CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Reinforcement Learning for Optimized Heat Plant Planning

Scheduling strategies in heat plant planning with long horizon forecasts and MILP optimality comparisons

Master's thesis in Computer science and engineering

Karl Göransson Kjellmer
Love Lindqvist

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Reinforcement Learning for Optimized Heat Plant Planning

Scheduling strategies in heat plant planning with long horizon forecasts and MILP optimality comparisons

Karl Göransson Kjellmer
Love Lindqvist

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Reinforcement Learning for Optimized Heat Plant Planning
Scheduling strategies in heat plant planning with long horizon forecasts and MILP optimality comparisons
Karl Göransson Kjellmer, Love Lindqvist

Supervisor: Kilian Tamino Freitag, Department of Electrical Engineering
Advisor: Christian Falk, Sigholm Tech AB
Examiner: Morteza Haghir, Department of Computer Science

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Reinforcement Learning for Optimized Heat Plant Planning
Scheduling strategies in heat plant planning with long horizon forecasts and MILP optimality comparisons
Karl Göransson Kjellmer
Love Lindqvist
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

District Heating Plants (DHPs) supply a significant share of heat for buildings in Sweden and present a non-trivial scheduling problem. Decisions about which production unit to run when and at which level must be balanced against time-varying electricity prices, heat demand, thermal storage dynamics, and unit-level operational constraints. Industry formulates this as a Mixed-Integer Linear Programming (MILP) problem, but MILP scales poorly with system size and horizon length. This thesis investigates whether Deep Reinforcement Learning (DRL), trained with week-ahead price and demand forecasts, can produce schedules competitive with a MILP baseline on the same problem instance. Two simulated DHP configurations of differing complexity are introduced, one corresponding to a household and one to a small town. Tested with a DRL Proximal Policy Optimization (PPO) agent that uses a convolutional encoder for forecast time series. Ablations on the action distribution (Gaussian, Beta, Kumaraswamy), the forecast encoder, the mechanism for enforcing demand satisfaction, and a transfer-learning study across plant configurations are presented. Against a rolling-horizon MILP on identical validation data, the trained agents learned a promising strategy reaching within 8.1% of MILP operating cost on the simpler configuration and effectively match it on the more complex one. However, the resulting schedules exhibit high-frequency unit switching and under-use of accumulator capacity which need to be addressed, still highlighting the potential of Reinforcement Learning (RL) in this domain.

Keywords: Reinforcement Learning, Proximal Policy Optimization, District Heating, Mixed-Integer Linear Programming, Scheduling, Distributions, Convolutional Neural Network, Deep Reinforcement Learning, Long Horizon

Acknowledgements

First of all we want to thank Sigholm Tech AB (Sigholm) for giving us this thesis opportunity and supporting us with data, compute, and an enjoyable time at their offices. At Sigholm we want to especially thank our industrial supervisors Christian, Jan Marten, and Ylva, who, during the project encouraged us and brought lots of ideas and interesting discussions. This thesis would not be possible without them. We want to give an extra thanks to Per, who, with patience, guided us through the workings of both heating plants and their MILP solver. Secondly we want to thank our Chalmers supervisor Kilian, who has been a beam of encouraging light during the entire process. The engagement he has shown in this project and his belief in us made this thesis what it is. Last of all we want to thank our examiner Morteza who has given us important direction and importantly, introduced us to Kilian.

Karl Kjellmer & Love Lindqvist, Gothenburg, 2026-06-03

Contents

List of Figures

List of Tables

List of Acronyms

ACC Accumulator. xi, 14, 15, 17, 21

BIO Biofuel burner. xi, xv, 15–17

HP Heat Pump. xi, 14–16, 45

PEAK Peak oil burner. 16

TUR Turbine. xi, 16, 45

CNN Convolutional Neural Network. xii, xiii, xv, 4, 5, 11, 12, 22, 24, 25, 32, 33, 38, 39, 42, 45

DHP District Heating Plant. v, xi, 1–4, 11, 13, 19, 21, 47

DRL Deep Reinforcement Learning. v, 1–4, 29, 41, 42, 47

DT Digital Twin. 3, 13, 15

EK Expert Knowledge. 3, 23

GAE Generalised Advantage Estimation. 7

MDP Markov Decision Process. 6

MILP Mixed-Integer Linear Programming. v, xi, xii, 1–4, 13, 18, 24–26, 29–33, 41, 42, 47, 48

MLP Multi Layer Perceptron. xii, 32, 33, 42

MWh Megawatt Hour. 13, 16, 21

PDF Probability Density Function. xi, 8–11

PPO Proximal Policy Optimization. v, 3, 5, 7, 8, 11, 13, 23, 41

RL Reinforcement Learning. v, xi, xii, 2, 3, 5, 6, 13, 15, 20, 22, 25, 26, 29–32, 38, 41, 42, 47, 48

Sigholm Sigholm Tech AB. vii, 1, 13, 20, 21, 25

1

Introduction

This thesis was carried out in collaboration with Sigholm Tech AB (Sigholm) with the objective of investigating whether Deep Reinforcement Learning (DRL), trained with week-ahead forecasts, can match the scheduling quality of the Mixed-Integer Linear Programming (MILP) solvers currently used in industry.

1.1 Background

District heating supplies around 9% of global building and industrial heat, roughly 90% of which is currently produced from fossil fuels [1]. The core idea of District Heating Plants (DHPs) is to centralise heat production at one or more plants and distribute hot water through insulated pipes to consumers, replacing thousands of individual furnaces and boilers with a single network. This enables the use of energy sources that are impractical for individual buildings, such as industrial surplus heat, large-scale heat pumps, and municipal waste incineration [2]. Sweden is actually uniquely situated in that it already has very widespread DHP infrastructure [3].

Two features turn operation of DHPs into a non-trivial optimisation problem. First, both heat demand and the price of inputs vary on multiple timescales: heat demand mostly follows daily and seasonal patterns driven by outdoor temperature, while electricity prices are set by market conditions [4]. Second, the thermal accumulator decouples production from consumption in time: heat produced when electricity is cheap or demand is low can be stored and discharged later when conditions are less favourable. This load-shifting capability is widely exploited in modern DHPs to reduce operating cost, smooth equipment operation, and absorb variability from renewable electricity [5], [6]. Plants also retain peak units, typically oil burners, that are expensive to run most of the year but are needed to cover demand on the coldest days when cheaper base-load units are saturated [2], [7]. An example of how a plant can look can be seen in Figure 1.1. The operating cost of the plant therefore depends not only on *which* units are used but on *when* they are used relative to forecasted prices, demand, and weather.

The problem this thesis addresses is one of scheduling: given forecasts of demand and prices over the coming week, decide for each hour and each unit how much heat to produce, in order to minimise operating cost subject to demand-satisfaction and unit-level constraints. In current industrial practice this problem is formulated as a MILP problem and solved with commercial branch-and-bound solvers such as

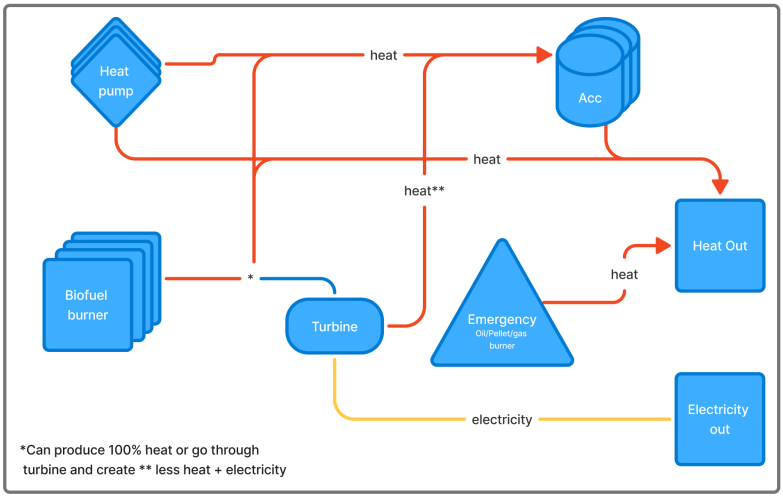

Figure 1.1: DHP configuration example, with different kinds of heat producers as well as electricity production. All input resources are implicit

Gurobi [8], HiGHS [9], or IBM CPLEX [10], [11]. MILP is the natural fit because the relevant structure is largely linear: fuel costs scale linearly with consumption, capacity and ramp limits are linear constraints, and on/off and start-up decisions are naturally encoded as binary variables, while nonlinearities such as start-up cost curves can be captured through piecewise-linear approximations. Decades of solver development have made MILP highly practical for problems of moderate size [12].

The trade-off for accuracy is computational. MILP is NP-hard [13], and thus the solve time scales poorly with system size [14]. Methods less limited by computational complexity could create schedules on longer time horizons, and enable running more experiments on the same data.

Reinforcement Learning (RL), and more specifically DRL, presents another approach to solving these DHP planning problems [15]. Instead of evaluating all variables, relations are instead represented by a deep network capable of representing high complexity. This shifts the computational burden offline: training is expensive, but inference is a single forward pass through a neural network, independent of horizon length.

1.2 Prior work

Existing work applying RL to heat and energy scheduling shares three recurring limitations: agents are typically trained without access to future information, optimisation horizons are short (at most a few days), and direct benchmarking against MILP on the same problem instance is rare. This thesis is positioned against each of these limitations in turn.

Franzoso et al. [15] applied DRL to optimise multi-energy systems, with the goal of extracting fundamental correlations between variables from the trained network as a rule-set for optimal decision making. Their approach reduced consumption, costs,

and emissions by 15–20%. However, the authors note that their agent takes only the current state and past experiences into account; it has no ability to access future information and therefore cannot compensate for predicted changes in supply or demand.

Deng et al. [16] applied DRL, including Proximal Policy Optimization (PPO) [17], to a district heating system in Espoo, Finland, with a detailed Digital Twin (DT) and an Expert Knowledge (EK) module to ensure actionability. Their agents outperformed manual baselines as well as set point strategies, however, due to computational constraints, "*only short-term (2-day) optimisation [were] conducted*" [16]. Wang et al. [18] similarly applied PPO to the energy management of a multi-energy building connected to a low-temperature district heating system, demonstrating that PPO is viable for continuous heat scheduling and can outperform classical methods in computational time.

O'Malley and de Mars [19] conducted an extensive comparison of RL and MILP methods for the unit commitment problem in low-carbon power systems, testing both deterministic and stochastic MILP solvers against model-free and lookahead RL agents across a large set of instances. Their results showed that MILP methods decisively outperformed RL in solution quality, and that significant improvements in RL scalability and reliability are still required. Notably, they found that a hybrid approach using RL to warm-start the stochastic MILP outperformed both methods individually, suggesting that the two paradigms can be complementary rather than competitive. Their finding directly motivates the need for honest, instance-level benchmarking: claims that RL can replace MILP must be evaluated against the same MILP solution on the same problem, not against weaker baselines.

Taken together, the reviewed literature establishes a pattern in which each of the three limitations above is typically present.

1.3 Objective and problem statement

The purpose of this thesis is to investigate whether DRL is a viable alternative to MILP for week-ahead scheduling in DHPs. Concretely, the problem to be solved is: given hourly forecasts of electricity price and heat demand for the coming week, decide for each hour and each controllable unit how much heat to produce, in order to minimise operating cost subject to demand-satisfaction and unit-level constraints.

The central research question is:

> *Can a DRL agent, trained with week-ahead forecast observations, produce schedules that are competitive with those produced by a MILP solver on the same problem instance?*

This question is motivated by a broader practical concern. MILP solvers are the current industry standard for DHP scheduling, but their solve time scales poorly with system size and horizon length, forcing operators to shorten horizons at the cost of under-utilising thermal storage. If DRL can match MILP, it opens the door to replacing or augmenting MILP in production systems and ultimately longer

planning horizons, faster inference, and possibly new strategies utilizing uncertainty in forecasts. Establishing instance-level competitiveness on representative configurations is a necessary first step toward that broader goal; it is not, by itself, sufficient to claim general replaceability, which would require evidence across a wider range of DHPs.

The specific contributions of this thesis are:

1. A simulation environment for two DHP configurations of differing complexity (*Household* and *Town*), together with a reward function aligned with the objective function of an industrial MILP formulation.
2. A DRL agent that takes week-ahead price and demand forecasts as first-class observations, with a convolutional encoder for the forecast time series.
3. Empirical ablations on the action distribution (Gaussian, Beta, Kumaraswamy), the encoder architecture (Convolutional Neural Network (CNN) vs. raw input), and the mechanism for encouraging/enforcing demand satisfaction (penalty-based vs. active enforcement).
4. A transfer-learning study examining whether forecast representations learned in one plant configuration transfer to another.
5. A direct, instance-level benchmark of the trained agents against a rolling-horizon MILP on identical validation data spanning multiple weather scenarios.

2

Theory

This chapter introduces the theory needed to understand the methods and results presented in this thesis. It covers the fundamentals of RL and the PPO algorithm, the action distributions used by policy gradient methods, and CNN together with transfer learning.

2.1 Reinforcement Learning

RL is a framing of learning from interaction to achieve a goal [20]. The, so called, agent interacts with the system, called environment, to receive a feedback signal. This signal is usually called the reward, this interaction can be seen in Figure 2.1.

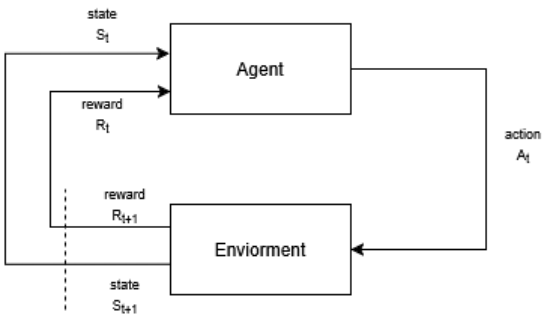

Figure 2.1: The agent-environment interaction in RL [20].

More concretely, an RL agent works in discrete timesteps $t = 0, 1, 2, ...$, where at every time step the agent receives a state $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of all possible states. The agent then takes an action $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of available actions for the agent in state S_t. Action A_t is sent to the environment which, after performing said action, gives a reward R_{t+1} and updates the state to S_{t+1} [20].

At each time step the agent utilises a mapping from states to probabilities of selecting each action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(a|s)$ is the probability that action $A_t = a$ is chosen at state $S_t = s$. An RL method describes how the agent changes and adapts its policy as a result of its experience interacting with the environment. The agents goal can roughly be described as maximizing the total reward the agent receives in the long run [20].

Formally, RL is defined by the Markov Decision Process (MDP), which formalizes the idea that the environment's future evolution depends only on the current state and action, not on the history of how that state was reached. This is known as the *Markov property*:

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_0, A_0, \ldots, S_t, A_t). \tag{2.1}$$

Formally, an MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{P}$ is the state-transition probability function, $\mathcal{R}$ is the reward function, and γ is the discount factor. The dynamics of the MDP are fully captured by:

$$p(s', r \mid s, a) = \mathcal{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a). \tag{2.2}$$

This function describes the probability p of transitioning to state s' and receiving reward r, given that the agent took action a in state s [20].

2.1.1 Discounted Return

At t, the agent receives reward R_{t+1} from the environment. Rather than summing all future rewards, a discount factor $\gamma \in [0, 1)$ is introduced to weight near-term rewards more heavily than distant ones. The discounted return G_t is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.3}$$

Intuitively, γ controls how "far-sighted" the agent is. A value close to 0 makes the agent "near sighted", only caring about immediate rewards, while a value close to 1 makes the agent consider long-term consequences more equally. The discount factor also ensures mathematical convergence of the infinite sum for bounded rewards [20].

2.1.2 Policy and Value Functions

To evaluate how good a given state or action is under π, two value functions are commonly defined.

The *state-value function* $V^\pi(s)$ expresses the expected discounted return when starting from state s and following π thereafter:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \tag{2.4}$$

The *action-value function* $Q^\pi(s, a)$ similarly expresses the expected return, conditioned on taking action a first:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]. \tag{2.5}$$

The difference between these two quantities is known as the *advantage function*:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{2.6}$$

The advantage $A^\pi(s, a)$ captures how much better action a is compared to the average action the agent would take in state s. This quantity plays a central role in policy gradient methods, as it provides a lower-variance signal then using only $Q^\pi(s, a)$ function for updating the policy [20].

2.1.3 Proximal Policy Optimization

PPO is a policy gradient algorithm introduced by Schulman et al. [17] that aims to improve training stability by limiting how much the policy can change in a single update. Unconstrained policy updates can be destructive; a large step in parameter space may drastically worsen the policy, from which recovery is slow or impossible.

PPO addresses this by optimizing a clipped surrogate objective. Let $\hat{r}_t(\theta)$ denote the probability ratio between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$:

$$\hat{r}_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{2.7}$$

The clipped objective $L^{\text{CLIP}}(\theta)$ is then

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\hat{r}_t(\theta)\, \hat{A}_t,\ \text{clip}(r_t(\theta),\, 1-\epsilon,\, 1+\epsilon)\, \hat{A}_t \right) \right], \tag{2.8}$$

where $\hat{A}_t$ is an estimate of the advantage function and ϵ is a small hyperparameter, typically 0.1–0.2. The clipping prevents $r_t(\theta)$ from moving too far from 1, thereby keeping the new policy close to the old one. Intuitively, if an action was better than expected ($\hat{A}_t > 0$), the update is capped so the policy does not over-commit to it; symmetrically for bad actions.

$\hat{A}_t$ is calculated as the agent interacts with the environment in fixed-length rollouts of T timesteps, collecting trajectories of (s_t, a_t, r_t, s_{t+1}) tuples. After each rollout, advantage estimates $\hat{A}_t$ are computed using Generalised Advantage Estimation (GAE) [21]:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l\, \delta_{t+l}, \qquad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{2.9}$$

where $\lambda \in [0, 1)$ is the GAE parameter controlling the bias-variance trade-off in the advantage estimate. The collected trajectories are then used to update both the actor and critic for K epochs over mini-batches of size M. This process is shown in Algorithm 1.

In practice, PPO combines the clipped policy objective with a value function loss L^{VF} and an entropy bonus $S[\pi_\theta]$:

$$L(\theta) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]. \tag{2.10}$$

The value function loss trains a critic to estimate $V(s_t)$, while the entropy bonus encourages exploration by penalizing overly deterministic policies. c_1 and c_2 are tunable coefficients [17].

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize policy parameters θ_0 and value function parameters ϕ_0
for iteration $= 1, 2, \ldots$ **do**
    for actor $= 1, \ldots, N$ **do**
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
        Store trajectories: $(s_t, a_t, r_t)_{t=1}^{T}$
    end for
    Compute advantage estimates $\hat{A}_t$ for all timesteps
    for epoch $= 1, \ldots, K$ **do**
        Sample mini-batches size $M \leqslant NT$ from collected trajectories
        Update θ by maximizing $L(\theta)$ via gradient ascent
    end for
    $\theta_{\text{old}} \leftarrow \theta$
end for

2.2 Action Distributions in Policy Gradient Methods

In policy gradient methods such as PPO, the actor network outputs the parameters of a probability distribution over the action space. During training, actions are sampled from this distribution, introducing the stochasticity needed for exploration [17]. The choice of distribution has significant implications for both training stability and the quality of the learned policy when the action space is bounded: a Gaussian policy has infinite support and must be clipped to fit a bounded action space, introducing an estimation bias that alternative distributions can eliminate [22].

2.2.1 Gaussian Distribution

The default distribution used to sample actions in PPO is the Gaussian, or Normal, distribution $\mathcal{N}(\mu, \sigma)$ where the mean μ is the state-dependant output of the actors policy and the standard deviation σ is a global learned parameter and not state dependant [17].

The Probability Density Function (PDF) is:

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}},$$

with the mode η being equal to the mean μ.

The Gaussian distribution has infinite support, i.e. is defined on $(-\infty, \infty)$. When actions are bounded, as is the case of this thesis, the sampled actions must be clipped or squashed before being applied to the environment. This introduces a mismatch between the distribution the agent believes it is sampling from and the distribution that is actually being sampled. An example of this can be seen in Figure 2.2, where for $\sigma = 1.5$ a majority of sampled values are outside the bound $[-1, 1]$. This can

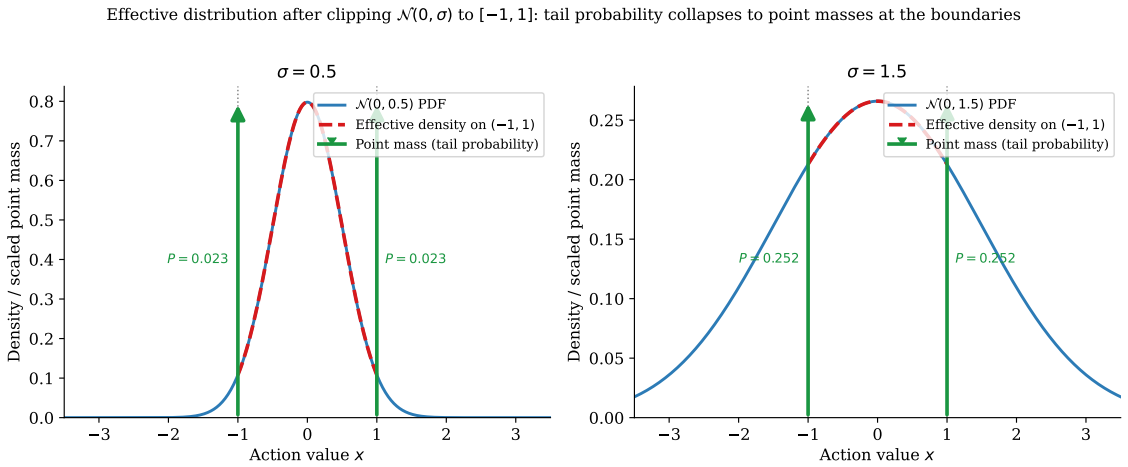

Figure 2.2: Figure showing how, for larger standard deviation, the probability distribution of sampling from a Gaussian distribution with mean zero, bounded on the interval $[-1, 1]$ is very far from the PDF of the Gaussian distribution.

cause a bias in the policy gradient which motivates the use of distribution that are natively defined on a bounded interval [22]. As can be seen in 2.2 the clipping problem gets significantly worse with larger standard deviation σ. Thus, another solution to this clipping problem could be to bound σ such that it is prevented from growing too large.

2.2.2 Beta Distribution

The beta distribution is defined on the interval $(0, 1)$ and is characterized by the two parameters $\alpha > 0$ and $\beta > 0$. It is a natural candidate for bounded continuous action spaces as actions can be sampled from $[0, 1]$ and rescaled to fit the action space without clipping.

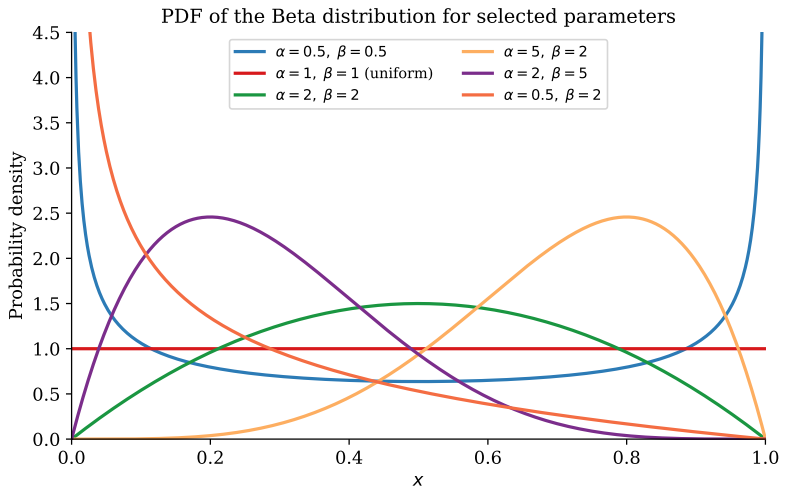

Figure 2.3: Figure showing the PDF of the Beta distribution for some sample α and β

The Beta Distribution is usually understood as the probability of success where $\alpha - 1$ and $\beta - 1$ represent the count of successes and failures respectively from prior knowledge.

For some random variable $x \in [0, 1]$ the Beta PDF is given by:

$$h(x : \alpha, \beta) = \frac{\Gamma(\alpha\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{2.11}$$

where Γ is the Gamma function which extends the factorial to the real numbers [22]. The mode η is given by

$$\eta = \frac{\alpha - 1}{\alpha + \beta - 2} \text{for } \alpha, \beta > 1 \tag{2.12}$$

and since it is undefined when $\alpha \leqslant 1$, $\beta \leqslant 1$, and a well-defined mode for all $\alpha, \beta \in \mathbb{R}^+$ is required, the definition is extended to

$$\eta = \begin{cases} \frac{\alpha-1}{\alpha+\beta-2} & \text{if } \alpha, \beta > 1 \\ 0 & \text{if } \alpha < \beta \\ 1 & \text{otherwise.} \end{cases} \tag{2.13}$$

This roughly matches the behaviour seen in Figure 2.3. Note that this definition defines the mode to 1 when the beta distribution is uniform or mirrored, i.e. $\alpha = \beta$.

For the Beta distribution, all relevant expressions have no closed form solution meaning that their computation load is non trivial.

2.2.3 Kumaraswamy Distribution

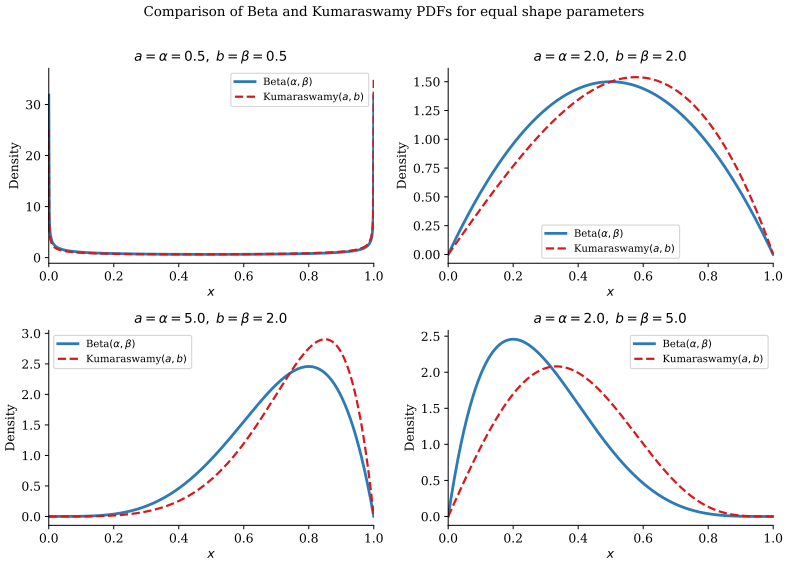

Figure 2.4: Figure showing the comparison of beta and Kumaraswamy distribution for different $a = \alpha$ and $b = \beta$

To mitigate the computational cost of the Beta distribution, the Kumaraswamy distribution is introduced as an alternative [23]. The Kumaraswamy distribution is defined on $(0, 1)$ and exhibits very similar behaviour to the Beta distribution, as can be seen in Figure 2.4. Its key advantage is that its PDF and log-probability all have closed-form expressions, making it significantly cheaper to evaluate than the Beta distribution.

The PDF for some $a, b > 0$ and a random variable x is defined by:

$$h(x : a, b) = abx^{a-1}(1 - x^a)^{b-1}. \tag{2.14}$$

The mode η is given by

$$\eta = (\frac{a-1}{ab-1})^{\frac{1}{a}} \text{for } a, b \geqslant 1, (a, b) \neq (1, 1), \tag{2.15}$$

and extended as:

$$\begin{cases} (\frac{a-1}{ab-1})^{\frac{1}{a}} & \text{if } a, b \geqslant 1, (a, b) \neq (1, 1) \\ 0 & \text{if } a < b \\ 1 & \text{otherwise.} \end{cases} \tag{2.16}$$

2.2.4 Comparison and suitability

Property	Gaussian	Beta	Kumaraswamy
Support	$(-\infty, \infty)$	$(0, 1)$	$(0, 1)$
Requires clipping	Yes	No	No
Closed form log-prob	Yes	No	Yes
Closed form Entropy	Yes	No	No
Closed form PDF	Yes	No	Yes

Table 2.1: Table showing the pros and cons of the three distributions discussed in this thesis

The Gaussian distribution is the default in most PPO implementations due to its simplicity, but the clipping required for bounded actions introduces gradient bias. The Beta distribution resolves this but lacks closed-form expressions for the PDF, log-probability, and entropy. The Kumaraswamy distribution offers a practical middle ground: it closely approximates the Beta distribution's expressiveness while having fully closed-form expressions for log-probability and PDF. For the DHP-scheduling problem in this thesis, where the actions are bounded, both Beta and Kumaraswamy are theoretically preferable to the Gaussian distribution. This does however need to be empirically validated.

2.3 Convolutional Neural Networks

CNNs were first introduced by LeCun et al. [24] and later formalised in the LeNet architecture [25]. Rather than treating an input as a flat vector of independent

features, a CNN applies a set of learnable filters that slide across the input, computing the dot product between the filter weights and each local receptive field. This operation, the discrete convolution, produces a feature map that captures how strongly each spatial or temporal location activates a given filter pattern [24].

For a one-dimensional input sequence $\mathbf{x} \in \mathbb{R}^L$ and a learnable filter $\mathbf{w} \in \mathbb{R}^k$ of size k, the output of a convolutional layer at position t is

$$y_t = \sigma\left(\sum_{j=0}^{k-1} w_j \, x_{t+j} + b\right), \tag{2.17}$$

where b is a bias term and $\sigma(\cdot)$ is a nonlinear activation function. Because the same filter weights $\mathbf{w}$ are applied at every position, the number of parameters is independent of the input length L, which gives CNNs strong inductive biases toward local pattern detection and translation invariance while remaining parameter-efficient [25].

2.4 Transfer Learning

Transfer learning, as described by Pan and Yang [26], refers to a family of methods that improve learning on a target task by transferring knowledge acquired from a source task, rather than training from scratch. The two tasks need not be identical, but a degree of relatedness between their domains is required for the transfer to be beneficial. In the context of deep neural networks, Yosinski et al. [27] show that early layers tend to learn general, broadly applicable features while later layers become increasingly specialised to the source task. This has a practical implication: the weights of early layers can often be reused in a new network trained on a related task, either as a fixed feature extractor or as an initialisation that is subsequently fine-tuned. Their experiments demonstrate that both strategies can improve performance and generalisation compared to random initialisation, with the relative benefit depending on how similar the source and target domains are.

3

Methods

This chapter describes the methods proposed in this thesis. It covers the design of two simulated DHP environments, the reward function and observation space presented to the agent, the training procedure based on PPO, and the evaluation protocol used to compare the trained agents against the MILP baseline.

3.1 Environment design

The design of the environments affects how efficiently the RL agents can learn. This section details and justifies the design decisions made. The systems were based on fictionalised but physically grounded DTs provided by Sigholm, which were made to imitate a small Swedish plant.

Two different environments were modelled: "Household" and "Town". The Household environment is a simpler model with only one unit, while the Town environment is more complex, containing more units and interactions. This was to investigate more general approaches to efficiently learn DHPs.

For both environments, the reward of an action has two components: costs and penalties. Costs are real world costs of consumed materials, such as oil or electricity. Penalties are put in place to regulate behaviour. This includes discouraging illegal actions such as not meeting demand. A portion of the penalties are opt-costs meant to regulate behaviour which is permitted but unwanted. In contrast to other penalties, opt-costs are not arbitrarily decided, but a reflection of real world wear and tear. For the Household and Town environments, these opt-costs are comprised of start costs, stop costs, and rate change costs. All opt-costs were modelled according to existing DTs.

Both environments contain one or more units to generate heat, an accumulator to store the heat, and a heat demand which must be satisfied at all times. The units, accumulator, and consumers are all connected on one grid, where heat can be transferred without loss. The specifics of the environments are described in the following chapters, however generally units consume some resource at a cost and outputs heat with a specified efficiency. Heat is in most cases the level of the unit u, $level_u$, measured in Megawatt Hours (MWhs). Costs have the unit €/MWh. For some resources, the energy contents measured in MWhs are empirically determined rather than theoretically calculated. This leads to some units having energy efficiency

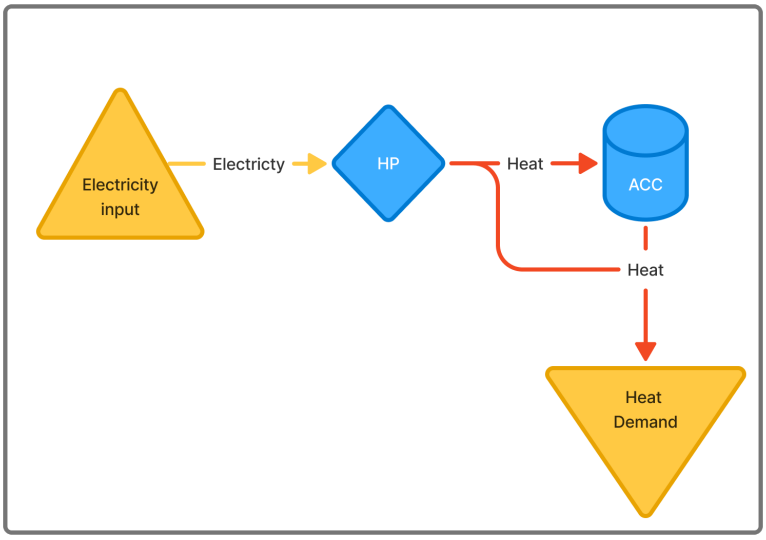

Figure 3.1: A modelled Household with an the **HP** and an the **ACC**

coefficients $\geqslant 1$. This reflects practical innovations improving on the empirically determined value, not a violation of any thermodynamic laws.

3.1.1 Household environment

The Household environment consists of a Heat Pump (**HP**) and an Accumulator (**ACC**). The **HP** consumes electricity, which comes at a varying cost, to generate heat. Heat is either consumed by consumers, by a varying amount, or stored in the **ACC**. The **ACC** has maximum input and output rates. If the **HP** is not able to meet consumers demands, heat is instead drawn from the **ACC**. An illustration of this plant is shown in Figure 3.1.

This system was designed to have only one action: the power level of the **HP**. The **ACC** is acted upon by the rest of the system, consuming or releasing heat to satisfy the balance of the system. Beyond the price of electricity, the **HP** has start and stop costs, which are incurred whenever the **HP** goes from producing some heat to producing no heat, or vice versa. These values are shown in Table 3.1.

The **HP** for the Household environment has a minimum operating level when turned on. The selected approach is to clamp any action value less than the minimum operating value down to zero.

The modelling of the system includes penalties for reaching illegal or infeasible states. For the Household environment, this means overfilling the **ACC** or not meeting

Parameter	Symbol	Value
Change cost **HP**	$c_{\Delta}^{\mathbf{HP}}$	0
Change rate **HP**		inf
Start cost **HP**	$c_{1}^{\mathbf{HP}}$	10
Stop cost **HP**	$c_{0}^{\mathbf{HP}}$	10
Efficiency **HP**	$\eta_{\mathbf{HP}}$	3

Table 3.1: Plant specific parameters used in training for the plant. All values were provided by Sigholm

demand. To ensure that demand is met at all times, the penalty for not meeting demand needs to be large, relative to the per-step cost of meeting the demand. However, large penalties overshadow the differences in price. This means that RL agents quickly learn to satisfy demand, with no regard for the price of electricity. As a method for ensuring feasibility of the solution, an optional soft boundary is added which aims to prevent actions which would put the environment into an unpermitted state. This boundary is referred to as "Active Environment". If an agent attempts to take an action which would overfill the **ACC**, or not meet demand, an action to ensure viability is instead selected. If the **ACC** would overfill, the **HP** is turned off entirely by the environment; if demand would not be met, the **HP** is turned on at maximum power by the environment. The algorithm for this is outlined in Algorithm 2. MaxCharge and MaxDischarge each calculate the maximum input and output respectively to the **ACC** based on the available constraints.

Algorithm 2 Demand satisfaction mechanism for Basic environment

Require: the **HP** level $level_{HP}$, the **ACC** level $level_{ACC}$, current demand d
  if $level_{HP} + \text{MaxDischarge}(level_{ACC}) < d$ **then**
    $level_{HP} \leftarrow HP_{max}$
  else if $level_{HP} + level_{ACC} - d > \text{MaxCharge}(level_{ACC})$ **then**
    $level_{HP} \leftarrow 0$
  end if

3.1.2 Town environment

The provided DT for the Town environment models several units. Some of these units are removed or modelled as passive attributes of other units to simplify the environment without affecting the action space. The resulting Town environment is described below.

As with the Household environment, there is varying demands, and electricity costs, along with an **ACC** with input and output limitations. Beyond this, four units are modelled:

- **HP**: a unit consuming electricity to generate heat;
- Biofuel burner (**BIO**): a unit burning biofuel to produce steam, generating some heat in the process;

- Turbine (**TUR**): a unit which consumes steam and generates either electricity and heat, or only heat via a heat exchanger;

- Peak oil burner (**PEAK**): a unit consuming oil to generate heat.

An illustration of this system is shown in Figure 3.2. Additionally, some limitations

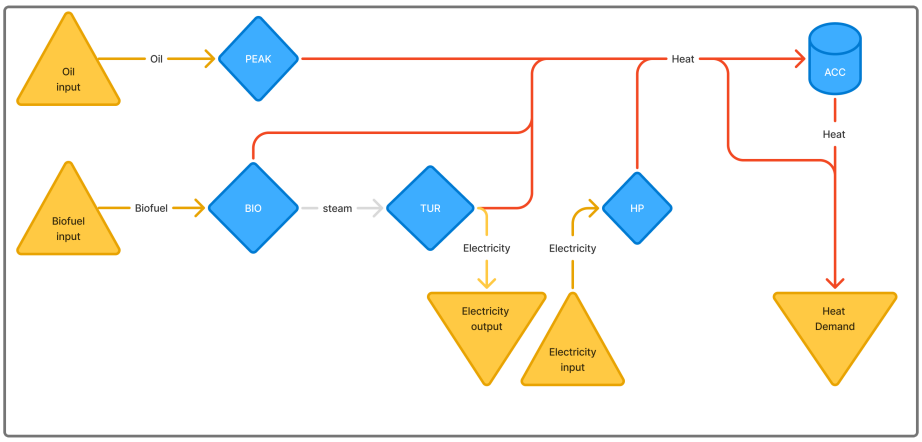

Figure 3.2: The Town plant. The **BIO** outputs both heat and steam, the latter of which is later consumed by the **TUR**

.

are modelled on the units: the **PEAK** has a rate change opt-cost, applied for each unit of energy that the operating level is changed; the **BIO** has a rate change opt-cost, similar to the **PEAK**, along with a rate change limit that limits the allowed change in operating level over one hour.

When the **BIO** generates heat, it also generates steam at a factor of $\frac{1}{0.18}$. This steam is later consumed by **TUR**. The action of the **TUR** is the fraction of steam to run through the turbine, rather than through the internal heat exchanger. the **TUR** has an efficiency of 1, and a set maximum number of MWh units which can be turned into electricity rather than heat.

As with the Household environment, an optional boundary, later referred to as "Active Environment", can be activated to prevent infeasible solutions. Due to the rate change limits of the **BIO** along with its interactions with **TUR**, only the **HP** and the **PEAK** will attempt to compensate for an over- or under-production. Both of these units will attempt to reduce or increase their operating level only by the necessary amount.

Parameter	Symbol	Value
Change cost **BIO**	$c_{\Delta}^{\mathbf{BIO}}$	50
Change cost **TUR**	$c_{\Delta}^{\mathbf{TUR}}$	0
Change cost **HP**	$c_{\Delta}^{\mathbf{HP}}$	10
Change cost **PEAK**	$c_{\Delta}^{\mathbf{PEAK}}$	7
Change rate **BIO**		20
Change rate **TUR**		inf
Change rate **HP**		inf
Change rate **PEAK**		inf
Start Cost **BIO**	$c_{s}^{\mathbf{BIO}}$	10000
Start Cost **TUR**	$c_{s}^{\mathbf{TUR}}$	15000
Start Cost **HP**	$c_{s}^{\mathbf{HP}}$	100
Start Cost **PEAK**	$c_{s}^{\mathbf{PEAK}}$	10000
Oil fuel Price		1152
Bio fuel Price		400
Efficiency **BIO**	$\eta_{\mathbf{BIO}}$	$0.18 \cdot 0.87$
Efficiency **TUR**	$\eta_{\mathbf{TUR}}$	1
Efficiency **HP**	$\eta_{\mathbf{HP}}$	3
Efficiency **PEAK**	$\eta_{\mathbf{PEAK}}$	0.9

Table 3.2: Plant specific parameters used in training for the Town plant. All values were provided by Sigholm. For **BIO**, the efficiency values are the efficiencies for steam and heat generation respectively.

Algorithm 3 Demand satisfaction mechanism for Town environment

Require: Residual imbalance δ
1: **for** $u \in [\textsc{Peak}, \text{HP}]$ **do**
2: **if** $\delta = 0$ **then**
3: **return**
4: **end if**
5: $\delta \leftarrow u.resolve(\delta)$
6: **end for**

The Active environment process is described in Algorithm 3. Here, the residual imbalance δ is the sum of heat produced by all units, including the accumulator as resolved by the system. If the actions and the **ACC** by themselves are able to satisfy the system balance, then $\delta = 0$. The *resolve* algorithm for the units is described in Algorithm 4. Here min_u and max_u is the minimum and maximum operating level of the unit u respectively. Line 10 in Algorithm 4 updates the operating level of the unit u.

Algorithm 4 Resolve function for unit u

Require: Residual imbalance δ

$\ell' \leftarrow \textsc{Clip}(level_u - \delta,\ 0,\ max_u)$
if $\ell' < min_u$ **then**
    if $\delta > 0$ **then**
        $\ell' \leftarrow 0$
    else
        $\ell' \leftarrow min_u$
    end if
end if
$\Delta \leftarrow level_u - \ell'$
$level_u \leftarrow \ell'$
return $\delta - \Delta$

Note that Algorithm 3 does not guarantee demand satisfaction in all cases. In cases where demand is not met, a penalty as described in Section 3.2.4 is given.

3.1.3 Weather Generation

To generate realistic synthetic data price and demand data, at its base, a proprietary weather scenario generator is used to produce synthetic but physically plausible perturbations of historical weather data. Given a base weather time series, the generator injects a configurable number of discrete weather events into the data, producing N distinct scenarios for testing, and generating data series of arbitrary lengths.

3.2 Reward Function

For an agent to learn effectively, it is important that there is a clear link between what it observes, the actions it takes, and the reward it receives. As the primary goal of this project is to match the performance of existing MILP-based systems, the reward function is designed to mirror the objective function and penalty structure of the MILP solver. In general, the optimization objective can be understood as minimizing material consumption (electricity, oil, pellets, etc.) while satisfying heat demand and maintaining operational stability. Additional penalty terms for overflow and underflow of production were introduced to enforce the strict supply-demand balance required by the MILP solver.

3.2.1 Start and Stop Cost

The start and stop opt-costs r_1^u penalizes the agent for starting a unit u from a standstill, i.e. transitioning from a fully off-state to any positive operational level, or turning a unit off. This opt-cost reflects real-world startup costs such as fuel ignition, thermal ramping, and mechanical wear associated with cold starts. It is defined as:

$$r_1^u = \begin{cases} c_1^u & \text{if } level_u(t-1) = 0 \text{ and } level_u(t) > 0 \\ c_0^u & \text{if } level_u(t-1) > 0 \text{ and } level_u(t) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where c_1^u and c_0^u are the fixed start cost and stop cost respectively for unit u. By penalizing unnecessary startups and shutdowns, this term encourages the agent to prefer strategies that keep units running stably rather than switching them on and off repeatedly, which would be operationally undesirable.

3.2.2 Change Cost

The change cost r_2^u penalizes changes in the operational level of a unit u between consecutive timesteps. While the start and stop costs capture the binary event of a unit switching on or off, the change cost captures the magnitude of any adjustment, whether the unit is ramping up or down. It is defined as:

$$r_2^u = |level_u(t-1) - level_u(t)| \cdot c_\Delta^u \tag{3.2}$$

where c_Δ^u is the per-unit change cost coefficient for unit u. This term promotes smooth, stable control policies and discourages erratic behaviour that would be impractical in a real DHP.

3.2.3 Material Consumption Cost

The material consumption cost r_3^u captures the direct operating cost of running unit u at a given level at time t. This is the primary cost term in the reward function, as reducing fuel and energy consumption is the central objective of the optimization. It is defined as:

$$r_3^u = \frac{level_u(t)}{\eta_u} \cdot p_u \tag{3.3}$$

where η_u is the efficiency of unit u and p_u is the current price of the material consumed (e.g., electricity, gas, or pellets). The efficiency term accounts for the fact that units do not convert input energy to heat output perfectly; a lower efficiency requires more material input to achieve the same heat output, and is therefore penalized more heavily.

3.2.4 Underflow Penalty

The underflow penalty r_4 is incurred when total heat production falls short of demand. Failing to meet demand is the most critical failure mode in a DHP, as it directly impacts end users. It is defined as:

$$\Delta_h = demand(t) - \sum_u heat_u(t) \tag{3.4}$$

$$r_4 = \max(0,\ \Delta_h) \cdot c_u + \begin{cases} k_u \text{ if } \quad \Delta_h > 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{3.5}$$

where c_u is the underflow penalty coefficient and k_u the underflow constant. The underflow is modelled to be proportional to the amount of heat not met.

3.2.5 Overflow Penalty

The overflow penalty r_5 is incurred when the total heat production makes it so there is more heat produced then can be covered by the demand or stored in the accumulator. This type of failure is critical as it can be very hard and and expensive to get rid of excess heat and might cause critical issues. It is described as:

$$r_5 = \begin{cases} k_o \text{ if } \quad \Delta_h < 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{3.6}$$

where k_o is the overflow constant.

3.2.6 Total Reward

The total reward at each timestep is the negated sum of all cost and penalty terms, converting the cost minimization problem into the reward maximization objective used by RL algorithms:

$$\mathcal{R}(t) = -\left(\sum_u (r_1^u + r_2^u + r_3^u) + r_4 + r_5\right) \tag{3.7}$$

3.3 Observation Space

For the agent to make informed decisions, it must have access to both the current state of the plant and relevant forecasts of future conditions. The observation space is therefore designed to reflect the information available to the MILP: Knowledge of what each unit is currently doing, how much energy is stored in the accumulator, and projections for electricity prices and demands over the coming week.

The information available to the agent can be divided into two categories: *forecast signals* (P_e, D), which inform the agent about future conditions it should plan for, and *state signals* (P_o, P_p, S_l), which describe the current physical and economic state of the plant. The full state is described in detail in the following subsections.

3.3.1 Price Predictions

Electricity price predictions are generated using a proprietary forecasting model provided by Sigholm, which takes weather data as input and outputs hourly electricity prices scaled to the SE3 (Southern Swedish bidding zone) spot price area. The model

captures the dependence of electricity prices on weather conditions, reflecting how factors such as temperature and solar radiation influence both supply (e.g. renewable generation) and demand in the Nordic electricity market.

$$P_e(t) = \{p_t,\ p_{t+1},\ \ldots,\ p_{t+h-1}\} \tag{3.8}$$

where $h = 168$ corresponds to a one-week lookahead horizon. Providing the agent with a full week of price predictions allows it to learn to shift production load towards low-price periods and reduce consumption during price peaks.

As no prediction models were provided for the price of oil or pellets they are modelled as static scalars in the state:

$$P_o = \{p_o\}, P_p = \{p_p\}, \tag{3.9}$$

where p_o and p_p is the price of oil and pellets given the values $p_o \approx 1100$ €/MWh and $p_p \approx 400$ €/MWh respectively (They are approximated here to not leave out industry secrets from Sigholm).

3.3.2 Demand Predictions

Heat demand predictions are generated by a proprietary demand forecasting model trained on historical weather data and real measured heat demand from a DHP analogous to the Town plant configuration. The model therefore captures both the seasonal and short-term dependence of heat demand on outdoor temperature and other weather variables.

Similarly to the price predictions, the demand observation at timestep t is a forward-looking window of predicted hourly demand values:

$$D(t) = \{d_t,\ d_{t+1},\ \ldots,\ d_{t+h-1}\} \tag{3.10}$$

where $h = 168$ corresponds to the same one-week lookahead horizon used for price predictions. Exposing the agent to future demand allows it to anticipate periods of high load and plan unit commitment accordingly, rather than reacting only to the current demand signal.

3.3.3 Unit State

In addition to the forecast signals, the agent must be aware of the current physical state of the plant in order to make informed decisions. This includes the operational level of each controllable unit as well as the current charge level of the **ACC** accumulator, both of which are encoded as scalar values in the observation space.

The unit and global state observation is defined as:

$$S_l = \left(\bigcup_{u \in \mathcal{U}} \{level_u(t)\} \right) \cup \{level_{acc}(t)\} \tag{3.11}$$

where $level_u(t) \in [0, 1]$ is the normalised operational level of unit u at time t, and $level_{acc}(t) \in [0, 1]$ is the normalised charge level of the accumulator. Including the accumulator state is important as it represents stored thermal energy that can be discharged during peak demand or high-price periods, and the agent must learn to manage it in conjunction with the unit schedules.

3.3.4 Total State

The complete observation presented to the agent at each timestep t is the concatenation of all signal components described above:

$$\mathcal{S}(t) = [P_e(t);\ D(t);\ P_o; P_p;\ S_l(t)] \tag{3.12}$$

The total observation vector has a fixed dimensionality of:

$$|\mathcal{S}| = \underbrace{h}_{P_e} + \underbrace{h}_{D} + \underbrace{1}_{P_o} + \underbrace{1}_{P_p} + \underbrace{|\mathcal{U}| + 1}_{S_l} = 2h + |\mathcal{U}| + 3 \tag{3.13}$$

for $h = 168$ and the Town plant with $|\mathcal{U}| = 4$ controllable units, this yields a total observation size of $|\mathcal{S}| = 2 \times 168 + 4 + 3 = 343$.

3.3.5 Encoding Observations

The forecast signals $P_e(t)$ and $D(t)$ are time series of length $h = 168$. To extract meaningful temporal patterns, a separate CNN encoder is used to compress the forecast data into a latent representation before it is passed to the policy network.

This idea draws from two established uses of CNNs: their successful application as visual encoders in image-based RL [28], where a CNN compresses raw pixel observations into compact state representations before the policy network, and their strong empirical performance on sequence modelling tasks, where 1D convolutions have been shown to match or outperform recurrent architectures for capturing local temporal patterns in time series data [29].

The two forecast signals are stacked as separate channels, forming an input tensor of shape $[2, h]$, analogous to a two-channel time series image. The CNN processes both channels jointly, allowing it to learn cross-signal relationships. The encoder outputs a single latent vector of size V:

$$z(t) = \mathrm{CNN}_\theta([P_e(t),\ D(t)]), \quad z(t) \in \mathbb{R}^V \tag{3.14}$$

The final observation fed to the policy network is then the concatenation of this latent vector with the scalar state signals:

$$\tilde{\mathcal{S}}(t) = [z(t);\ P_o;\ P_p;\ S_l(t)], \quad \tilde{\mathcal{S}}(t) \in \mathbb{R}^{V+|\mathcal{U}|+2} \tag{3.15}$$

The scalar signals P_o, P_p, and $S_l(t)$ are not passed through the encoder, as they carry no temporal structure that convolution could exploit. The encoder weights θ are trained end-to-end with the policy, allowing the latent representation to be shaped by the reinforcement learning objective rather than a fixed reconstruction loss.

To investigate whether the encoder learns a representation that generalises across plant configurations, experiments were conducted in which the encoder was first trained on the basic environment, then frozen and transferred to the medium environment. The motivation is that both environments share the same forecast signals, so a well-trained encoder may learn features that are plant-agnostic, such as price peaks, demand ramps, or high-cost coincidence events. If transferring a frozen encoder incurs no significant loss in performance compared to training end-to-end from scratch, it would reduce the effective training time in the medium environment, and by extension possibly other untested configurations, by eliminating the need to re-learn the forecast representation.

3.4 Training

Training is performed using the PPO algorithm.

The full training procedure follows the weather curriculum described in Section 3.1.3: intensity is ramped progressively from *low* to *extreme* over the course of training, exposing the agent to increasingly challenging scenarios before evaluation. The complete set of hyperparameters is reported in Table 3.3.

3.4.1 Gaussian Clipping in continuous actions

Actions are sampled from a Gaussian distribution $\mathcal{N}(\mu_u, \sigma_u)$ per unit, with μ_u output by the actor network and σ_u a globally learned parameter. Sampled actions are clipped to $[-1, 1]$ before being passed to the environment. As discussed in Section 2.2, this clipping introduces a mismatch between the distribution the agent believes it is sampling from and the distribution of actions actually applied. While bounded distributions such as Beta or Kumaraswamy would eliminate this bias, the Gaussian distribution was retained for its simplicity.

The main issue regarding clipping that was registered during the process of this project was that it caused σ_u to grow exponentially, and at large σ_u the training became unstable. This was solved by bounding σ_u by $\ln(\sigma_u) \leqslant 2$. When the Gaussian distribution is mentioned in the result it should be assumed that its standard deviation is clipped unless otherwise stated.

3.4.2 Active Environment or Penalties

Experiments were conducted to explore whether a more "active" environment, one that directly enforces constraints rather than penalising violations, would lead to better training outcomes. This idea was partly inspired by Deng et al.[16] and their use of an EK module to enforce feasibility, and partly motivated by a practical issue

Table 3.3: PPO training hyperparameters.

Hyperparameter	**Symbol**	**Value**	**Value from**
Discount factor	γ	0.99	From [17]
GAE parameter	λ	0.95	From [17]
Training steps		10,000,000	Searched
PPO clip parameter	ϵ	0.1	Searched
Learning rate		$0.0001 \rightarrow 0.000001$	Searched
Rollout length	T	2048	Searched
Mini-batch size	M	512	Searched
Update epochs	K	10	From [17]
Entropy coefficient	c_2	$0.2 \rightarrow 0.0001$	Searched
Value loss coefficient	c_1	0.75	Searched
CNN latent size	V	256	Searched
Action distribution		Gaussian $\mathcal{N}(\mu_u, \sigma_u)$	From [17]

observed during early experiments: agents consistently failed to achieve 100% demand satisfaction during validation, making it difficult to perform a fair comparison against MILP, which satisfies demand by construction.

Two approaches were therefore evaluated:

Penalty-based: unmet demand is penalised via the reward term r_4, and the agent is left to discover demand satisfaction through optimisation. This preserves the agent's autonomy but offers no hard guarantee of demand satisfaction.

Forceful: after the agent's actions are resolved, a post-processing step corrects any residual imbalance by adjusting unit outputs directly, as described in Algorithm 3. This guarantees demand is met at every timestep regardless of the agent's policy.

The penalty-based approach gives the agent a meaningful learning signal for demand satisfaction, but risks the agent settling into a policy that accepts occasional shortfalls when the penalty is outweighed by savings elsewhere. The forceful approach eliminates shortfalls entirely, but because the correction is applied silently outside the reward signal, the agent receives no direct gradient information about why a correction was needed. The agent simply observes a different cost than it would have incurred had it acted correctly. In practice this means the agent may become reliant on the mechanism rather than internalising demand satisfaction as part of its policy.

Note that for validation "force demand met" is always turned on to increase probability of demand being met at all times.

3.4.3 Transfer Learning

The application of transfer learning was dependent on both environments. A CNN, *source*, was trained on one environment. After this, a new CNN network, *target*, was created to be trained on a different environment. This entailed a new observation- and action-space. However, the time series observations fed to the CNN layers would be the same. The weights of the convolutional layers were copied from the *source* to

the *target*, and these layers were subsequently frozen. The linear projection layer at the head of the CNN was left randomly initialised, allowing the *target* to learn how to map the transferred features to its new observation space. The optimizer was then rebuilt to exclude the frozen parameters from training.

3.5 Validation

To validate the RL agent, this paper aims to compare RL generated solution against an optimal solution. Such an optimal solution could be generated by the MILP solver provided by Sigholm.

An issue arises with the comparison with the MILP solver, as it is set to end each run with the accumulator set at a predetermined level a'. This is due to the episodic construction on the MILP problem formulation; if no constraint is set, emptying the accumulator by the end of the run is almost always more efficient. To combat this, the validation set was constructed as one long continuous schedule, reducing the effect of differences in the ending accumulator level.

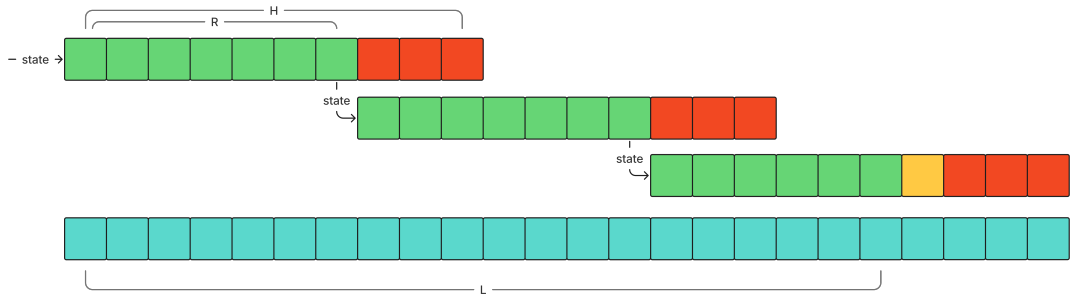

Figure 3.3: Describing the process of taking part of the results generated by one MILP run and stitch it together to form a longer continuous MILP solution. The top three rows represents one MILP pass each and the bottom row the continuous price series. Green represent the used part of each MILP run while red or yellow is computed but discarded. Red due to R, and yellow due to L.

Using the MILP solver to solve for such an extensive time window would be computationally infeasible due to the inherent complexity of the problem. As such, the validation set was constructed of several runs using a "Rolling Horizon" approach, described in Algorithm 5. In the SCOPE step, the model is updated with the scoped prices and demands. In the ADVANCE step, R values from the solution segment are saved and the accumulator level of the model is updated to reflect the solution. An intuitive example of how this works can be seen in Figure 3.3.

Setting the horizon H to L equates to solving for the full scope. The resolution R controls how much of each solution should be collected. Setting R to H equates to forcing the produced solution to reach an accumulator level of a' at the end of each solution segment. Reducing H improves running time at the cost of total running

Algorithm 5 Rolling Horizon Scheduling

Require: Total schedule length L, Horizon H, Resolution R, Model $\mathcal{M}$ *with* $R \leqslant H \leqslant L$

$n \leftarrow \lceil L/R \rceil$
$\Theta \leftarrow \textsc{ExtractData}(\mathcal{M})$ ▷ Extract all full data series
for $i = 0, 1, \ldots, n-1$ **do**
  $t \leftarrow i \cdot R$
  $M \leftarrow \textsc{Scope}(\mathcal{M}, \Theta, t, H)$ ▷ Restrict model to window $[t,\ t+H]$
  $x^* \leftarrow \textsc{Solve}(\mathcal{M})$
  $M \leftarrow \textsc{Advance}(\mathcal{M}, x^*, R)$ ▷ Commit first R steps of x^*
end for

cost. Reducing R negates the effect of the end accumulator level a' at the cost of running time.

Additionally, during the ADVANCE step the MILP solver is not able to have set starting levels for any units apart from the accumulator. Instead, the MILP solver is able to select the operating level of each unit, regardless of any constraints. Reductions in R thus improves the solution, as it is less bounded by the end accumulator level a' or unit constraints. This can thus increase the returned value beyond feasibility as change rate constraints can be ignored more often. For the Validation sets used in Chapter 4, the values in Table 3.4 were used for L, H, and R.

Name	Description	Value	Note
L	Total length	38,640	230 weeks
H	Horizon	840	5 weeks
R	Resolution	672	4 weeks

Table 3.4: Rolling horizon parameters for Algorithm 5, with $R \leqslant H \leqslant L$.

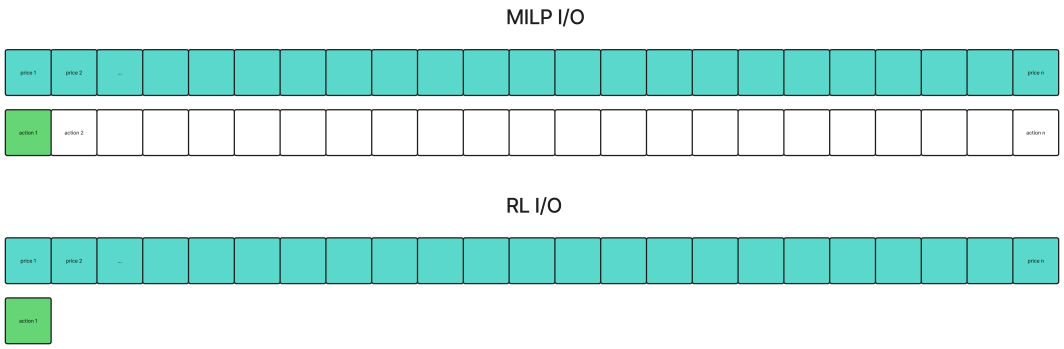

Figure 3.4: Showing the input (Turquoise) and output below as Green (used) and White (discarded) for MILP and RL , in this example the MILP solver and RL has the exact same input and output as all but one value of the MILP solver is discarded.

The main metric used to produce the result is “accuracy”, this is calculated as:

$$Accuracy = \frac{SUM_{MILP}}{SUM_{RL}} \tag{3.16}$$

Where SUM sums the reward or material cost from the entire validation set of length L. The data used for validation is real electricity-, real weather-, and synthetic demand-data.

We differentiate between “total accuracy” and “cost accuracy” where the total accuracy is calculated from the sum of the entire reward and the cost accuracy is calculated only from the commodities (electricity, oil, pellets) part of the reward.

4

Results

This chapter will be introduced with some sample behaviour for the best performing agent on the Household configuration as well as for the Town configuration. Then we show ablations for the methods presented. Each ablation contains five model instances trained with different seeds to ensure realistic confidence bounds. Plant parameters used for the Household plants in the chapter can be seen in Table 3.1. Similarly for the Town configuration in Table 3.2. Hyperparameters used during training of all models can be seen in Table 3.3. When tuning hyperparameters, learning rate and entropy were the most impactful.

4.1 Sample Behaviour of Best Models

This section provides a qualitative, granular view of how the best-performing DRL agents operate against the MILP baseline. The episodes shown here also illustrate samples intended to build intuition for the agents' decision-making characteristics.

4.1.1 Household

This section show results of the best RL agent trained and validated in the Household environment. Figure 4.1 shows the running cost of the RL agent, with a total difference of 8.10% compared to the MILP solver.

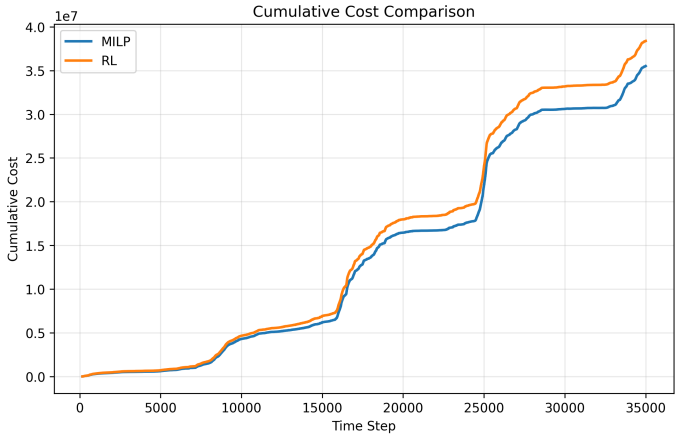

Figure 4.1: The cumulative cost of the RL agent (orange) and the MILP solver (blue). Total running cost of the RL agent is 38, 391, 644. Total running cost for the MILP solver is 35, 515, 079

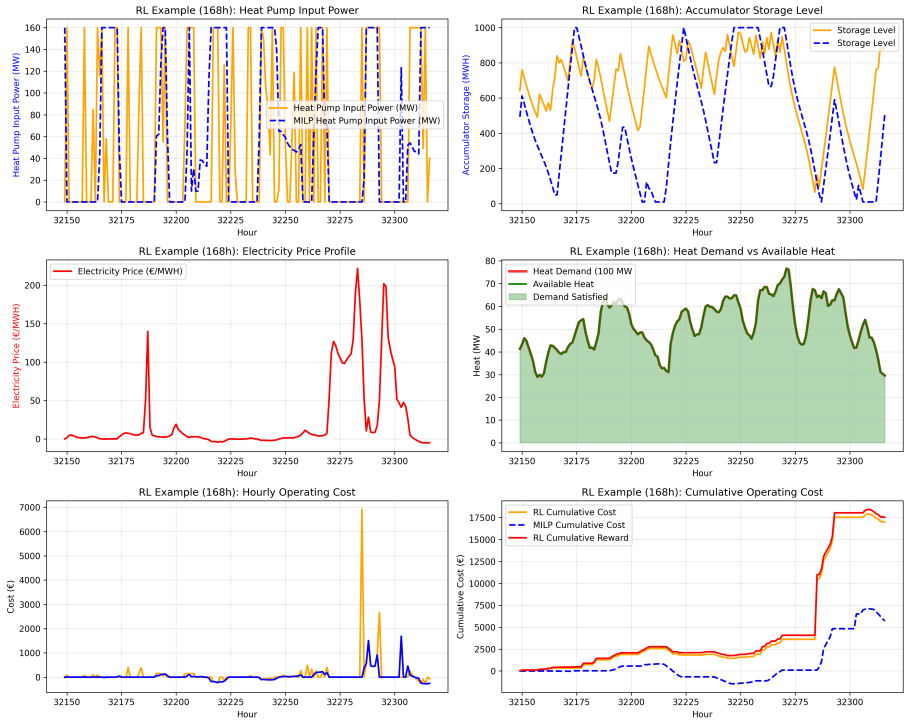

Figure 4.2: Worst-case week (in terms of cost gap to the MILP baseline) from the validation set, shown for the Household plant configuration. *Top left:* Heat pump input power for the RL agent (orange) and MILP (dashed blue); both schedules concentrate operation near the 160 MW upper bound, but the RL agent exhibits more frequent on/off transitions. *Top right:* Accumulator storage trajectories for the two controllers. *Middle left:* Day-ahead spot price profile, dominated by two sharp peaks around hours 32 285 and 32 295. *Middle right:* Heat demand versus available supplied heat; demand is met throughout the episode. *Bottom left:* Hourly operating cost for RL (Yellow) and MILP (Blue). *Bottom right:* Cumulative operating cost for both controllers together with the RL cumulative reward.

Figure 4.2 shows the validation-set week with the largest cost gap between the RL agent and the MILP baseline. Despite this being the worst case for the agent, the two controllers share the same qualitative strategy: both rely heavily on the heat pump near its 160 MW upper bound, and both draw the accumulator down ahead of the price peaks around hours 32 285 and 32 295, with charging concentrated in the preceding low-price window. Heat demand is satisfied throughout the episode for both schedules.

The cost gap is driven almost entirely by behaviour during and immediately around the price peaks. The MILP commits to a longer pre-charge of the accumulator, allowing it to ride through the peak hours with the heat pump off, whereas the RL agent enters the peak with less stored energy and is forced to operate the heat pump at high power during expensive hours, producing the two large cost spikes visible in the bottom-left panel. As a result, the cumulative cost curves diverge sharply at the price peak, and the RL agent ends the week roughly a factor of two above the MILP.

4.1.2 Town

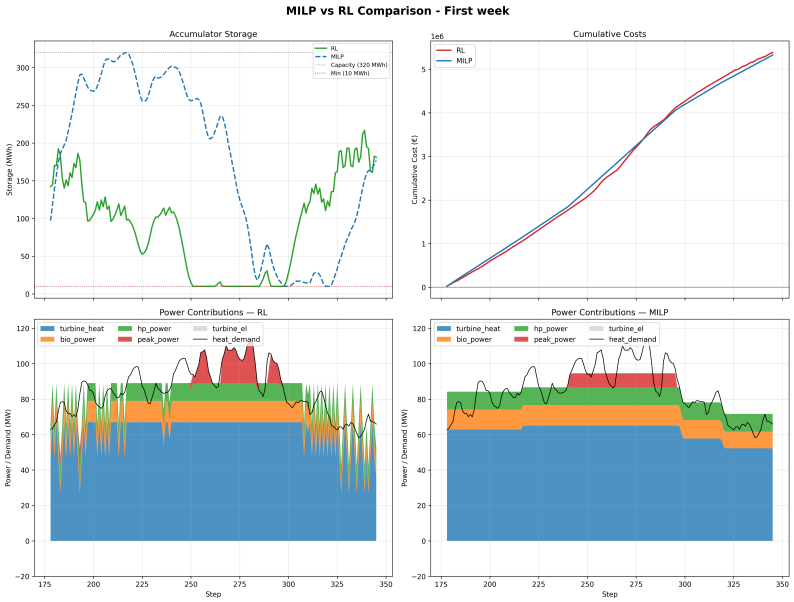

Figure 4.3: Sample week comparison between the best-performing RL agent and the MILP baseline on the Town plant configuration. *Top left:* Accumulator storage trajectories for MILP (green) and RL agent (blue). The MILP exploits a large fraction of the 320 MWH capacity, while the RL agent operates at a consistently lower storage level with higher-frequency fluctuations. *Top right:* Cumulative operating cost over the week; despite the behavioural differences, the two trajectories track each other closely and finish within a small margin. *Bottom:* Per-step power contributions by source for RL (left) and MILP (right). The MILP dispatches units in stable, block-wise patterns, whereas the RL agent switches units at high frequency.

This section show results of the Best RL agent trained and validated in the Town environment. Figure 3.2 is sampled as the worst performing week calculated as largest difference between MILP and RL.

The Town configuration in Figure 4.3 exposes a more nuanced picture. On the aggregate metric cumulative operating cost, the two controllers are nearly indistinguishable over the week, with the RL trajectory closely shadowing the MILP and finishing within a small margin.

A difference in behaviour is reflected in the unit-schedule breakdown in the bottom panels: the MILP holds each generation source on stable, block-wise plateaus, whereas the RL agent toggles units on a step-by-step basis.

Though the behaviour is different, the middle right graph shows that the RL reward is still competitive with the MILP even though it incurs more penalties from the reward function due to the erratic behaviour.

The middle right graph in Figure 4.3 shows that RL crosses the MILP line. This behaviour is explained by looking at the top right graph showing the accumulator. There the RLs accumulator is being charged more than the MILPs during those hours.

4.2 Time Series Encoding

In this section, the comparative difference can be seen between training with a Multi Layer Perceptron (MLP) or encoding the data with a CNN before feeding it to the MLP. The network architectures can be seen in Table 4.1.

Model	**Channels**	**Kernels**	**Policy network**	**Total parameters**
Household_MLP	–	–	$1024 \rightarrow 1024$	2,795,523
Household_CNN	[32, 32, 64, 64, 128]	[5, 5, 5, 5, 3]	$1024 \rightarrow 1024$	2,724,403
Town_MLP	–	–	$1024 \rightarrow 1024$	2,741,769
Town_CNN	[32, 32, 64, 64, 128]	[5, 5, 5, 5, 3]	$1024 \rightarrow 1024$	2,739,721

Table 4.1: Architecture and parameter count for the two RL configurations on both Environment configurations. “Total parameters” count the parameters for both the policy and value networks

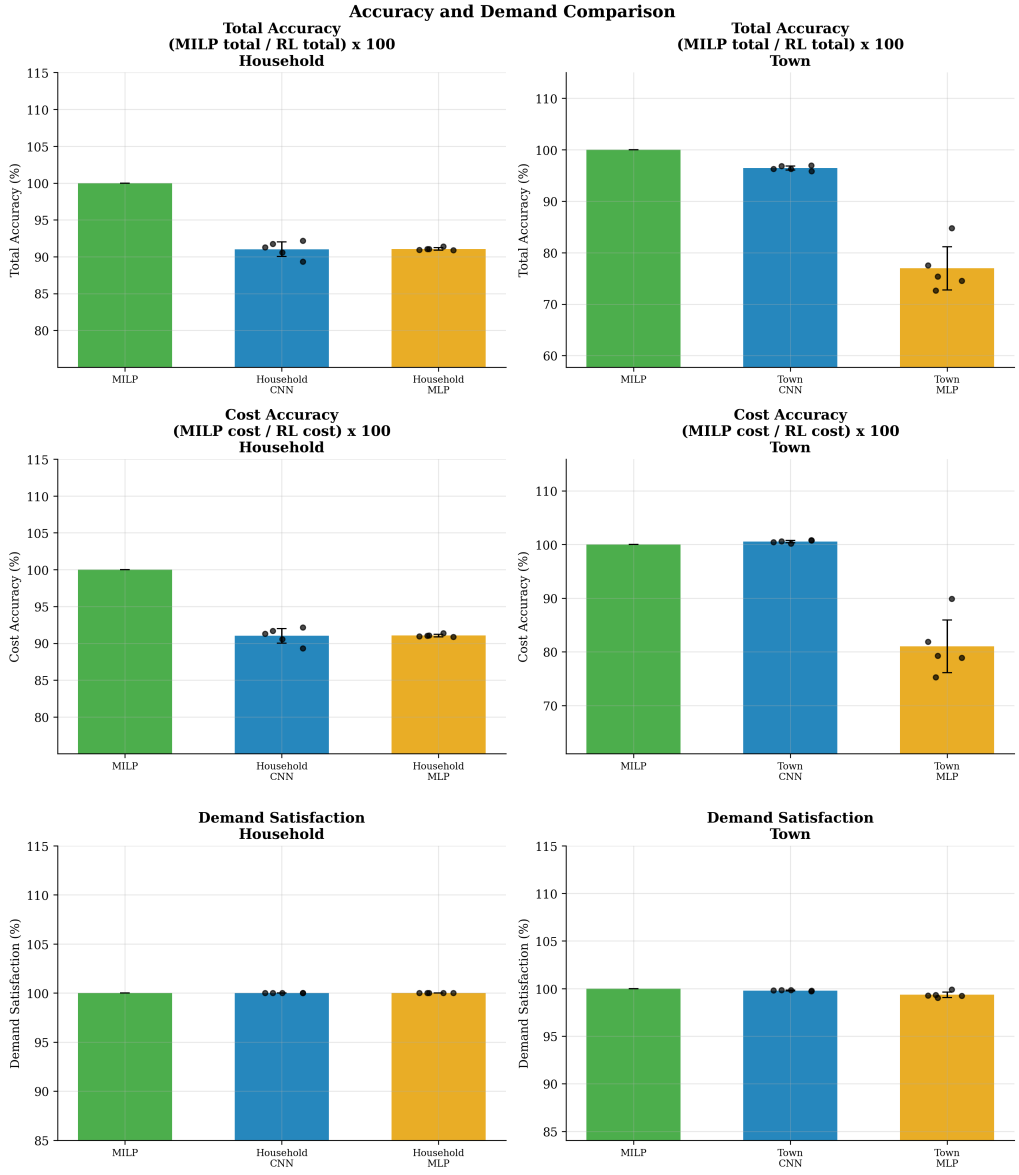

Figure 4.4: Comparison between a CNN encoder and a flat-vector MLP for processing the week-ahead forecast signals, evaluated on both plant configurations. Top: total accuracy relative to the MILP baseline. Middle: cost-only accuracy, isolating the material consumption component of the reward. Bottom: demand satisfaction rate. Left column: Household plant, comparing Household_cnn against Household_mlp. Right column: Town plant, comparing Town_cnn against Town_mlp. Each bar shows the mean over five seeds with 95% confidence intervals; individual seed results are overlaid as dots.

As can be seen in Figure 4.4, the CNN performs far better in the Town environment, as compared to the MLP. This both in total results, and variance. For the Household environment, the CNN outperforms slightly on average, but with far higher variance.

4.3 Distribution Comparison

In this section, results of comparing different distribution for sampling actions during training are shown.

4.3.1 Beta and Kumaraswamy

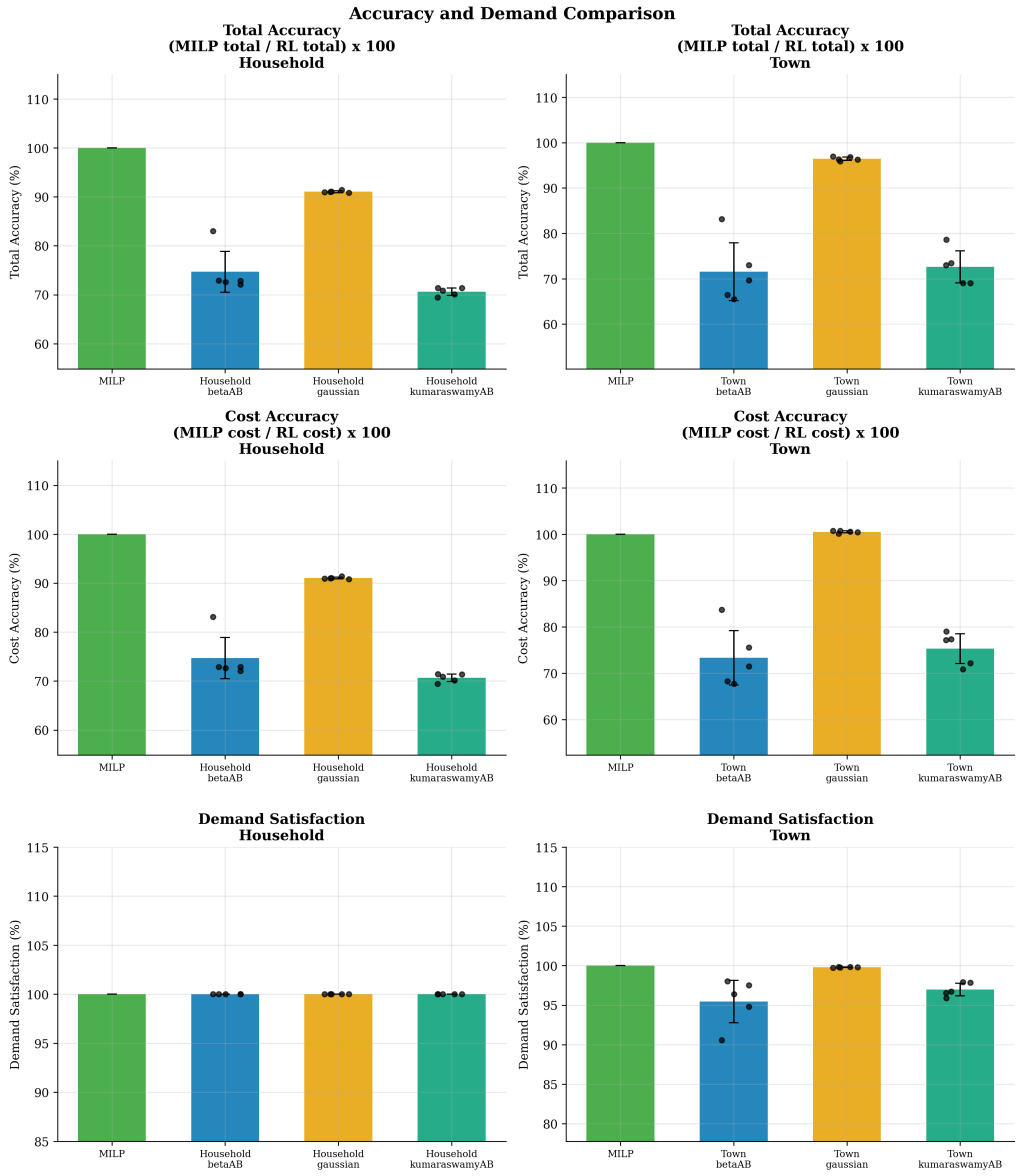

Figure 4.5: Comparison of action distributions, Gaussian (with bounded standard deviation), Beta, and Kumaraswamy. Evaluated on both plant configurations. Top: total accuracy relative to the MILP baseline. Middle: cost-only accuracy. Bottom: demand satisfaction rate. Left column: Household plant. Right column: Town plant. Each bar shows the mean over five seeds with 95% confidence intervals; individual seed results are overlaid as dots.

As shown in Figure 4.5, the Gaussian agent achieves a higher MILP accuracy than both bounded distributions on both plant configurations, despite the theoretical gradient bias introduced by action clipping. On the Household plant, the Gaussian reaches a cost accuracy of 91.1 ± 0.2, compared to 74.7 ± 4.2 for Beta and 70.6 ± 0.8 for Kumaraswamy. The same ordering holds on the Town plant, where the Gaussian achieves 100.2 ± 0.1 against 84.4 ± 2.0 for Beta and 85.7 ± 1.3 for Kumaraswamy. It is also clear that Beta on both environments, and Kumaraswamy on the Town configuration in particular, exhibit notably higher variance across seeds, suggesting less stable training. Demand satisfaction remains high across all three distributions on both plants, so the differences are primarily in cost efficiency rather than feasibility.

4.3.2 Clipped standard deviation in Gaussian distribution

As discussed in Section 2.2.1, the clipping that the Gaussian distribution requires for bounded actions can cause the learned standard deviation σ_u to grow without bound, which in turn worsens the clipping problem and destabilises training. To investigate whether bounding σ_u resolves this, two otherwise identical Gaussian agents were trained in the Household plant: one with $\ln(\sigma_u) \leqslant 2$ (*Clipping*) and one with no bound on σ_u (*Noclip*).

Table 4.2: Validation Accuracy Metrics for Household plant comparing standard deviation clipping or not

Config	Total Acc. (%)	Cost Acc. (%)	Demand Sat. (%)
MILP	100.0	100.0	100.0
Bounded σ	91.1 ± 0.2	91.1 ± 0.2	100.0
Unbounded σ	90.6 ± 0.4	90.4 ± 0.4	100.0

As can be seen in Table 4.2 the clipping configuration slightly outperforms the *Unbounded* configuration, both in mean accuracy and in tightness of the confidence interval. Figure 4.6 shows that the standard deviation in the unbounded case grows toward 300, which is very large given that the action space is $[-1, 1]$. The bounded configuration on the other hand stays at the imposed bound, confirming that the bound is in fact active during training.

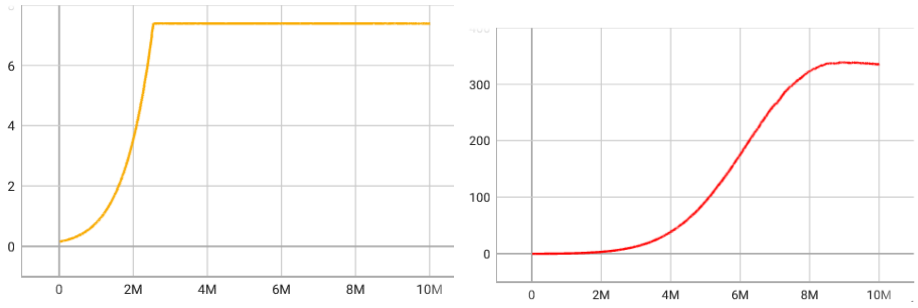

Figure 4.6: Comparison of standard deviation during training for Gaussian where it is bounded (left) and unbounded (right), (right) has sigmoid shape due to aneling of hyper parameters, otherwise it would be exponential.

4.4 Active environment or penalties

This experiment compares the "forceful" formulation, described in Section 3.4.2, with the purely penalty-based formulation in which demand violations are only discouraged through the reward function. Both configurations were evaluated on both plant configurations across five seeds each. The penalty values used during training are listed in Table 4.3.

Affected Penalty	**Variable**	**Value**
Underflow Coefficient	c_u	1500
Underflow Constant	k_u	5000
Overflow Constant	k_o	5000

Table 4.3: Penalty values used during training.

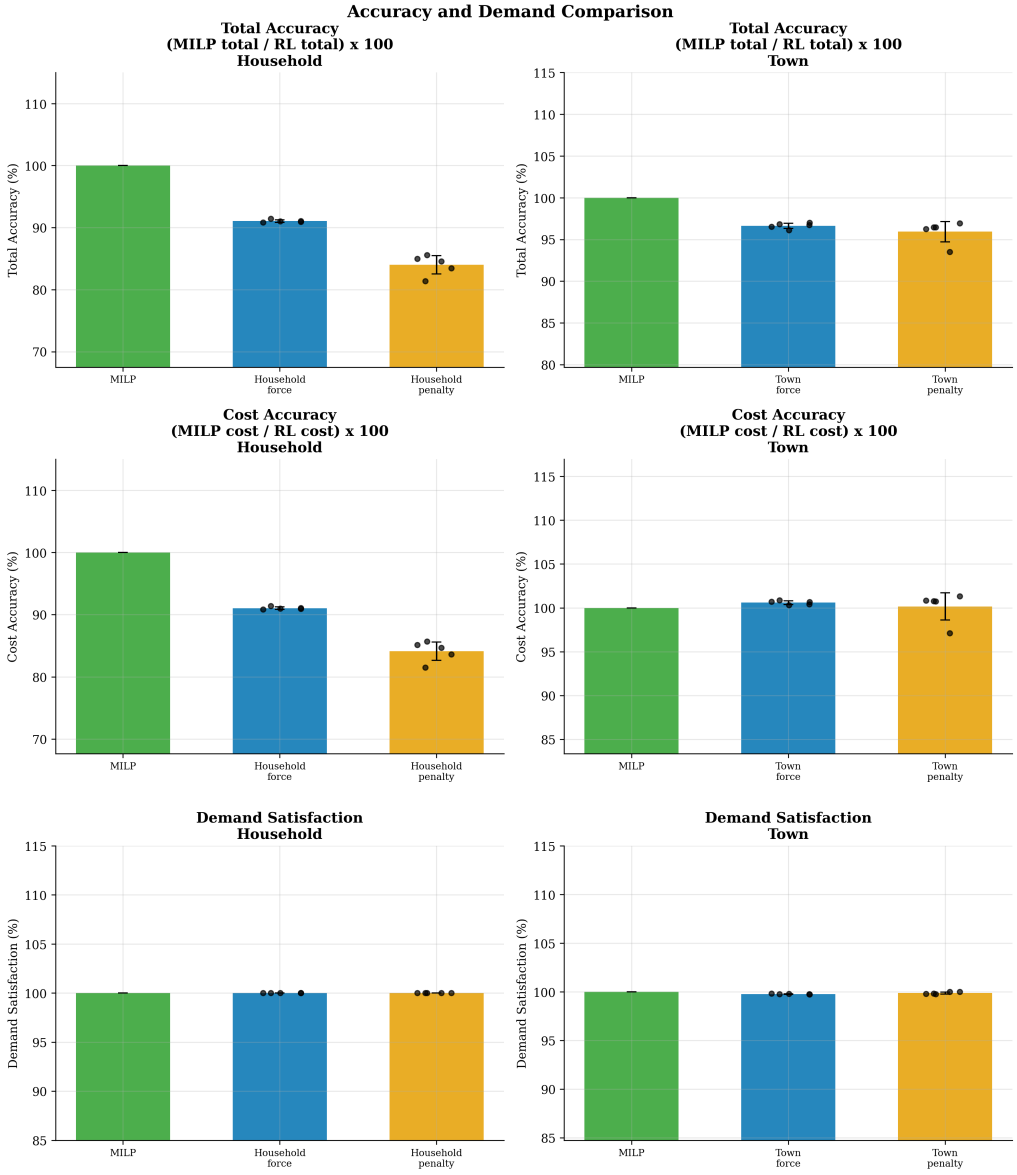

Figure 4.7: Comparison between the "forcefull" and penalty-based formulations for enforcing demand satisfaction, evaluated on both plant configurations. Top: total accuracy relative to the MILP baseline. Middle: cost-only accuracy. Bottom: demand satisfaction rate. Left column: Household plant, comparing Household_force against Household_penalty. Right column: Town plant, comparing Town_force against Town_penalty. Each bar shows the mean over five seeds with 95% confidence intervals; individual seed results are overlaid as dots.

The results in Figure 4.7 differ between the two plants. On the Household plant, the active environment configuration outperforms the penalty-based one, reaching 91.1 ± 0.2 against 84.0 ± 1.5 total accuracy. The confidence intervals do not overlap, indicating a robust difference rather than seed noise. On the Town plant, the gap narrows substantially: the two configurations achieve 100.2 ± 0.1 and 100.0 ± 0.7 respectively, with overlapping confidence intervals.

4.5 Transfer Learning

Figure 4.8 shows the accuracy, cost, and demand satisfaction of the experiment with transfer learning. Figure 4.9 shows the reward during training for on the Town environment for the transferred CNN and two comparable RL agents. The total training time over 10′000′000 timesteps, for the same models as shown in Figure 4.9, is shown in Table 4.4. It should be noted that as experiments were run on spot instance compute environments, the RL agents may have been trained on different hardware.

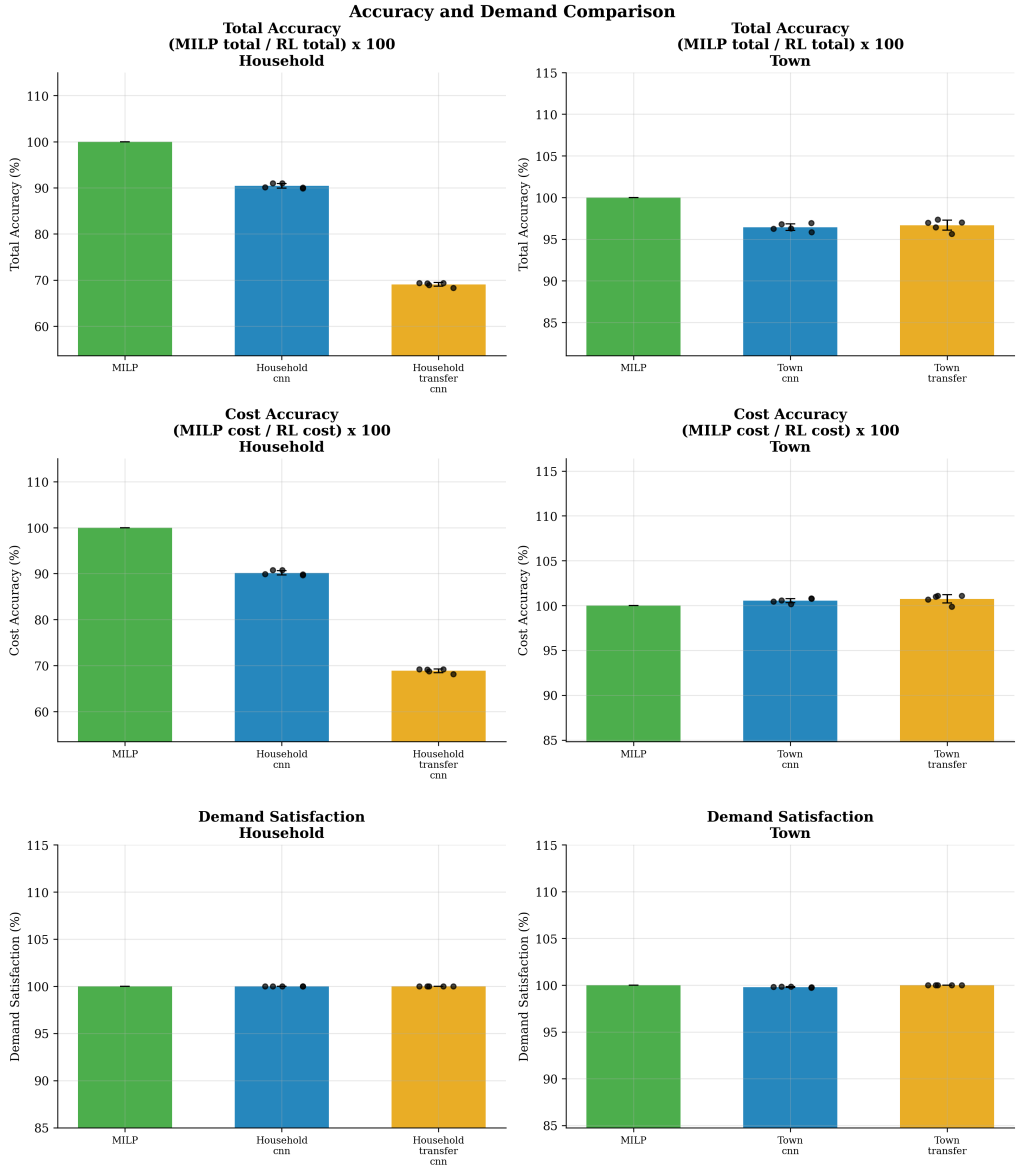

Figure 4.8: Comparison total accuracy (top), cost part of reward accuracy (middle), and the demand satisfaction (bottom) between. The left side shows CNNs trained on the Household environment, and trained on the Town environment then transferred. Right side shows CNNs trained in the Town environment, and trained on the Household environment then transferred

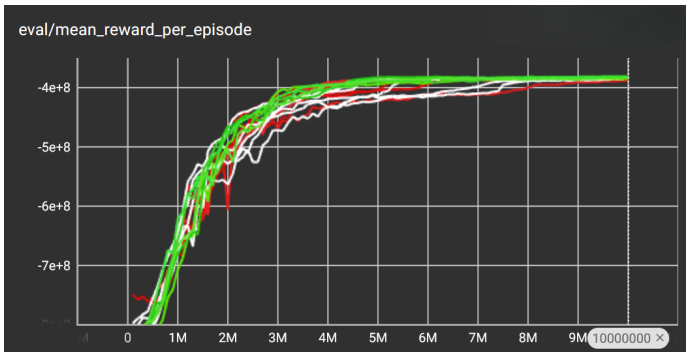

Figure 4.9: Reward (negative) over timesteps for CNN models in the Town environment. Three experiments are included, each with five runs: Transferred model trained in the Household environment (green); one from the active environment, with active environment (white); one from the distribution experiments, with Gaussian (red). All models used clipped Gaussian with active environment.

Table 4.4: Model training time in hours for the transferred CNN, and two baseline CNNs.

#	**transfer_cnn**	**cnn_gaussian**	**cnn_active**
1	7.418	8.311	8.341
2	8.137	8.352	8.533
3	8.204	8.536	8.601
4	8.499	12.07	11.95
5	8.917	12.20	12.11

5

Discussion

The results across both plant configurations point to a consistent overall picture: DRL agents trained with PPO are capable of producing solutions that are rather good on costs, while satisfying demand at rates that are competitive with the provided MILP solver. The agents achieve cost accuracies of within 8.1% on the Household plant, and above 100% on the Town plant. The reward accuracies are within 7.9% on the Household plant, and 2.7% on the Town plant. However, this cost competitiveness was achieved through operationally problematic behaviour that the cost metric alone does not capture. Of note is that 8.1% is too low to use in real world applications as the increase in cost would be to large. What 8.1% does say is that it is possible to learn this system and further investigation is needed to make RL a competitive option.

5.1 Cost Accuracy Versus Operational Quality

The cost metrics present an optimistic picture, but Figure 4.3 reveals an important limitation that the cost metric alone does not capture. While the DRL agent matches the MILP closely in cumulative cost over the validation week, it does so through operationally problematic behaviour: the agent switches units' operating levels at high frequency and maintains a consistently lower accumulator level. The MILP, by contrast, dispatches units in stable block-wise patterns with smooth, deliberate charge-discharge cycles that exploit the full accumulator capacity. While not fully utilizing the accumulator is not a catastrophic problem, it indicates that there is still room for improvement.

This pattern is consistent with the findings of O'Malley and de Mars [19], who reported that MILP decisively outperforms RL on solution quality in unit commitment problems and that significant improvements in RL reliability remain necessary. Our results add nuance to their conclusion: on a pure cost basis the gap can be closed, but the operational characteristics of the resulting schedule are not equivalent. Part of the reason the RL agent can occasionally match or exceed the MILP on cost is precisely that it is willing to ignore the implicit cost of high-frequency switching, granting it more freedom to chase short-term price opportunities. A deployed system would not have that freedom; mechanical wear, thermal stress on burners, and minimum-runtime constraints are real and would impose costs the current setup fails to accommodate successfully [30].

This reveals that cost accuracy is a necessary but not sufficient metric for evaluating DRL agents in this domain. A deployed system would need to satisfy both economic performance and operational smoothness, and the current agents satisfy only the former. The erratic switching behaviour would impose significant real-world costs through mechanical wear that are not sufficiently captured in the reward signal. In fact, this behavioural difference is part of why the RL agent can occasionally match or exceed the MILP solver on cost, like seen in Figure 4.3. By ignoring the implicit cost of switching frequency it has more freedom to chase short-term price opportunities.

This is likely a reward-design problem rather than a fundamental failure of the DRL approach. The start, stop, and rate-change opt-costs are present in the reward function and use the same values as the MILP solver, yet fail to discourage erratic behaviour (the fact that we in some cases achieve accuracy above 100% proves this). Two possible remedies are worth investigating in future work: increasing the change-cost coefficients c^{u}_{Δ} relative to the material-cost terms, or adding an explicit penalty on switching-frequency rather than only on switching-magnitude. Ahmed et al. [31] show some methods for this frequency regularisation, stating that third order derivatives provide the most effective jerk minimization. Note that penalties targeting behaviour frequency, would require extending the observation to include a short action history in order to preserve the Markov property.

As a note on reward design: It is not impossible that, as the test we performed to make sure our environments matched the MILP solver was only conducted in the direction of taking MILP produced action in the RL environment and not vice versa (as it is impossible in the current setup), the MILP environment has constraints that are more restrictive then the RL in regards to erratic behaviour.

5.2 Time series encoding

As seen in Figure 4.4, the CNN encoder produced a small improvement over the flat-vector MLP on the Household plant. On the Town plant, the gap in means is larger. There is also a great variance for the MLP.

This matches the intuition motivating the encoder design. When the action space is simple, the policy network has sufficient capacity to implicitly extract useful features from the raw forecast vector. As the environment grows more complex, receiving a compact and structured latent representation of the forecast becomes more important. The parameter counts in 4.1 show that the CNN does not introduce additional capacity relative to the MLP (the totals are within roughly 2% of each other) showing that the improved stability does not come from increased expressivity.

5.3 Distribution comparison

The Beta and Kumaraswamy results in Figure 4.5 are contrary to our initial hypothesis. Rather than the bounded distributions outperforming the Gaussian by

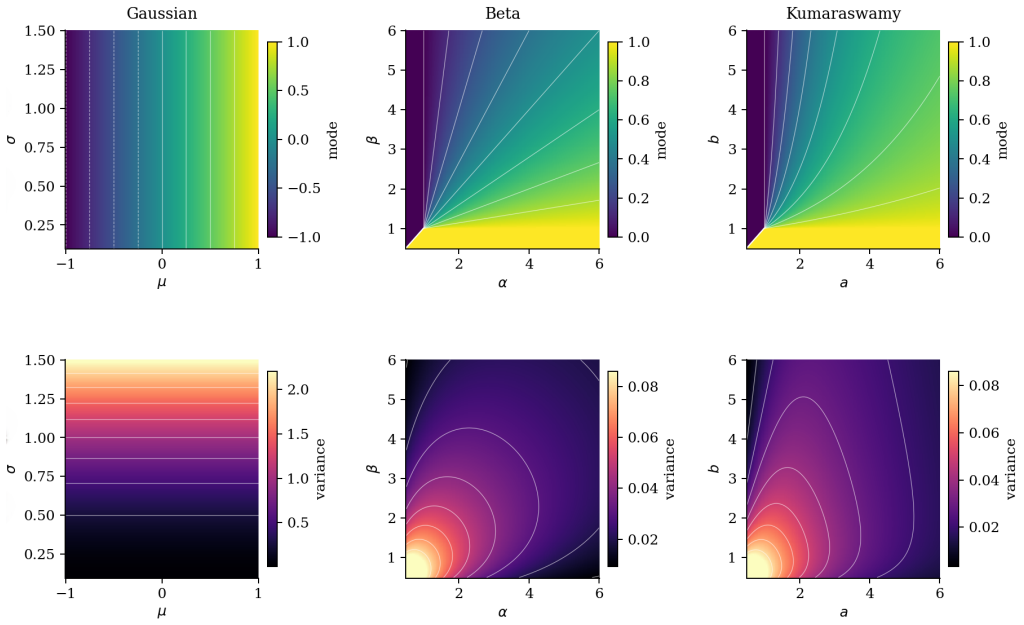

Figure 5.1: Showing the relationship of distributions Gaussian, Beta and Kumaraswamys parameters to their mode (top row) and variance (bottom row).

eliminating the clipping-induced gradient bias, the Gaussian agent achieved significantly better total cost in both plant configurations.

The explanation we propose for this lies in the fact that both Beta and Kumaraswamy are characterized by two shape parameters, α and β (or a and b), which jointly and non-linearly determine the shape of the distribution. Unlike the Gaussian, where the mean μ directly corresponds to the most likely action and σ controls spread, the relationship between the shape parameters and the resulting mode and variance of the Beta and Kumaraswamy distributions is less transparent. This relation is visualized in Figure 5.1 where it is clear that coupled changes to α/a and β/b has to be made to change the mode. This may make optimization harder potentially slowing convergence or leading to suboptimal local minima, particularly early in training when the parameters are far from a useful configuration.

A secondary observation is that the Beta and Kumaraswamy distributions have a natural interpretation as models of the probability of success and failure, where the shape parameters encode prior counts of outcomes rather than direct distributional moments. While this property is not exploited in our formulation, it raises the question of how an agent that could leverage this structure would perform. This remains an interesting direction for future work, but is outside the scope of this thesis.

The clipping ablation in Section 4.3.2 shows that bounding $\ln(\sigma_u) \leqslant 2$ produces a small but consistent improvement over the unbounded configuration, and a notably tighter confidence interval. The training curves in Figure 4.6 confirm that the bound is active throughout training, and that without it σ_u grows to values that are

very large relative to the $[-1, 1]$ action range. This supports the choice to use the clipped variant as the Gaussian baseline. In preliminary runs without learning-rate and entropy-coefficient annealing, σ_u grew even more aggressively and produced numerical instabilities, which is why annealing was retained throughout all reported experiments.

5.4 Demand Enforcement

The comparison between the forceful and penalty-based formulations in Figure 4.7 yielded different conclusions depending on the plant configuration. On Household, the forceful formulation substantially outperformed the penalty-based one (91.1% vs. 84.0% total accuracy), with non-overlapping confidence intervals. On Town, the two formulations converged to overlapping confidence intervals (100.2% vs. 100.0%), though Town_penalty showed wider variance.

The worse performance of the penalty agent can be attributed to the competing reward signals discussed in Section 3.4.2: If the underflow penalty is set too low, the agent discovers policies that accept occasional shortfalls in exchange for short-term cost savings. If set too high, the penalty dominates the cost signal and the agent becomes overly conservative, neglecting the price-arbitrage strategies that distinguish a good schedule from a merely feasible one. Tuning a penalty that avoids both pitfalls is difficult, and the forceful formulation sidesteps the problem entirely by guaranteeing feasibility through post-processing.

We propose two non-exclusive explanations for why the gap closes on the Town plant. First, the more expressive policy, operating over four actions with the support of a CNN encoder, may be better able to internalize demand satisfaction as part of its learned strategy, reducing its reliance on the forceful mechanism even in the penalty-based setting. Second, the Town plant enforces strict change-rate limits on BIO (Table 3.2), inherently limiting how far production can deviate between timesteps. This makes over- and underproduction less likely by construction, narrowing the practical gap between the two configurations. The Household plant has no such rate limits, so the same effect cannot dampen the difference there.

It is also possible that the fact that the oil boiler (as well as all other price signals) is so expensive that it dominates the reward signal in a way that is not possible for the comparatively less expensive heat pump in the basic environment, and thus the possible issue of the agent becoming conservative is not as pronounced. The same behaviour as in the basic case might be seen if the penalties were orders of magnitude larger. As seen in Table 5.1 we see that when the oil boiler is turned on the cost almost doubles from when it is turned off. The penalties are also very low but that might be more an artifact of the validation then how they look during training.

One concern raised in Section 3.4.2 remains valid: because the forceful correction is applied silently outside the reward signal, the agent receives no direct gradient information about why the correction was needed and may become reliant on the mechanism rather than internalising demand satisfaction. We did not design a

Table 5.1: Sample hour cost breakdown sampled from week seen in Figure 4.3

Hour	Sum	Oil cost	Pellet cost	Electricity cost	Total penalties
275	56 217	23 961	30 651	159	9
285	31 110	0	30 651	392	67

dedicated experiment to test this as early testing proved that this was the most viable strategy to get comparable (close to, or at 100% demand met) results.

5.5 Transfer Learning

As can be seen in Figure 4.8, there is a discrepancy between the effects of transfer learning for the Household and Town environments. When transfer learning was applied, the Household environment saw a drop in accuracy, while the Town environment was relatively unaffected.

The loss in accuracy for the Household transferred CNNs can be explained by differences in the domains. For the Household environment, choosing at which times to turn the **HP** on is vital. However, for the Town environment, agents rarely turned the **HP** off as electricity is cheap compared to other fuel sources available. Moreover, agents in the Town environment rarely utilised the **TUR**, as the heat was generally more valuable than any electricity generated. With this is mind, it seems plausible that a CNN trained on the Town environment would not learn to interpret the electricity prices very well, instead only focusing on extracting features from the demand series. Thus, the transfer of knowledge from the Town environment to the Household environment would largely ignore vital features of the time series. Without relevant features extracted, the Household agent is functionally blind to any changes in price.

Conversely, in the Household environment electricity prices are more relevant to decision-making. It seems that a CNN trained in the Household environment does learn to extract relevant features from both time series. This seems to enable a knowledge transfer from the Household environment to the Town environment. However, the improvement shown in Figure 4.8 is not great enough to make a definitive statement of any positive effects.

As mentioned in Section 4.5, definitive statements about improvements in training time are not possible as the different experiments were potentially run on different hardware. However, Table 4.4 does seem to indicate both a slight improvement and lower variance in training time. Moreover, Figure 4.9 shows a noticeable improvement in convergence time for the transferred agent, as compared to other similar agents. This suggests that transfer learning can be used to speed up training and improve stability.

The results of the transfer learning experiments do show some promise for practical applications: when training a base encoder, such an encoder is not necessarily limited by the complexity of the environment.

6

Conclusion

The main research question asks:

> *Can a DRL agent, trained with week-ahead forecast observations, produce schedules that are competitive with those produced by a MILP solver on the same problem instance?*

On a pure cost basis, the answer is *mostly* and *yes* for the configurations tested: The best DRL agents reach within 8.1% of MILP cost on the Household plant and effectively match MILP on the Town plant, while satisfying demand at competitive rates. This is a promising result given that the DRL agents produce week-long schedules in a single forward pass, whereas the MILP solver requires the rolling-horizon procedure described in Section 3.5 and considerable compute per solution.

On an operational-quality basis, the answer is not yet. The high-frequency unit switching and under-use of accumulator capacity documented in Section 4.1 mean that the DRL schedules, despite being cost-competitive, are not equivalent to MILP schedules in ways that matter for deployment. Closing this gap is paramount for the feasibility of using RL as a drop-in replacement for MILP .

While this thesis cannot give a definitive general answer to the motivating question, the results suggest that it will be possible for RL to replace MILP in heat plant planning in the future. However, more work is needed to generalise to more plant configurations and guarantee operational quality at all times.

6.1 Future work

Only two environments were modelled for this project. While these differ in scope and complexity, this limitation does increase the risk that the strategies outlined are not generally applicable. As future work, more complex environments could be modelled containing a greater variety of units. Specifically, there is an interest in modelling those DHPs which regularly prove costly for a MILP solver to solve. Such an addition could solidify the methods presented in this thesis, and potentially show generally applicable methods.

For this thesis, predictions were assumed to be perfectly accurate. This is to say that previously predicted values are the true future values. Without this assumption, different one week projections would be required for each timestep. Adding variance

to the price predictions could prove relevant, as it would more closely model the real world lack of precise information. RL agents trained on unreliable predictions could potentially outperform MILP solvers, as the RL agent could theoretically learn to "trust" different values more, while the MILP solver weights all values equally.

As mentioned by O'Malley and de Mars [19], RL could be used in conjunction with MILP solvers, where an RL agent aids in finding sufficient solutions, later fine tuned by the MILP solver. Such a system could prove efficient in reducing MILP computational cost without loss of accuracy.

We also sincerely believe that more complex reward shaping for encouraging "correct" behaviour should be explored. While we had an idea of exactly matching the opt-cost in the MILP to penalties in RL, it is a natural though that they necessary do not need to be the same for optimal behaviour.

Bibliography

[1] F. Voswinkel, C. Delmastro, B. Reindenbach, and O. Kvanström, *Opportunities for district heating in the changing energy landscape – analysis - iea*, Dec. 2025. [Online]. Available: https://www.iea.org/commentaries/opportunities-for-district-heating-in-the-changing-energy-landscape.

[2] S. Werner, "District heating and cooling," in *Reference Module in Earth Systems and Environmental Sciences*, Elsevier, 2013, ISBN: 978-0-12-409548-9. DOI: https://doi.org/10.1016/B978-0-12-409548-9.01094-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124095489010940.

[3] A. Dzebo and B. Nykvist, *Swedish heat energy system – new tensions and lock-ins after a successful transition*, https://www.sei.org/publications/swedish-heat-energy-system-new-tensions-and-lock-ins-after-a-successful-transition/, Policy brief based on Dzebo & Nykvist (2017), Energy Research & Social Science, 29, 2017.

[4] Nord Pool, *Price formation: Day-ahead market*, https://www.nordpoolgroup.com/en/the-power-market/Day-ahead-market/Price-formation/, Accessed: 2026-05-19, 2024.

[5] S. Ghane, "Reinforcement learning-based control for collective heating systems," Doctor of Applied Engineering thesis, University of Antwerp, Antwerp, Belgium, Jun. 2025. [Online]. Available: https://repository.uantwerpen.be/docman/irua/243fbbmotoMe0.

[6] E. Guelpa and V. Verda, "Thermal energy storage in district heating and cooling systems: A review," *Applied Energy*, vol. 252, p. 113 474, 2019. DOI: 10.1016/j.apenergy.2019.113474.

[7] D. Henning and A. Gebremedhin, "District heating and cooling enable efficient energy resource utilisation," in *Sustainable Energy - Recent Studies*, A. Gebremedhin, Ed., London: IntechOpen, 2012, ch. 1. DOI: 10.5772/51837. [Online]. Available: https://doi.org/10.5772/51837.

[8] Gurobi Optimization, *Integer linear programming*, https://www.gurobi.com/faqs/integer-linear-programming/, Accessed: 2026-05-19, 2024.

[9] Q. Huangfu and J. A. J. Hall, "Parallelizing the dual revised simplex method," *Mathematical Programming Computation*, vol. 10, no. 1, pp. 119–142, 2018. DOI: 10.1007/s12532-017-0130-5.

[10] IBM, *IBM ILOG CPLEX Optimization Studio*, Version 22.1, 2024.

[11] Y. Chen et al., "Security-constrained unit commitment for electricity market: Modeling, solution methods, and future challenges," *IEEE Transactions on*

Power Systems, vol. 38, no. 5, pp. 4668–4681, 2023. DOI: 10.1109/TPWRS.2022.3213001.

[12] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 449–481, ISBN: 978-3-642-38189-8. DOI: 10.1007/978-3-642-38189-8_18. [Online]. Available: https://doi.org/10.1007/978-3-642-38189-8_18.

[13] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer US, 1972, pp. 85–103, ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9.

[14] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 258–271, ISBN: 9781450383837. DOI: 10.1145/3452296.3472902. [Online]. Available: https://doi.org/10.1145/3452296.3472902.

[15] A. Franzoso, G. Fambri, and M. Badami, "Deep reinforcement learning as a tool for the analysis and optimization of energy flows in multi-energy systems," *Energy Conversion and Management*, vol. 341, p. 120 095, Oct. 2025. DOI: 10.1016/j.enconman.2025.120095.

[16] J. Deng et al., "Deep reinforcement learning for fuel cost optimization in district heating," *Sustainable Cities and Society*, vol. 99, p. 104 955, 2023, ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2023.104955. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670723005668.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[18] J. Wang, Y. Wang, D. Qiu, H. Su, G. Strbac, and Z. Gao, "Resilient energy management of a multi-energy building under low-temperature district heating: A deep reinforcement learning approach," *Applied Energy*, vol. 378, 2025. DOI: 10.1016/j.apenergy.2024.124780.

[19] C. O'Malley, P. de Mars, L. Badesa, and G. Strbac, *Reinforcement learning and mixed-integer programming for power plant scheduling in low carbon systems: Comparison and hybridisation*, 2022. arXiv: 2212.04824 [eess.SY]. [Online]. Available: https://arxiv.org/abs/2212.04824.

[20] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.

[21] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," Jun. 2015. DOI: 10.48550/arXiv.1506.02438.

[22] I. G. B. Petrazzini and E. A. Antonelo, *Proximal policy optimization with continuous bounded action space via the beta distribution*, 2021. arXiv: 2111.02202 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2111.02202.

[23] P. Kumaraswamy, "A generalized probability density function for double-bounded random processes," *Journal of Hydrology*, vol. 46, no. 1, pp. 79–88, 1980, ISSN: 0022-1694. DOI: https://doi.org/10.1016/0022-1694(80)90036-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022169480900360.

[24] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*, MIT Press, 1995.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. DOI: 10.1109/TKDE.2009.191.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *CoRR*, vol. abs/1411.1792, 2014. arXiv: 1411.1792. [Online]. Available: http://arxiv.org/abs/1411.1792.

[28] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[29] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018. [Online]. Available: https://arxiv.org/abs/1803.01271.

[30] N. Kumar, P. Besuner, S. Lefton, D. Agan, and D. Hilleman, "Power plant cycling costs," National Renewable Energy Laboratory (NREL), Subcontract Report NREL/SR-5500-55433, 2012. [Online]. Available: https://docs.nrel.gov/docs/fy12osti/55433.pdf.

[31] F. Ahmed, A. Dixit, and J. Brusey, *Higher-order action regularization in deep reinforcement learning: From continuous control to building energy management*, 2026. arXiv: 2601.02061 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2601.02061.