UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Yaroslava Mykhailenko

Optimization of Battery Energy Storage System in the Estonian Energy Markets using Reinforcement Learning

Master's Thesis (30 ECTS)

Supervisor(s):
Victor Henrique Cabral Pinheiro, DSc
Jean-Baptiste Scellier, MSc
Tambet Matiisen, MSc

Tartu 2025

Optimization of Battery Energy Storage System in the Estonian Energy Markets using Reinforcement Learning

Abstract:

The integration of renewable energy sources into electricity markets has increased the need for efficient energy management solutions. Battery Energy Storage Systems (BESS) help balance fluctuating supply and demand by storing excess energy and supplying it during shortages, so determining an optimal charge-discharge schedule becomes a core optimization task for market participants. Currently, Eesti Energia addresses this task with linear optimization techniques that are effective yet limited, because they cannot fully capture nonlinear battery dynamics or adapt to complex market patterns.

This thesis explores the potential of model-free reinforcement learning (RL) algorithms, specifically Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO), to optimize battery trading strategies in Estonia's day-ahead electricity market. The primary objective is to assess whether RL models can potentially surpass traditional linear optimization benchmarks in terms of profitability and decision quality.

Results demonstrate that in a six-hour trading horizon the optimized DDPG agent consistently outperformed PPO and closely approached the performance of the linear optimizer, capturing 85.5% of its profit. When extended to a full 24-hour horizon, its relative performance fell to 65%. Qualitative analysis of evaluation logs confirmed market-aware behavior, with the agent charging when prices were low, discharging near peaks, and preserving capacity for anticipated high-price periods.

Overall, the findings suggest that, when carefully tuned, model-free RL can provide a competitive alternative to linear optimization for battery trading in volatile electricity markets, with the potential to account for nonlinear battery aging and integrate multi-market signals. This lays the groundwork for future applications in even more dynamic settings such as balancing markets.

Keywords: Reinforcement learning, Battery Energy Storage System, day-ahead market, electricity price arbitrage, Deep Deterministic Policy Gradient, Proximal Policy Optimization

CERCS: P176 – Artificial Intelligence; T140 – Energy Research.

Akuenergiasalvestussüsteemi optimeerimine Eesti energiaturgudel stiimulõppe abil

Lühikokkuvõte:

Taastuvenergiaallikate kasvav osakaal elektriturul suurendab vajadust tõhusate energiahalduslahenduste järele. Akuenergiasalvestussüsteemid (AESS) aitavad tasakaalustada kõikuvaid nõudluse-pakkumise mustreid, salvestades üleliigset energiat ning vabastades seda nappuse korral; seetõttu on optimaalne laadimis-tühjendamisgraafik turuosaliste jaoks keskne optimeerimisülesanne. Praegu kasutab Eesti Energia selleks lineaarseid optimeerimismeetodeid, mis on küll tulemuslikud, kuid piiratud – need ei kirjelda täielikult aku mittelineaarset vananemist ega kohane keerukate turumustritega.

Käesolevas magistritöös uuritakse mudelivabade stiimulõppe algoritmide – eeskätt Deep Deterministic Policy Gradient (DDPG) ja Proximal Policy Optimization (PPO) – potentsiaali AESS-i kauplemisstrateegiate optimeerimisel Eesti päev-ette turul. Eesmärk on hinnata, kas stiimulõppe mudelid suudavad ületada traditsioonilisi lineaarsel optimeerimisel põhinevaid meetodeid tulususe ja otsustuskvaliteedi osas.

Katsetulemused näitavad, et 6-tunnise kauplemishorisondi puhul ületas hoolikalt häälestatud DDPG-agent järjekindlalt PPO-d ning saavutas 85,5 % lineaarse optimeerija kasumist. 24-tunnise horisondi korral langes suhteline tulemus 65 %-ni. Logide kvalitatiivne analüüs kinnitas turuteadlikku käitumist: agent laadis akut madalate hindade juures, tühjendas hinnatippude eel ning säilitas mahutavust oodatavate kõrgete hindade jaoks.

Kokkuvõttes viitavad tulemused, et hoolikalt häälestatud mudelivaba RL võib volatiilsel elektriturul pakkuda lineaarsele optimeerimisele konkurentsivõimelist alternatiivi ning võimaldab arvesse võtta nii aku mittelineaarset vananemist kui ka mitme turu signaale. See loob tugeva lähtekoha tulevasteks rakendusteks veelgi dünaamilisemates keskkondades, nagu näiteks tasakaalustusturud.

Võtmesõnad: tugevdamisõpe, akuenergiasalvestussüsteem, päeva-ette turg, elektrihinna arbitraaž, Deep Deterministic Policy Gradient, Proximal Policy Optimization

CERCS: P176 - Tehisintellekt, T140 – Energeetika

Table of Contents

1. Introduction

In recent years, the world's energy systems have been going through a major shift as countries seek to cut greenhouse gas emissions and mitigate climate change, pushing for a faster switch to cleaner energy. In Europe, the focus has shifted towards renewable energy sources (RES), which are now a key part of the future electricity grid. According to [1] solar and wind made up a record 30% of the EU's electricity in 2024, up from 27% in 2023, with solar generation increasing by approximately 22% year-on-year. Estonia, like many European countries, is at a turning point in its energy transition, aiming to operate entirely on renewable electricity by 2030 [2]. By 2024, renewables accounted for around 52% of Estonia's electricity production, with wind and solar each surpassing 1 TWh for the first time, driven by wind capacity expansion to 694 MW and investments totaling €244 million in renewable projects [3], [4], [5]. However, the increased reliance on weather-dependent renewables brings challenges: when the sun shines and the wind blow, electricity production exceeds demand, driving prices to a minimum, but during periods of low renewable production, prices spike sharply. This volatility creates significant price spreads, defined as the difference between the highest and lowest day-ahead electricity prices within a single day. For instance, as highlighted in Figure 1, in the third quarter of 2024, Estonia experienced a price spread of 200.45 €/MWh, whereas countries with higher share of renewables, such as Bulgaria and Romania, reached spreads of 250–350 €/MWh [6].

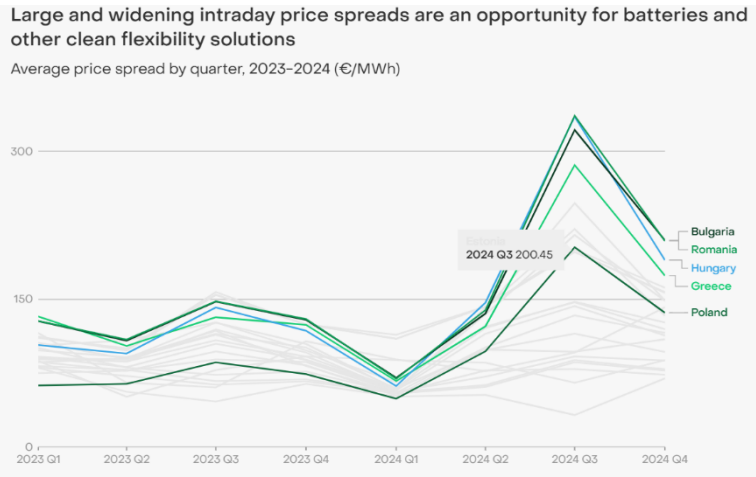

Figure 1. Average price spread by quarter in years 2023-2024 [6].

The challenges associated with integrating renewables became even more pronounced after Estonia disconnected from Russia's electricity grid in February 2025, highlighting the need for robust and stable energy management. The inherently uncontrollable nature of renewable

production further increased reliance on short-term markets, such as the day-ahead and other ancillary markets, essential for maintaining grid stability through rapid-response solutions.

To address this, energy storage technologies are gaining prominence: today, options include pumped hydro storage and, increasingly, battery energy storage systems, which charge during energy surplus and discharge during deficits, smoothing out grid fluctuations. The cost of batteries has significantly decreased in recent years [7], making them more widespread in countries with a high-RES share, such as Estonia, where BESS are becoming an attractive asset for investors due to their ability to generate revenue through electricity market arbitrage and grid balancing services. Projects like the 26.5 MW/53.1 MWh system in Auvere launched in February 2025, are already operational, with plans to deploy an additional 200 MW/400 MWh in Kiisa and Aruküla by 2026 [8]. However, realizing this potential requires sophisticated management strategies: accurate forecasting of market conditions and optimal management of charge-discharge cycles remain key to maximizing profits. To this end, companies have begun developing algorithms to optimize BESS trading. For example, Eesti Energia implemented a linear optimization algorithm, outlined conceptually in [9], focusing on both day-ahead (DA) and balancing markets, such as the manual frequency restoration reserve (mFRR), to determine optimal charging and discharging schedules based on price forecasts. Yet, such optimization approaches not fully account for nonlinear effects, such as battery degradation dependent on charge levels, and cannot adapt to complex market patterns, like the correlation between low day-ahead prices and high mFRR spreads, because they make optimal trades based solely on the information available at the time. When bidding for the DA market (tomorrow), they lack data on balancing markets prices since forecasts for those are not available yet, leading them to often use full capacity whenever an opportunity arises, which misses potential profits from better-timed trades and ultimately limits overall earnings. These limitations highlight the need for a more dynamic approach, leading to the exploration of reinforcement learning as an alternative. Unlike static optimization, RL offers an adaptive logic that can model and address the shortcomings of optimization-based approaches by learning from diverse market interactions over time, allowing it to adjust to the volatility and unpredictability of prices in energy markets while also anticipating future price shifts to make smarter and forward-looking decisions. Companies like Capalo AI are already successfully applying RL to optimize BESS, raising €3.8 million in 2025 and serving clients such as Taaleri Energia and MW Storage, claiming significant profitability improvements for flexible assets, which highlights RL's potential [10].

Inspired by previous knowledge, this thesis, conducted in collaboration with Eesti Energia, explores the opportunities of RL to optimize battery usage on the DA and mFRR markets, comparing its performance with the existing linear optimization approach to determine whether it is able to better manage battery usage and maximize financial returns. Due to the complex nature of RL algorithms and time constraints, the experimental analysis focuses exclusively on the DA market, laying the foundation for future research on balancing markets such as mFRR.

The thesis is structured as follows: Chapter 2 provides the necessary background on Estonia's electricity markets, linear optimization, and the reinforcement learning fundamentals; Chapter 3 details the methodology, including the environment design, model implementation, training configurations, and evaluation procedures; Chapter 4 presents the experimental results the experimental results, supported by analysis and comparisons to benchmark method. Finally, Chapter 5 discusses the findings and outlines opportunities for future research.

2. Background

This chapter begins with an overview of Estonia's electricity markets and their key components. It then presents the linear optimization approach used by Eesti Energia, which serves as the benchmark method in this study. Finally, it introduces the fundamentals of reinforcement learning, followed by a detailed explanation of the specific algorithms applied: Deep Deterministic Policy Gradient and Proximal Policy Optimization.

2.1 Electricity markets in Estonia

Estonia, as a member state of the European Union, is fully integrated into the EU's internal energy market, which aims to promote competition and secure energy supply across member states. In 2025, Estonia synchronized its electricity grid with the European ENTSO-E network, marking a key step in strengthening cross-border cooperation and enabling future participation in common European balancing markets.

Electricity in Estonia can be traded either on organized markets, such as Nord Pool, or through bilateral contracts. Nord Pool is the main exchange platform facilitating transactions between producers and consumers in the Nordic and Baltic regions. Our main focus is trading within the open market, as it plays a key role in maintaining the stability of the energy system. The country operates several markets: the day-ahead market, which forms the basis of supply and demand planning for the following day; the intraday market (ID), which allows for operational adjustments closer to real-time delivery; and balancing markets, managed by the transmission system operator (TSO) to maintain real-time network equilibrium. In addition, there are imbalance settlement mechanisms to address situations where market participants fail to meet their obligations, such as when they sell electricity that they cannot supply to the grid or buy electricity that they do not ultimately consume.

The following subsections provide an overview of these markets, their operational logic, and their role in ensuring system balance.

2.1.1 Day-ahead market

The day-ahead market is the main platform for electricity trading one day before physical delivery, ensuring that producers, consumers, and other market participants secure a financially binding schedule for each hour of the following day. At its core, the DA market's purpose is to match total supply with total demand for every hour based on bids submitted by participants, who specify how much electricity they are willing to buy or sell and at what price. Because

this market sets the base schedule for power production and consumption, it typically covers the largest trading volumes compared to other short-term electricity markets. According to Nord Pool's 2024 annual report [11], the DA market had the largest trading volume, amounting to 1036 TWh.

In Europe, the DA market operates through organized exchanges (such as Nord Pool in the Nordic–Baltic region). Under a common framework known as the Single Day-Ahead Coupling (SDAC), these exchanges coordinate auctions across multiple countries [12]. To manage grid constraints effectively, Europe is divided into bidding zones - geographical areas within which electricity is traded without internal transmission constraints, meaning all participants in the zone face the same market price for each hour. These zones are typically aligned with national borders, although in some countries, like Sweden, Norway, internal zones exist due to known transmission bottlenecks within the country [13]. When cross-border transmission capacity becomes limited, each bidding zone can settle at a distinct price. Estonia, for example, is its own bidding zone; on days with congestion between Estonia and neighboring zones like Finland or Latvia, the Estonian zone may settle at a different price.

To participate in the day-ahead market in Nord Pool, market participants rely on key market signals published throughout the day before delivery. At 10:00 CET, available transmission capacities on interconnectors and within the grid are published, giving buyers and sellers crucial information about potential cross-border flows and congestion risks. Based on this, participants have until the gate closure time at 12:00 CET to submit their final bids for each hour of the following day. After this deadline, no further changes to bids are allowed. The central market operator then aggregates these bids – both sell and buy – from all bidding zones and use to construct supply and demand curves for every hour of the next day. The intersection of these curves in each zone determines the equilibrium price and traded volume; however, cross-border flows must also be optimized so that electricity can move from lower-price zones to higher-price zones. This optimization is carried out by EUPHEMIA, the clearing algorithm responsible for handling the complexities of transmission constraints and maximizing social welfare across all coupled markets [14]. Final day-ahead prices and accepted volumes are typically published at 12:45 CET or later, giving participants a clear reference for their next-day production schedules or consumption plans [15]. Figure 2 shows the whole timeline for day-ahead auction process.

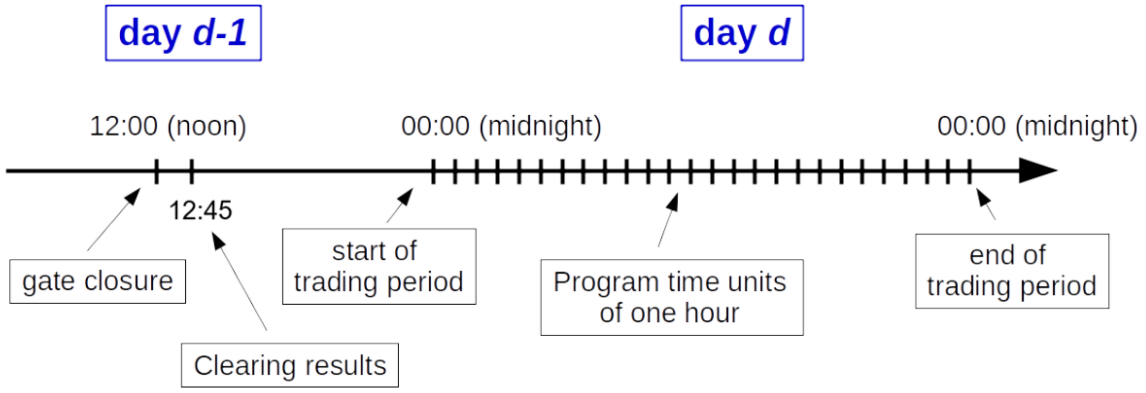

Figure 2. Daily timeline for the day-ahead auction [16].

In DA market, each hourly bid typically specifies (i) the volume in MWh that a participant wants to sell or buy, and (ii) the minimum (for sellers) or maximum (for buyers) acceptable price in €/MWh. There is also the option to use more complex bid types, such as block orders that span multiple consecutive hours when operational or cost constraints extend beyond a single hour [17]. However, the standard practice remains hourly bids with a granularity of one hour, meaning that bids are submitted and clearing prices are calculated separately for each of the 24 hours of the following day. Furthermore, the day-ahead market enforces pan-European price limits: as of 2025, the maximum price can rise to 4000 €/MWh in most bidding zones, while the minimum price generally sits at -500 €/MWh [18]. Negative prices can occur during periods of electricity oversupply – such as when there is strong wind or solar generation combined with low demand – and when export capacity to neighboring zones is insufficient. In such situations, producers may be forced to pay others to take their electricity, leading to negative market prices.

Once the hourly prices for the next day are published, all accepted bids become financially binding: a seller must deliver the committed volume of electricity, and a buyer must pay for and take delivery of the volume they have purchased. However, there is one important point to consider: these bids are based on forecasts of demand and generation, which are inherently uncertain. Unforeseen factors, such as weather changes, technical failures, or shifts in demand, can cause deviations between expected and actual electricity schedule. To correct these discrepancies, additional markets are used. Typically, the result of the day-ahead market is used as the baseline for planning the next 24-hour period, with later changes or new opportunities realized in Nord Pool's intraday market and balancing markets [15]. Although the ID market plays an important role in short-term electricity trading, allowing participants to continuously

adjust their positions closer to real time, it is not analyzed in detail in this thesis, as our focus is on day-ahead optimization with the intention to extend this approach to balancing markets in future work. For more information on ID trading structure and functionality, see [19].

2.1.2 Balancing markets

Balancing markets form the final operational layer of the electricity trading system, operating in near real-time to correct deviations that arise after day-ahead and intraday markets have set their schedules. While DA and ID trades establish financially binding commitments for energy supply and demand, actual production and consumption often diverge from forecasts due to unexpected events described earlier. Such deviations are known as imbalances, and they can cause the system frequency to drive from its nominal level of 50 Hz in Europe [20]. Imbalances can occur in either direction: a positive imbalance happens when generation exceeds consumption, whereas a negative imbalance occurs when demand is greater than supply [21]. The responsibility for maintaining this real-time balance lies with the transmission system operator, which must continuously monitor the grid and activate reserves when needed [22]. Depending on the situation, TSOs apply either upward regulation (to increase generation or reduce consumption) or downward regulation (to reduce generation or increase consumption) through balancing markets to restore equilibrium and ensure secure grid operation. In Estonia, this role is fulfilled by the national TSO, Elering.

The structure of balancing markets in Estonia, as across most of Europe, is guided by two core EU regulations [23], [24], [25]. Under these regulations, three main products are employed to maintain system stability: frequency containment reserve (FCR), automatic frequency restoration reserve (aFRR), and manual frequency restoration reserve (mFRR). Each of these products is operated through a two-phase market structure composed of a (i) capacity market, which ensures availability, and an (ii) energy market, which activates actual delivery. This structure and timeline of balancing markets in relation to other electricity markets is illustrated in Figure 3.

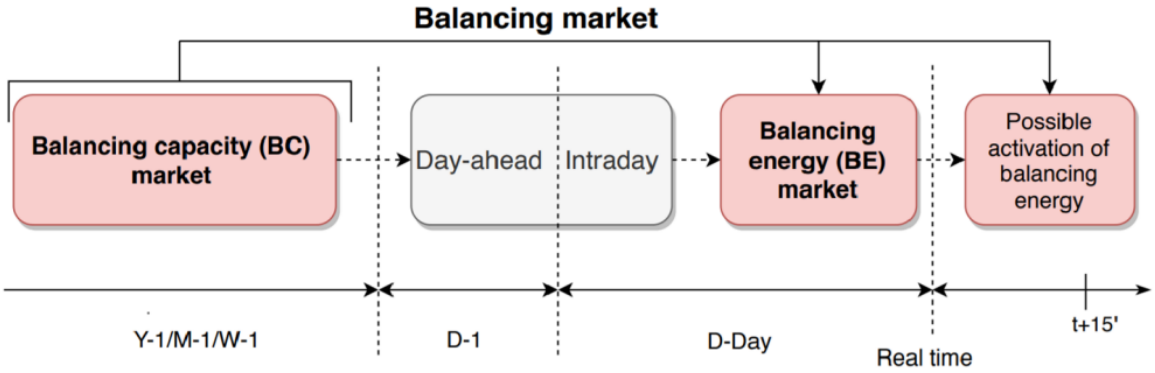

Figure 3. Timeline of balancing capacity and energy markets in relation to day-ahead and intraday markets [26].

The balancing process begins a day in advance with the capacity markets, where TSO purchases reserve availability for the next 24 hours. This ensures that sufficient capacity is on standby to respond to imbalances. In the first stage, the required volume of FCR reserves is purchased, followed by simultaneous capacity auctions for aFRR and mFRR. Participants submit bids specifying the amount of reserve capacity (in MW) they can offer and at what price (€/MWh). Bids for FCR are symmetric, meaning the provider must be able to both increase (upward regulation) and decrease (downward regulation) their output depending on the direction of frequency deviation. In contrast, aFRR and mFRR bids are mono-directional, submitted separately for upward or downward regulation. All the bids are sorted by price, and the cheapest ones are selected until the required volume is reached. Suppliers whose bids are selected are paid for being ready for activation, even if they are not later activated.

After the capacity reserved, the second phase is activated - the balancing energy markets. This stage operates continuously throughout the day, responding to imbalances in the system in real time. Market participants have to submit their energy bids to the European balancing platforms: PICASSO for aFRR [27] and MARI for mFRR [28]. Initial bids must be submitted by 16:30 CET on the day before delivery, with final updates allowed up to 25 minutes before each 15-minute market time unit (MTU) [25]. Energy bids indicate the quantity of energy (in MWh) a participant is willing to provide in a specific direction (up or down) and the corresponding price (€/MWh). During real-time operations, Elering monitors the system frequency and calculates the imbalance volumes. When an imbalance is detected, Elering begins activating balancing reserves based on hierarchical logic and merit-order pricing, selecting the most cost-efficient resources first.

In terms of real-time operation, activation follows a clearly defined order depending on the severity and urgency of the imbalance. mFRR is usually activated first due to the larger volume of available resources and lower cost, although its response time can be up to 12.5 minutes [29]. It is activated manually by the TSO control center. If the imbalance requires faster intervention or persists after mFRR activation, aFRR is used. aFRR is faster (with a response time of about 5 minutes) and is triggered automatically via the TSO dispatch system [30]. If necessary, FCR provides the fastest response (within 30 seconds) and is triggered locally and automatically by each asset, without the need for centralized activation [31]. The amount of energy supplied by the asset is proportional to the deviation from the desired frequency, which is 50 Hz. It is important to note that aFRR and mFRR involve both capacity and energy markets, while FCR involves only the capacity market: once a participant is selected and paid for availability, the actual energy is activated automatically based on physical measurements, and no additional energy trading is required.

Until recently, Estonia and the other Baltic countries had only one active balancing market: the manual frequency restoration reserve. This was the main tool used to keep the system balanced. However, after disconnecting from the Russian electricity system and synchronizing with the continental European grid in 2025, the situation began to change. New balancing products such as aFRR and FCR are now being introduced step by step in line with European rules and market structures. Although not all components are fully operational yet, the system is still being tested and adjusted, and Estonia is expected to implement all three balancing markets – FCR, aFRR, and mFRR – and join the European balancing platforms MARI and PICASSO within the next few months [32], [33].

2.1.3 Imbalance price

The imbalance price is the rate at which electricity imbalances are settled between market participants and the TSO [34]. It applies when participants deviate from their scheduled electricity production or consumption in the day-ahead or intraday markets. These deviations must be balanced in real time to ensure system stability.

Estonia uses the Single Imbalance Price model, as required by the EU Electricity Balancing Guideline [24]. This means both over and under deliveries are settled using the same price within each 15-minute imbalance settlement period. The imbalance price is typically based on: (i) the marginal price of activated balancing energy – the price of the last accepted offer used to balance the system in the required direction; (ii) the direction of the system imbalance –

indicates whether the system was in surplus or shortage and determines which type of regulation was needed (upward or downward); (iii) the neutrality component is a fee or adjustment applied to ensure that the TSO remains financially neutral. It balances out any surplus or deficit resulting from the difference between what the TSO pays and receives for balancing energy, redistributing the difference fairly among market participants [35].

This pricing mechanism provides a transparent and efficient incentive for market participants to maintain a balance between their scheduled and actual positions.

2.2 Linear optimization

Optimization problems occur in various engineering and decision-making contexts, where the idea is to find the most efficient operation under a set of known constraints. In these kinds of problems, the task is to figure out the best decision or action from a set of feasible alternatives by minimizing or maximizing an objective function that quantifies the cost, profit, return, or performance involved. In general, an optimization problem is defined as [36]:

$$\min_x f_0(x) \text{ subject to } f_i(x) \leq b_i, i = 1, \dots, m,$$

where $x = (x_1, \dots, x_n)$ represents the decision variables, $f_0(x)$ is the objective function, and $f_i(x) \leq b_i$ define the constraints that the solution must satisfy.

A special case of this formulation is linear optimization, or linear programming (LP), in which both the objective function and all constraints are linear functions of the decision variables. Here, linearity means that each function f_i satisfies:

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y), \forall x, y \in R^n, \alpha, \beta \in R.$$

Linear programming is widely used due to its simplicity, deterministic structure, clear interpretability, guaranteed global optimality and clear interpretability of results. LP problems are especially suited for operational planning tasks in which a linear approximation can represent the relationships among several decision variables; examples are scheduling, production planning, and energy market optimization.

Eesti Energia has developed a linear optimization model specifically designed for battery energy storage system to efficiently manage battery trading across different electricity markets. The purpose of this LP model is to minimize the total net trading cost within a predetermined planning horizon, using the forecasts of electricity prices. It determines the most profitable

sequence of charging and discharging actions, respecting the physical and operational constraints of the battery system.

The optimization procedure takes as input electricity prices for each discrete hourly timestep within a forecasting horizon. Specifically, the model utilizes hourly forecasts of consumption prices p_t^{cons} and production prices p_t^{prod} for each timestep $t \in \{1,2,\dots,T\}$. Additionally, the optimization receives key battery parameters, including battery capacity C_{max}, maximum charging and discharging power P_{ch} and P_{dis}, charging and discharging efficiencies η_{ch} and η_{dis}, battery depreciation rate δ, and initial state of charge SOC_0.

Formally, the objective function of the linear optimization model is:

$$\min \sum_{t=1}^{T} \left(p_t^{\text{cons}} \cdot E_t^{\text{in}} - p_t^{\text{prod}} \cdot E_t^{\text{out}} \right) + \delta \sum_{t=1}^{T} D_t,$$

where E_t^{in} and E_t^{out} represent the energy volumes imported from and exported to the grid at timestep t, respectively. The term D_t refers to the discharged energy from the battery, with the parameter δ accounting for battery depreciation costs due to cycling.

This optimization is constrained by several operational limits. First, the battery's state-of-charge (SOC) dynamics must reflect the net effect of charging and discharging over each interval, adjusted by efficiency:

$$SOC_t = SOC_{t-1} + \eta_{\text{ch}} \cdot C_{t-1:t} - \frac{D_{t-1:t}}{\eta_{\text{dis}}}, \forall t \in \{1,\dots,T\},$$

where $C_{t-1:t}$ and $D_{t-1:t}$ denote the energy charged to and discharged from the battery during the interval $(t-1, t]$.

The battery SOC must stay within its operational limits:

$$0 \leq SOC_t \leq C_{\text{max}}, \forall t$$

Charging and discharging activities must also respect the battery's power limitations:

$$0 \leq C_{t-1:t} \leq P_{\text{ch}}, 0 \leq D_{t-1:t} \leq P_{\text{dis}}, \forall t$$

To prevent simultaneous charging and discharging, binary variables b_t^{ch} and b_t^{dis} are used to activate only one action per timestep:

$$b_t^{\text{ch}} + b_t^{\text{dis}} = 1, \forall t$$

and the variables $C_{t-1:t}$ and $D_{t-1:t}$ are then linearly linked to these binary variables to ensure mutual exclusivity.

Finally, the energy imported from and exported to the grid E_t^{in} and E_t^{out} are computed based on these optimized charging and discharging actions. Such linear formulation hence provides a clear and deterministic framework under which cost-optimal battery trading strategies can be formulated given the market forecasts and technical constraints. This approach serves as the reference model against which the experimental results are compared.

2.3 Reinforcement learning

This section introduces the main ideas behind reinforcement learning. It starts with core terminology and learning paradigms, then moves through fundamental concepts such as policies, value functions, policy gradients, and different ways of collecting training data. It also explains the distinction between model-free and model-based approaches, as well as value-based and policy-based methods. Finally, the section presents the two specific algorithms used in this thesis: DDPG and PPO.

2.3.1 General principles

Reinforcement learning is a subfield of machine learning (ML) concerned primarily with the problem of sequential decision-making, where an autonomous agent interacts with an environment in order to maximize a cumulative numerical reward [37]. This concept fundamentally differs from the well-known supervised and unsupervised learning principles. While supervised learning requires access to labeled input-output pairs, and unsupervised learning focuses on finding structure in unlabeled data, RL does not require labeled training examples; instead, the agent learns optimal behaviors directly through trial-and-error interactions, continuously adjusting its actions based on the feedback provided by the environment. More specifically, at each discrete time step t, the agent observes the current state of the environment s_t, selects an action a_t, and subsequently receives an immediate scalar reward r_{t+1}, along with the next state s_{t+1} resulting from its action. This process is illustrated on Figure 4. This feedback loop is repeated over time and allows the agent to learn which actions resulted in higher rewards in different situations. The goal of an agent is to develop a policy – a strategy for selecting actions – that maximizes the total cumulative reward (also called ‘return”).

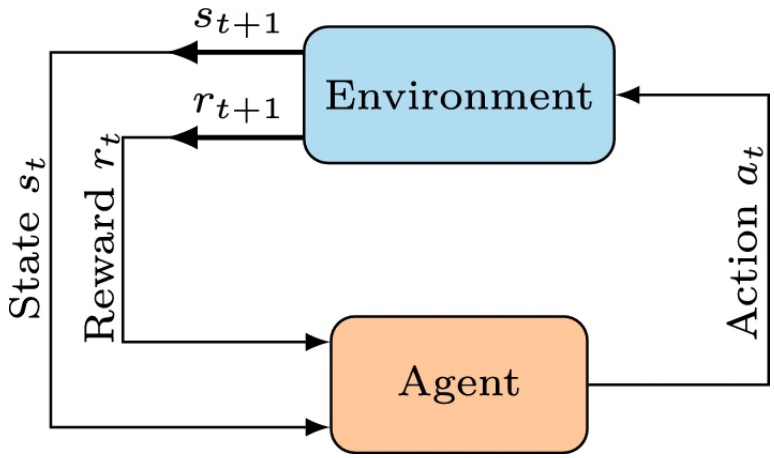

Figure 4. Reinforcement learning interaction loop [38].

In RL, the environment represents everything external to the agent; essentially, it is the "world" in which the agent operates. State s is defined as a complete description of the environment. It contains all the information needed to determine what happens next. For example, in a grid-world navigation task, a state could include the agent's current position and the locations of nearby obstacles. In practice, agents do not always have access to the full state. Instead, they receive an observation, which may be a subset of the true state. Whether the environment is fully or partially observable depends on the setting, but this distinction often has little impact, as most RL methods work similarly in both cases.

Another important aspect is the action space, which defines the set of all possible actions the agent can take at each time step. Depending on the problem, this space can be either discrete – consisting of a finite number of distinct options (e.g., move left, move right) – or continuous – allowing the actions to be selected from a real-valued range (e.g., how much energy to charge or discharge from a battery). This distinction is important because some RL algorithms are directly applicable only to discrete action spaces [39] and require significant modifications or alternatives to work with continuous actions [40].

As the agent interacts with the environment over time, it generates a sequence of experiences known as a trajectory, usually referred to as an episode or rollout. A trajectory τ is typically defined as a sequence of states and actions:

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

It starts from the initial state s_0 that is sampled from a predefined start-state distribution, sometimes denoted p_0, and proceeds step-by-step as the agent selects actions and transitions through the environment. The transition to the next state s_{t+1} may be either deterministic,

$s_{t+1} = f(s_t, a_t)$ or stochastic, $s_{t+1} \sim P(\cdot\,|s_t, a_t)$, i.e., drawn from a probability distribution that captures the uncertainty in environmental dynamics [41].

At each time step, the agent receives a scalar reward r_t, which reflects the immediate consequence of its previous action. In general, the reward function can be defined as:

$$r_t = R(s_t, a_t, s_{t+1})$$

meaning that the reward may depend not only on the current state and the chosen action, but also on the resulting next state. However, in practice, this function is often simplified to depend only on the state-action pair $r_t = R(s_t, a_t)$, the current state $r_t = R(s_t)$, or the next state $r_t = R(s_{t+1})$.

Regardless of how the reward is calculated at each step, the overall goal of the agent is to learn to maximize the return – the total accumulated reward over time. To express this more generally, the notation $R(\tau)$ will be used to express the total reward collected along a trajectory.

There are two types of returns: finite-horizon undiscounted return and infinite-horizon discounted return. The first type is used in environments where interactions take place over a known, limited number of steps T. In this case, the return is simply the sum of rewards received during the episode:

$$R(\tau) = \sum_{t=0}^{T} r(t)$$

The infinite-horizon discounted return is used in continuing tasks without a fixed endpoint. To account for the potentially infinite length of interaction and to reflect the intuition that immediate rewards are often more valuable than distant ones, the formulation uses a discount factor $\gamma \epsilon (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

The discount factor determines how much future rewards contribute to the overall return. With γ close to 0 the agent acts as short-sighted, focusing primarily on immediate outcomes, whereas a value closer to 1 encourages long-term planning.

In summary, the agent-environment interaction described above is commonly formalized using Markov Decision Processes (MDPs), which provide a mathematical framework for sequential

decision-making under uncertainty. This framework assumes the Markov property, meaning that each transition depends only on the current state and action, not the full history. For more information see [37], [41].

2.3.2 Policy and value functions

In reinforcement learning, a policy defines the agent's strategy for selecting actions based on the current state. Depending on the task, policies can be categorized as either deterministic or stochastic [37]. A deterministic policy, typically written as $a_t = \mu(s_t)$, maps each state s_t to a single action a_t. This approach often suits tasks with continuous action spaces, where directly computing the best action can be simpler or more efficient to implement. In contrast, a stochastic policy, written as $a_t \sim \pi(\cdot\,|s_t)$, assigns probabilities to possible actions for each state and is particularly useful in discrete or highly uncertain environments, as the randomness can help with exploration or handle existing variability in outcomes. In deep reinforcement learning (DRL), these policies are commonly parameterized by a function approximator, such as a neural network, whose parameters (e.g., weights and biases) are denoted by θ or φ. Thus, a deterministic policy might appear as $a_t = \mu_\theta(s_t)$, while a stochastic policy is expressed as $a_t \sim \pi_\theta(\cdot\,|s_t)$. Adjusting θ via gradient-based or other optimization methods allows the agent to iteratively refine its strategy over time [41].

Whether the policy is deterministic or stochastic, it encodes the agent's behavior, specifying which action to take – or how likely each action is – in each state. To evaluate this behavior, reinforcement learning uses value functions that estimate "how good" it is for the agent to be in a given state or to take a particular action, measured in terms of expected return – that is, the total reward the agent can expect to accumulate by following the policy thereafter.

Value functions appear in two common forms: the state-value function $V^\pi(s)$, which estimates the expected return from state s when following policy π, and the action-value function $Q^\pi(s,a)$ which measures the return from taking action a in state s, then continuing under π [37].

Mathematically, $V^\pi(s)$ and $Q^\pi(s,a)$ are defined as:

$$V^\pi(s) = \mathrm{E}_{\tau\sim\pi}\,[\,R(\tau)\mid s_0 = s\,],$$

$$Q^\pi(s,a) = \mathrm{E}_{\tau\sim\pi}\,[\,R(\tau)\mid s_0 = s, a_0 = a],$$

where $E_{\tau\sim\pi}[\cdot]$ is the expectation taken over future trajectories under policy π. In essence, $V^{\pi}(s)$ indicates how favorable it is to be in state s when following π, while $Q^{\pi}(s,a)$ refines that notion to specific actions in a given state. These functions are inherently policy-specific, as the expected return depends on the policy followed.

All RL algorithms ultimately aim to identify an optimal policy π^*, which maximizes expected return from any state. The corresponding optimal value functions are defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s),$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

By definition, the optimal action-value function $Q^*(s,a)$ represents the expected return obtained by acting a in state s, and then acting according to the π^* thereafter. The optimal policy, in each state, selects the action that maximizes this expected return. Thus, once $Q^*(s,a)$ is known, the agent can recover π^* by simply choosing:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a)$$

Following the policy $\pi^*(s)$ leads to the best possible long-term outcomes under the assumptions of an MDP. As a result, finding such optimal behavior – either by learning value functions or by directly improving the policy – is one of the main goals of reinforcement learning.

2.3.3 Online vs offline learning

In practice, RL agents typically learn from batches of experiences – collections of state, action, and reward tuples gathered across multiple trajectories. However, a fundamental question is how the agent obtains these batches in the first place. As shown in Figure 5, there are two approaches for collecting experience and training the agent: online and offline learning.

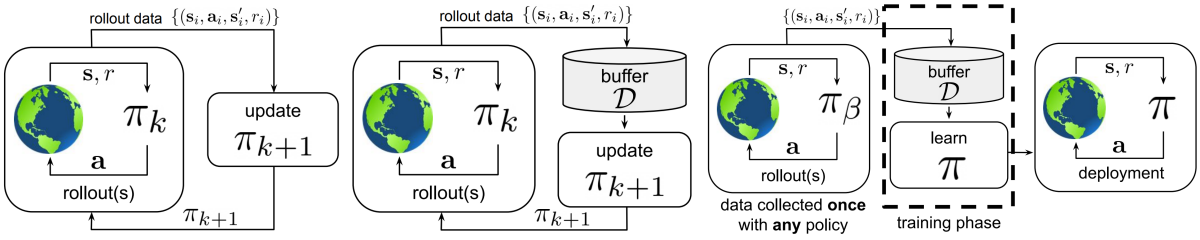

Figure 5. Comparison of RL paradigms. Online on-policy (left), where the agent learns from data collected by its current policy; online off-policy (center), where the agent stores past

interactions in a replay buffer D and learns from them later; and offline RL (right), where the agent is trained on a fixed dataset collected beforehand. Note that off-policy methods still involve interaction with the environment, unlike offline learning, which does not [43].

In online RL, the agent continually interacts with the environment and updates its policy either immediately based on new experience, or by learning from stored data using a replay buffer. This can occur in real-world settings, such as controlling a robot or optimizing an industrial process, where the agent explores and adapts its behavior based on real-time interactions with the physical environment. However, this process can often be highly costly or even dangerous [44]. Alternatively, the agent may operate in a simulator that approximates the real environment's dynamics. There, the agent may, then, explore multiple scenarios faster and more safely but risking a "sim-to-real" gap, if the model of the environment is imperfect [45]. Online RL is frequently employed in domains where agents can safely or efficiently run large numbers of trials, e.g., game environments, like DeepMind's Atari 2600 [46].

In contrast, offline reinforcement learning (also referred to as batch RL) relies on a fixed dataset of historical interactions, collected beforehand by other actors, through human demonstrations or past logs, without further interaction with the environment [43]. So, the optimal strategy is selected based on the trajectories present in the dataset. While this approach handles the risks of real-time trial-and-error, it depends on data coverage and quality of the existing dataset. If certain parts of the state-action space are missing or underrepresented, the policy learned may fail in those conditions. This is known as the distributional shift problem [47]. Offline RL has gained popularity in many real-world applications, such as in healthcare, where agents are trained on historical patient data to learn optimal treatment strategies without putting real patients at risk [48].

2.3.4 Model-free and model-based

Another important distinction amongst RL algorithms lies in whether they rely on an explicit model of the environment's dynamics. Algorithms that use such a model are referred to as model-based methods, whereas those that do not are known as model-free methods. Having a model allows an agent to plan ahead by simulating the consequences of potential actions before executing them in the real environment, which can improve data efficiency. The main difficulty is that a high-fidelity model is rarely available and must be learned from data, which can introduce bias: the agent may optimize for the model rather than for reality, resulting in sub-optimal or unsafe behavior.

Model-free methods take a different approach – they learn optimal behavior directly from interactions with the environment, focusing on estimating the value of actions based on the rewards received. Despite the fact that these methods may require significantly more data and real environment interactions for the agent to converge to a satisfactory policy, they are typically easier to implement and tune [49]. However, since they rely heavily on the reward signal to guide learning, it is important that the reward function is well-designed and encourages an effective learning process. Model-free approaches are more commonly used in RL and are generally considered to be capable of providing better performance [50].

2.3.5 Value-based and policy-based methods

Value-based and policy-based methods are two core branches of model-free reinforcement learning, each representing a distinct way of approaching the control problem.

Value-based methods aim to learn a parametric approximation $Q_\theta(s, a)$ of the optimal action-value function $Q^*(s, a)$. The update rule is built around the Bellman optimality equation, which defines a consistency relation between the value of a current state–action pair and the expected return of the best next action:

$$Q^*(s, a) = E_{s' \sim P}\left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right],$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot\,|s, a)$, indicating that the next state s' is sampled from the environment's transition dynamics [41].

In practice, algorithms minimize the difference between the current estimate $Q_\theta(s, a)$ and the target value defined by the right-hand side of the equation above. Because the Bellman target uses a maximum over actions, rather than the action taken, the update can be performed off-policy. This allows the algorithm to use data collected at any point during training, regardless of the behavior policy.

Once the Q-function is learned, a policy can be derived by selecting, in each state, the action with the highest estimated value:

$$\pi(s) = \arg\max_a Q_\theta(s, a)$$

Value-based methods are particularly effective in environments with discrete action spaces, where selecting the action that maximizes $Q_\theta(s, a)$ is straightforward. They have achieved impressive results in a variety of benchmark tasks [51], [52]. However, applying value-based

methods to continuous action spaces is significantly more challenging, as there is no simple way to compute the maximum over an infinite number of possible actions. Approximate solutions, such as discretizing the action space, often lead to poor performance due to loss of precision or exponential growth in the number of action choices.

Policy-based methods take a fundamentally different approach by directly modeling and optimizing the policy itself without explicitly representing value functions. Methods in this family define the policy as $\pi_\theta(a|s)$ – either as a probability distribution over actions (in the stochastic case) or as a deterministic mapping from states to actions. The main objective is to find the parameters θ that maximize the expected return, defined as:

$$J(\pi_\theta) = \mathrm{E}_{\tau\sim\pi_\theta}[R(\tau)], \text{ where } R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

and τ represents a trajectory of states and actions, sampled according to the current policy $\pi_\theta(a|s)$ and $R(\tau)$ is the cumulative reward. To optimize this objective, policy-based methods use gradient ascent, directly updating the policy parameters θ in the direction that increases the expected reward:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_t},$$

where α is the learning rate. This procedure involves calculating the gradient of the policy performance $\nabla_\theta J(\pi_\theta)$, known as the policy gradient, and algorithms that optimize the policy this manner are called policy gradient algorithms. In theory, repeatedly applying this update rule allows the parameters θ to converge towards values that maximize the objective $J(\pi_\theta)$. However, computing the gradient analytically presents two significant challenges. Firstly, it would require evaluating the probability of all possible trajectories, which quickly becomes computationally infeasible due to the exponential growth of possible state-action sequences. Secondly, it necessitates differentiating through the environment's dynamics, which typically are unknown or non-differentiable, especially in real-world scenarios.

Therefore, the Policy Gradient Theorem is used to provide an alternative formulation of the policy gradient [53]:

$$\nabla_\theta J(\pi_\theta) = \mathrm{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

This formulation expresses the gradient as an expectation that analytically would require an integral. In practice, the value of this expectation is typically estimated by sampling a fixed number of trajectories and using their sample mean as an approximation of the policy gradient:

$$\hat{g} = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

where D is the set of trajectories and $|D|$ is its cardinality. This is the simplest formulation of the policy gradient that is used by the REINFORCE algorithm [54], which is a well-known and widely used baseline within RL literature.

Policy-based methods are particularly well-suited to environments with continuous action spaces, as they can naturally model distributions over actions and often exhibit smoother learning dynamics than value-based methods, especially in high-dimensional or complex action spaces. In recent years, many variations of policy gradient algorithms have been developed – each estimating and improving the policy in its own way. However, pure policy-based methods often suffer from high variance and sample inefficiency, which has given rise to hybrid actor-critic frameworks that use a value-function critic to stabilize and accelerate learning.

2.3.6 Actor-critic

Actor-critic methods represent an important advancement in RL because they combine the strengths of policy-based and value-based approaches. The intuition is simple: learning a value function alongside the policy helps guide and stabilize policy updates, reducing gradient variance [55].

This class of algorithms is built around two components: an actor and a critic. The actor selects actions according to the current policy $\pi_\theta(a|s)$ and aims to maximize the expected cumulative reward. The critic, in turn, estimates a value function – typically $V(s)$ or $Q(s,a)$ – to quantify how good those actions are and to provide feedback that improves the policy over time [56]. In many actor-critic variants, the full return $R(\tau)$ in the basic policy-gradient formula is often replaced with a value estimated by the critic. This can be either the action-value $Q(s,a)$, or the advantage function $A(s,a) = Q(s,a) - V(s)$:

$$\nabla_\theta J(\pi_\theta) = \mathrm{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right]$$

In DRL, the actor and the critic are usually implemented as neural networks and may either be completely separate or share some layers, especially in environments where they receive the same inputs [57]. During training, these two networks are updated concurrently. The critic first computes a temporal-difference (TD) error – the gap between its current value estimate and the reward plus the discounted value of the next state – and minimizes this error with gradient descent, progressively improving its predictions. Simultaneously, the actor performs gradient ascent, using the feedback from the critic's TD error to favor actions that lead to higher returns. Unlike traditional policy gradient methods that wait until the end of an episode, actor-critic updates the policy incrementally at each time step using immediate feedback from the critic. The entire process can be visually illustrated in Figure 6.

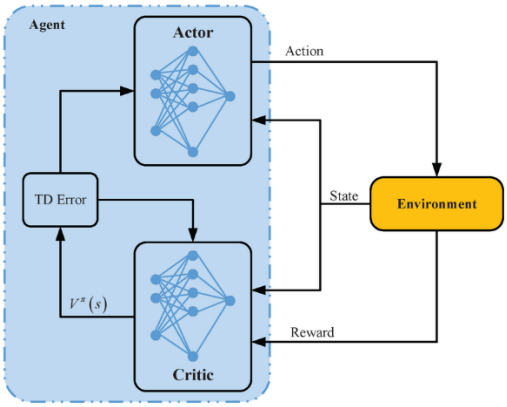

Figure 6. Actor-critic interaction with the environment [58].

This continuous feedback allows both components to evolve together, resulting in lower-variance updates and typically faster, more stable learning. Training stability, however, still depends on balancing actor and critic learning rates – if either network learns too quickly, it can destabilize the other. Despite this, the idea of combining policy- and value-based methods is now considered standard for solving RL problems. Most modern algorithms rely on actor-critics and expand this basic idea into more sophisticated and complex techniques.

2.3.7 DDPG

The introduction of neural networks in the Deep Q-Network (DQN) algorithm [51] was an important step in reinforcement learning, enabling value function approximation in high-dimensional spaces. However, DQN is limited to discrete actions, which makes it unsuitable for problems that involve continuous control, such as robotics or autonomous driving. To solve this, the Deep Deterministic Policy Gradient algorithm was introduced by Lillicrap et al., [59]. It is based on the Deterministic Policy Gradient method [60], which learns deterministic

policies by directly adjusting actions through gradient ascent, and it integrates key innovations from Deep Q-Networks (DQN), into an actor-critic architecture, effectively handling environments with continuous action spaces. These advancements categorize the DDPG as a model-free, off-policy actor-critic learning method.

The actor in this algorithm is a deterministic policy network, $\mu_\theta(s)$ which maps each state s directly to a specific action a in the continuous action space. The critic, on the other hand, is a Q-function approximator, $Q_\varphi(s,a)$, that evaluates how good a particular action is when taken in a given state. To solve the main challenge of balancing exploration and exploitation in continuous spaces, DDPG adds noise to the output of the deterministic policy. This noise is typically generated using a time-correlated Ornstein-Uhlenbeck process, but more recent findings suggest that simple uncorrelated Gaussian noise with zero mean can also be effective [61].

Similar to DQN and due to off-policy nature, DDPG employs an Experience Replay Buffer to store transitions (s,a,r,s'), collected during the learning process. By randomly sampling minibatches of these stored experiences, the algorithm breaks the sequential nature of the data, to benefit from learning across a set of uncorrelated transitions.

Additionally, DDPG make use of target networks for both the actor and critic, ${\mu'}_{\theta'}(s)$ and ${Q'}_{\varphi'}(s,a)$ which are delayed copies of the main networks $\mu_\theta(s)$ and $Q_\varphi(s,a)$. Unlike DQN, these target networks are updated continuously using a soft update strategy, where only a small fraction of the main network weights are added to the target networks at each step. This method further improves stability by preventing rapid oscillations and divergence in learning. This soft update helps to stabilize the learning and produce more consistent policy and value estimates.

Formally, DDPG uses the following learning algorithm to update the actor and critic at each time step [59]:

i. initialize the networks μ_θ and Q_φ with random weights.
ii. initialize the target networks as copies of the original ones, ${\mu'}_{\theta'} \leftarrow \mu_\theta, {Q'}_{\varphi'} \leftarrow Q_\varphi$.
iii. initialize the experience replay buffer D to store transitions.
iv. at each time step, observe the current s_t and select an action using the actor network with added exploration noise:

$$a_t = \mu(s_t|\theta^\mu) + \mathrm{N}_t,$$

where N_t is stochastic noise from an Ornstein-Uhlenbeck process or Gaussian distribution.

v. execute action a_t, observe the reward r_t and next state s_{t+1}.

vi. store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D.

vii. sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from D.

viii. compute the target Q-value for each sample using the target networks. If s_{i+1} is a terminal state than $y_i = r_i$, otherwise:

$$y_i = r_i + \gamma\, Q'_{\varphi'}\left(s_{i+1}, \mu'_{\theta'}(s_{i+1})\right)$$

The target value used for computing the critic's loss combines the immediate reward r_i with the discounted estimate of future return. To compute this return, the agent first asses the next state s_{i+1} through the target actor to obtain the next action and then evaluates the resulting state-action pair using the target critic network.

ix. update the critic network by minimizing the loss function L, which is Mean Squared Bellman Error:

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(Q_\varphi(s_i, a_i) - y_i\right)^2$$

x. update the actor network to maximize the expected discounted reward using the deterministic policy gradient:

$$\nabla_\theta J = \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \nabla_a Q_\varphi(s_i, a)\,|_{a=\mu_\theta(s_i)}\; \nabla_\theta \mu_\theta(s_i),$$

xi. Update the target networks of both actor and critic via soft updates:

$$\varphi' \leftarrow\ \tau\varphi + (1-\tau)\varphi',\ \theta' \leftarrow\ \tau\theta + (1-\tau)\theta'$$

with target smoothing factor $\tau \ll 1$

2.3.8 PPO

Proximal Policy Optimization is a RL algorithm introduced by Schulman et al. in 2017 [62]. Like other policy gradient methods, it directly optimizes a policy's objective function, but it was specifically designed to address the high variance and unstable updates found in earlier approaches such as REINFORCE and Actor-Critic.

PPO is still considered as Actor-Critic algorithm, as it combines a stochastic policy (actor) with a value function (critic) used to estimate advantages. What distinguishes PPO is its use of a principled optimization procedure that ensures each parameter update stays within a trust

region of the previous policy iteration. This trust-region constraint prevents large, destabilizing jumps and enables more reliable and stable training.

There are two main variants of PPO: PPO-Penalty and PPO-Clip. This thesis adopts the PPO-Clip implementation of the trust-region, as it is primarily employed by OpenAI and has demonstrated good performance and stability across a variety of tasks as referenced in [63].

The central idea in PPO-Clip is the clipped surrogate objective, which limits policy updates within a controlled range. Mathematically, the PPO-Clip objective function can be defined as:

$$L^{clip}(\theta) = E_t[\min(p_t(\theta)A_t, clip(p_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

where $p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies, and ϵ is a clipping hyperparameter that controls the maximum allowed deviation from the previous policy. The term A_t is the advantage function, often computed as $A_t = \hat{R}_t - V(s_t)$, where $\hat{R}_t$ is the estimated return at t, and $V(s_t)$ is the critic's value estimate. It measures how much better or worse acting a_t in state s_t was compared to the average.

Since the objective takes the minimum between the unclipped and clipped terms, and the clip function ensures the ratio stays $[1-\epsilon, 1+\epsilon]$, the optimizer is no longer motivated to push policy updates to extremes in order to obtain greater rewards, even if they appear to yield higher advantages [64].

In addition to stabilizing policy updates, another important aspect worth mentioning is how PPO handles the trade-off between exploration and exploitation. Because the policy is stochastic, it naturally explores widely early in training; as learning progresses, it tends to exploit high-reward actions more often [65]. To delay premature convergence, PPO often adds an entropy bonus to the objective. Entropy quantifies the uncertainty of the policy's action distribution: higher entropy means more randomness and thus more exploration. By rewarding entropy, the algorithm discourages the policy from becoming too deterministic too quickly, reducing the risk of settling on a sub-optimal solution.

PPO uses the following iterative algorithm to update the policy and value networks during training:

i. Initialize the policy network $\pi_\theta(a \mid s)$ and the value function network $V_\phi(s)$ with random parameters θ_0 and ϕ_0.

ii. For each iteration k, run the current policy $\pi_k = \pi(\theta_k)$ in the environment and collect a batch of trajectories $\mathcal{D}_k = \{\tau_i\}$, where each $\tau_i = (s_0, a_0, r_0, \ldots, s_T)$.

iii. For each trajectory τ, compute the rewards-to-go $\hat{R}_t$ from time step t where

$$\hat{R}_t = \sum_{t'=t}^{T} R\left(s_{t'}, a_{t'}, s_{t'+1}\right)$$

iv. Compute the advantage estimates $\widehat{A_t}$ using any technique (e.g. Generalized Advantage Estimation [66]) based on the current value function $V_{\phi_k}(s)$.

v. Update the policy parameters θ_{k+1} by maximizing the PPO-Clip surrogate objective:

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \frac{1}{| \mathcal{D}_k | T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(p_t(\theta)\hat{A}^{\pi_{\theta_k}}(s_t, a_t), \operatorname{clip}(p_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_{\theta_k}}(s_t, a_t)\right)$$

where $p(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$. This optimization is typically done using stochastic gradient ascent with the Adam optimizer.

vi. Update the value function parameters ϕ_{k+1} by minimizing the mean squared error between the predicted state values and the computed returns:

$$\phi_{k+1} = \underset{\phi}{\operatorname{argmin}} \frac{1}{| \mathcal{D}_k | T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

This is typically done via gradient descent.

Although PPO has limitations, including sensitivity to hyperparameter tuning, a conservative clipped objective that tends to prioritize stability over exploration, and potentially slower learning compared to other policy gradient methods, it remains the most widely used on-policy reinforcement learning algorithm.

3. Methodology

The initial goal of this research was to train a reinforcement learning agent capable of optimizing battery trading strategies in different Estonian electricity markets. As mentioned, in Chapter 2, Eesti Energia already employs a strong method based on linear optimization for optimizing battery trading decisions. But despite its efficiency, it faces several limitations. For example, the linear optimizer works solely on the basis of available forecasts, unable to effectively incorporate uncertain or delayed market information, such as future intraday or mFRR market prices, which are often correlated with day-ahead price patterns. In addition, linear constraints limit the modeling of more complex nonlinear phenomena, such as battery degradation, which varies nonlinearly with the state of charge. This is where RL can offer a distinct advantage, as it is naturally suited to environments with delayed feedback, complex dynamics, and high levels of uncertainty in both inputs and outcomes. It is particularly advantageous when considering simultaneous price forecasts for multiple interconnected markets. However, given the complexity of RL and the nature of electricity markets, this thesis limits its scope to the day-ahead market, laying a foundational framework for subsequent expansion to other electricity markets.

3.1 Data

RL models require huge amounts of samples for effective training, typically on the order of millions of interactions [67]. As mentioned earlier, it would be possible to train an RL model directly in a real live market environment, continuously receiving new data as time progresses. However, this approach is impractical, as it would expose the system to financial risks and require an unfeasible amount of time to collect enough real-world interactions for effective learning. Therefore, in practice, training is usually performed either using large volumes of historical data or through the creation of environment simulators that can generate synthetic experiences for the agent.

Our research focuses on optimizing battery energy storage system in Estonia's DA electricity market. The core dataset available consists of hourly day-ahead market prices and corresponding forecasts for the period from 2022 to 2024, where the price forecasts were provided by Eesti Energia, and the actual prices were sourced from the Nord Pool exchange. This historical dataset included approximately 50000 hourly data points, which was not enough to train a powerful RL model. To overcome this data limitation, a custom synthetic data

generator was developed that is designed to produce realistic price and forecast scenarios based on historical electricity market behaviors.

To build the generator, the historical data was cleaned by removing unnecessary columns and calculating the residuals - the difference between the actual market prices and the forecasted prices. Capturing residuals was important because they reflect realistic forecast errors linked to specific price patterns, preserving the relationship between actual prices and their predictions.

Next, the data was reformatted so that each sample represented a full day: 24 consecutive hourly actual prices and their corresponding residuals. To account for seasonal variations in electricity price dynamics, the dataset was grouped by month. For each month, we created a multivariate normal distribution, using the monthly median as the mean and computing the covariance matrix to capture both the dependencies between different hours within a day and the relationship between prices and their forecast errors.

During training, synthetic data were sampled from these monthly distributions. Each generated sample consisted of 48 values: the first 24 values represented synthetic actual prices, and the next 24 values represented their corresponding residuals. Synthetic forecasts were then reconstructed by subtracting the residuals from the actual prices.

This approach allows the RL agent to request new, realistic 24-hour sequences of prices and forecasts at any time, ensuring a continuous and diverse data stream for training without being limited by the historical dataset.

3.2 Environment setup

A custom reinforcement learning environment was developed based on the Gymnasium interface [68] to mimic the trading behavior of a battery energy storage system on the Estonian day-ahead market. It is made to simulate the battery's physical behavior under operational limitations as well as the process of making economic decisions in a volatile electricity market.

3.2.1 Parameters

The environment is configured using a set of fixed parameters that define the characteristics of both the battery and the market prices. This battery configuration is based on the technical specifications of the energy storage system installed at the Auvere industrial complex in Ida-

Virumaa, which, as of its commissioning in February 2025, is the largest battery storage facility in Estonia [69]. The Table 1 below summarizes these main settings:

Table 1. Simulation parameters used to define battery and market characteristics.

Parameter	Value	Unit	Description
MAX_BATTERY_CAPACITY	50	MWh	Maximum energy storage capacity of the battery
MAX_POWER_CHARGE	26	MW	Maximum charging or discharging power per hour
INITIAL_BATTERY_LEVEL	20	MWh	Initial battery level at the start of training
DEPRECIATION_COEFF	10	€/MWh	Cost of battery degradation per unit of discharged energy
IMBALANCE_PENALTY	50	€/MWh	Penalty cost for imbalance between planned and actual trading
PRICE_MIN, PRICE_MAX	[-500, 800]	€/MWh	Minimum and maximum possible electricity market price

In reality, certain values, such as battery depreciation or imbalance penalties, would typically be derived from operational performance and real market behavior.

These parameters define the operational space within which the agent must learn to optimize the trading strategy.

3.2.2 State and action spaces

The agent receives the state parameters generated within our custom battery trading environment. In our ideal environment design, at each decision step, the agent's observation includes:

- the current battery state of charge (SOC)
- the 24 hours forecasted electricity prices

These selected states describe both the internal status of the battery and the expected market conditions for the upcoming trading day, which will guide the agent to learn good actions across various possible sets of observations.

However, it is worth noting that throughout the experimental phase, the observation space was simplified for training purposes. Instead of providing the forecasted prices, actual market prices were used (in different variations), allowing the agent to operate with fully known price information. Nevertheless, the intended final environment setup assumes that the agent relies exclusively on forecasted price data, reflecting real-world operational constraints.

The agent operates within a continuous action space, where each action consists of a 24-dimensional vector. Each element of this vector specifies the amount of energy to be charged or discharged during a specific hour of the upcoming trading day. Positive values correspond to charging - buying electricity, while negative values correspond to discharging - selling electricity. All actions are constrained by the battery's maximum allowable charge or discharge of ±26 MW per hour.

3.2.3 Execution logic

After the agent selects an action representing its planned energy schedule for the upcoming day, the environment executes several internal processes to reflect both economic outcomes of trading decisions and the physical constraints of battery operation.

First, the environment calculates bid revenues, which represent the theoretical profit or loss that would be obtained if the agent's proposed actions were executed exactly as planned under the actual cleared prices from the day-ahead market. The bid revenue is computed as the sum of the negative product between the scheduled energy quantities and corresponding actual hourly market prices, ensuring that energy purchases are treated as costs and energy sales as revenues:

$$\text{Bid Revenues} = -\sum_{t=1}^{24} (q_t \times p_t)$$

where q_t is the action at hour t and p_t is the actual market price at hour t.

Due to the physical limitations of the battery, the agent's initially proposed actions might not be entirely executable. The environment, therefore, calculates a valid trading schedule that meets the battery constraints (maximum battery capacity and power limits). The validation logic dynamically considers the SOC:

- For charging actions, the agent can charge the battery only up to its maximum capacity. If the proposed amount exceeds the available space - maximum capacity minus current SOC, it is automatically reduced.
- For discharging actions, the agent can discharge only as much energy as is currently stored in the battery. If the proposed discharge exceeds the available energy, the action is clipped accordingly.

Any discrepancies between the proposed actions and valid schedule result in imbalances. Such imbalances lead to financial penalties or reduced revenues:

- Underconsumption (charging less than planned) results in surplus energy sold back to the market at a reduced price.
- Underproduction (discharging less than planned) results in shortfalls covered at a higher cost.

Imbalance revenues are computed as:

$$\text{Imbalance} = \sum_{t=1}^{24} \begin{cases} \left(|\, q_t - v_t \,| \times (p_t - \text{IMBALANCE\_PENALTY})\right) & \text{if } q_t > 0 \text{ and } v_t < q_t \\ -|\, q_t - v_t \,| \times (p_t + \text{IMBALANCE\_PENALTY}) & \text{if } q_t < 0 \text{ and } v_t > q_t \end{cases}$$

where v_t is the valid scheduled quantity at hour t.

Additionally, the environment considers battery depreciation, a cost proportional to the energy discharged from the battery (reflecting wear and tear). It is calculated as:

$$\text{Depreciation} = \text{DEPRECIATION\_COEFF} \times \sum_{t=1}^{24} |\, v_t \,|_{\text{discharge}}$$

The final daily reward for the agent combines all these financial outcomes:

$$\text{Reward} = \text{Bid Revenues} + \text{Imbalance Revenues} - \text{Depreciation Cost}$$

As a result, it reflects the agent's ability to plan profitable actions under the given market forecasts and operational constraints.

After calculating the reward, the environment transitions to the next state. The battery's SOC is updated to the final level achieved at the end of the trading day, and a new set of forecasted prices for the upcoming day is generated using our synthetic data generator. Actual market prices are also generated internally for reward calculation but are not

provided to the agent. To ensure that generated prices represent accurately historical seasonal changes, the environment monitors the advancement of simulated days and shifts the current sample month after every 30 days, therefore initiating a new monthly distribution.

3.2.4 Reward scaling

Reward scaling is an important “best practice” in reinforcement learning to improve the stability of training, speed up convergence, and prevent problems caused by large or highly variable reward values. Proper scaling ensures that the agent receives consistent and meaningful learning signals.

In this study, two different methods were used: dynamic spread-based scaling and fixed constant scaling.

3.2.4.1 Spread-based scaling

The spread-based scaling method dynamically normalizes the reward based on the observed daily market conditions. This aligns the agent's feedback with the relative profitability potential of each trading day.

To achieve this, two boundary scenarios were estimated:

- Best-case reward: The battery is assumed to charge at the lowest-priced hours and discharge during the highest-priced hours. This represents the maximum potential profit that can realistically be made on that day, after accounting for battery depreciation costs. Only spreads larger than the depreciation threshold is considered profitable and contribute to the best-case estimate.
- Worst-case reward: The battery is assumed to do the opposite - charging at the highest prices and discharging at the lowest, leading to the worst possible financial outcome.

The actual daily reward resulting from the agent’s actions is then scaled to the range [−1, +1] based on these extreme scenarios, although in some cases the scaled reward may be slightly larger than this range. Specifically:

- If the best-case reward is positive, the actual reward is linearly scaled between [-1, 1], using min-max normalization:

$$\text{Scaled Reward} = 2 \times \frac{\text{Actual Reward} - \text{Min Reward}}{\text{Max Reward} - \text{Min Reward}} - 1$$

- If the best-case reward is non-positive, indicating that even under perfect trading no profit is achievable, the rewards are scaled between [-1, 0].

$$\text{Scaled Reward} = \frac{\text{Actual Reward} - \text{Min Reward}}{0 - \text{Min Reward}} - 1$$

This dynamic normalization ensures that the agent is evaluated relative to the day's trading conditions, not in absolute financial terms.

3.2.4.2 Fixed scaling

Another scaling approach that was considered is fixed reward scaling, which applies a constant factor to normalize all daily rewards independently of specific market conditions. The idea behind this method is to select a realistic scaling constant C that reflects the typical magnitude of daily profits or losses and then normalize all rewards by dividing them by this value.

To identify a suitable value for C, the distribution of unscaled rewards was analyzed during preliminary evaluations of the environment, see Figure 7. The analysis showed that most daily financial profits fell between [−10000, -10000], with the rare extreme values exceeding this range.

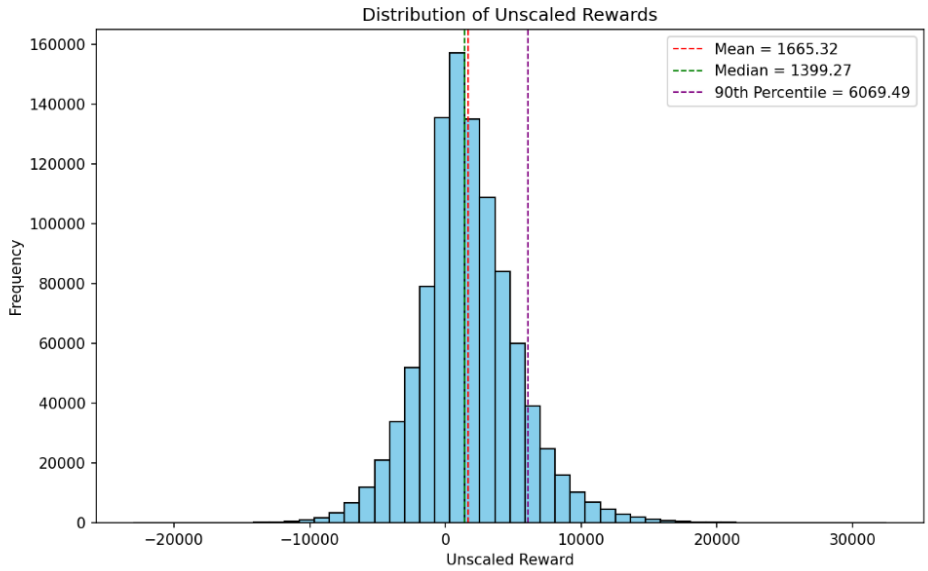

Figure 7. Distribution of the unscaled daily rewards, collected during preliminary environment evaluation

Based on these observations, the 90th percentile was selected as the scaling constant, and daily rewards were normalized by dividing each value by this constant.

Formally,

$$\text{Scaled Reward} = \frac{\text{Actual Reward}}{C}, \text{where } C = 6000$$

As a result, the majority of rewards are scaled into the range [-1, 1], while more extreme outcomes may map slightly beyond these limits.

3.3 Algorithms

The focus of this thesis was on applying reinforcement learning methods to optimize battery trading in an electricity market, rather than developing new neural network architectures. As a result, most of our experiments used standard DDPG and PPO architectures provided by the CleanRL library [70]. This codebase was chosen for its ease of modification, and transparency, allowing for rapid experimentation and adaptation to the specifics of our environment without introducing unnecessary implementation complexity.

3.3.1 DDPG architecture

For Deep Deterministic Policy Gradient experiments, the original actor-critic network architecture from the CleanRL implementation was used. Both networks, actor and critic, use fully connected neural network layers with straightforward structures.

Actor network takes the environment's state observation as input and outputs continuous actions. It consists of two hidden layers, each with 256 neurons and ReLU activation functions. The output layer uses a tan activation followed by scaling and shifting to map actions into the environment’s original action space range.

The critic network takes both the state and action as inputs, which are concatenated into a single vector, and outputs a scalar Q-value estimating the expected return of the action taken in the given state. Like the actor, it also contains two hidden layers with 256 neurons each, followed by ReLU activations, and a linear output layer.

A graphical representation of this standard DDPG architecture is shown in Figure 8.

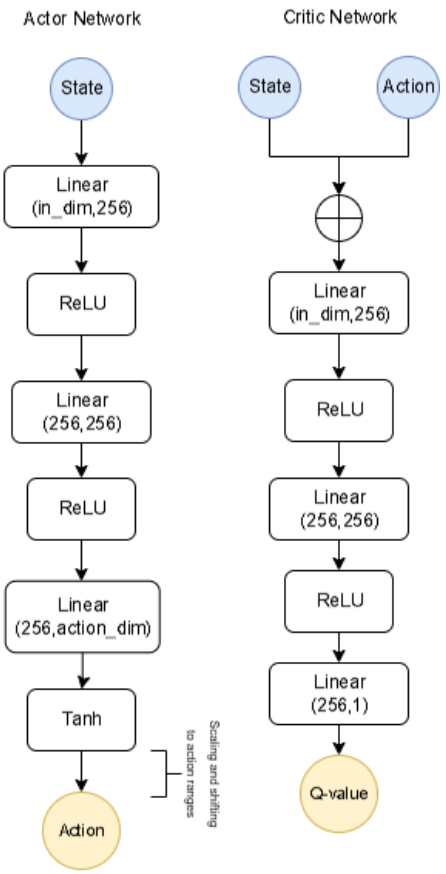

Figure 8. Actor and Critic network architecture in DDPG

While most experiments used this standard architecture, experiments replacing ReLU activations with tanh in both the actor and critic networks were performed, but this did not yield improvements. At later stages, the network architecture was slightly modified. First, LayerNorm layers were inserted after each hidden linear layer in both the actor and critic, helping stabilize training. Second, the critic now processes the observation alone in its first layer and concatenates the action only after that transformation, rather than feeding state + action together from the start. These adjustments impacted our final results.

3.3.2 PPO architecture

For Proximal Policy Optimization experiments, the standard actor-critic network architecture provided by the CleanRL library was adopted as the baseline. PPO utilizes a stochastic actor, which outputs parameters of a probability distribution, from which actions are sampled, along with a critic network that estimates the value of given states.

A graphical representation of the PPO architecture used in our study is shown in Figure 9.

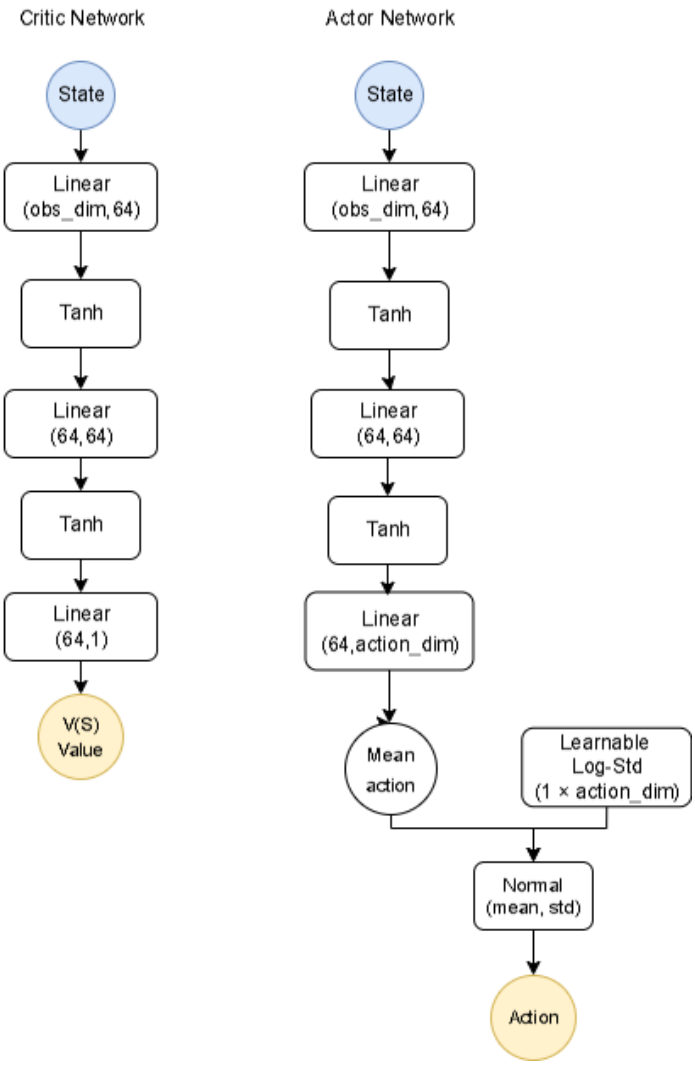

Figure 9. Actor and Critic network architecture in PPO

The architecture consists of two main components:

- Actor network processes the environment's state observation through two fully connected layers, each containing 64 neurons and tanh activations. The final output from the actor consists of mean action values produced by a linear layer, paired with learnable log-standard deviation parameters for the action distribution. Actions are sampled from a Gaussian distribution parameterized by these outputs.
- Critic network also consists of two fully connected layers with 64 neurons each and tanh activations. It outputs a single scalar value representing the estimated state-value function, indicating the expected future return from the given state.

3.4 Training setup

This section describes how the RL agents were trained, including the structure of training episodes and the final hyperparameter configurations used for the DDPG and PPO models.

3.4.1 Training episodes

The training process for reinforcement learning models can be organized into episodes. The way episodes are structured significantly impacts how the model learns and generalizes. This study examined two different approaches to structuring episodes: a fixed-length episodic setup and a continuous setup.

In the fixed-length episodic setup, each episode covered a 3-day horizon. The agent received cumulative rewards over this period without applying discounting ($\gamma = 1.0$). This formulation allowed the agent to optimize short-term trading strategies without considering effects beyond the 3-day horizon. Such setup was used only during early experiments to focus on short-term profit strategies and reward tuning. After each episode, the environment was completely reset, including the battery state and price sequences. However, this artificial reset was later found to be unrealistic, as real-world batteries operate continuously without returning to a predefined starting state.

In the continuous setup, the environment operated without predefined episode boundaries, simulating uninterrupted battery operation over a long-time horizon. The agent continuously interacted with the environment, and learning proceeded until a fixed number of learning steps was reached. Here, the discount factor of $\gamma = 0.9$ was used to balance short and medium-term rewards. It approximately corresponds to a 10-day horizon, where $T \approx \frac{1}{1-\gamma} = \frac{1}{1-0.9} = 10$. Alternative values for γ were tested, but both higher and lower settings led to worse performance. This setup better reflected real-world battery conditions, so most of the later experiments used it to support long-term planning.

3.4.2 Training configuration

The configurations presented here correspond to the final training settings used for the best DDPG and PPO models. While many initial experiments relied on the default hyperparameters from the codebase, several key parameters were iteratively adjusted to improve performance based on empirical observations. The DDPG model was trained in a single continuous environment for 1 million timesteps. Learning began after a warm-up phase of 50000 steps,

during which actions were sampled randomly. A batch size of 256 was used for training, with experiences drawn from a replay buffer containing up to 1 million transitions. The actor and critic networks were optimized using the Adam optimizer with learning rates of 1e-3 and 1e-4, respectively. Soft target updates were applied with a smoothing coefficient of $\tau = 0.01$. For exploration, Gaussian noise scaled to 10% of the action range was added to the policy output. Table 2 summarizes the main hyperparameters of DDPG:

Table 2: Final parameters used in the best DDPG model version

Hyperparameter	Value
Total timesteps	1000000
Discount factor (γ)	0.9
Learning rate	1e-3(actor), 1e-4 (critic)
Replay buffer size	1000000
Batch size	256
Learning starts	50000
Policy updates frequency	Every 2 steps
Target smoothing (τ)	0.01
Exploration noise scale	0.1

The PPO model was also trained for 1 million steps using 8 parallel environments. Each rollout collected 1024 steps per environment, resulting in a batch size of 8192. The actor-critic network was updated for 10 epochs after each rollout, using 32 minibatches per epoch, each containing 256 samples. The policy was optimized with the Adam optimizer and a learning rate of 5e-4, linearly annealed over the training. A clipped surrogate objective was used with a clipping coefficient of 0.3 to stabilize updates. Entropy regularization was also applied to encourage exploration, with the coefficient gradually decaying from 0.1 to 0.001.

Table 3: Final parameters used in the best PPO model version

Hyperparameter	Value
Total timesteps	1000000
Number of environments	8
Rollout steps per env	1024
Batch size	8192
Number of minibatches	32
Update epochs	10
Discount factor (γ)	0.9
GAE lambda	0.95
Learning rate	5e-4 annealed
Clipping coefficient	0.3
Value loss coefficient	0.5
Entropy coefficient	0.1 → 0.001

3.5 Evaluation

This section outlines the metrics and procedures used to assess agent performance in both fixed-length episodic training and the continuous interaction setting. It explains how final profitability results were obtained and compared across models, along with additional training metrics recorded to monitor learning progress.

3.5.1 Episodic setup

To measure the performance under the fixed-length episode design, the "episodic return" metric was collected during training. It captures the total cumulative reward obtained over a full 3-day episode when a termination state is reached. This metric is used to compare results between episodic and continuous setups in Section 4.7. A comparable metric was calculated in the continuous environment by summing daily rewards every three steps. For the final evaluation, 5 episodes were run without noise and scaling, and the actual financial revenues were calculated as the average return over these 3-day periods.

3.5.2 Continuous setup

As mentioned, in a continuous setting, the agent interacts with the environment without episodic resets, receiving a reward after each simulated trading day. However, daily rewards, especially when they are affected by reward scaling, exploration noise, and fluctuating market

prices, are not a reliable metric for assessing the long-term performance of the learned policy. Therefore, a periodic offline evaluation procedure was implemented.

At fixed intervals during training, namely every 10000 steps, the current model weights were frozen, and a deterministic evaluation was performed by turning off exploration noise and removing reward scaling to reflect actual financial returns of the trading strategy. The policy was evaluated over four representative months (January, April, July, and October) to capture seasonal price variations. For each month, the model was run on a fixed 15-day sequence of synthetic price data and accumulating the total returns. To ensure comparability across different models and training stages, the same price samples and environment conditions were used in all evaluation runs.

The main evaluation metric, the average return over 4 months, was calculated as:

$$\text{Mean Return} = \frac{1}{4}\sum_{m=1}^{4} R^{(m)}$$

where $R^{(m)}$ is the total profit accumulated over 15 consecutive trading days in month m.

This metric served as the basis for evaluating our DDPG and PPO models, as well as for comparing their performance against the benchmark linear optimization strategy.

In addition to the main evaluation procedure, various auxiliary training metrics were logged to monitor the model's behavior during training. These included value and actor losses for both used models, as well as a number of diagnostic indicators for PPO, such as entropy, explained variance and fraction of the data that triggered the clipped objective. However, these metrics were used only for monitoring purposes and were not considered in the final model comparison, as the main focus was on the resulted financial profit achieved by each model.

4. Results and discussion

This chapter sequentially presents and analyzes the experiments conducted during the research to optimize the BESS using reinforcement learning algorithms in the day-ahead electricity market. It outlines the methodological evolution, starting from the initial setup with DDPG, followed by environment simplifications, a detailed exploration of the impact of various architectural and hyperparameter choices for both DDPG and PPO, and concludes with the final results. These results are compared to the baseline linear optimization model (hereinafter referred to as the Optimizer) provided by Eesti Energia (see Section 2.2).

4.1 Implementation details

The initial training was conducted on a local machine equipped with a standard CPU and 16 GB of RAM. However, due to the computational demands of the model, each training session, on average, required over 6.5 hours to complete, even in the simplified setup. For faster experiments, training was later migrated to a more powerful cloud environment. A single-node Databricks cluster with 28 GB of RAM, 4 CPU cores, and 1 NVIDIA Tesla T4 GPU was used. This helped reduce the average training time to approximately 3.5 hours.

In total, over 70 training runs were performed, with the most effective results summarized in the subsequent sections. All experiments were systematically logged using Weights & Biases ("wandB") framework to track performance metrics and hyperparameter variations, visualize learning curves and ensure full reproducibility of experiments. The experiments were conducted with Python 3.10.4 and used various libraries for both data manipulation and reinforcement learning-specific tasks. Table 4 lists these libraries and their respective versions.

Table 4. Python libraries used for conducting the experiments.

Library	Version
numpy	1.24.4
pandas	1.3.5
scipy	1.14.1
gymnasium	0.28.1
stable-baselines3	2.0.0
pytorch	1.12.1
wandB	0.19.1

In all experiments, the random seed was fixed to the value 1 for reproducibility, except in cases where the models were run with different samplings to reflect the variability of the results.

4.2 Initial setup

At the beginning, the experiments were designed using the DDPG algorithm due to its simplicity and as a way to approach the problem with continuous actions. The baseline model incorporated default algorithm parameters provided by the CleanRL library and utilized a spread-based scaling approach for rewards(see Section 3.2.4) to stabilize training. The first iteration of the environment was built to operate using 24-hour forecasts of electricity prices as input. However, early experiments showed the huge learning difficulties due to the complex nature of both forecasts and size of input, indicated by declining financial returns.

To simplify the training, two key changes were made:

- model input was shortened from 24-hour to 6-hour prices window (ultimately, it was reset to 24-hour for the final experiments once modelling was optimized, as will be shown in Section 4.6).
- forecasts were replaced by actual historical prices, to remove noise and uncertainty from the input space.

These adjustments led to immediate improvements: returns turned positive, confirming the benefit of simplifying the environment for further experiments.

4.3 Observation space experiments

The choice of state representation is important in RL, because it can significantly affect the agent's ability to anticipate future price changes and respond accordingly.

To explore how different observation spaces influence the model performance, two state configurations were tested:

- 6 hours of today’s prices + state of charge (SOC) – the base configuration used in the initial model.
- 6 hours of today’s prices + 6 hours of tomorrow’s prices + SOC – an extended input structure that increases the observation window to 12 hours.

In both cases, the reward was computed only based on today's prices, while tomorrow’s prices were provided as a static input, serving as a hint of future market conditions. Because the

generator samples each day prices from a monthly distribution independently, no inter-day price correlation is available in the raw data, so the agent cannot deduce tomorrow's trend from today's observations. Therefore, an additional experiment was conducted to test whether explicitly providing the next-day prices would help the model exploit any short-term structure that might still emerge during training and, in turn, improve its charging-discharging schedule.

Table 5. Comparative analysis of the influence of different observation space formats on model performance.

Model	January	April	July	October	Mean return for 4 months in euro
Optimizer, today+tomorrow input	16775.6	20981.7	57277.6	62400.7	39358.9
Optimizer, today-only input	10697.5	22322.1	44382.4	62367.1	34942.3
RL, today+tomorrow input	9809.0	11063.3	36996.7	50156.1	27006.3
RL, today-only input	13479.4	10176.4	43903.2	44324.3	27970.8

Table 5 summarizes the comparative performance of these two configurations, including their evaluation against the Optimizer. Each value represents the total profit, measured over 15 days for a specific month. The final column shows the mean monthly profit across the four months.

Experiments showed that adding tomorrow's prices to the observation space does not yield a consistent profit gain for the RL agent. In contrast, the linear optimizer benefits from this additional information, since it is explicitly designed to optimize over a fixed and known price horizon. Even so, the extended input structure (today + tomorrow) was retained in all subsequent experiments, under the assumption that access to future prices is generally preferable and could help the model become more robust and adaptable to varied market conditions.

4.4 PPO model tuning

The CleanRL implementation of Proximal Policy Optimization was adopted as a second baseline. The default configuration, which included observation normalization, reward scaling, where rewards are divided by the standard deviation of a rolling discounted return, and no entropy regularization, failed to produce a profitable strategy. The agent's performance

stagnated at a mean 15-day profit of under €1000, averaged across four months. Following accepted implementation guidelines [71] the observation normalization was kept but replaced the original reward scaling with the spread-based reward scaling scheme already used for DDPG.

A heuristic hyperparameter search was then conducted across the parameters that most strongly affect PPO in continuous-control tasks:

Entropy coefficient

The entropy coefficient was varied in the range [0, 0.1]. Constantly high entropy values during training quickly degraded the model performance, while extremely low or zero entropy led to premature convergence and insufficient exploration. A linearly decaying entropy coefficient from 0.1 to 0.001 was found to provide the optimal balance between exploration and stability.

Initialization gain in the final actor layer

To evaluate how the scale of the initial action means affects learning, the initialization gain of the final layer in the actor network was explored within the range [0.01, 0.15]. This gain sets the magnitude of the layer's initial weights, and therefore the size of the action means at the start of training, while the log-standard deviation vector remains at zero until it is updated during learning. A default low gain of 0.01 resulted in near-zero initial means, so the agent began with tiny, almost deterministic buy/sell actions; exploration was safe but slow, and early returns remained modest. On the other hand, higher gains led to large initial means, causing overly aggressive actions, higher reward variance and worse learning behavior. The best results were obtained with the gain of 0.05.

Neural network size

To explore whether increasing the complexity of the neural network would allow the agent to capture more complex trading strategies, three configurations were tested: the default 64×64, 128×128 and 256×256. Deeper networks showed no improvement in returns but only increased the training time. Therefore, the simpler 64×64 architecture from CleanRL was retained as the optimal one.

Number of parallel environments

The impact of varying the amount of data collected for each update was studied by increasing the number of parallel simulation environments from 1 to 8. Using more parallel rollouts

sharply reduced the gradient variance, resulting in smoother and more stable learning curves, although it did not directly improve the final evaluation performance. Therefore, 8 environments were adopted for all subsequent experiments.

Learning rate and clipping coefficient

In experiments with PPO, tuning the learning rate (LR) and the clipping coefficient (ε) played a key role in the agent's performance. The LR controls the step size of policy updates and governs the speed of adaptation, while ε limits the magnitude of policy shifts to avoid large and destabilizing changes. These parameters needed to be tuned together: a high LR accelerates learning but risks instability, while a strict ε prevents erratic updates but may slow progress if too restrictive. Testing showed that setting both parameters too high, such as $lr = 0.01$, $\varepsilon = 0.3$, caused sharp drops in learning curves due to unstable updates. Following practical guidance from the original paper [62], LR values in the range [0.01, 0.0005] were evaluated using linear annealing throughout training, along with $\varepsilon \in \{0.2, 0.3\}$. The combination $LR = 0.0005$ and $\varepsilon = 0.3$ proved to be the most effective, achieving stable convergence and higher trading revenues.

As seen in the evaluation results represented in Figure 10, the mean 15-day profit (4-month average) metric is plotted against training steps in order to show the impact of our tuning efforts on the agent's battery trading performance. The basic PPO configuration quickly converged to an unprofitable policy that had a mean profit of only €730. Applying the custom reward scaling improved returns to €2050 but got the agent to plateau early on with the sub-optimal strategy. Introducing a decaying entropy coefficient enhanced exploration and performance to €16000, while employing 8 parallel environments stabilized the evaluation curve. The best results came from the tuned learning rate and clipping coefficient, with a final mean profit of €23830, reflecting the most effective trading strategy.

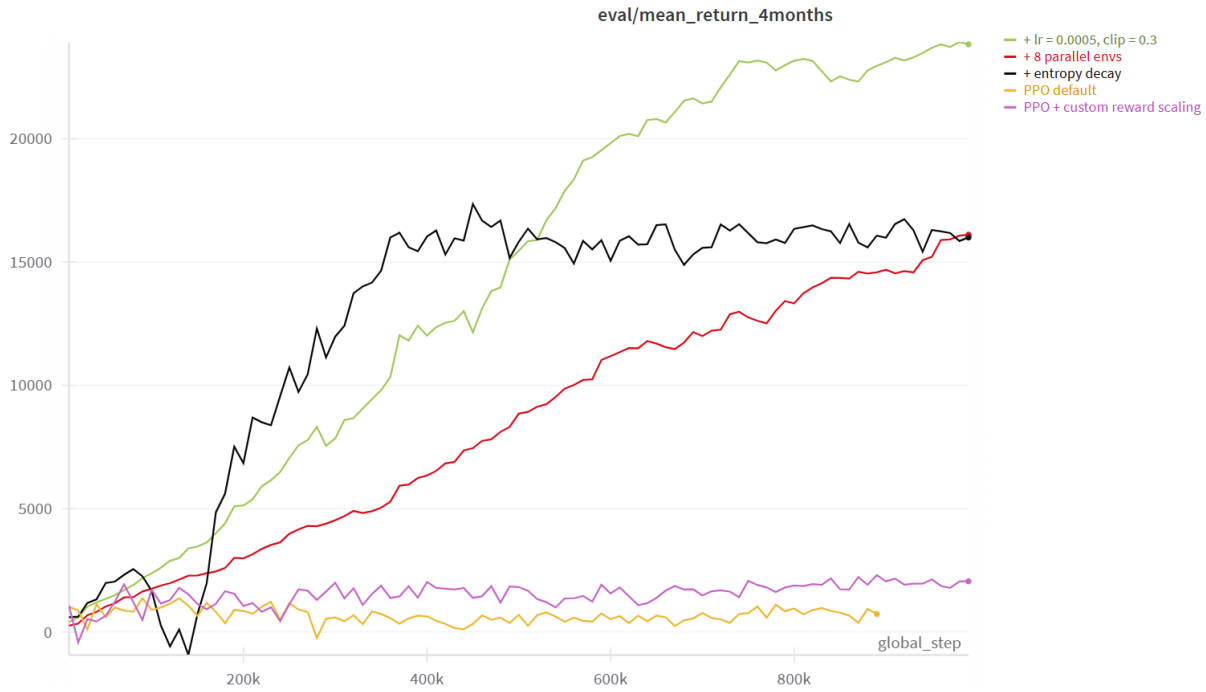

Figure 10. Evaluation performance curves of different PPO configurations, showing the mean 15-day profit in euro averaged over four months against training steps.

Table 6 compares the performance of the best PPO configuration (see Section 3.4.2) with the original DDPG baseline. Even with careful tuning, the PPO model underperforms by approximately 12% in terms of mean revenues, with lower profits in every individual month.

Table 6. Comparative performance analysis of Optimizer, the best PPO configuration and the default DDPG model.

Model	January	April	July	October	Mean return for 4 months in euro
Best PPO	6639.8	9777.6	32706.0	46197.7	23830.3
Default DDPG	9809.0	11063.3	36996.7	50156.1	27006.3
Optimizer	16775.57	20981.70	57277.61	62400.75	39358.9

Based on these results, the focus was shifted back to the DDPG algorithm and its tuning, aiming to reduce the gap with the established benchmark of linear optimization.

4.5 DDPG model tuning

To further improve the performance of the DDPG model, a series of isolated modifications to key training parameters was made. Initial experiments included changes to the learning rate within the range of [1e-4, 1e-3], tuning exploration noise between 0.05 and 0.2, introducing a decaying noise schedule, and switching from Gaussian noise to Ornstein-Uhlenbeck action noise. However, none of these individual modifications produced meaningful or consistent improvements over the default baseline.

Returning to the baseline configuration and following the approach from the original DDPG paper [59], different learning rates were introduced for the actor and critic networks. The original assumption was that the critic should adapt faster because the actor relies on its gradients and used LR for critic as 1e-3 and LR for actor as 1e-4. Tests confirmed that this configuration was preferable to using the same learning rate for both networks. However, in this study, a reversed configuration showed the most optimal final returns. It is believed that a more stable critic, with a lower LR, could provide steady gradients to the actor for stable policy improvement, even if that implies delayed adaptation to changes in reward.

Another major turning point in DDPG came from correcting the reward scaling approach. The initial spread-based method normalized each reward based on the estimated best- and worst-case returns for each trading day. As a result, the same raw reward could be scaled differently across training steps. Figure 11 shows a concrete example of this issue: the reward of 7500 euro was scaled down to 0.79, while a smaller reward of 5500 euro was scaled up to 1.57 due to shifts in internal scaling ranges, creating conflicting learning signals for the agent.

```
Step = 990160, Unscaled reward = 5554.745934303075, Scaled reward = 1.5757
Step = 990168, Unscaled reward = 394.4106782431809, Scaled reward = 0.2828
Step = 990176, Unscaled reward = 4659.553581141945, Scaled reward = 1.1349
Step = 990184, Unscaled reward = 7536.777998521511, Scaled reward = 0.7902
```

Figure 11. Fragment of training logs illustrating the issue with spread-based reward scaling. The same actual reward is scaled to very different values depending on the internal scaling range of each trading day.

To address this, the dynamic scaling was replaced with a fixed reward scaling method, dividing all rewards by a predefined constant, explained in Section 3.2.4. At the same time, the neural network architecture was improved by introducing Layer Normalization into both the actor and

critic networks and refining the input processing, as described in Section 3.3.1. All these architectural and optimization adjustments led to the best DDPG results achieved in this study.

Figure 12 illustrates a clear improvement over the default baseline. Both configurations were analyzed with multiple seeds to illustrate the variance due to sampling. The shaded regions around each curve depict this variance. The tuned DDPG model (see Section 3.4.2) achieved much higher mean revenues of 33633 euro, compared to the default DDPG of 27006 euro on the last evaluation steps; it reached near-optimal performance around 700k training steps, after which the evaluation curve was steady across different seeds. Therefore, it is not necessary for the model to be run for the entire million steps.

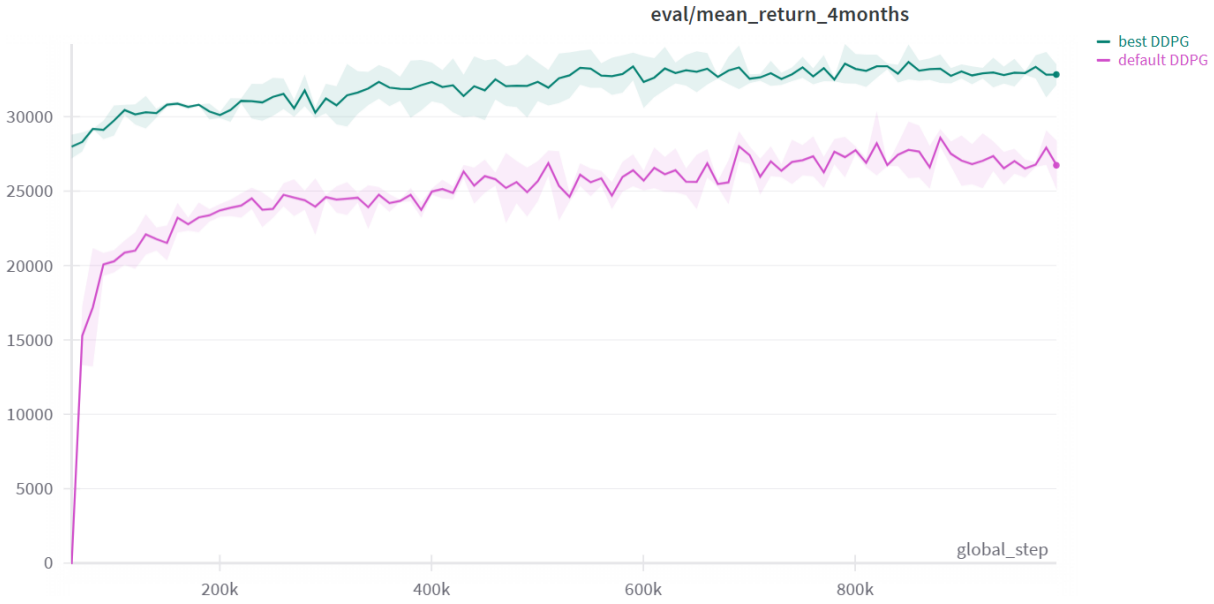

Figure 12. Evaluation performance of default and best-tuned DDPG models across multiple random seeds, showing mean 15-day profit averaged over four months against training steps.

Based on the results in Table 7, the tuned DDPG model caused an increase of almost 24.5% in mean revenues compared to what a default setup could obtain, consistently obtaining higher total profits across all recorded months. Moreover, the performance gap between the best DDPG model and the linear optimization benchmark was narrowed, specifically, the baseline DDPG model initially reached only about 68.6% of the optimizer's mean returns, while the tuned model version improved this ratio to ~85.5%.

Table 7. Comparative performance analysis of the linear optimization benchmark, best and default DDPG models

Model	January	April	July	October	Mean return for 4 months in euro
Optimizer	16775.57	20981.70	57277.61	62400.75	39358.9
Best DDPG	14629.66	17465.62	46183.61	56255.20	33633.5
Default DDPG	9809.04	11063.34	36996.67	50156.13	27006.3

4.6 24-hour setup

After achieving the most efficient DDPG configuration in a simplified environment, the setup was extended to a more complex scenario using a full 24-hour price schedule. Specifically, the observation space was expanded from the previous 6+6-hour format to 24 actual market prices for today and 24 for tomorrow, resulting in 49-dimensional vector, with one feature reserved for SOC. To accommodate this change and maintain consistent reward magnitudes, the reward-scaling constant was proportionally increased by a factor of four relative to the simplified scenario. The evaluation results for the 24-hour setup are presented in Table 8 below.

Table 8. Comparative performance analysis of linear optimization benchmark and the best DDPG model with a full 24-hour price input.

Model on 24 hours	January	April	July	October	Mean return for 4 months in euro
Optimizer	121771.85	66855.21	141198.49	121008.71	112708.56
Best DDPG	76470.85	42362.73	90989.37	82585.98	73102.23

Within this setting, the mean revenues of the model dropped to approximately 65% of the optimizer's mean returns, compared to 85% previously achieved in the simplified 6-hour setup. This performance degradation is likely due to the increased complexity introduced by extending both the input and output spaces. First, increasing the input vector complicates the

task of identifying reliable patterns in price data and extracting meaningful signals. Second, predicting 24 interdependent actions simultaneously increases the risk of cumulative errors, meaning early inaccuracies in trading decisions can negatively impact subsequent actions. Finally, it becomes more difficult to assign credit to specific decisions, as the model finds it hard to determine exactly which specific actions were responsible for the eventual profit or loss. Together, these factors limit the agent's capability to adapt well. Further tuning of the model parameters and possible architectural modifications are required to improve the performance of the model in this extended 24-hour scenario.

To understand the learned decision-making behavior of DDPG agent, battery trading actions were visualized for one concrete evaluation day. Figure 13 shows the actual day-ahead prices provided to both models, alongside the trading schedules generated by the RL policy and the optimizer benchmark.

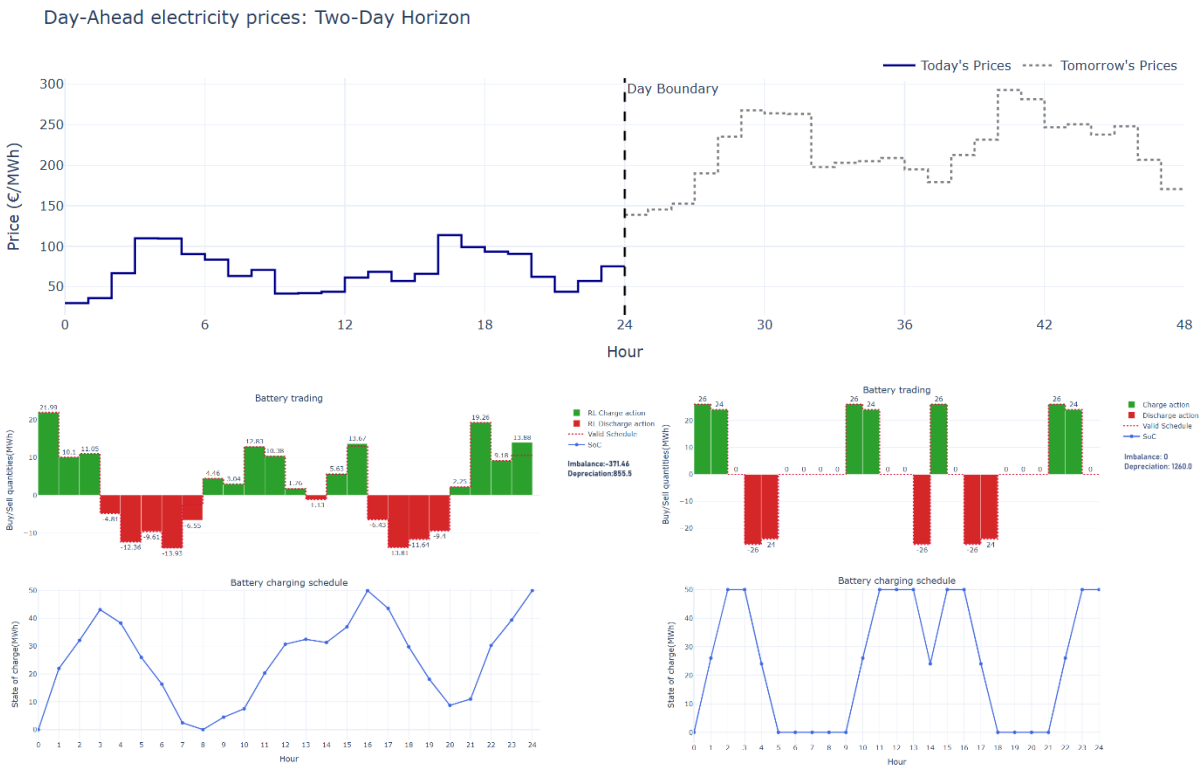

Figure 13. Comparison of battery trading schedules between RL (left) and Optimizer (right) for a single day. The top chart shows day-ahead electricity prices for today and tomorrow.

The RL policy demonstrates good general market awareness, notably charging during periods of relatively low electricity prices, such as midnight to 3 AM and again between 9-11 AM, and discharging close to local price peaks at hours 4-5 AM and 5-7 PM. Furthermore, the RL agent

strategically accumulates the battery to its maximum capacity of 50 MWh toward the day's end, effectively preparing for the significantly higher prices anticipated on the following day.

However, the main problem RL has faced is its inability to focus trades strictly around global price extrema. Instead, the agent spreads its energy trading actions throughout the day, missing the opportunity to maximize gains by fully charging at the lowest prices and fully discharging at peak prices. Particularly problematic are the minor, poorly timed discharges during moderate-priced intervals, such as between 5-9 AM, when inaction would have been economically preferable. Such trading behavior may result from limited feature information available to the agent, which now relies only on price arrays, making it difficult to clearly differentiate between moderately and extremely favorable hours. The issue could potentially be solved through additional feature engineering, e.g. including the rank of prices to explicitly indicate the cheapest and most expensive hours of the day, guiding the RL agent toward more concentrated and economically optimal decisions.

4.7 Episodic setup

To justify the design choice behind the continuous training setup, an episodic training formulation used in early-stage experiments was evaluated (see Section 3.4.1). Under this structure, several reward-shaping methods were explored: (i) delayed rewards [72], where no intermediate rewards were given, and cumulative daily revenues were provided only at the end of a three-day period as feedback. This approach aimed at encouraging long-term strategies rather than short-term profit maximization. (ii) immediate rewards and bonus structures to favor optimal hourly trading decisions. Initially, fixed bonuses rewarded favorable buy/sell decisions. These later evolved into proportional bonuses reflecting trading volume during profitable periods.

While these experiments resulted in improved episodic performance during training, the setup ended up being incompatible with real-world battery operations: the artificial reset of battery state and the imposed finite trading horizon forced the agent to fully discharge the battery on the last day to maximize short-term profits.

The Figure 14 below, compares the episodic and continuous setups using the metric of “3-day episodic return” during training (see Section 3.5.1). Each curve reflects performance over consecutive 3-day trading windows. Smoothing was applied to reduce the noise of fluctuating episodic returns and to highlight the learning trend. The continuous setup outperformed the episodic one, achieving higher returns.

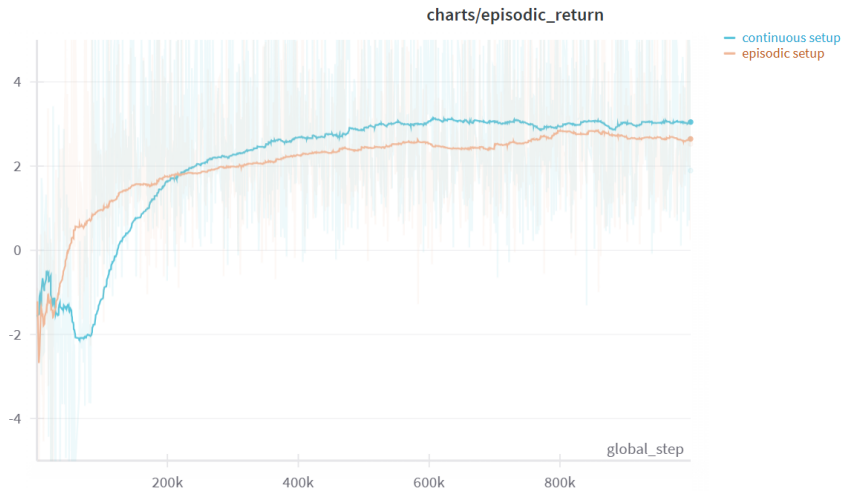

Figure 14. Comparison of episodic vs continuous setups based on the 3-day return metric.

The internal trading logs also showed improved battery conservation strategies, suggesting better long-term operational planning. The model evaluation further confirmed these results: the model trained under the continuous configuration achieved an average 3-day return of 2425 euro, while that trained by the episodic setup achieved only 1653 euro.

This ablation demonstrates the benefits of the continuous approach, which was therefore used in the following experiments.

4.8 Reinforcement learning limits & prospects

The thesis findings confirm that while RL can perform well in simplified settings, scaling to realistic complexities remains difficult. As noted in article [73], deep RL often underperforms compared to other methods and lacks robustness required for wide deployment. Research [74] also shows that RL results are hard to reproduce and vary a lot depending on how experiments are set up. Another study [75] adds that even small changes in the task can greatly affect results, highlighting how much trial-and-error is involved. In fact, RL is an empirical science with no guaranteed formulas for success, and even hyperparameter optimization alone often requires months of iterative experimentation [76]. This makes RL a powerful but time-consuming and resource-intensive approach.

On the other hand, two possible advantages of RL are worth considering. Firstly, RL may handle uncertainty in future electricity prices more effectively than traditional optimization methods. Optimization treats its price forecast as ground truth; if that forecast is off by, e.g., 20–30%, the resulting schedule can be suboptimal or even misleading. By training on the distribution of possible price trajectories, an RL agent can learn strategies that are more robust

to such uncertainty. When forecast variance is high, this ability to learn from variability may allow RL to outperform simpler, deterministic approaches. In contrast, if the forecast error is small, optimization remains preferable due to its transparency and theoretical guarantees.

Secondly, while optimization fundamentally depends on having access to future prices, RL could, at least in principle, learn policies that rely on historical information or additional context such as weather forecasts. Although the RL model in this thesis is still considered to rely on forecasted future prices, the framework is more flexible in terms of potential data sources and problem formulations.

Overall, the results show that RL models, particularly DDPG, when well-tuned, can offer a viable alternative to linear optimization for battery trading in volatile electricity markets, while potentially remaining able to incorporate nonlinear battery ageing and future multi-market signals, as mentioned in the beginning of Chapter 3. Due to time constraints, further work is needed to refine tuning for the day-ahead horizon and to extend the approach to the mFRR market.

5. Conclusion

This study explored reinforcement learning as an alternative approach to optimizing Battery Energy Storage System trading on Estonia's day-ahead electricity market, in comparison to the existing linear optimization method used by Eesti Energia. The performance of two RL algorithms, Deep Deterministic Policy Gradient and Proximal Policy Optimization, was evaluated across varying task complexities.

Under numerous experiment settings, DDPG performed better than PPO across the board. In a simplified trading setup of 6-hour horizon, PPO achieved 60.5% of the optimizer's performance, whereas the best results were obtained with a tuned DDPG model, achieving 85.5% of the linear optimization benchmark's returns, with average monthly profits of 33633 euro compared to 39358 euro for the optimizer. When the input and output spaces were expanded to a full 24-hour format, DDPG's relative performance declined to roughly 65%, underscoring increased challenges associated with higher-dimensional input and output spaces. The agent failed to concentrate trades around global price extrema, often performing smaller and poorly timed trading actions, thus limiting potential profits. Nevertheless, results prove that RL is a viable alternative to the optimizer, although further experimentation and tuning is needed. Future work should address the above limitations by further tuning model parameters and improving network architectures to better handle increased complexity, as well as performing additional feature engineering in the observation space. After that, moving from historical to forecasted price inputs is essential for realistic deployment, and from there, extending the model to multi-market scenarios, particularly balancing markets such as mFRR.

References

[1] M. Lempriere, “EU’s solar and wind growth pushes fossil-fuel power to lowest level in 40 years,” Carbon Brief. Accessed: May 14, 2025. [Online]. Available: https://www.carbonbrief.org/eus-solar-and-wind-growth-pushes-fossil-fuel-power-to-lowest-level-in-40-years/

[2] “Estonia plans to cover its annual consumption with renewable electricity by 2030 | Ministry of Economic Affairs and Communications.” Accessed: May 14, 2025. [Online]. Available: https://www.mkm.ee/uudised/eesti-plaanib-2030-aastal-katta-oma-aastase-tarbimise-taastuvelektriga-0

[3] “Electricity Data Explorer,” Ember. Accessed: May 14, 2025. [Online]. Available: https://ember-energy.org/data/electricity-data-explorer

[4] “Estonia: Solar, wind energy production exceeds 1 TWh in 2024 – Eesti Tuuleenergia Assotsiatsioon.” Accessed: May 14, 2025. [Online]. Available: https://tuuleenergia.ee/estonia-solar-wind-energy-production-exceeds-1-twh-in-2024/?lang=en

[5] Fuad, “Estonia’s Renewable Energy Leap: Milestones of 2024.” Accessed: May 14, 2025. [Online]. Available: https://www.pvknowhow.com/news/estonias-renewable-energy-leap-milestone-of-2024/

[6] “European Electricity Review 2025,” Ember. Accessed: May 14, 2025. [Online]. Available: https://ember-energy.org/latest-insights/european-electricity-review-2025

[7] H. Ritchie, “The price of batteries has declined by 97% in the last three decades,” *Our World Data*, Jun. 2021, Accessed: May 14, 2025. [Online]. Available: https://ourworldindata.org/battery-price-decline

[8] G. Heynes, “Baltic Storage Platform breaks ground on 400MWh BESS in Estonia,” Energy-Storage.News. Accessed: May 14, 2025. [Online]. Available: https://www.energy-storage.news/baltic-storage-platform-breaks-ground-on-400mwh-bess-in-estonia/

[9] N. Shahroudi, “Optimisation of Battery Energy Storage System in the Estonian Energy Markets”.

[10] “Finnish startup secures €3.8M to scale AI-driven virtual power plant for energy storage - ArcticStartup.” Accessed: May 14, 2025. [Online]. Available: https://arcticstartup.com/capalo-ai-raises-e3-8-million-seed/

[11] “Nord Pool reports encouraging growth in 2024.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/message-center-container/newsroom/exchange-message-list/2025/q1/nord-pool-reports-encouraging-growth-in-2024/

[12] “Single Day-Ahead Coupling (SDAC).” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/the-power-market/Day-ahead-market/single-day-ahead-coupling/

[13] “Markets divided into bidding areas.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/the-power-market/Bidding-areas/

[14] “Public Description Single Price Coupling Algorithm.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/globalassets/download-center/single-day-ahead-coupling/euphemia-public-description.pdf

[15] “The main arena for trading power.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/the-power-market/Day-ahead-market/

[16] “Electricity spot markets.” Accessed: May 14, 2025. [Online]. Available: https://pierrepinson.com/index.php/teaching/module-2-electricity-spot-markets-e-g-day-ahead/

[17] “Order types and start trading.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/trading/Day-ahead-trading/Order-types/

[18] “How members use the single hourly order.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/trading/Day-ahead-trading/Order-types/Hourly-bid/

[19] “Intraday trading.” Accessed: May 14, 2025. [Online]. Available: https://www.nordpoolgroup.com/en/trading/intraday-trading/

[20] G. Neidhöfer, “50-Hz Frequency [History],” *IEEE Power Energy Mag.*, vol. 9, no. 4, pp. 66–81, Jul. 2011, doi: 10.1109/MPE.2011.941165.

[21] “Balancing Energy Markets in Belgium: R1 (FCR), R2 (aFRR), R3 (mFRR) explained.” Accessed: May 14, 2025. [Online]. Available: https://www.next-kraftwerke.be/en/knowledge-hub/balancing-markets/

[22] “Balancing Market Best Practice.” Accessed: May 14, 2025. [Online]. Available: https://elering.ee/remit-maaruse-kohane-turujarelevalve

[23] “Regulation - 2017/1485 - EN - EUR-Lex.” Accessed: May 14, 2025. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2017/1485/oj/eng

[24] “Regulation - 2017/2195 - EN - EUR-Lex.” Accessed: May 14, 2025. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2017/2195/oj/eng

[25] “Eleringi reserviturgude Q&A.” Accessed: May 14, 2025. [Online]. Available: https://elering.ee/reguleerimisteenuste-energia-ja-voimsusturg

[26] K. Poplavskaya, J. Lago, and L. De Vries, “Effect of market design on strategic bidding behavior: Model-based analysis of European electricity balancing markets,” *Appl. Energy*, vol. 270, p. 115130, Jul. 2020, doi: 10.1016/j.apenergy.2020.115130.

[27] “The Platform for the International Coordination of Automated Frequency Restoration and Stable System Operation (PICASSO).” Accessed: May 14, 2025. [Online]. Available: https://www.entsoe.eu/network_codes/eb/picasso/

[28] “Manually Activated Reserves Initiative.” Accessed: May 14, 2025. [Online]. Available: https://www.entsoe.eu/network_codes/eb/mari/

[29] “mFRR standardtoode.” Accessed: May 15, 2025. [Online]. Available: https://elering.ee/sites/default/files/public/Teenused/mFRR%20standardtoode%2002.2025.pdf

[30] “aFRR pilot end report.” Accessed: May 15, 2025. [Online]. Available: https://elering.ee/sites/default/files/public/Teenused/aFRR%20standardtoode%2002.2025.pdf

[31] “FCR_standardtoode.” Accessed: May 15, 2025. [Online]. Available: https://elering.ee/sites/default/files/public/Teenused/FCR_standardtoode.pdf 2

[32] K. Mee, “Utilitas was the first in Estonia to be prequalified as a manual frequency reserve provider,” Utilitas. Accessed: May 14, 2025. [Online]. Available: https://utilitas.ee/en/elering-was-the-first-to-prequalify-utilitas-as-a-manual-frequency-reserve-provider/

[33] “Elektrisüsteemi_bilansi_tagamise_ehk_tasakaalustamise_eeskirjad.”

[34] “Imbalance prices - BTD.” Accessed: May 15, 2025. [Online]. Available: https://baltic.transparency-dashboard.eu/node/42

[35] “ELEKTRITURU KÄSIRAAMAT.” Accessed: May 15, 2025. [Online]. Available: https://elering.ee/elektrituru-kasiraamat

[36] S. P. Boyd and L. Vandenberghe, *Convex optimization*, Version 29. Cambridge New York Melbourne New Delhi Singapore: Cambridge University Press, 2023.

[37] R. S. Sutton and A. G. Barto, “Reinforcement Learning: An Introduction”.

[38] “What is Deep Reinforcement Learning (RL),” Matrix Alchemy - RenoCrypt. Accessed: May 14, 2025. [Online]. Available: https://www.renocrypt.com/posts/reinforcement-learning/

[39] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, “Q-Learning Algorithms: A Comprehensive Classification and Applications,” *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[40] G. Liu, G. Chen, and V. Huang, “Policy ensemble gradient for continuous control problems in deep reinforcement learning,” *Neurocomputing*, vol. 548, p. 126381, Sep. 2023, doi: 10.1016/j.neucom.2023.126381.

[41] “Key Concepts in RL — Spinning Up documentation.” Accessed: May 14, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[42] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. in Wiley Series in Probability and Statistics, no. v. 414. Hoboken: John Wiley & Sons, Inc, 2009.

[43] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems,” 2020, *arXiv*. doi: 10.48550/ARXIV.2005.01643.

[44] H. Zheng, X. Luo, P. Wei, X. Song, D. Li, and J. Jiang, “Adaptive Policy Learning for Offline-to-Online Reinforcement Learning,” Mar. 14, 2023, *arXiv*: arXiv:2303.07693. doi: 10.48550/arXiv.2303.07693.

[45] Y. Chebotar *et al.*, “Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience,” 2018, *arXiv*. doi: 10.48550/ARXIV.1810.05687.

[46] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” Dec. 19, 2013, *arXiv*: arXiv:1312.5602. doi: 10.48550/arXiv.1312.5602.

[47] Y. Yue, B. Kang, X. Ma, Z. Xu, G. Huang, and S. Yan, “Boosting Offline Reinforcement Learning via Data Rebalancing,” Oct. 17, 2022, *arXiv*: arXiv:2210.09241. doi: 10.48550/arXiv.2210.09241.

[48] M. Komorowski, L. A. Celi, O. Badawi, A. C. Gordon, and A. A. Faisal, “The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care,” *Nat. Med.*, vol. 24, no. 11, pp. 1716–1720, Nov. 2018, doi: 10.1038/s41591-018-0213-5.

[49] “Kinds of RL Algorithms — Spinning Up documentation.” Accessed: May 14, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#model-free-vs-model-based-rl

[50] S. Tu and B. Recht, “The Gap Between Model-Based and Model-Free Methods on the Linear Quadratic Regulator: An Asymptotic Viewpoint”.

[51] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[52] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning,” Dec. 08, 2015, *arXiv*: arXiv:1509.06461. doi: 10.48550/arXiv.1509.06461.

[53] “Intro to Policy Optimization — Spinning Up documentation.” Accessed: May 15, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#deriving-the-simplest-policy-gradient

[54] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Mach. Learn.*, vol. 8, no. 3–4, pp. 229–256, May 1992, doi: 10.1007/BF00992696.

[55] L. Weng, “Policy Gradient Algorithms.” Accessed: May 14, 2025. [Online]. Available: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

[56] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012, doi: 10.1109/TSMCC.2012.2218595.

[57] L. Graesser and W. L. Keng, *Foundations of deep reinforcement learning: theory and practice in Python*. in Addison Wesley data & analytics series. Boston Amsterdam London: Addison-Wesley, 2020.

[58] H. T. H. Giang, T. N. K. Hoan, P. D. Thanh, and I. Koo, "Hybrid NOMA/OMA-Based Dynamic Power Allocation Scheme Using Deep Reinforcement Learning in 5G Networks," *Appl. Sci.*, vol. 10, no. 12, p. 4236, Jun. 2020, doi: 10.3390/app10124236.

[59] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv*. doi: 10.48550/ARXIV.1509.02971.

[60] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms".

[61] "Deep Deterministic Policy Gradient — Spinning Up documentation." Accessed: May 14, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[62] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017, *arXiv*. doi: 10.48550/ARXIV.1707.06347.

[63] M. Andrychowicz *et al.*, "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study," Jun. 10, 2020, *arXiv*: arXiv:2006.05990. doi: 10.48550/arXiv.2006.05990.

[64] J. Achiam, "Simplified PPO-Clip Objective". Accessed: May 14, 2025. [Online]. Available: https://drive.google.com/file/d/1PDzn9RPvaXjJFZkGeapMHbHGiWWW20Ey/view

[65] "Proximal Policy Optimization — Spinning Up documentation." Accessed: May 14, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[66] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," Oct. 20, 2018, *arXiv*: arXiv:1506.02438. doi: 10.48550/arXiv.1506.02438.

[67] "Reinforcement Learning Tips and Tricks." Accessed: May 14, 2025. [Online]. Available: https://stable-baselines.readthedocs.io/en/master/guide/rl_tips.html

[68] "Content Environment Creation." Accessed: May 14, 2025. [Online]. Available: https://www.gymlibrary.dev/content/environment_creation/

[69] "Eesti Energia has begun the configuration of the energy storage facility at the Auvere industrial complex," Enefit AS. Accessed: May 14, 2025. [Online]. Available: https://www.enefit.ee/en/-/uudised/eesti-energia-alustas-eesti-suurima-energiasalvesti-seadistustoodega

[70] "CleanRL - Overview." Accessed: May 14, 2025. [Online]. Available: https://docs.cleanrl.dev/

[71] “The 37 Implementation Details of Proximal Policy Optimization.” Accessed: May 14, 2025. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[72] B. Han, Z. Ren, Z. Wu, Y. Zhou, and J. Peng, “Off-Policy Reinforcement Learning with Delayed Rewards,” Jun. 22, 2021, *arXiv*: arXiv:2106.11854. doi: 10.48550/arXiv.2106.11854.

[73] “Deep Reinforcement Learning Doesn’t Work Yet.” Accessed: May 14, 2025. [Online]. Available: http://www.alexirpan.com/2018/02/14/rl-hard.html

[74] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep Reinforcement Learning that Matters,” Jan. 30, 2019, *arXiv*: arXiv:1709.06560. doi: 10.48550/arXiv.1709.06560.

[75] A. R. Mahmood, D. Korenkevych, B. J. Komer, and J. Bergstra, “Setting up a Reinforcement Learning Task with a Real-World Robot,” 2018, *arXiv*. doi: 10.48550/ARXIV.1803.07067.

[76] J. Baptista, “PPO Hyperparameter Optimization,” Joel’s PhD Blog. Accessed: May 14, 2025. [Online]. Available: https://joel-baptista.github.io/phd-weekly-report/posts/hyper-op/

License

Non-exclusive licence to reproduce thesis and make thesis public

Yaroslava Mykhailenko,

(*author's name*)

herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Optimization of Battery Energy Storage System in the Estonian Energy Markets using Reinforcement Learning,

(*title of thesis*)

supervised by

Victor Henrique Cabral Pinheiro, Tambet Matiisen and Jean-Baptiste Scellier

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Yaroslava Mykhailenko

15/05/2025