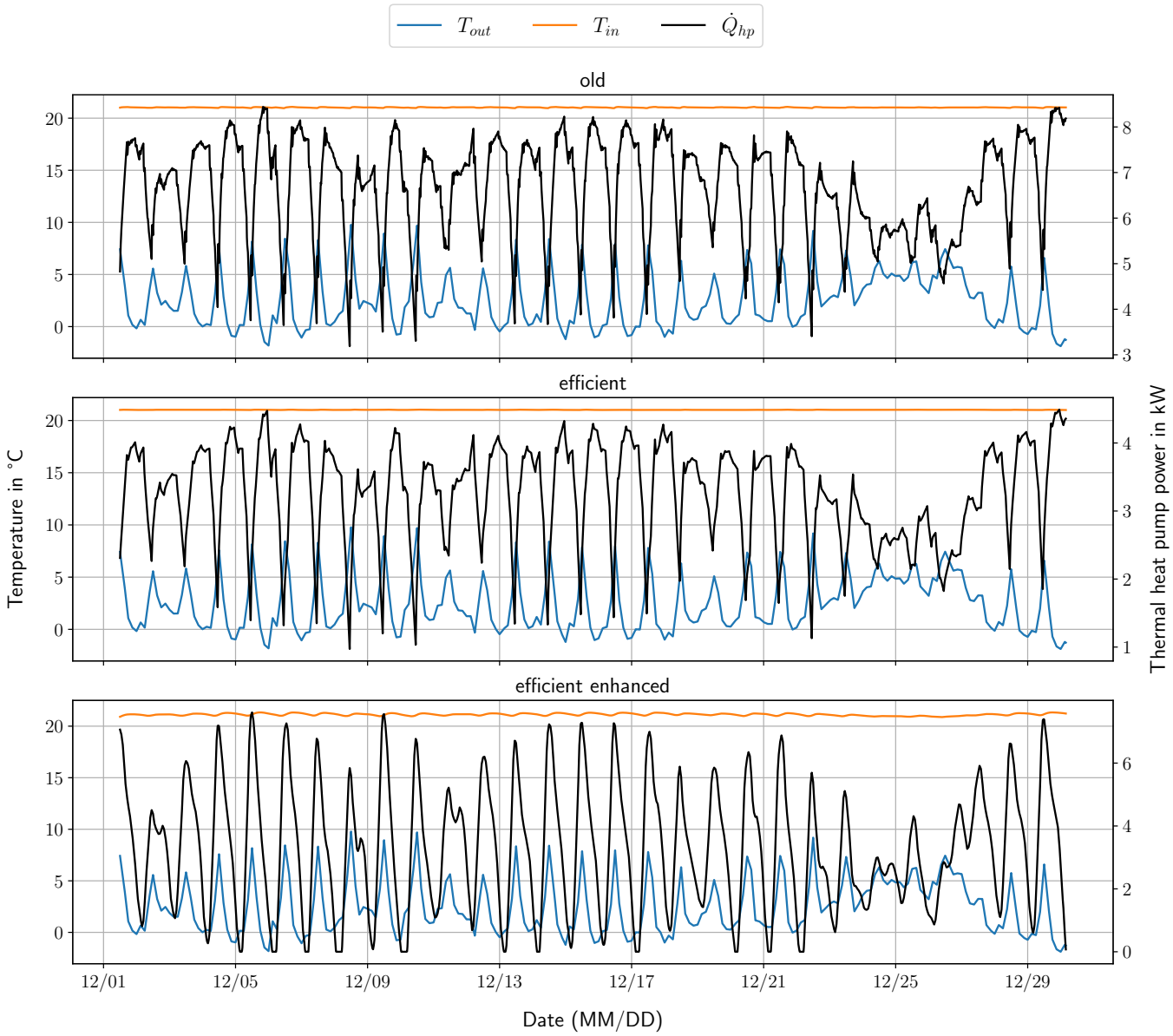

Figure 24: Control Strategies for December