UiA University of Agder

Master

Active network management with decision transformer

RUBEN SEVLAND, VEGARD SVENSLI ÅSVESTAD

SUPERVISOR
Ole-Christoffer Granmo, Per-Arne Andersen

University of Agder, 2024
Faculty of Engineering and Science
Department of Information and Communication Technology

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Abstract

This thesis analyzes the implementation of a DT model for ANM in power grids, focusing on active network management with intermittent renewable energy sources. Considering the increasing implementation of renewable sources and DES systems, efficient and robust algorithms are important for optimal grid management.
We start by evaluating two state-of-the-art RL models, PPO and SAC, as a baseline for expected performance. The SAC model showed superior performance and was used as the teacher model for dataset generation. This dataset was used in the training of the DT model.
Our methodology additionally includes PSO to fine-tune the output of our DT. This approach utilizes PSO's ability to understand non-differentiable problem spaces, complementing the DT's predictive accuracy. We evaluate the performance of the DT and the enhanced DT + PSO model in various grid scenarios, especially focusing on the model's behavior during the critical transitional periods between different stages of power demand and generation.
Key findings indicate that while the DT aligns closely with the baseline models in terms of performance, it has problems adapting to the sudden shifts between stages A and B in the grid simulation. However, the integration of PSO fine-tuning shows a slight performance improvement, demonstrating the potential of this hybrid approach in improving decision-making in grid management.
This thesis contributes to the field by not only introducing a new approach for ANM but also highlighting the importance of model adaptability in various grid scenarios. It lays the groundwork for future research in implementing advanced machine learning techniques for efficient power grid management, especially now as the world transitions towards more sustainable energy systems.

Abbreviations

ANM	-	Active Network Management
CPU	-	Central Processing Unit
D4RL	-	Datasets for Reinforcement Learning
DES	-	Distributed Energy Storage
DP	-	Dynamic Programming
DQN	-	Deep Q Network
DSO	-	Distribution System Operator
DT	-	Decision Transformer
EV	-	Electric Vehicle
GPT	-	Generative Pre-trained Transformer
GPU	-	Graphical Processing Unit
LLM	-	Large Language Model
MDP	-	Markov Decision Process
ME-TRPO	-	Model-Ensemble Trust-Region Policy Optimization
MSE	-	Mean squared error
PPO	-	Proximal Policy Optimization
PSO	-	Particle Swarm Optimization
PV	-	Photovoltaic
RL	-	Reinforcement Learning
RNN	-	Recurrent Neural Network
SAC	-	Soft Actor-Critic
SARSA	-	State–Action–Reward–State–Action
SB3	-	Stable Baselines 3
SGD	-	Stochastic Gradient Decent
SoC	-	State of Charge
TD	-	Temporal-Difference
TRPO	-	Trust Region Policy Optimization
TSO	-	Transmission System Operator

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

According to the US Department of Energy's vision report 'Grid 2030', disturbances in power distribution and quality issues can result in a cost of $119-$188 billion to certain industries annually [1]. With the on-going environmental pressure, considering the increasing accumulation of CO_2 emissions in the atmosphere, the sloping depletion of fossil fuels, and the fact that the world's population continues to grow and urbanize; a global paradigm shift in the energy sector is needed [2]. A self-evident step is to continue increasing the utilization of renewable energy sources, and an equally important measure is to ensure optimal management of the electricity flow in the power grid by scheduling the energy generation based on demand (otherwise known as *active network managment*) [3]. However, trying to improve energy systems by implementing such changes is a difficult task when considering both generation and demand [4, 5].

The set of operational optimization issues that occurs when scheduling power generation is getting more complex, which in turn makes it more challenging to efficiently control energy systems. There have already been radical operational and structural improvements in power grids in the last two decades [6]. As the electricity market has been more liberalized, a competitive aspect was introduced in how they were managed, resulting in network enhancements and more affordable energy generation. In addition, power networks are going through a shift from the traditional model of centralized power generation based on fossil fuel, to a smarter, more decentralized network of renewable sources, with the arrival of distributed generators, such as photovoltaic panels (PVs) and wind turbines [7]. For example, virtual microgrids have developed local ecosystems in which consumers are also producers [8, 9]. It is further anticipated an increase of distributed generators in the near future [10], as well as an expansion of considerable loads as a result of rapid growth in the electric vehicle market [11, 12]. Furthermore, distributed energy storage (DES) is also becoming more available with the implementation of technologies like batteries [5, 13], vehicle fleets [14] and power-to-gas [15]. These changes have introduced an increase in complex decision-making challenges, such as voltage coordination, transmission line congestion, overvoltages, etc., for system operators. Therefore, the need for robust and efficient algorithms to solve such challenges, and that can take account of the intermittent nature of renewable energy sources, is becoming increasingly crucial in order to ensure a smooth transition to sustainable energy systems; some of which can benefit from recent improvements in the area of reinforcement learning (RL) research.

RL is a dynamic and exciting field of machine learning that seeks to emulate the way humans learn, and it presents a viable solution to a broad spectrum of intricate decision-making predicaments. For years, RL techniques have played an important role in the management of electricity networks [16].

In this paper, we will introduce an RL method utilizing the transformer architecture for ANM problems. The transformer architecture is most commonly associated with large language models(LLM). The architecture is utilized as an RL method by treating the problem as a set of sequential decision-making tasks in the management of power grids [3]. In literature related to power systems, ANM pertains to the development of control systems that regulate the generators, the loads, and the DES devices that are connected to the grid. This is important to avoid problems at the distribution level, and for increasing profitability by avoiding energy loss [17]. The operations that are done by network operators, may result in decreasing power generation compared to the maximum pro-

duction, in regards to the available resources. This is commonly known as *curtailment*, which can be a requirement in certain situations. However, curtailment, together with losses in transmission and storage, is the main cause of energy loss that ANM wish to minimize. Furthermore, the ANM model shall ensure a robust and reliable operation of the power grid.

1.1 Contributions and Scope of the Thesis

The main contributions of this thesis are:

- Creation of an offline dataset for RL training on active network management.
- The implementation of a decision transformer for active network management with the inclusion of intermittent renewable energy sources.
- A novel approach to use particle swarm optimization (PSO) to fine-tune the decision transformer output.

1.2 Report Outline

The rest of this thesis is organized as follows: Chapter 2 provides a thorough review of current state-of-the-art RL approaches for power grid management. Chapter 3 covers relevant preliminaries that help in understanding the contributions of this work, and the scaled-down power grid application that the algorithms are tested on. Results and limitations of using a decision transformer for Active Network Management (ANM) is discussed in Chapter 4. Finally, chapter 5 concludes the latter, and includes relevant future work.

Chapter 2

Background

2.1 Previous Work

RL has demonstrated exceptional capability in solving a variety of realistic control and decision-making problems in the presence of ambiguity. Despite its proven potential, its utilization in power grid management is not yet well explored. Nevertheless, RL has been applied to consider the generator's load frequency control issue [18], the unit commitment problem [19], power system transient stability enhancement [20], and address individual preferences for private customers in the electricity market [21]. In addition, RL has been implemented to deal with the issues regarding auction-based energy pricing [22] and economic dispatch [23]. A Q-learning approach is introduced in [24] to manage reactive power control and limited load flow problems in power grids. An optimization scheme based on RL designed for tackling the actions of microgrid consumers is designed in [25]; which attempts to handle the unpredictability of the environment and renewable energy. In [26], RL is compared to a predictive control model for solving the problem of power grid damping. Furthermore, some RL approaches are mentioned in [27] where advancements in intelligent microgrid controlling are reviewed, whereas [28] reviews various RL approaches for demand response in the power grid. However, the revised papers mostly utilize basic RL algorithms, such as Q-learning and SARSA methods [29], which are dependant on the memory-extensive tabular representation of the state-action value function Q. These tabular methods introduce some limitations in their applicability to large-scale problems characterized by high-dimensional state-action spaces, which is often the case in real-world cases. Memory usage in these situations becomes difficult, and the compute time makes it an unattractive approach. However, regression tools can be implemented to replace the tabular representation of Q to increase the applicability of RL approaches to environments with state spaces of any size.
Moreover, a deep Q-learning approach is utilized in [30] to attempt optimal power management of hybrid electric buses. In [21], RL in combination with deep neural networks is designed for smart grids to enable optimal incentive rates regarding the demand-response problem in real-time. Deep RL is tested for short-term voltage control by dynamic load shedding in [31], whereas two RL approaches using deep Q-learning and Gibbs deep policy gradient are implemented onto physical models in [32]. In [33], residential demand response using RL is tackled, and [34] proposes a non-tabular solution for optimal operation and maintenance of power grids, by merging RL with an artificial neural network. In [35] and [36], the same gym environment uses simulations in reliability and stability studies, respectively.
Furthermore, [37] studies optimal energy management strategies using reinforcement learning. And we conclusively refer to [38] and [39] which both provide comprehensive overviews of reinforcement learning algorithms applied to power systems.

2.2 Active Network Management

The power network management at utility-scale ($\geq$ 0.5MW) is controlled centrally, with subdivided responsibility for handling different voltage levels. The high-voltage network, which is used for long-distance transmission, is managed by the Transmission System Operator (TSO). On the other hand, the Distribution System Operator (DSO) takes care of the medium and low-voltage transmission networks, which are used for local power distribution.
Both TSO and DSO have two main jobs: to make sure the power network functions safely and reliably, and to maintain the network frequency steady at 50Hz [40]. To do this, they need to balance the amount of electricity being generated with the amount being used at all times. If there's too much electricity being made, the frequency goes up. If the demand is too high, the frequency goes down. Even a small change in frequency of 0.5 Hz, can cause big problems, especially for industrial equipment like induction motors that rely on this steady frequency.
To maintain this balance, the operators use special agreements, called reserve contracts, with large-scale electricity producers or consumers (who deal with at least 0.5 megawatts). These contracts let the operators either increase or curtail electricity production, or temporarily reduce the amount of electricity used by big consumers.
Currently, the operations to take are decided by analyzing historical load data along with production plans issued by utility-scale producers/consumers to get insights into future power deficits and/or surpluses, and plan accordingly.

2.3 Reinforcement Learning

ML algorithms are categorized into three groups based on their input data and how data is processed for it to learn: supervised learning, unsupervised learning, and RL. Supervised learning uses labeled data that consists of input samples paired with their corresponding desired output i.e. "labels", and aims to establish a method that maps the training input to labeled outputs, using a predetermined evaluation index [41]. Deep neural network is an example of a supervised learning method that has attracted more attention in recent times [42]. Unsupervised learning tries to discover relationships, structures, or patterns within unlabeled data; typically utilizing techniques like dimensionality reduction [43], clustering [44], and association rule learning methods [45].
Unlike supervised and unsupervised learning, RL is regarded as *active learning*. It is commonly described as a game based on the interaction between an agent and environment, in which the agent learns — through trial and error — to make decisions in its corresponding environment that maximize cumulative rewards over time [16]. For each step, the agent performs an action based on its current observation of the environment; receives feedback from the environment in the shape of a reward or penalty, and updates its decision-making policy according to this feedback. The final objective is to maximize the expected long-term reward by learning the optimal policy.

2.3.1 Markov Decision Process (MDP)

Formally, the interaction between an agent and environment is described by an MDP [46], $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R})$, where $\mathcal{S}$ and $\mathcal{A}$ denotes the state and the action space respectively. The state transition probability is $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, i.e., $\Pr(s'|s, a)$ is the probability that s' will be the next state if action a is executed in state s. The discount factor is represented by γ. $\mathcal{P} : \mathcal{S} \to [0, 1]$ further describes the initial distribution of states $\mathcal{S}$, and the feedback/reward is expressed by $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The agent's decision process is entirely captured by a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps every state to an action. The set of all policies that remains fixed or unchanged throughout the learning process (commonly known as *stationary policies*) are denoted by π. A policy that intuitively maximizes the cumulative (discounted) score long-term is called optimal.
The so-called game of RL can be generalized to the steps shown in algorithm 1:
The game of RL is also visualized in 2.1. An agent, operating within an environment M, executes a policy $\pi : S \to A$. The agent's interaction with the environment results in the accumulation of

Algorithm 1 RL pseudo code

for each episode **do**

$s_t \in \mathcal{S}$	▷ Agent observes environment's state
$a_t = \pi(s_t) \in \mathcal{A}$	▷ Agent chooses an action
$r_{t+1} = R(s_t, a_t)$	▷ Agent receives a feedback
$s_{t+1} \sim \mathcal{P}(\cdot\|a_t, s_t)$	▷ Agent transitions to new state

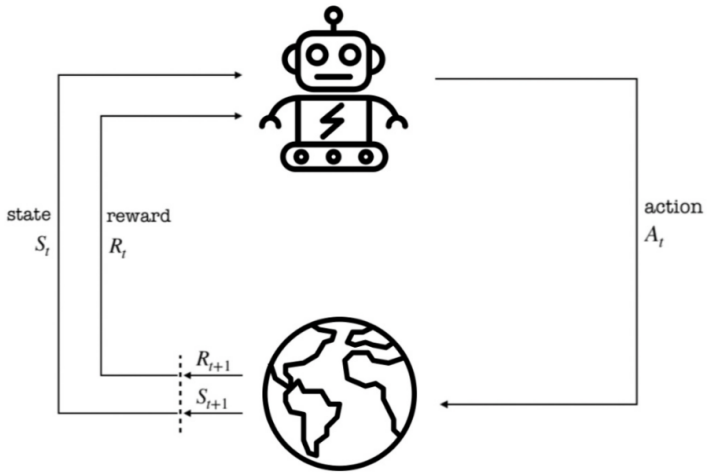


Figure 2.1: Interaction between the agent and the environment, adapted from Ref. [16]

a stochastic return $\mathcal{G} = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$, where $r_t = R(s_t, a_t)$ signifies the reward obtained by taking action a_t in state s_t. An essential goal in RL is to find the optimal policy, denoted as π^*, which aims to maximize the discounted sum of rewards.

$$\pi^* = \underset{\pi:\mathcal{S}\to\mathcal{A}}{arg\ max}\ E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \right] \tag{2.1}$$

Value Functions: Value functions serve as estimates for the expected rewards within a specified time scope, considering the current state $s \in S$. These functions play a crucial role in the formulation of the optimal policy. The state-value function $\mathcal{V}^\pi$ for a policy π is expressed as

$$V^\pi(s) = E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s \right] \tag{2.2}$$

The term $V^\pi(s)$ describes the expected return from a given state $s \in S$. The optimal value function, V^* coordinates with the value function associated with an optimal policy π^*, expressed as $V^* = \underset{\pi:S\to A}{max} V^\pi(s)$.
When V^* is known, the optimal policy can be inferred by using a greedy action a based on V^*. In the context of RL setups, where essential model information like transition probabilities and reward distributions is often unavailable; deducing the optimal policy directly from the state-value function introduces a considerable challenge. Instead, we use something called the state-action value function, denoted as $Q^\pi(s, a)$. This function helps us assess how good our chosen policy (π) is. It does so by adding up all rewards an agent gets when it takes an action (a_t) in the current state (s_t) according to the policy. Mathematically, the state-action value function is expressed as follows:

$$Q^\pi(s, a) := \mathcal{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, a_0 = a \right] \tag{2.3}$$

And the corresponding optimal Q-function is defined as

$$Q^*(s, a) := \underset{\pi:S\to A}{max} Q^\pi(s, a) \tag{2.4}$$

Note that once Q^* is available, we can retrieve the optimal policy using $\pi^*(s) = \arg\max_{a \in A} Q^{\pi^*}(s, a)$. This retrieval process stands in contrast to the state-value function scenario, because it doesn't require the inclusion of model information.

Dynamic Programming(DP) becomes cruical when dealing with a known MDP, $\mathcal{M}$. In this context, determining the optimal policy π^* is called a *planning problem*, and DP algorithms offer an efficient solution to solve this. The computation or estimation of value functions for a policy π is simplified by solving the corresponding Bellman equations 2.5 and 2.6 [[47], [48]]:

$$V^{\pi}(s) = E_{s' \sim P(\cdot|s,\pi(s))}[R(s, \pi(s)) + \gamma V^{\pi}(s^{'})] \tag{2.5}$$

$$Q^{\pi}(s, a) = E_{s' \sim P(\cdot|s,a)}[R(s, a) + \gamma Q^{\pi}(s^{'}, \pi(s^{'}))] \tag{2.6}$$

Derived from the Markov property [46] of the MDP and the definition of the value function, these steps are commonly known as policy evaluation. Once a policy π is evaluated, an improved policy can be derived with the greedy policy

$$\pi^{'}(s) = \max_{a \in A} E_{s' \sim P(\cdot|s,a)}[R(s, a) + \gamma Q^{\pi}(s')] \tag{2.7}$$

or $\pi'(s) = \arg\max_{a \in A} Q^{\pi}(s, a)$, a process referred to as policy improvement. Various DP algorithms, including policy and value iterations, are dependent on different combinations of alternating iterations of policy evaluation and improvement [48].

2.3.2 RL methods

RL can be subdivided into three primary classifications: Value-based, Policy-based, and Model-based methods. The difference between these is based on what they optimize. Value-based focus value functions, Policy-based on policies, and Model-based on internal environment models; which shows the variety in approaches to decision-making and learning strategies in dynamic environments.

Model-Based

Model-based methods in RL are widely different from model-free approaches. Instead of only relying on data collected from the environment, model-based methods involve creating a model of the environment's dynamics.
Model-based RL functions under the principle that understanding and modeling the environment's behavior can help make informed decisions. Instead of learning directly from interaction data, these methods try to create a representation of how the environment responds to different actions [49].
The main element of model-based methods is the environment model [50]. This model, commonly represented as a transition function or a predictive model, simulates the dynamics of the environment. It estimates the state's behavior when a specific action is taken.
Model-based methods utilize planning algorithms to optimize decision-making. These algorithms use already learned models to estimate future trajectories, allowing agents to make informed choices to maximize expected rewards.
In model-based RL, exploration strategies are central for collecting data to refine the environment model. These strategies help the agent to further explore actions that lead to informative experiences.
Model-based methods can be more sample-efficient than some model-free counterparts because they make use of the learned model to estimate potential outcomes, which reduces the need for extensive interaction with the real environment [51].
Several model-based algorithms exist, such as the Dyna algorithm [52], Model-Ensemble Trust-Region Policy Optimization (ME-TRPO) [53], and Stochastic Lower Bound Optimization (SLBO) [54]. These methods, with the use of the environment model, improve decision-making and can be applied to various domains, including robotics and autonomous systems.

2.3.3 Value based

Value-based methods in reinforcement learning center around estimating value functions, which compute the expected cumulative rewards linked to specific states and/or actions. These methods focus on calculating the desirability of states or state-action pairs, introducing these value assessments to guide decisions toward actions that promise higher-value outcomes.
Algorithms like SARSA [55] and Q-learning [56] seek to learn state or action-value functions and then make action selections based on these values.
Temporal-Difference (TD) Learning, a common policy evaluation algorithm, iteratively estimates the state value function V^π for a given policy π. The TD learning update rule originates from the squared-Bellman error and is expressed by ([16]):

$$V_{k+1}(s_k) = V_k(s_k) + \alpha_k \left(R(s_k, \pi(s_k)) + \gamma V_k(s_{k+1}) - V_k(s_k)\right) \tag{2.8}$$

Here, $s_k \sim d^\pi$, $s_{k+1} \sim P(\cdot|s_k, \pi(s_k))$, and α_k denotes the learning rate. The update occurs directly after the transition $(s_k, a_k = \pi(s_k), r_{k+1}, s_{k+1})$. The updated term, $\mathcal{R}(s_k, \pi(s_k)) + \gamma V_k(s_{k+1}) = V_k(s_k)$, is the Temporal-Difference (TD) error, measuring the difference between the current estimated value $V_k(s_k)$ and the improved estimate $\mathcal{R}(s_k, \pi(s_k)) + \gamma V_k(s_{k+1})$. The TD update converges to V^π almost surely if the step-size satisfies the Robbins-Monro rule [48].
SARSA, which is an on-policy method, updates the Q-value by following sequences of experiences, employing the update rule:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left(R(s_k, a_k) + \gamma Q_k(s_{k+1}, \pi(s_{k+1})) - Q_k(s_k, a_k)\right) \tag{2.9}$$

SARSA runs TD-learning to evaluate the state-action value function Q^π relating to the current policy π, computes an improved policy using Q^π, and alternates both steps to find Q^*.
Q-learning, an off-policy method, updates the Q-values using the equation:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left(R(s_k, a_k) + \gamma \max Q_k(s_{k+1}, a) - Q_k(s_k, a_k)\right) \tag{2.10}$$

Q-learning assesses the policy based on experiences from any behavior policy, which allows the use of a database of past experiences; this is referred to as the experience replay buffer. By contrast, SARSA produces a new experience each time a policy is updated.
Deep Q Network (DQN) extends RL to manage problems with a larger state and/or action space. DQN implements neural networks to estimate the Q-function. The DQN algorithm [57] trains a neural networks parameterized by θ using the loss function:

$$L(\theta) := \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r(s,a) + \gamma \max Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2.11}$$

D is an experience replay buffer that stores past experiences to reduce associations between observations. The online and target variables, θ and θ^-, respectively, are updated periodically. Augmentations like Double DQN [58] and Dueling DQN [59] improve DQN's design by covering overestimation issues and decoupling the value and advantage functions.

2.3.4 Policy-Based

Policy-based methods in RL take a unique approach by directly learning the policy, described as $\pi(a|s;\theta)$, with respect to the parameterized function θ. In contrast to value-based methods, policy-based approaches prove highly capable in handling continuous action spaces and can adjust to stochastic policies. The objective function

$$J(\theta) := \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{2.12}$$

include the stationary distribution $d^{\pi_\theta}(s)$ of the Markov chain for π_θ. The policy gradient theorem, which is the core of many policy gradient algorithms, states that the gradient of the objective is given by

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla \ln \pi_\theta(a|s) Q^{\pi_\theta}(s,a)] = \mathbb{E}_{\pi_\theta}[G_t \nabla \ln \pi_\theta(a_t|s_t)] \tag{2.13}$$

where G_t represents the discounted cumulative return starting from time step t.
Policy gradient methods seek a local maximum in $J(\theta)$ by increasing the gradient of the policy with respect to the parameters θ. This approach allows for direct policy optimization without the need for value function estimation. The importance of policy-based methods is particularly noticeable in scenarios that involve continuous action spaces and stochastic policies. The parameterized policies that are often used in policy-based methods are adjusted during training to enhance performance. The balance between exploration and exploitation is achieved by adapting the policy's stochasticity, enabling efficient exploration while utilizing the knowledge that is gained through learning.
This method is advantageous in tasks where uncertainty is apparent, making it well-suited for learning stochastic policies, in which actions are linked to set probabilities. Well-known policy-based algorithms, including REINFORCE, Trust Region Policy Optimization (TRPO) [60], and Proximal Policy Optimization (PPO), have seen success across various domains.
Off-policy methods, on the other hand, learn a policy independently from the policy being executed, allowing the agent to learn from actions that have not been executed. This flexibility can lead to more efficient learning as the agent can evaluate and improve its strategy using data from different policies, improving its ability to optimize decision-making for various scenarios. This method is useful in environments where collecting new data is costly or impractical.

Proximal Policy Optimization (PPO) PPO [61] is a popular RL algorithm that is designed to account for the challenge of training a policy that performs well, and maintaining stability during the learning process. PPO can do this by focusing on the relative change between the old and the new policy, using a technique called clipped surrogate objective functions.
At the core of PPO is the idea of probability ratio, denoted as $r_k(\theta)$, which is the likelihood of taking a particular action a in a given state s under the new policy (π_θ) compared to the old policy (π_{θ_k}). This ratio, defined as:

$$r_k(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \tag{2.14}$$

explains how the new policy prioritizes a certain action in comparison to the old policy. If $r_k(\theta)$ is higher than 1, it indicates a preference for the action under the new policy, while a value smaller than 1 means a preference for the old policy.
To avoid that the new policy makes large, sudden changes, PPO implements a constraint on the probability ratio, with a hyperparameter ϵ (epsilon) determining the acceptable range. The constraint makes sure that $r_k(\theta)$ remains within a specific interval around 1, defined as $[1-\epsilon, 1+\epsilon]$.
This constraint is an important feature of PPO and leads to more stable training. It prevents the policy from diverging significantly from the old policy, mitigating the risk of training becoming unnecessarily volatile.
PPO's goal is to maximize the expected value of a concrete objective function, denoted as L, with respect to the new policy parameters (θ_{k+1}). The objective function L is defined as follows:

$$L(s,a,\theta_k,\theta) = \min\{r_k(\theta)A^{\pi_{\theta_k}}(s,a), \text{clip}(r_k(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_k}}(s,a)\} \tag{2.15}$$

In this equation, $A^{\pi_{\theta_k}}(s,a)$ represents the advantage function, which estimates how much better or worse an action is compared to the average action executed under the old policy.
The "clip" function in the objective function makes sure that the probability ratio $r_k(\theta)$ is restricted within the interval $[1-\epsilon, 1+\epsilon]$. This clipping mechanism helps maintain the stability of policy updates by preventing unreasonable policy changes that could lead to unpredictable training behavior.
To update the policy, PPO seeks to maximize the expected value of the objective function L in regards to the new policy parameters (θ_{k+1}). This is typically accomplished through many iterations of stochastic gradient descent (SGD), often using minibatches of data to improve computational efficiency.

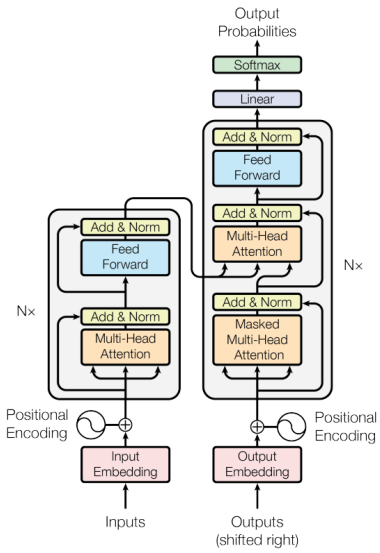


Figure 2.2: Model architecture of the transformer [63]

Soft Actor-Critic (SAC) SAC [62] is an RL algorithm that has a unique approach to learning optimal policies. Unlike many other algorithms, SAC implements the policy's entropy into the reward function. SAC is an off-policy actor-critic model functioning within the framework of maximum entropy RL. Its special feature lies in the emphasis on both exploration and efficient learning in complex environments.

The main goal of SAC is to find the optimal policy (π) that maximizes the expected return. This return is influenced by two critical components: the standard reward (R) and the entropy ($\mathcal{H}$) of the policy. The inclusion of entropy in the objective function plays an important role in encouraging exploration. The mathematical representation of this objective is expressed as:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma' \mathcal{R}(S_{t+1}) + \alpha \mathcal{H}(\pi(\cdot|a_t)) \right] \tag{2.16}$$

where $\mathcal{R}(s_{t+1})$ denotes the standard reward at the next time step (t+1), signifying how beneficial the agent's action is in the current state (s_t). $\mathcal{H}(\pi(\cdot|a_t))$ is the entropy of the policy (π) given a state (s_t), that indicates the level of uncertainty or stochasticity among the policy's possible actions. The parameter α serves as a crucial trade-off hyperparameter. It controls the balance between the reward and the entropy in the objective; altering the value of α allows us to control the emphasis on exploration (higher entropy) versus exploitation (higher reward).

By encouraging higher entropy, SAC motivates the policy to explore more, especially when the optimal action is indefinite; it also enables it to explore multiple modes of near-optimal strategies, which helps it to adapt and learn various behaviors.

2.4 Transformer

Most well-known models for transforming sequences in neural networks utilize an encoder-decoder configuration, wherein the encoder translates an input sequence of symbol representations $(x_1, ..., x_n)$ into a series of continuous representations denoted as $z = (z_1, ..., z_n)$. Subsequently, leveraging z, the decoder produces an output sequence $(y_1, ..., y_m)$ of symbols iteratively. At each step, the model operates in an auto-regressive manner, incorporating previously generated symbols as additional input during the generation of the subsequent symbol. The Transformer adheres to this overarching design, employing stacked self-attention and point-wise, fully connected layers for both the encoder and decoder. These components are illustrated in the left and right halves of Fig. 2.2, respectively.

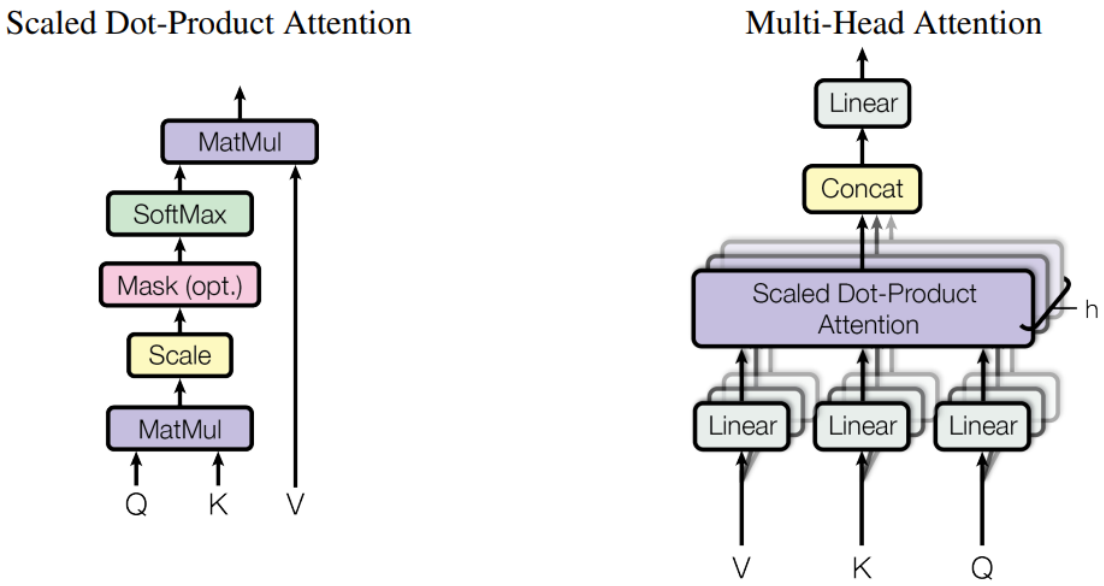


Figure 2.3: Scaled Dot-Product Attention (left); Multi-Head Attention (right) [63]

2.4.1 Model Architecture

Encoder: Comprising a stack of N identical layers, the encoder exhibits a dual-sub-layer structure. The initial sub-layer introduces a multi-head self-attention mechanism, while the second sub-layer use a simple position-wise fully connected feed-forward network. Each sub-layer is contained by a residual connection [64], followed by layer normalization [65]. Specifically, the output of each sub-layer adheres to the expression LayerNorm(x + Sublayer(x)), where Sublayer(x) denotes the function executed by the sub-layer itself. To facilitate these residual connections, all sub-layers within the model, including the embedding layers, yield outputs of dimension d_m.

Decoder: Correspondingly composed of N identical layers, the decoder introduces an additional sub-layer to each encoder layer. This newly added sub-layer conducts multi-head attention over the output generated by the encoder stack. Analogous to the encoder, residual connections surround each sub-layer, followed by layer normalization. Further, a modification is applied to the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This strategic masking, coupled with the fact that the output embeddings are displaced by one position, ensures that predictions for position i rely solely on the established outputs at positions preceding i.

2.4.2 Attention

An important part of the Transformer architecture is the attention mechanism which gives the model the ability to selectively focus on different parts of the input sequence. In contrast to recurrent neural networks(RNNs), which process sequences in a linear fashion, the attention mechanism allows Transformers to decide how important each element is in the sequence based on the task it is solving. This mechanism helps the model to capture long-range dependencies with very good efficiency.
An attention function can be describes as an opreation that maps a query and a collection of key-value pairs to an output, in which the query, keys, values, and output are all represented as vectors. The output is determined by a weighted summation of the values, with each value's weight decided by a compatibility function that involves the query and its corresponding key.

Scaled Dot-Product

The attention mechanism employed is termed "Scaled Dot-Product Attention". The input involves queries and keys of dimension d_k. The computation involves obtaining dot products between the query and all keys, dividing each by $\sqrt{d_k}$, and subsequently applying a softmax function to derive

the weights associated with the values. In practical application, the attention function is computed concurrently on a group of queries, consolidated into a matrix Q. Simultaneously, the keys and values are assembled into matrices K and V.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.17}$$

The most commonly used attention functions are additive attention and dot-product (multiplicative) attention. Dot-product attention closely resembles the algorithm in equation 2.17, with the exception of the scaling factor $\frac{1}{\sqrt{d_k}}$. Additive attention, on the other hand, computes the compatibility function with the use of a feed-forward network featuring a single hidden layer.
Although the performance of the two mechanisms is comparable for small values of d_k, additive attention surpasses dot-product attention without scaling for larger values of d_k [66]. This inconsistency in performance is due to the likelihood of dot products reaching substantial magnitudes for large d_k values, thereby pushing the softmax function into regions characterized by exceedingly small gradients. As a countermeasure to this occurence, a scaling factor of $\frac{1}{\sqrt{d_k}}$ is included to normalize the dot products.

Multi-head Attention

Rather than carrying out a singular attention function with model-dimensional keys, values, and queries, a more favorable approach is the linear projection of queries, keys, and values h times. This projection introduce distinct, learned linear projections to dimensions d_q, d_k, and d_v, respectively. The attention function is then independently applied in parallel to each of these projected versions of queries, keys, and values, resulting in d_v-dimensional output values. These outputs are subsequently linked and exposed to an additional projection, peaking in the final values. The utilization of multi-head attention makes the model able to collectively consider information from various representation subspaces at different positions. In contrast, employing a single attention head and subsequently averaging hinders this capacity.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, ..., h_h) \cdot W^O \tag{2.18}$$

where $h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and the projection parameter matrices $W_i^Q = \mathbb{R}^{d_m \times d_k}$,$W_i^K = \mathbb{R}^{d_m \times d_k}$,$W_i^V = \mathbb{R}^{d_m \times d_v}$.
For a model with n parallel attention layers, the dimensionality of each attention layer can be reduced to $d_k = d_v = d_m/n$, which will result in a computational cost equal to that of single-head attention with full dimensionality (d_m).

2.4.3 Embeddings

Similar to commonly used sequence transduction models, learned embeddings are employed to transform input tokens and output tokens into vectors of dimension d_m. Additionally, a frequently learned linear transformation and softmax function are applied to convert the decoder output into probabilities for the next token prediction. Notably, an alternative configuration is to adopt a shared weight matrix for both embedding layers and the pre-softmax linear transformation, a design choice akin to [67]. Within the embedding layers, the weights are scaled by the square root of the model dimensions d_m.

2.4.4 Positional Encoding

In the absence of recurrence and convolution within the model, including information about the relative or absolute position of tokens in the sequence becomes essential for the model to effectively utilize the sequence order. Accordingly, 'positional encodings' are introduced into the input embeddings at the base of both the encoder and decoder stacks. These positional encodings share the same dimension, d_m, as the embeddings, to enable summation between the two. Numerous alternatives exist for positional encodings, including both learned and fixed approaches [68].

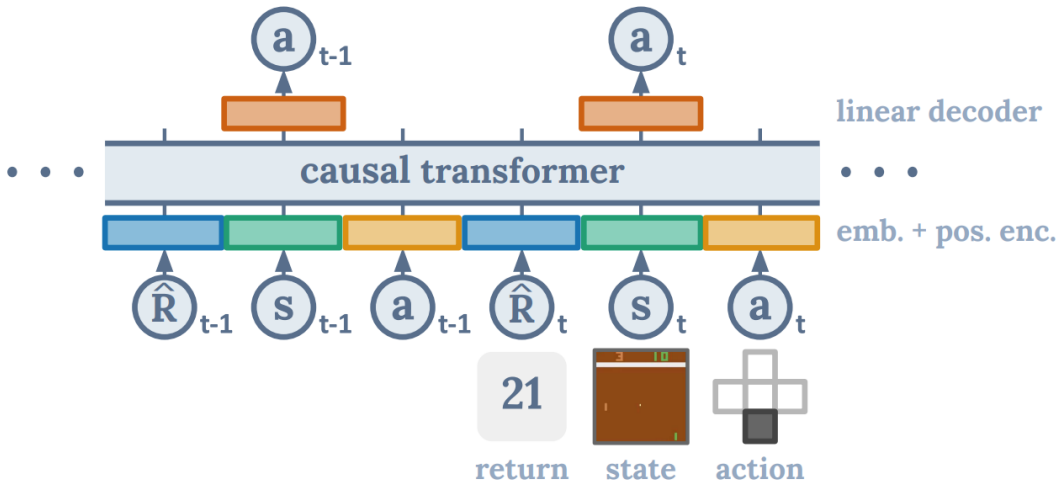


Figure 2.4: The Decision Transformer architecture involves the incorporation of states, actions, and returns, which are processed through modality-specific linear embeddings. Additionally, a positional episodic timestep encoding is introduced. Tokens are then input into a GPT architecture, enabling the autoregressive prediction of actions through the utilization of a causal self-attention mask [71]

.

2.5 Decision Transformer

Recent research has demonstrated the capability of transformers to effectively model high-dimensional distributions of semantic concepts on a large scale. This includes notable achievements such as proficient zero-shot generalization in language [69] and successful out-of-distribution image generation [70]. Given the varied success of these applications, the focus has also been directed toward assessing their suitability for addressing sequential decision-making problems formulated within the framework of RL. The approach of *Decision Transformers* leverages the simplicity and scalability inherent in the Transformer architecture, along with the advancements in language modeling exemplified by models such as GPT-x and BERT.
Diverging from previous methods in reinforcement learning that involve fitting value functions or computing policy gradients, Decision Transformer adopts a distinctive approach by directly producing optimal actions through the utilization of a causally masked Transformer, visualized in Fig. 2.4. Through the conditioning of an autoregressive model on the desired return (reward), historical states, and actions, the Decision Transformer model has the capacity to generate future actions that lead to the desired return.

Algorithm 2 Decition transformer psuedo code

function DECITIONTRANSFORMER($\hat{R}$, s, a, t)
$\text{pos}_{emb} \leftarrow \text{Embedding}(t)$ ▷ Learnt position embedding
$s_{emb} \leftarrow \text{Linear}(s) + \text{pos}_{emb}$ ▷ Linear state embedding
$a_{emb} \leftarrow \text{Linear}(a) + \text{pos}_{emb}$ ▷ Linear action embedding
$\hat{R}_{emb} \leftarrow \text{Linear}(\hat{R}) + \text{pos}_{emb}$ ▷ Linear reward embedding

$\text{input}_{emb} \leftarrow \text{Concat}(\hat{R}_{emb}, s_{emb}, a_{emb})$ ▷ Inpute embeddings
$\text{hidden\_state} = \text{Transformer}(\text{input}_{emb})$ ▷ Hidden states

$\text{pred}_a \leftarrow \text{Linear}(\text{hidden\_state}[\text{a}])$ ▷ Action prediction from hidden states(action index)
$\text{pred}_r \leftarrow \text{Linear}(\text{hidden\_state}[\hat{R}])$ ▷ Reward prediction from hidden states(reward index)

2.5.1 Trajectory Representation

Modeling rewards poses a nontrivial challenge, as the objective is to have the model generate actions based on future desired returns rather than past rewards. Consequently, instead of directly

inputting rewards, the model is supplied with the returns-to-go, denoted as $\mathcal{R}_t = \sum^T t' = trt'$. This results in the subsequent trajectory representation, which lends itself to autoregressive training and generation:

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ..., \hat{R}_T, s_T, a_T) \tag{2.19}$$

During the testing phase, the desired performance and the environment's starting state can be specified as conditioning information to initiate the generation process. Following the execution of the generated action for the current state, the target return is reduced by the obtained reward, and this process is repeated until episode termination.

2.5.2 Positional Encoding

A unique embedding is learned for every timestep and appended to each token. This differs from the conventional positional embedding typically employed in transformer models, as here, one timestep is equivalent to three distinct tokens. These tokens undergo processing by a GPT 2 model, as referenced in [72], which utilizes autoregressive modeling to estimate subsequent action tokens.

2.6 Optimizers

Various optimization algorithms exist, each with distinct advantages and limitations, suited to different types of problems. In this context, we explore Gradient Descent [73] and Quasi-Newton methods [74] alongside Particle Swarm Optimization (PSO) [75] for our active power line management project.

Gradient Descent is a fundamental approach used primarily in machine learning and deep learning for minimizing the cost function. The core principle involves updating the parameters iteratively in the opposite direction of the gradient of the cost function with respect to the parameters. Its simplicity is a major advantage, but it can struggle with non-convex functions, potentially getting trapped in local minima.

Quasi-Newton methods, including the Broyden–Fletcher–Goldfarb–Shanno algorithm [76], are advanced optimization techniques that approximate the Newton method for finding stationary points of functions. They are particularly effective for large-scale optimization problems due to their low memory requirements and ability to handle complex, nonlinear objectives. However, they require the objective function to be differentiable and can be computationally intensive.

Furthermore, PSO is particularly advantageous with its proficiency in handling complex, multi-dimensional optimization problems. Its strength lies in efficiently exploring search spaces without necessitating gradient information, making it an interesting alternative for intricate applications like power line management.

2.6.1 Particle Swarm Optimization

PSO is a computational algorithm used in optimization problems, characterized by its efficiency and simplicity. It's a versatile method that's especially useful where problem spaces are complex, and where analytical gradients are not available [77].

In PSO, a set of candidate solutions (the 'swarm'), referred to as particles, navigate the search space to find an optimal solution. Each particle's stochastic movement is influenced by two primary factors: its personal best position and the global best position found by the swarm. This mechanism enables the swarm to explore and exploit the search space effectively [78].

The algorithm operates by initializing particles randomly within the search space. These particles iteratively tweak their velocities and positions based on their experiences and those of their neighbors. The velocity update is influenced by the historical best positions of the particles, guiding them towards promising regions.

Formally, we define the cost function as $f : \mathbb{R}^n \to \mathbb{R}$, which needs to be minimized without known gradients. This function evaluates a candidate solution (a vector in $\mathbb{R}^n$) and outputs a real value representing its quality. The objective is to find a vector $\mathbf{a}$ such that $f(\mathbf{a}) \leq f(\mathbf{b})$ for all $\mathbf{b}$ in

the search space, signifying that $\mathbf{a}$ is the global minimum. Within the swarm of P particles, each particle i has a position $\mathbf{x}_i \in \mathbb{R}^n$ and a velocity $\mathbf{v}_i \in \mathbb{R}^n$, with $\mathbf{p}_i$ denoting its best known position and $\mathbf{g}$ representing the best known position across the swarm. The PSO algorithm utilizes these definitions to iteratively minimize the cost function see Fig. 3.

Algorithm 3 PSO pseudo code

for each particle $i \in P$ **do**
  $\mathbf{x}_i \leftarrow U_{(\mathbf{b}_{min},\mathbf{b}_{max})}$ ▷ Sample point from a uniform distrobution
  $\mathbf{p}_i \leftarrow \mathbf{x}_i$ ▷ Initilize particle best known position
  $\mathbf{v}_i \leftarrow U_{(-\lvert\mathbf{b}_{min}-\mathbf{b}_{max}\rvert,\lvert\mathbf{b}_{min}-\mathbf{b}_{max}\rvert)}$ ▷ Sample velocity from from a uniform distrobution
  if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then**
    $\mathbf{g} \leftarrow \mathbf{p}_i$ ▷ Initilize swarm best known position
while not terminated **do**
  for each particle $i \in P$ **do**
    for dimension $d \in \mathbf{x}_i$ **do**
      $r_p \leftarrow U_{(0,1)}$ ▷ Sample random cognitive scalar
      $r_g \leftarrow U_{(0,1)}$ ▷ Sample random social scalar
      $\mathbf{v}_{i,d} \leftarrow w\mathbf{v}_{i,d} + \phi_p r_p(\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g(\mathbf{g}_d - \mathbf{x}_{i,d})$ ▷ Update velocity
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$ ▷ Update position
    if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then** ▷ Evaluate new position
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$ ▷ Update best position
      if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then** ▷ Evaluate new best position
        $\mathbf{g} \leftarrow \mathbf{p}_i$ ▷ Update best swarm position

$\mathbf{b}_{min}$ and $\mathbf{b}_{max}$ define the search space's lower and upper bounds, setting limits within which particles can move. The parameter w, known as the inertia weight, balances exploration and exploitation by affecting the momentum of particles. The cognitive coefficient (ϕ_p) and social coefficient (ϕ_g) lead particles towards their personal best and the swarm's best positions, respectively. The selection of w, ϕ_p, and ϕ_g is important, as it dictates the PSO method's effectiveness and behavior. The algorithm terminates either after a set number of iterations or when reaching a satisfactory solution as defined by the objective function.

Chapter 3

Method

3.1 Experiment setup

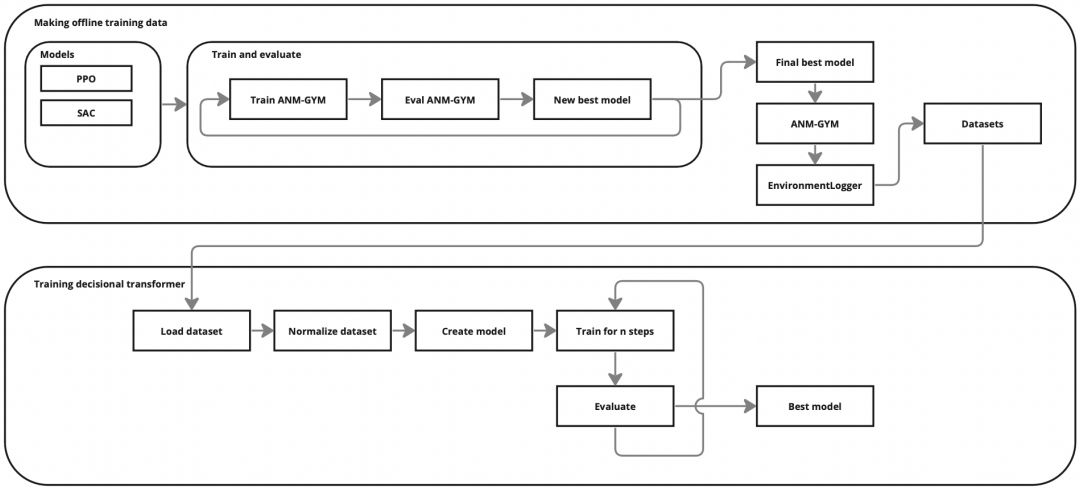


Figure 3.1: Project flowchart

An overview of the setup for how the decision transformer was trained to perform active network management with intermittent renewables sources is shown in the form of a flowchart in Fig.3.1. The setup consists of two major parts; dataset generation from baseline models, and decision transformer training. The generation of the dataset is done similarly to how the datasets in the D4RL [79] catalog are generated. Two state-of-the-art RL models; PPO and SAC, were chosen from the *stable baselines3 zoo* [80]. The choice of models is discussed in 5.1. These models both serve as a baseline for expected performance, and SAC is further used as a teacher model to generate the offline dataset. The teach model choice was determined by the best performance reached after training, which was based on evaluation runs. Multiple models were chosen to give a better baseline metric, as well as to give the option to choose one over the other if there was found to be a performance gap. This model was then used to generate the trajectories $\boldsymbol{\tau} = [\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{t}]$ by capturing the rollouts as well as the environment metrics. This would compose the offline training set $\mathbf{D} = [\boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_n]$ for the decision transformer.

Two versions of the DT architecture were trained; one that only predicts action(DT_a), and one that predicts action and reward(DT_{ar}) which would be used in conjunction with PSO. Training of the DTs was performed by initializing the model with its hyperparameters. Then loading and creating normalized batches from the trajectories in the offline dataset. The model was trained on these for multiple steps and periodically deployed in the online environment to evaluate its performance. The weights of the best-performing models were saved to be evaluated further.

3.2 ANM-GYM

A fork of ANM-Gym [81] is used to evaluate our agent and answer our research question. Some minor deprecations and bugs were resolved for the gym to be usable. The simulation backend of the environment is based on the work of Gemine et al.[3], which has been adapted by Henry et al [82] to follow the structure of OpenAI environments for RL training. This framework allows for the imitation of real-world power systems, and the creation of synthetic networks of varying complexity. These power network simulations are focused on the DSO/TSO level, meaning that only the larger components of the grid are considered, such as power plants, power heavy industry, and the aggregation of smaller loads such as residential homes.

3.2.1 Components of the environment

The ANM framework allows the creation of environments using five distinct component types: passive loads, generators, DES, transmission lines, and busbars. These components fit into two categories; network linking components, and power components. The set of power components(D) consists of passive loads(D_l), generators(D_g), and energy storage(D_e). The components in D_l can only consume power from the grid, D_g can only inject power into the grid, and D_e can both consume and produce power, depending on its state of charge(SoC). There is also a requirement for all networks to include a slack bus, which is responsible for keeping the balance between load and generation. It will consume surplus power from the generators and supply the required power if needed to fulfill a power deficit. The slack bus is an imitation of the connection to the power grid, which will always have far greater capacity than the local grid.
Network linking components are used to construct the network topology, which is represented as a directed graph, $G = (V, E)$, with nodes and edges representing busbars and transmission lines respectively. The set of edges(E), represents the connections between the integer set of nodes(V). Each connection is represented as $E_{i,j}$ where i and j are integers describing which nodes the edge is connected between. The need to represent the network as a directed graph arises from modeling of transmission line losses which entails that power injected at one side of the transmission line is not equal to the power received at the other end.

3.2.2 Environment simulation

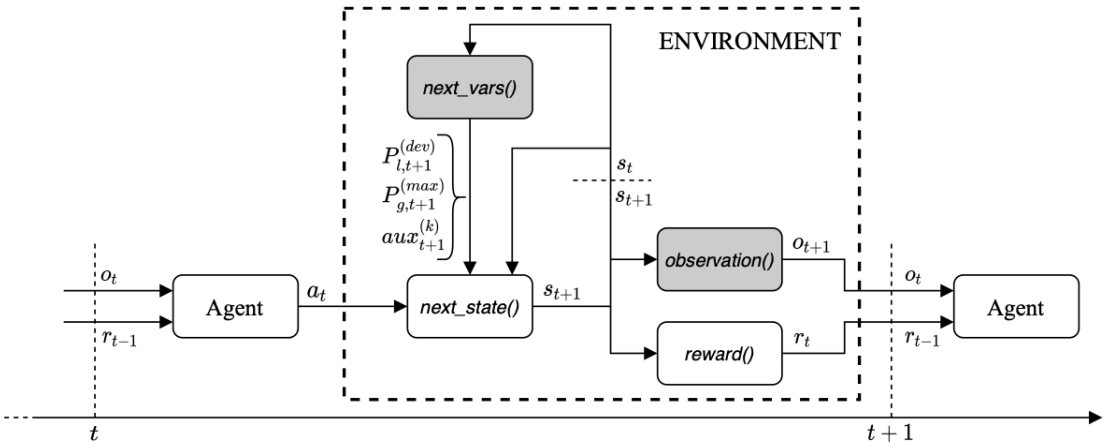


Figure 3.2: Flowchart of how new state and rewards are calculated based on agent action in the ANM environment, taken from [82].

The environment can be divided into several smaller components, each with a dedicated function. The `next_state` block is responsible for the deterministic simulation of the power network given the agent's output action. This is done by solving for the power flow using Newton's method. The loads and available renewable energy are subject to change for each time step, the logic defining the temporal change of these values is defined in the `next_vars` block. These changes are then used

in the generation of the next state vector, s_{t+1}. It is important to note that the agent requires some information about how these variables change in order to keep it Markovian. This is ensured through the introduction of an auxiliary vector, aux, containing all necessary information to infer the next state. The state is further used to calculate the reward for the current time step, r_t, using the reward function defined in the `reward()` block. The state can be observed by removing some elements from the vector, forming an observation state, o_{s+1}. This is done to simulate components outside the agent's control. However, this was not done for the ANM6 network, as it is a relatively small network.

3.2.3 State vector

The state vector, $\mathbf{s} = [P_{0...n}, Q_{0...n}, SoC, P^{max}_{0...n}, \kappa]$, includes active($P_{0...n}$) and reactive power($Q_{0...n}$) injections from devices into the grid; the SoC of DES units, and the maximum production capacity of generators($P^{max}_{0...n}$). To capture temporal dependencies and ensure Markovian properties, an auxiliary variable(κ), is included. This variable enables the modeling of additional temporal factors that influence future power injections and maximum production, ensuring that the state transitions are solely conditioned on the current state of the environment and the action taken by the agent.

3.2.4 Action vector

The action vector, denoted as $\mathbf{a}$, encompasses the set of control variables that the agent can manipulate in each time step. These variables, represented by the action vector $\mathbf{a} = [P^g_{0...n}, Q^g_{0...n}, P^{DES}, Q^{DES}$, consist of active power injections setpoints of generators ($P^g_{0...n}$), reactive power setpoint of generators ($Q^g_{0...n}$), active power injections for DES units(P^{DES}), and reactive power setpoint of DES units(Q^{DES}]).
These variables are constrained by the physical limits of the components shown in Table 3.1, available renewable power,$\overline{P}^{RES}_{0...n}$, as well as the current SoC of DES units. The action space is bounded by these constraints, ensuring that the agent's choices adhere to the practical limitations of the power system.
However, the agent is still able to choose actions that are outside of this feasible space. These occurrences are resolved by adjusting to the closest valid point according to Euclidean distance in the subsequent state update. This approach ensures that the agent's decisions align with the actual physical constraints of the power system.

3.2.5 Reward function

The reward function of the environment is defined to incentivize two types of behaviors, with network stability being the main priority and a goal of minimizing energy loss and renewable energy curtailment being a secondary goal. The function is defined as follows:

$$r_t = \begin{cases} \text{clip}(|l|\Delta t, 0, c_1) + \text{clip}(p\lambda\Delta t, 0, c_2), & \text{if } s_{t+1} \notin S_{term} \\ \dfrac{c_2}{1-\lambda} & \text{if } s_{t+1} \in S_{term} \end{cases} \tag{3.1}$$

where energy loss l, and penalties p are defined as:

$$l = \sum_{v=0}^{V} \max(0, P^{RES}_{max} - P^{RES}) + \sum_{e=0}^{E} P_{loss} \tag{3.2}$$

$$p = \sum_{i=0}^{N} \max(0, |u_i| - u_i^{max}) + \max(0, u_i^{min} - |u_i|)) + \sum_{j=0}^{line} \max(0, |k_{ij}| - k_{ij}^{max}) \tag{3.3}$$

with P^{RES}_{max} and P^{RES} being the maximum available renewable power and the renewable generator set point, respectively. P_{loss} are the losses resulting from power transmission. u_i is the voltage of the i-th bus and the max/min superscripts denote the maximum and minimum operating limits for

the voltage not to collapse. k_{ij} is the apparent power flowing though line ij with the subscript being the line capacity. The reward is bound between the values $[c_1, c_2]$ which are hyperparameters used to limit the reward received each transition. A terminal state S_{term} is reached when the network solver is unable to converge to a solution for s_{t+1}. This is usually caused by a collapse in voltage on one or more of the busbars. This is most commonly due to a shortage in reactive power [83].

3.2.6 Example environment ANM6

The ANM6 environment consists of 6 busses with 2 passive loads, 2 renewable sources, one DES unit, and one fossil generator used as the slack generator. These buses were interconnected with 4 transmission lines and one transformer. The two intermittent renewable sources consist of a solar farm and a wind turbine farm. This topology is illustrated in Fig. 3.3. The properties of these components can be found in Table 3.1. The DES unit has additional parameters, SoC_{max} which is set to 100MWh.

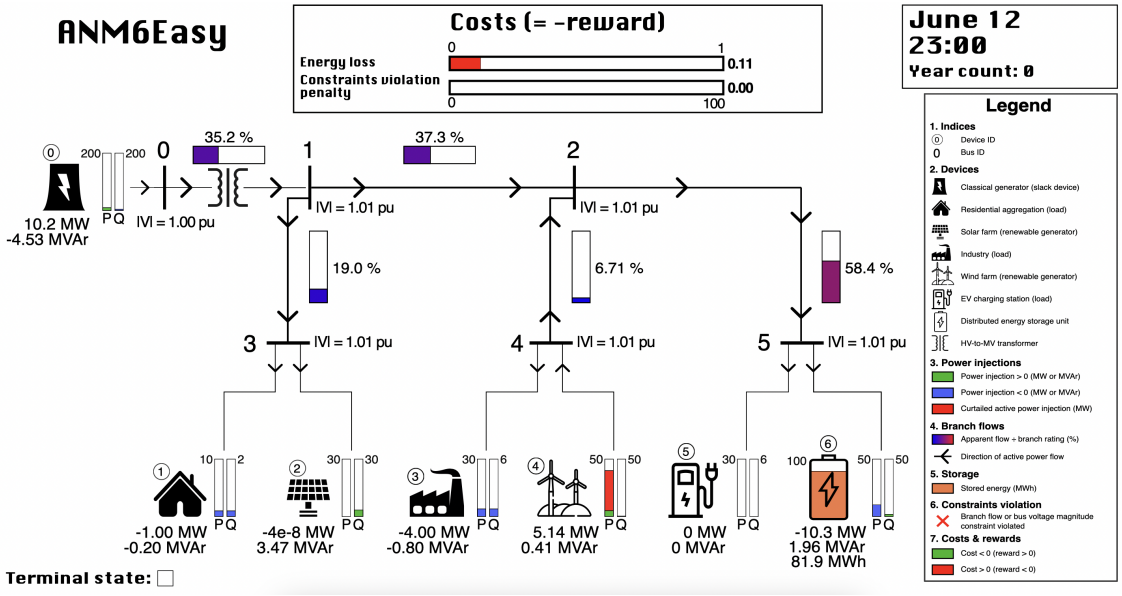


Figure 3.3: Example of the UI of the gym environment, showing the topology of ANM 6. Each component is explained in the illustration legend.

Table 3.1: Power network parameters

Bus	Type	P_{max}	P_{min}	P_c	Q_{max}	Q_{min}	Q_c
3	Solar	30	0	20	30	-30	$\pm$15
4	Wind	50	0	30	50	-50	20
5	DES	50	-50	$\pm$30	50	-50	$\pm$25
3	Residential	0	-10				
4	Industry	0	-30				
5	EV park	0	-30				

Each component has a defined maximum and minimum value for active and reactive power. There is also a limit to how much power a component can sink or inject simultaneously. This relationship can be seen in the PQ capacity diagram in Fig 3.4. These limits arise as the components in actuality are limited by their apparent power rating defined in equation (3.4). So the real limits are curved, however, for simplicity, the limit used is a linear slope between P_c and Q_c. The DES unit is also constrained by its SoC, not being able to inject any active or reactive power if no energy is stored, as well as not being able to sink any reactive or active power if full.

$$S = \sqrt{P^2 + Q^2} \tag{3.4}$$

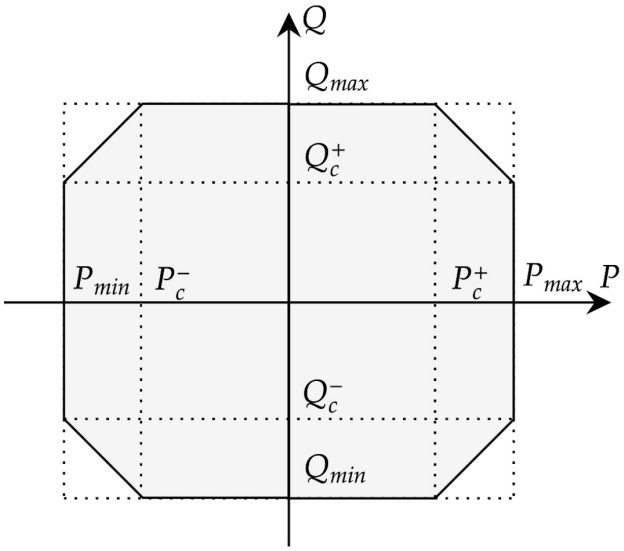


Figure 3.4: PQ capacity diagram

3.2.7 ANM6 Easy scenario

The *ANM6 Easy* scenario, designed by [82] is a combination of the ANM6 topology along with a set of production and load curves. This is meant to simulate a potential scenario seen on a normal day of a grid operator. The problem can be broken down into three stages, with a transitional period between each stage, as shown in Fig. 3.5. The stages are referred to as stages A, B, and C, and are highlighted as green, red, and blue sections, respectively. These stages are:

- Stage A: This stage simulates a night with high winds and low power consumption. This means that there is excess power in the grid which can be stored in the DES unit.
- Stage B: This stage simulates when people are going to and returning from work. This period has low wind and solar production and slightly higher demand from industrial and residential loads. The demand for the EV park is at its peak when cars are charged after the commute. This creates a high demand for bus 5, and the DES has to be unitized to not exceed the capacity of the transmission line going into this bus.
- Stage C: In this stage, the EVs are fully charged, so there is no demand for the EV park. People are at work so industrial demand is at its peak, and the residential demand is minimal. During this period solar and wind farms are at peak production.

Each day is represented in 15-minute intervals, resulting in a total of 96 steps per day. This was chosen as it is a common resolution used by DSO for energy reserve markets [84].

3.2.8 Clipped Reward

Periodic evaluations were performed every 40k steps to assess the performance of the agents in the ANM6-Easy task. During these evaluations, the training is paused, and the current baseline policy, π_θ, is used to perform $N_r = 5$ rollouts, each spanning $T = 5000$ timesteps, in a separate instance of the environment. The performance metric, $\mathcal{R}(s, a)$, is calculated as the expected return from the initial state, approximated by the average discounted reward($\tilde{R}_{\pi_\theta}$) over these rollouts:

$$\tilde{R}_{\pi_\theta} \approx \frac{1}{N_r} \sum_{i=1}^{N_r} \sum_{t=0}^{T-1} \gamma^t r_t^i \tag{3.5}$$

Here, r_t^i are the rewards during the i^{th} rollout. Given that the rewards are bounded by r_{clip}, approximating the true infinite discounted return may introduce an error. The potential deviation is bounded, as shown in the equations below:

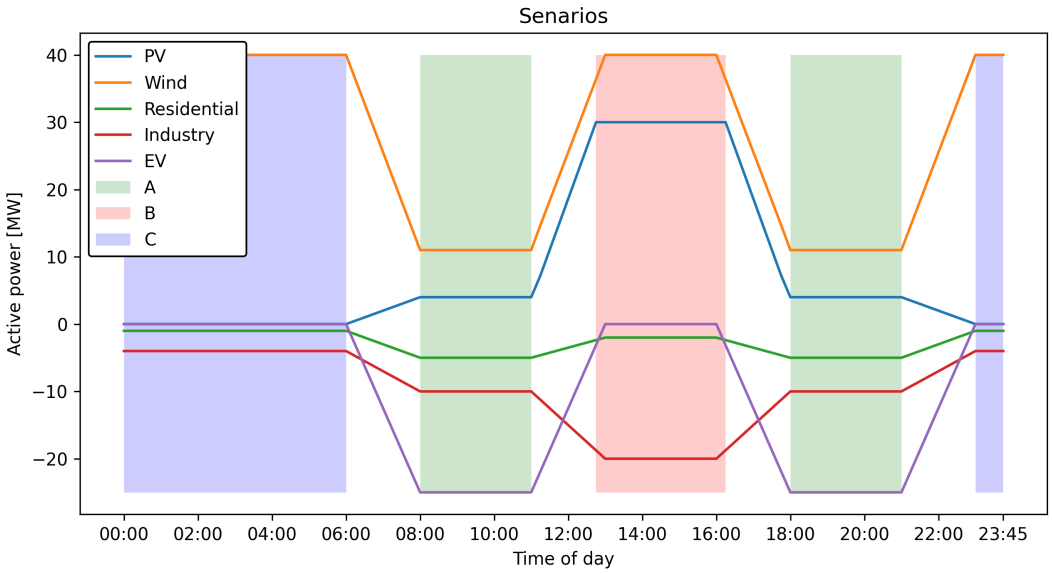


Figure 3.5: Showing load and maximum available active power throughout a 24-hour period.

$$\tilde{R}_{\pi_\theta}(s) \leq r_{clip}\frac{1}{1-\gamma} \quad \textit{and} \quad \sum_{t=0}^{T-1}\gamma^t r_t \leq r_{clip}\left(\frac{1}{1-\gamma} - \frac{\gamma^T}{1-\gamma}\right) \tag{3.6}$$

3.3 Baseline models training

3.3.1 Training

The models were trained using the `learn()` method from SB3 with a time and normalization wrapper around the environment. The normalization wrapper was used to normalize the action space to be within the range [-1,1], and The time wrapper was used to define the maximum length of each episode, which was set to 5000 steps. The default SB3 hyperparameters shown in Table 3.2 were left unchanged for training as they were sufficient to generate good results in the environment. However, it should still be noted that better performance is likely possible through hyperparameter optimization. The purpose of the models was to serve as a baseline and not a lower bound.

Table 3.2: Baseline models hyperparameters

(a) PPO

Hyperparameter	Value
Adam learning rate	3×10^{-4}
Discount factor (γ)	0.995
Minibatch size	64
Entropy coefficient	0
Clip ratio	0.2
Number of epochs	10
Value function coefficient	0.5
GAE lambda	0.95
Max gradient clip	0.5

(b) SAC

Hyperparameter	Value
Adam learning rate	3×10^{-4}
Discount factor (γ)	0.995
Minibatch size	256
Entropy coefficient	auto
Buffer size	1×10^{6}
Learning start	100
Soft update coefficient	5×10^{-3}
Target update frequency	1
Gradient step	1

3.3.2 Evaluation

A callback function was used to evaluate the agent every 40k timesteps. The evaluation consisted of averaging the episode reward from five runs each with a length of 3k steps. The episode reward is an accumulation of the step rewards in the episode. The difference between the environment used for evaluation and the one used for training is that the latter returns normalized rewards instead of discounted rewards. The criterion for when to stop training was derived from the return of the evaluations, where training was stopped once the difference between consecutive evaluations was negligible. This was found to be around 700k timesteps.

3.3.3 Dataset generation

The dataset, D, was generated by collecting the model rollouts, $a(s)$, the environment state, $s(a_{t-1})$, as well as the reward, $r(a, s)$, and data on whether or not the environment is in a terminal state, $s \in s_{term} \Rightarrow t = \top$. The evaluation environment was used to collect trajectories as the reward should be unnormalized. This allows for the dataset to be utilized by other models, leaving the option to normalize the values if needed. Furthermore, the unnormalized reward was required to calculate the returns-to-go value for the decision transformer. Returns-to-go at timestep, t, denoted as $\hat{R}_t$ is calculated using Eq. 3.7, where T is the last timestep of the trajectory. In addition to collecting the trajectories, the logger also collected metrics from the environment that were not a part of the state vector. This was done to allow for a more in-depth analysis of the model's performance in the environment.

$$\hat{R}_t = \sum_{t^*=t}^{T} r_{t^*} \tag{3.7}$$

3.4 Decision transformer

3.4.1 Training

The two DT configurations were trained similarly by backpropagating the loss calculated with mean squared error (MSE). The DT_a model only used the MSE between the masked predicted action and masked action from the dataset, whereas the DT_{ar} included an additional loss of the MSE between the predicted reward and the actual reward in the dataset. The target return-to-go was set to -130 and remained the same during training as it did during inference. This value was chosen to be slightly higher than the average reward of the dataset.
In the paper that introduced the DT model, [71], it is claimed that the prediction of reward and state had no effect on model performance. However, the prediction of reward is required for our approach to approximate the reward received from an action. This would then be used as the reward function for the PSO.

3.4.2 Context window

The model used a context window of size K for predicting future action and reward for DT_{ar}. This implies that the previous state, actions, and rewards-to-go need to be stored between each step for the K previous steps. This was achieved by using a deque buffer which removes the oldest trajectory when the buffer is full. An attention mask was also required since the buffer would be empty at the start of inference, and the empty values should not be included. This was achieved by initializing the buffer with a zero-mask, and updating this to 1 once a trajectory had been collected.

3.4.3 Optimizer

When training the decision transformer, we employed the AdamW(Adam with weight decay) optimizer with a learning rate of 10^{-4} and a weight decay of 10^{-3}. The AdamW optimizer was chosen

for its adaptive learning rate properties and weight decay capabilities, which have demonstrated effectiveness in enhancing training stability and convergence in deep neural network models [85].

3.4.4 Scheduler

To further optimize the training process, we implemented warm startup scheduling using the LamdaLE scheduler. The learning rate of the optimizer was linearly increased from 10^-5 to 10^-4 over a period of 10k steps. The smaller initial learning rate allows the optimizer to calculate a better approximation for the gradient by using more samples before the learning rate is large enough to alter the weights. This guards against the model being pushed in the wrong direction based on poor gradients [86].

3.4.5 Mini-batching

Training used a batch of 64 randomly chosen trajectories per epoch. These trajectories were normalized such that all values were within the range [-1,1] using min-max scaling. This is a well-established method to reduce training time [87]. A slice equal to the length of the context window, K was randomly selected from each trajectory. For the edge cases where the end of the selected slice was outside the range of the trajectory due to a terminal state being reached, the samples were padded with zeros to ensure each slice was of equal length. This ensures that the samples have start indexes in the whole range of the trajectory and that the terminal stage is not always found at the end of a slice. Only having the terminal state appear at the end of a slice could lead to a wrongful correlation between position embedding and terminal state.

3.4.6 Evaluation

The model was evaluated every 40k step. Evaluation of the model is done by taking the predicted actions from the model and denormalizing them back to their respective magnitudes before using them as input to the ANM gym environment. The model was evaluated until termination or reaching 5k steps. This was repeated 3 times, and the average episode length, step reward, and overall reward were used to determine the performance of the current epoch. If this was the best performance so far, the model parameters were saved as a benchmark for later epochs.

3.4.7 Hyperparameters

The hyperparameters shown in Table 3.3 are primarily related to the transformer's architecture. Hyperparameter search was not performed as it would only be limited to a shallow search due to slow training time in addition to the large amount of numbers iterations required to reach a stable region. For this reason, the only hyperparameters changed from the MuJoCo configuration in the original paper, were the number of layers and attention heads, which were both increased to 4.

Table 3.3: Decition transformer hyperparameters

Hyperparameter	Value
Number of layers	4
Number of attention heads	4
Embedding dimension	128
Context length(K)	48
Batch size	64
Activation function	ReLU
Grad normalisation clip	0.25
Learning rate(γ)	10^{-4}
Weight decay(λ)	10^{-3}

3.5 Particle Swarm Optimization fine-tuning

The PSO algorithm was implemented in accordance with the pseudo-code 3, in section 2.6.1. The original implementation aims to minimize the cost from the cost function, f. However, our goal was to maximize reward, so the implementation is modified accordingly by replacing f with the reward function $R(s, a)$. The parameters used in the implementation are listed in table 3.4. The initial best position, $\mathbf{g}$, was set to the predicted action by DT_{ar}. The upper and lower bounds of the uniform distribution were set to $\pm$ 0.2 of the initial action, clipped by the range [-1,1]. This was to ensure that the new action was not too different to the initial action. This was done since the local minima of the full range of actions might not account for future rewards. Therefore, the action space of the PSO was limited to a shallow region around the initial action. When evaluating a particle's position, only the last action of the DT context window was mutated. The deque buffer was only updated by $\mathbf{g}$ once the optimization had converged.

Table 3.4: PSO hyperparameters

Hyperparameters	Value
Number of particles(N)	15
Inertia (w)	0.05
Cognitive coefficient(ϕ_p)	0.6
Social coefficient(ϕ_g)	0.8
Upper bound (b_{max})	$\mathbf{a} + 0.2$
Lower bound (b_{min})	$\mathbf{a} - 0.2$
Convergence threshold(τ)	0.1
Max iterations(t_{max})	20

3.5.1 Convergence criteria

The criteria used to determine the convergence of the optimization was the sum of differences between the best particle positions of the current iteration compared to the previous was, smaller than some threshold, τ, defined mathematically as $\tau \geq \sum_{i=0}^{N}(\mathbf{p}_{i,t} - \mathbf{p}_{i,t-1})^2$. Additionally, an upper limit, t_{max}, was used to terminate the optimization if no convergence had been reached.

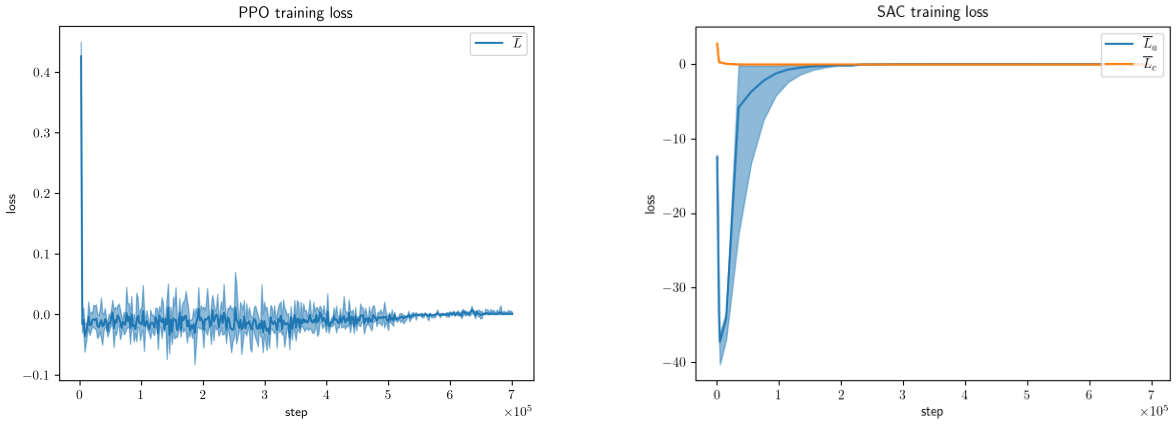
Chapter 4

Results

In this chapter, the results of the baseline model's training and evaluation are presented, and metrics for validation of the variance of the generated dataset are introduced. Next, the results from the training process of the decision transformer are presented. In addition, the performance of the decision transformers is evaluated by investigating the current flow in the network. Lastly, the results from the PSO fine-tuning are presented and compared with the non-fine-tuning results along with metrics for the decision transformer's performance as a cost function estimator.

4.1 Baseline training and evaluation

In this section, the performance of the baseline models is evaluated. Each figure represents an average performance over multiple independent runs. The shaded area indicates the maxima and minima values of these runs.



(a) **(b)**

Figure 4.1: Training loss PPO(a), and SAC(b)

Training loss for an RL model is generally not as interpretable as in supervised learning. However, it serves as an indicator of the stability of the learning process, with the expectation of having loss asymptotically decrease towards zero. The PPO loss can be observed to drop quickly and remain close to zero for the duration of the training with the variance being approximately 0.03 up until step 5×10^5, where the loss stabilizes. This roughly aligns with when the evaluation reward plateaus, seen in Fig. 4.2. It should be noted that the loss, L, depicted in 4.1a is a combination of losses as defined in (2.15), which simplifies to the following equation:

$$L = L_{policy} + c_{ent}L_{ent} + c_{vf}L_{vf} \tag{4.1}$$

where, L_{policy} is the policy loss, c_{ent} is the entropy loss coefficinet, L_{ent} is the entropy loss, c_{vf} is the value function coefficient, and L_{vf} is the value function loss.

The loss from the SAC training, shown in Fig. 4.1b, shows both the actor loss, L_a, and critic loss, L_c, rapidly moving towards zero. It should be noted that this figure contains fewer data points than the PPO loss since the SAC model was only set up to log the mean loss of multiple gradient steps.

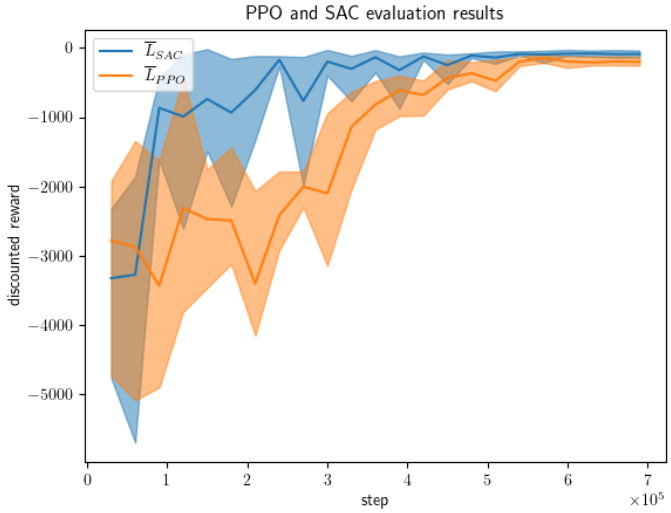
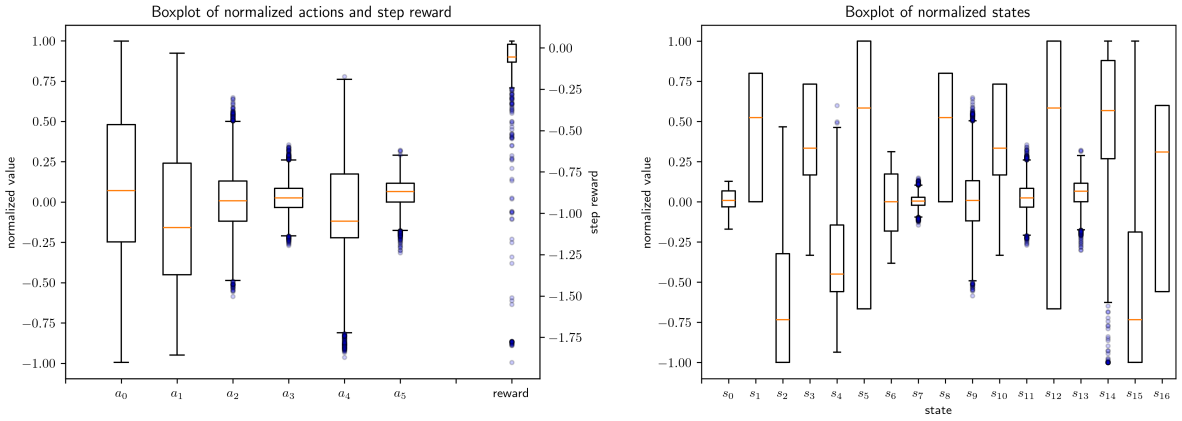


Figure 4.2: Average of the accumulated discounter rewards received in each episode of the evaluation runs.

The average discounted reward obtained from 3 independent training runs of the SAC and PPO models are shown in Fig.4.2. The performance of each independent PPO model is initially varied and sporadic. This settles down around the 350k step where the performance increases before plateauing after the 500k steps. A similar trend is observed for the SAC training, however, it starts to see relatively good performance after only 100k steps. Then the performance gradually increases for the remainder of the training.
Recall that at each evaluation step, 5 evaluation episodes are evaluated. The scalar average of these are again averages between the 3 runs. Hence, the variance of the 5 evaluation episodes is not captured in the figure.

4.1.1 Dataset variation



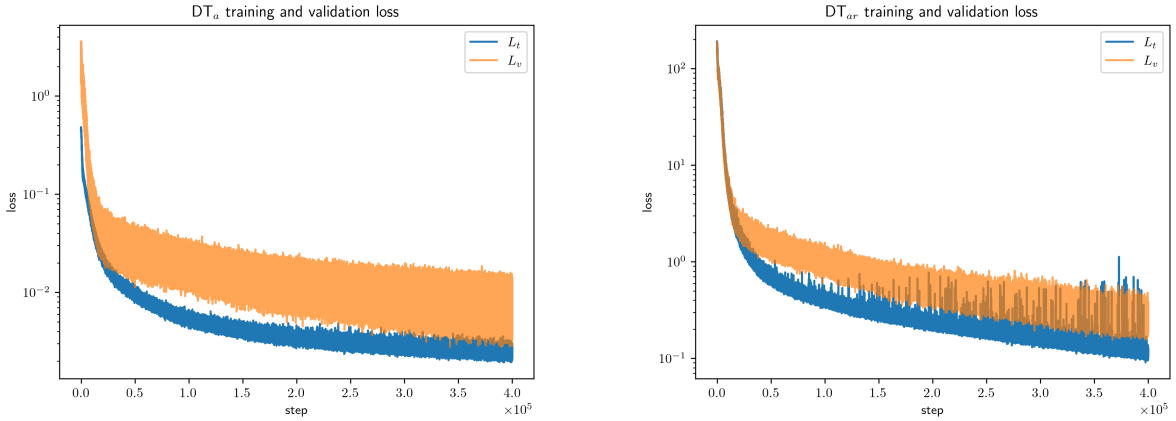
(a) **(b)**

Figure 4.3: Showing the distribution of trajectories in the dataset with the use of boxplot, displaying actions and rewards(a), and states(b). Note that s_17 has been left out of the figure since it always oscillates between [0,95]

The distribution of normalized values on the action and state vectors, as well as rewards contained in the offline training dataset, is shown in Fig.4.3 as a box plot. The top and bottom of the box

represent the 75th and 25th percentile of the data, whereas the orange line represents the median. The whiskers represent the 0th and 100th percentile, and the blue dots are outliers in the data.

4.2 Decision transformer training and evaluation



(a) **(b)**

Figure 4.4: Decision transformer training and evaluation loss during training.

The training loss of the two DT models DT$_a$, and DT$_{ar}$ are shown in figures 4.4a and 4.4b, respectively. The test and validation loss in both figures can be observed to decay exponentially. The loss of DT$_{ar}$ is higher than DT$_a$, which is to be expected as the loss value also includes the MSE of the predicted reward. The validation loss remains greater than the test loss which is indicative of a model training correctly. The decline of the validation loss in conjunction with test loss is an indication that the model can generalize to the trajectories in the holdout set.

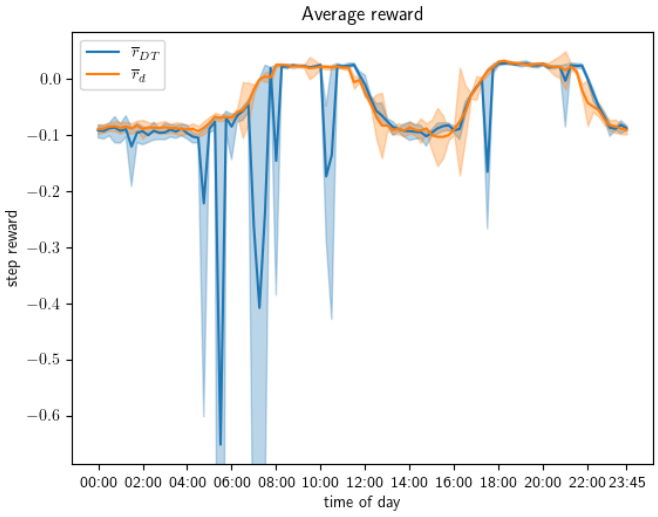


Figure 4.5: Comparison between the average reward found in an episode from the dataset,r_d, and the average reward achieved by the decision transformer r_{DT}

The average reward of a 1k step trajectory from the dataset is compared with the reward obtained by the agent in Fig. 4.5. This trajectory length roughly equates to a month of simulated operation, as each day consists of 96 steps. It can be seen that the reward in the dataset is consistent throughout, with there being little to no deviation from the average compared to the minima and maxima, shown as the shaded region. It can be seen that for large parts of the day, the DT agent has similar performance to the one in the dataset, however, between the hours 06:00 to 09:00 and, 16:00 to

19:00 the agent performs worse. Looking at the load chart in Fig. 3.5 it can be seen that these regions are the transitionary periods going into stages B and C, respectively.

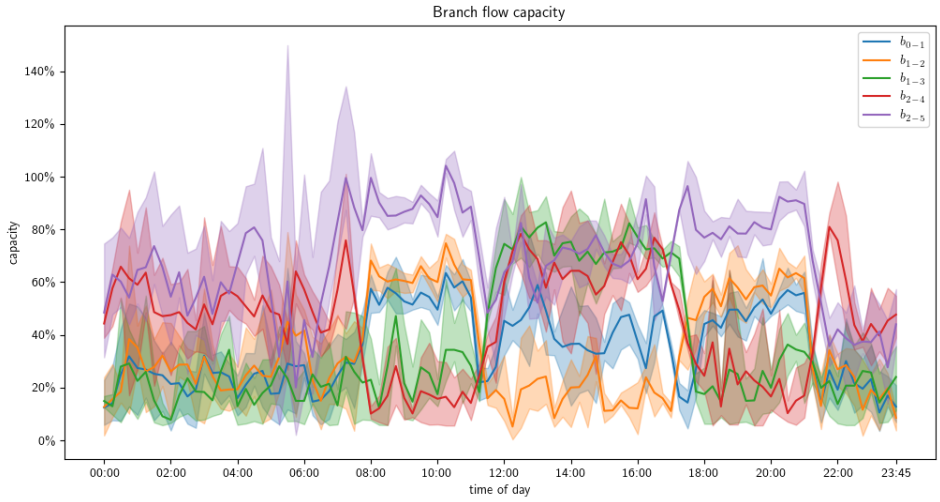


Figure 4.6: Showing the average power flow in each branch as a percentage of max capacity for a 1k trajectory, with the shaded area being the maximum and maximum observed in the trajectory

In the overview of the average branch power flow shown, in Fig. 4.6, it can be seen that the only branch exceeding the power limit of 100% is branch b_{2-5}. This occurs between 8:00 and 9:00, which is in the transition period between stages A and B. Looking at the stage of charge of the DES unit in that branch, shown in Fig. 4.7, it can be seen that the DES still charging as the load from the EV park, shown in red in the figure, is starting to increase. Comparing the charge of the DES in the dataset, SoC_d, to the charge from the DT, SoC_{DT}, it can be seen that DES is generally charged slower by the DT agent.

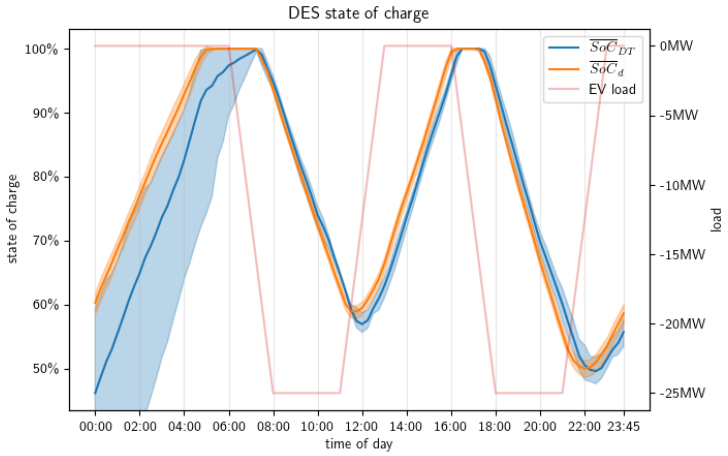
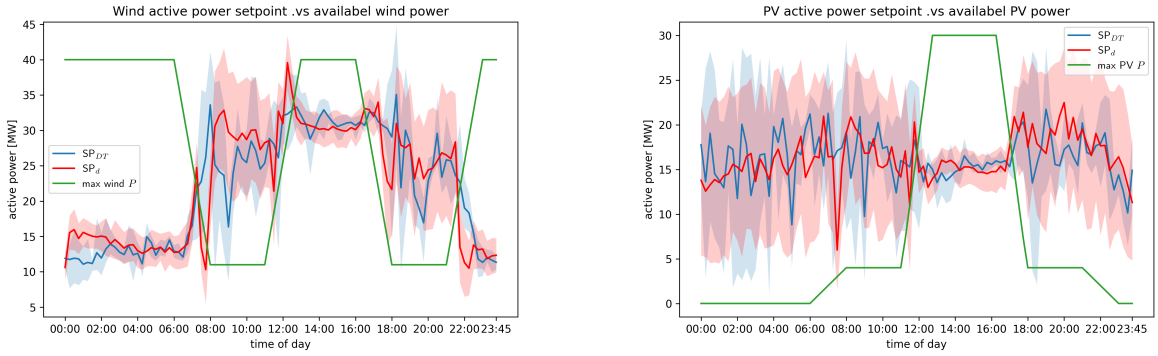


Figure 4.7: Showing the SoC of the DES unit connected to bus 5 compared with EV park load. Note that the SoC level does not start at 0%.

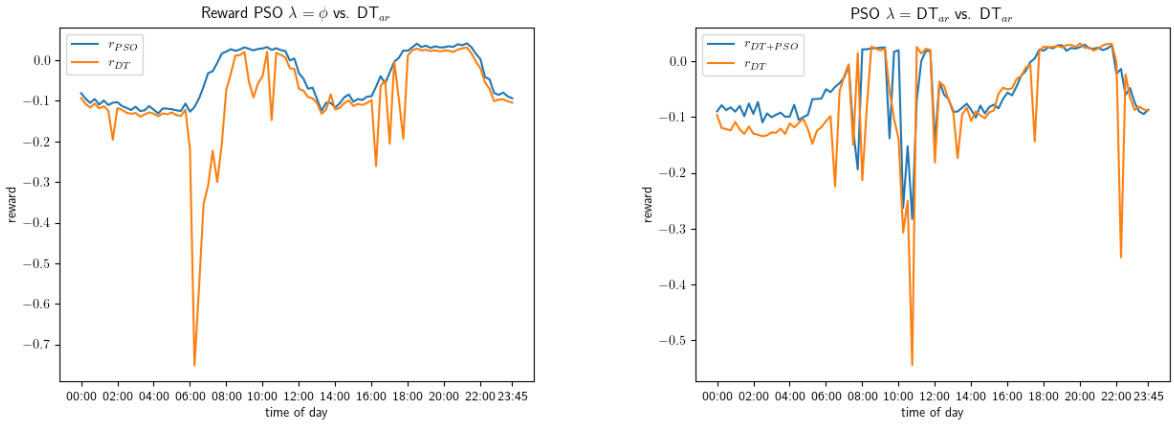
In the comparison of available power and the actions taken by the DT_a agent and the ones in the dataset, shown in Fig. 4.8, it can be seen that when the available power is close to zero the agents have setpoints higher than what is possible. Recall that this is resolved by taking the closest valid action based on Euclidean distance, so the input to the environment will be the max available power in these cases. Another observation is that the setpoint seems to change more sporadically in this region where the setpoint is set above the max available power.



(a) **(b)**

Figure 4.8: Comparing DT$_a$ agent's actions and dataset actions with available power for wind(a) and PV(b)

4.3 Particle Swarm Optimizer fine-tuning



(a) **(b)**

Figure 4.9: Showing original reward for action taken by DT and reward after PSO fine-tuning with a cost function,λ, equal to the simulation penalty function,ϕ defined in 3.2 (a), and the predicted reward from model DT$_{ar}$(b)

In Fig. 4.9a the reward from an action taken by the DT is compared with the reward after PSO fine-tuning with perfect knowledge. This refers to the fact that the cost function in the PSO can utilize the reward function from the environment, giving it the exact reward a different action would produce. This figure is meant to illustrate the PSO's ability to improve the performance of the agent given a good estimation of the reward from an alternative action. The original action taken by the DT is used as input to the PSO as an initial starting point. Note that this figure is only showing the reward from 96 steps as opposed to an average over many steps as in previous figures, this is due to the heavy computing cost of solving the network equations to calculate the reward each time a particle evaluates its position.
A comparison between the DT's predicted reward for an action, **a**, and the actual reward obtained from the simulator for the same **a** is shown in Fig.4.10. Recall that **a** represents the 6-dimensional vector composed of sets of P-Q set-points for each controllable component. The x-y directions in each subplot represent the normalized set-point concerning the limits of action space, A, for the reactive effect, Q, and active effect,P, respectively. The reward at each point is calculated by evaluating each point in a 50×50 grid with a Gaussian kernel to get a smooth transition between the discrete steps. While one set of P-Q points was evaluated, the other four values in **a** were kept as zero. The state vector, **s** when **a** was evaluated was kept constant and was sampled from the training dataset. The chosen sample was at 12:00 o'clock in simulation time, which is in the transition period between stages B and C. This was chosen as it is a period with both PV and wind power available, as well as moderate demand.

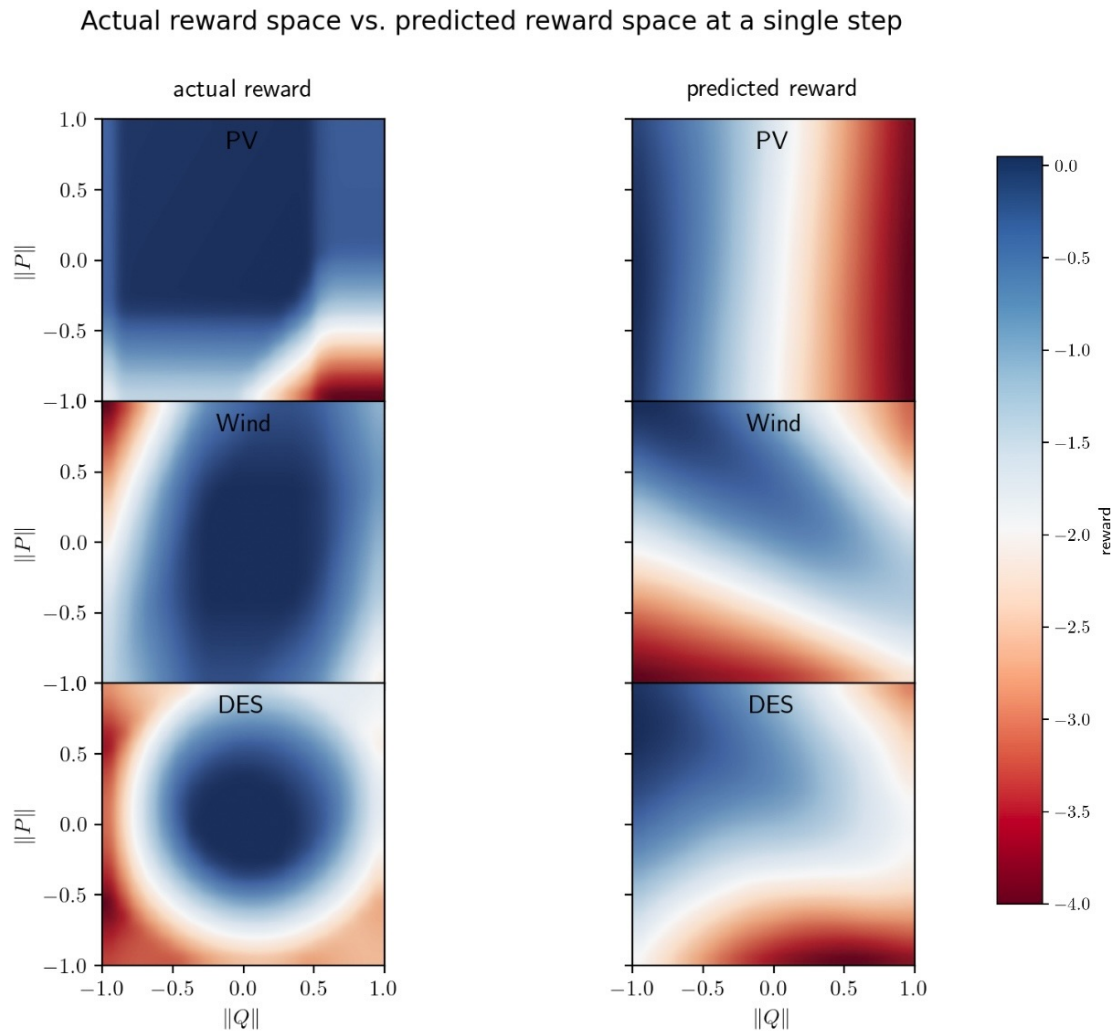


Figure 4.10: Showing the reward predicted by the DT and the actual reward from the environment for a subset of actions in the action space.

It can be seen that the maxima of the predicted rewards and the actual rewards are dissimilar. The maxima of the predicted reward is close to the center, whereas the maxima of the predicted reward is close to the upper right corner. This indicates that the DT favors power production and charging of the DES unit. However, it should be noted that the figure only displays an extremely small subset of actions compared to the actual size of the action space.

4.4 Summary

Table 4.1: Training time on a Nvidia A100 80GB GPU and Intel Xeon E5-2660 v4 CPU averaged over 3 runs.

Models	Training time
SAC	7h 8m ± 38m
PPO	6h 58m ± 14m
DT_a	3h 14m ± 12m
DT_{ar}	3h 27m ± 8m

The training times shown in Table 4.1 are included as a useful comparison for replication. However, the shorter training times of the DT should not be interpreted as an advantage, its offline training so the overhead of the environment is only present in the evaluation. Furthermore, the majority of the training of the baseline models was CPU-bound due to the gym environment not being able to

utilize the GPU.

Table 4.2: The average sum of discounted rewards for 1000 steps, except for PSO_{perf} which was extrapolated. DT_a denotes the model only trained on predict action where DT_{ar} is trained to predict both action and reward.

Model	Discounted reward
SAC (dataset)	**-131 ± 9.321**
PPO	-159 ± 4.412
DT_a	-143 ± 11.673
DT_{ar}	-158 ± 12.112
DT_{ar} + PSO	-138 ± 8.754
PSO_{perf}(extrapolated)	-46.5

The performance of all the models is summarised in table 4.2. The best performance was achieved with the SAC as highlighted in the table. The extrapolation of the performance of the PSO with perfect knowledge is included as a reference to achievable scores in the gym. However, it should be noted that extrapolating the performance over 1k steps from only 96 data points is likely to introduce a large uncertainty.
Some key findings are that DT_a has 9.2% worse performance than the dataset. Moreover, it can be seen that the performance of adding PSO to DT_{ar} yielded an improvement of 12.6%.

Chapter 5

Discussions

5.1 PPO & SAC

When using reinforcement learning, specifically Proximal Policy Optimization and Soft Actor-Critic, for active power grid management, it is important to acknowledge the complexity and sensitivity of these algorithms to hyperparameter tuning and procedural changes. This recognition plays a fundamental role in forming the quality of datasets used for training models like the Decision Transformer.

The complexity of tuning these algorithms comes from their numerous hyperparameters and implementation details. For instance, PPO's performance is dependent on factors like vectorized architecture, weight initialization, the epsilon parameter in the Adam optimizer, learning rate conditioning, Generalized Advantage Estimation (GAE), mini-batch updates, normalization of advantages, clipped surrogate objective, and value function loss clipping. Each of these components can notably influence the algorithm's efficiency and the resulting policy's quality.

Similarly, SAC's performance depends on its own hyperparameters, including the temperature parameter that controls the trade-off between exploration and exploitation, the learning rates of the actor and critic networks, and the architecture of these networks. The fine-tuning of these parameters can significantly affect the agent's learning curve and the stability of training.

Based on this complexity, achieving optimal performance in a specific task like power grid management is difficult. There is a good chance that better performance could be achieved with more precise hyperparameter tuning or the integration of additional methods. These improvements could potentially lead to a higher-quality dataset for training the DT, which is key since the DT's performance is directly influenced by the quality of the dataset it is trained on.

Nevertheless, the training loss seen in figs 4.1a and 4.1b indicate stable training. Perhaps the best evidence of the models successfully being trained is the evaluation performance seen in 4.2, which has a substantial increase from the initial discounted reward and reaches a plateau where the deviation between subsequent evaluations is minimized.

A notable drawback in our approach is the comprehensive array of hyperparameters that is used. These were mainly selected based on heuristic methods, drawing from their demonstrated performance in existing literature [82]. This reliance on heuristic selection indicates a potential area for more systematic, practical optimization in future research.

However, it is important to mention that our thesis does not aim for optimal performance but rather focuses on the relative performance based on the available dataset. This approach is useful for academic research, where the primary goal is to explore and understand the capabilities of these algorithms, and their limitations, in a specific context. The information attained from this research contributes to the broader understanding of how RL can be effectively applied to complex systems like power grids.

5.2 Decision transformer

5.2.1 Model choice

The DT was chosen for its strong performance in tasks with complex reward dynamics like the space reward key-to-door gym highlighted in [71]. This model's capability to understand RL as a sequence prediction problem makes it suitable for managing the sequential and restrictive conditions of power grid operations.
It directly models the cumulative future rewards as a portion of the input, letting the model account for long-term consequences of actions in a way that is both effective and computationally efficient. This ability is particularly important in the context of power grid management, where actions taken at one timestep can have far-reaching impacts due to the system's cyclical patterns and the limitations imposed by transmission line ratings.
The DT can effectively handle temporal dependencies and multi-stage decision-making in power grid management because it uses a transformer-like architecture, which is especially proficient at understanding sequences of data. In power grid management, decisions at one point in time can significantly affect future states and decisions. The transformer's ability to analyze these sequences helps it estimate the best actions to take at each step, considering both current conditions and their effects on future grid stability and energy distribution. This makes it theoretically well-suited for the sequential nature of managing a power grid, however, unable to surpass the results of traditional RL algorithms in our simulations.

5.2.2 Generalization Across Grid Scenarios

A fundamental strength of the DT model is its ability to generalize from training scenarios to new environments. The model's success in adapting to the cyclical nature of the power grid, while complying to transmission regulations, indicates robust generalization capabilities. However, the model's performance variance during transitions between stages B and C implies a need for further refinement. Improving generalization could involve integrating more diverse scenarios during training or developing methods to better handle transitional dynamics.

5.2.3 Reward-to-go vs. causality

DTs utilize RL as a sequence modeling problem, essentially transforming RL into a form of sequence prediction. Here, the concept of 'reward-to-go', which is the total accumulated future rewards from a given state, plays a vital role.
One argument against 'reward-to-go' is that it might violate causality by implementing future rewards into the current decision-making process. However, in the context of DTs, 'reward-to-go' is used more as a guiding heuristic rather than a definitive future prediction. It is like setting a performance target based on historical data, which impact but does not determine, the actions taken by the model.
This approach heavily leans on the quality and nature of the data which is used to compute 'reward-to-go'. In cases where the data is not representative of the possible range of scenarios the model might encounter, the effectiveness of the DT could be reduced. Furthermore, the reliance on historical data to set these 'rewards-to-go' is not always practical, especially in dynamic environments where future rewards are unpredictable.
While 'reward-to-go' in Decision Transformers seemingly challenge traditional rules of causality in RL, it operates within a framework where future rewards are used as a reference point rather than a deterministic factor in decision-making. This refined approach allows DTs to utilize the strength of transformer models for sequence prediction while acknowledging and working within the inherent uncertainties of RL environments.

5.2.4 Decision transformer training

In evaluating the robustness of our Decision Transformer models, DT_a, and DT_{ar}, an important measure to take is their potential overfitting to the training dataset. Overfitting is generally not

a concern in RL, but since our model is reliant on an offline dataset, there is a chance the model is only doing behavior cloning, meaning that it copies the actions seen in the dataset without generalization. This is usually analyzed by comparing the model's performance with a behavior cloning algorithm. However, we opted for a simpler approach of using a holdout set.
For our analysis, we reserved a holdout set, making up 20% of the total data, to compare the training loss (L_t) with the validation loss (L_v). The reasoning behind using a holdout set is to have a dedicated portion of data that the model has never seen during training, which serves as a proxy for real-world, unseen scenarios.
Referring to 4.4a and 4.4b, the training and validation losses both show an exponential decay, which is a good sign indicating that the model is learning and improving over time. If the model were noticeably overfitted, we would expect the validation loss to either plateau or increase after a certain point, as the model's predictions would start to diverge from the actual outcomes in the holdout set.
Furthermore, the training procedure's periodic evaluations on the separate environment instances help secure that the policy remains general and applicable beyond the training dataset. It is also worth mentioning that an overfitted model would likely perform well on the training set but poorly on the validation set, which is not the case here.

5.2.5 Transition challenges

The DT model has some challenges during the transitional periods between stages A and B. These stages are characterized by significant shifts in demand and production that are critical to the day-to-day operations simulated in the model.
Stage A, representing the morning and evening commute times, involves a spike in demand for low renewable energy generation, particularly from wind sources. The DT agent, which is trained to maximize rewards, have some problems here due to the sudden change from the over-supply in Stage A to the under-supply in Stage B. This situation is intensified by the high demand at branch b_{5-2}, as seen in 4.6 due to electric vehicle (EV) charging, placing further strain on the system.
It can be seen from Fig. 4.7 that the DT charges the DES slower compared to what is done in the dataset in the lower demand hours with high wind in Stage A. It is also apparent that the DES is still charging; putting additional load on branch $b_2 - 5$ which leads it to exceed its rating, resulting in high penalties.
Similarly, Stage B transitions to a period where EV charging demand decreases, but industrial demand increases while residential demand dips. Although renewable energy production rises, the DT agent must navigate a complex balance between varying demands and the sudden influx of renewable energy.
These transitional periods present an apparent contrast from the more stable conditions in other parts of the simulation, requiring the DT to adapt quickly to the dynamic environment. The reward figures associated with the DT model exhibit a notable dip in performance during these times (06:00 to 09:00 and 16:00 to 19:00), as illustrated in 4.5. The pronounced fluctuations in the graph reflect the model's difficulty in optimizing decisions amidst the competing demands of the grid.
The scenario-specific challenges during these periods are indicative of a realistic grid management situation where demand can fluctuate greatly, and production from renewable sources can be unpredictable. Despite these struggles, the DT's ability to learn from sequences of actions and rewards is still beneficial. It allows the model to potentially capture and predict patterns over longer-term horizons, even if short-term transitions may introduce some problems.

5.2.6 Decision transformer as reward function

There is a notable discrepancy between the predicted reward and the actual reward, as depicted in Fig. 4.10. However, the actual reward represents the instantaneous reward for that step but does not take into account how it may affect future actions. For instance, not charging the DES will lead to a penalty in the future, as there will be a power deficit in branch b_{2-5}. Looking at Fig. 4.9b it is also seen that the 12:00 o'clock region is one where there is no difference between reward for DT$_a r$

and DT with PSO. Since the PSO takes the predicted action from DT$_a r$ as the initial global best action, this might indicate that no higher rewards were found in the PSO search space.

5.3 PSO

The decision to employ PSO is mainly due to its simplicity and efficiency in handling complex, non-differentiable problem spaces; which is a common characteristic in power system optimization. Compared to gradient-based optimizers, PSO does not require derivative information, making it more fitting for our project where the problem space involves non-linear and potentially non-differentiable dynamics. Traditional optimizers like gradient descent or quasi-Newton methods are not as effective here, as they are dependent on gradient information that is difficult to estimate in such environments [88]. Furthermore, PSO's ability to explore a broad search space with a swarm of particles helps in avoiding local optima, a well-known constraint in single-point search methods like hill-climbing algorithms.
However, PSO does come with drawbacks, for example, its inability to guarantee convergence to a global optimum and potential inaccuracies in high-dimensional spaces. Even when considering these limitations, its robustness in exploring complex landscapes and ease of implementation make PSO a fitting choice for our project's unique requirements, creating a very good balance between computational feasibility and the ability to navigate through a multifaceted optimization problem.

5.3.1 Limitations of using PSO

The evaluation of the PSO performance gives a perfect reward function, seen in Fig. 4.9a shows that the PSO is able to improve upon an additional inferior action. This is especially apparent when comparing the extrapolated accumulated reward in Table 4.2, with the other models. However, it is important to note that the extrapolation of PSO's performance from a limited number of 96 data points can be misleading. The uncertainties in this process may lead to an overestimation of the optimization capabilities, particularly over a long operation horizon like 1k steps.

5.3.2 Reward Function

When designing a reward function for a RL environment it is important that this function does not only lead the agent toward the target goal but also embeds an understanding of the operational limitations and realities of the system that is being managed.
Our findings analyzing the agents' actions, as shown in 4.8 indicate that the reward function of the environment is not complex enough to encourage the desired behavior. To address this, we propose incorporating a penalty in the reward function for unrealistic power requests, aiming to align the agent's actions with the realistic constraints of power generation sources.
The reason for choosing this modification is twofold. Firstly, it addresses a habit observed in the agent's behavior, where it tends to request large amounts of power from renewable sources, without considering their actual availability, as shown in 4.8. This pattern indicates that the agent was not sufficiently factoring in auxiliary vectors, which provide information about available power, into its decision-making process. By penalizing unrealistic power requests, the reward function incentivizes the agent to gain a more nuanced understanding of the power generation capabilities and limitations of various sources.
Secondly, this penalty aims to minimize the difference between requested power and the actual power that is delivered, developing a more efficient energy management. In a scenario where a solar plant has a maximum capacity (P_{max}) of 20 MW but only 4 MW is available, penalizing a request for 20 MW encourages the agent to readjust its requests based on real-time availability.

5.4 Generalization abilities of the network

The generalization capabilities of our network model, especially in adapting to novel network architectures, continue to be a subject of speculative analysis. Although our thesis did not explicitly focus

on model generalization across varying grid topologies, it is important to recognize this feature for practical applications. Grid topologies are dynamic, and frequently changing due to maintenance, faults, or developmental upgrades. Thus, a model's ability to adapt to these changes is of significant importance for its real-world applicability.

It is hypothesized that the model at present might have some challenges with novel network structures, given its training on a specific grid configuration. In real-world applications, it is not enough for a power grid model to only understand but also adapt to these changes efficiently. The inability to do so could lead to suboptimal decision-making, making the model less efficient and reliable.

To deal with this limitation, future improvements of the model could be integrating additional information about grid topology into its learning model. This would mean expanding the context window of the model to also include dynamic topological data, enabling it to get an understanding of different grid structures. Such an approach could both improve the model's responsiveness to changes and also increase its predictive accuracy in various operational scenarios.

However, it's important to note that incorporating such topological flexibility will make the model much more complex. It would need the model to process a more diverse range of data inputs and likely, recalibrate its decision-making algorithms to consider different grid configurations. Despite these challenges, increasing the model's generalization capabilities is vital for its practical application in real-world grid management, where adaptability is as crucial as accuracy and efficiency.

Chapter 6

Conclusions

This thesis first developed an offline dataset designed for RL training, specifically addressing active network management with the implementation of intermittent renewable energy sources. The dataset was created through the evaluation and training of two state-of-the-art models, SAC and PPO. SAC showed a better performance of the two networks and was thus used to develop the offline dataset, which served as a teacher network for the DT.
A key focus of this research was the implementation and testing of two configurations of the DT. One DT_a was designed to predict actions based on trajectory contexts, while the other DT_{ar} also included reward prediction in the given states. Even though the DT models did not surpass the dataset's set performance benchmarks. The DT_a showed similar results with only a decrease in performance of 9.2% compared to SAC.
This thesis also introduces the use of PSO with the DT_{ar} model. This approach utilized the DT's reward prediction ability as a reward function, refining the decision-making process. Interestingly, training the DT to predict both rewards and actions showed a slight decrease in performance, the implementation of PSO notably increased the overall effectiveness of the model by 12.6%. This combination of DT and PSO underlines the potential of combining traditional optimization techniques with modern predictive models to tackle complex grid management challenges, marking a potential direction for future research in this field.

6.1 Future work

As discussed in subsection 5.3.2, future work should explore changing the reward function of the environment to see if this improves performance allowing for more complex topologies to be tested. To get closer to real-world operations, the environment and agent need to consider the monetary cost of actions such as curtailment and energy prices. In this case, operation costs should determine the performance of the agent.

Bibliography

[1] Jeffrey Taft. "The intelligent power grid." In: *IBM* (2006), pp. 1–5.

[2] Chengshan Wang et al. "Modeling and optimal operation of community integrated energy systems: A case study from China." In: *Applied energy* 230 (2018), pp. 1242–1254.

[3] Quentin Gemine, Damien Ernst, and Bertrand Cornélusse. "Active network management for electrical distribution systems: problem formulation, benchmark, and approximate solution." In: *Optimization and Engineering* 18 (2017), pp. 587–629.

[4] Massimiliano Manfren, Paola Caputo, and Gaia Costa. "Paradigm shift in urban energy systems through distributed generation: Methods and models." In: *Applied energy* 88.4 (2011), pp. 1032–1048.

[5] KC Divya and Jacob Østergaard. "Battery energy storage technology for power systems—An overview." In: *Electric power systems research* 79.4 (2009), pp. 511–520.

[6] Xi Fang et al. "Smart grid—The new and improved power grid: A survey." In: *IEEE communications surveys & tutorials* 14.4 (2011), pp. 944–980.

[7] Thomas Ackermann, Göran Andersson, and Lennart Söder. "Distributed generation: a definition." In: *Electric power systems research* 57.3 (2001), pp. 195–204.

[8] Robert H Lasseter. "Microgrids." In: *2002 IEEE power engineering society winter meeting. Conference proceedings (Cat. No. 02CH37309)*. Vol. 1. IEEE. 2002, pp. 305–308.

[9] Thomas Morstyn, Branislav Hredzak, and Vassilios G. Agelidis. "Control Strategies for Microgrids With Distributed Energy Storage Systems: An Overview." In: *IEEE Transactions on Smart Grid* 9.4 (2018), pp. 3652–3666. DOI: 10.1109/TSG.2016.2637958.

[10] Florin Capitanescu et al. "Assessing the potential of network reconfiguration to improve distributed generation hosting capacity in active distribution systems." In: *IEEE Transactions on Power Systems* 30.1 (2014), pp. 346–356.

[11] Nic Lutsey, Peter Slowik, and Lingzhi Jin. "Sustaining electric vehicle market growth in US cities." In: *International Council on Clean Transportation* (2016).

[12] Marie Rajon Bernard, Dale Hall, and Nic Lutsey. "Update on electric vehicle uptake in European cities." In: *International Council on Clean Transportation: Washington, DC, USA* (2021).

[13] Sam Koohi-Kamali et al. "Emergence of energy storage technologies as the solution for reliable operation of smart power systems: A review." In: *Renewable and Sustainable Energy Reviews* 25 (2013), pp. 135–165. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2013.03.056. URL: https://www.sciencedirect.com/science/article/pii/S1364032113002153.

[14] Mehdi Ferdowsi. "Vehicle fleet as a distributed energy storage system for the power grid." In: *2009 IEEE Power & Energy Society General Meeting*. 2009, pp. 1–2. DOI: 10.1109/PES.2009.5275495.

[15] Manuel Götz et al. "Renewable Power-to-Gas: A technological and economic review." In: *Renewable energy* 85 (2016), pp. 1371–1390.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] Simon Gill, Ivana Kockar, and Graham W Ault. "Dynamic optimal power flow for active distribution networks." In: *IEEE Transactions on Power Systems* 29.1 (2013), pp. 121–131.

[18] TP Imthias Ahamed, PS Nagendra Rao, and PS Sastry. "A reinforcement learning approach to automatic generation control." In: *Electric power systems research* 63.1 (2002), pp. 9–26.

[19] VP Jagathy Raj, EA Jasmin, and TP Imthias Ahamed. "Reinforcement Learning solution for Unit Commitment Problem through pursuit method." In: (2009).

[20] Mevludin Glavic, Damien Ernst, and Louis Wehenkel. "A reinforcement learning based discrete supplementary control for power system transient stability enhancement." In: *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications* 13.2 Sp. Iss. SI (2005).

[21] Renzhi Lu and Seung Ho Hong. "Incentive-based demand response for smart grid with reinforcement learning and deep neural network." In: *Applied energy* 236 (2019), pp. 937–949.

[22] Vishnuteja Nanduri and Tapas K Das. "A reinforcement learning model to assess market power under auction-based energy pricing." In: *IEEE transactions on Power Systems* 22.1 (2007), pp. 85–95.

[23] EA Jasmin, TP Imthias Ahamed, and VP Jagathy Raj. "Reinforcement learning approaches to economic dispatch problem." In: *International Journal of Electrical Power & Energy Systems* 33.4 (2011), pp. 836–845.

[24] John G Vlachogiannis and Nikos D Hatziargyriou. "Reinforcement learning for reactive power control." In: *IEEE transactions on power systems* 19.3 (2004), pp. 1317–1325.

[25] Elizaveta Kuznetsova et al. "Reinforcement learning for microgrid energy management." In: *Energy* 59 (2013), pp. 133–146.

[26] Damien Ernst et al. "Reinforcement learning versus model predictive control: a comparison on a power system problem." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2008), pp. 517–529.

[27] Magdi S Mahmoud, Nezar M Alyazidi, and Mohamed I Abouheaf. "Adaptive intelligent techniques for microgrid control systems: A survey." In: *International Journal of Electrical Power & Energy Systems* 90 (2017), pp. 292–305.

[28] José R Vázquez-Canteli and Zoltán Nagy. "Reinforcement learning for demand response: A review of algorithms and modeling techniques." In: *Applied energy* 235 (2019), pp. 1072–1089.

[29] Csaba Szepesvári. "Algorithms for reinforcement learning." In: *Synthesis lectures on artificial intelligence and machine learning* 4.1 (2010), pp. 1–103.

[30] Jingda Wu et al. "Continuous reinforcement learning of energy management with deep Q network for a power split hybrid electric bus." In: *Applied energy* 222 (2018), pp. 799–811.

[31] Jingyi Zhang et al. "Deep reinforcement leaming for short-term voltage control by dynamic load shedding in china southem power grid." In: *2018 International joint conference on neural networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[32] Tomah Sogabe et al. "Smart grid optimization by deep reinforcement learning over discrete and continuous action space." In: *2018 IEEE 7th World Conference on Photovoltaic Energy Conversion (WCPEC)(A Joint Conference of 45th IEEE PVSC, 28th PVSEC & 34th EU PVSEC)*. IEEE. 2018, pp. 3794–3796.

[33] Daniel O'Neill et al. "Residential demand response using reinforcement learning." In: *2010 First IEEE international conference on smart grid communications*. IEEE. 2010, pp. 409–414.

[34] Roberto Rocchetta et al. "A reinforcement learning framework for optimal operation and maintenance of power grids." In: *Applied energy* 241 (2019), pp. 291–301.

[35] Øystein Rognes Solheim, Boye Annfelt Høverstad, and Magnus Korpås. "Deep reinforcement learning applied to Monte Carlo power system reliability analysis." In: *2023 IEEE Belgrade PowerTech*. IEEE. 2023, pp. 01–08.

[36] Haotian Zhang et al. “Deep Reinforcement Learning Based Active Network Management and Emergency Load-Shedding Control for Power Systems.” In: *IEEE Transactions on Smart Grid* (2023).

[37] Haochen Hua et al. “Optimal energy management strategies for energy Internet via deep reinforcement learning approach.” In: *Applied energy* 239 (2019), pp. 598–609.

[38] Mevludin Glavic, Raphaël Fonteneau, and Damien Ernst. “Reinforcement learning for electric power system decision and control: Past considerations and perspectives.” In: *IFAC-PapersOnLine* 50.1 (2017), pp. 6918–6927.

[39] Zidong Zhang, Dongxia Zhang, and Robert C Qiu. “Deep reinforcement learning for power system applications: An overview.” In: *CSEE Journal of Power and Energy Systems* 6.1 (2019), pp. 213–225.

[40] Statnett. *Introduction to Norwegian reserve markets*. Statnett SF. Sept. 15, 2023. URL: https://www.statnett.no/globalassets/05-dokumentlisteblokker/introduksjon-til-reserver/how-the-reserve-markets-works.pdf (visited on 05/04/2023).

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[42] Di Cao et al. “Reinforcement learning and its applications in modern power and energy systems: A review.” In: *Journal of modern power systems and clean energy* 8.6 (2020), pp. 1029–1042.

[43] Manoranjan Dash, Hua Liu, and Jun Yao. “Dimensionality reduction of unsupervised data.” In: *Proceedings ninth ieee international conference on tools with artificial intelligence*. IEEE. 1997, pp. 532–539.

[44] Kristina P Sinaga and Miin-Shen Yang. “Unsupervised K-means clustering algorithm.” In: *IEEE access* 8 (2020), pp. 80716–80727.

[45] Krzysztof J Cios et al. “Unsupervised learning: association rules.” In: *Data Mining: A Knowledge Discovery Approach* (2007), pp. 289–306.

[46] Martin L Puterman. “Markov decision processes.” In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[47] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons, 2007.

[48] Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[49] Wen Sun et al. “Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches.” In: *Conference on learning theory*. PMLR. 2019, pp. 2898–2933.

[50] Stephen Tu and Benjamin Recht. “The gap between model-based and model-free methods on the linear quadratic regulator: An asymptotic viewpoint.” In: *Conference on Learning Theory*. PMLR. 2019, pp. 3036–3083.

[51] Tingwu Wang et al. “Benchmarking model-based reinforcement learning.” In: *arXiv preprint arXiv:1907.02057* (2019).

[52] Richard S Sutton. “Dyna, an integrated architecture for learning, planning, and reacting.” In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[53] Thanard Kurutach et al. “Model-ensemble trust-region policy optimization.” In: *arXiv preprint arXiv:1802.10592* (2018).

[54] Yuping Luo et al. “Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees.” In: *arXiv preprint arXiv:1807.03858* (2018).

[55] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[56] Christopher JCH Watkins and Peter Dayan. “Q-learning.” In: *Machine learning* 8 (1992), pp. 279–292.

[57] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *nature* 518.7540 (2015), pp. 529–533.

[58] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[59] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning." In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.

[60] John Schulman et al. "Trust region policy optimization." In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[61] John Schulman et al. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[62] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[63] Ashish Vaswani et al. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[64] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[65] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization." In: *arXiv preprint arXiv:1607.06450* (2016).

[66] Denny Britz et al. "Massive exploration of neural machine translation architectures." In: *arXiv preprint arXiv:1703.03906* (2017).

[67] Ofir Press and Lior Wolf. "Using the output embedding to improve language models." In: *arXiv preprint arXiv:1608.05859* (2016).

[68] Jonas Gehring et al. "Convolutional sequence to sequence learning." In: *International conference on machine learning*. PMLR. 2017, pp. 1243–1252.

[69] Tom Brown et al. "Language models are few-shot learners." In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[70] Aditya Ramesh et al. "Zero-shot text-to-image generation." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831.

[71] L Chen et al. *Decision transformer: Reinforcement learning via sequence modeling. CoRR abs/2106.01345 (2021)*.

[72] Alec Radford et al. "Improving language understanding by generative pre-training." In: (2018).

[73] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method." In: *Neurocomputing* 5.4-5 (1993), pp. 185–196.

[74] John E Dennis Jr and Jorge J Moré. "Quasi-Newton methods, motivation and theory." In: *SIAM review* 19.1 (1977), pp. 46–89.

[75] Yuhui Shi. "Particle swarm optimization." In: *IEEE connections* 2.1 (2004), pp. 8–13.

[76] John D Head and Michael C Zerner. "A Broyden—Fletcher—Goldfarb—Shanno optimization procedure for molecular geometries." In: *Chemical physics letters* 122.3 (1985), pp. 264–270.

[77] Frans Van Den Bergh. *An analysis of particle swarm optimizers*. University of Pretoria (South Africa), 2001.

[78] Maurice Clerc and James Kennedy. "The particle swarm-explosion, stability, and convergence in a multidimensional complex space." In: *IEEE transactions on Evolutionary Computation* 6.1 (2002), pp. 58–73.

[79] Justin Fu et al. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*. 2020. arXiv: 2004.07219 [cs.LG].

[80] Antonin Raffin. *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo. 2020.

[81] Robin Henry. *Gym-ANM*. https://github.com/robinhenry/gym-anm. 2020.

[82] Robin Henry and Damien Ernst. "Gym-ANM: Reinforcement Learning Environments for Active Network Management Tasks in Electricity Distribution Systems." In: *CoRR* abs/2103.07932 (2021). arXiv: 2103.07932. URL: https://arxiv.org/abs/2103.07932.

[83] John W Simpson-Porco, Florian Dörfler, and Francesco Bullo. "Voltage collapse in complex power grids." In: *Nature communications* 7.1 (2016), p. 10790.

[84] DM Greenwood et al. "Frequency response services designed for energy storage." In: *Applied Energy* 203 (2017), pp. 115–127.

[85] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization." In: *arXiv preprint arXiv:1711.05101* (2017).

[86] Priya Goyal et al. "Accurate, large minibatch sgd: Training imagenet in 1 hour." In: *arXiv preprint arXiv:1706.02677* (2017).

[87] Jorge Sola and Joaquin Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems." In: *IEEE Transactions on nuclear science* 44.3 (1997), pp. 1464–1468.

[88] Alaa Tharwat and Wolfram Schenck. "A conceptual and practical comparison of PSO-style optimization algorithms." In: *Expert Systems with Applications* 167 (2021), p. 114430.

Appendix A

Datasheet A

A.1 State Vector

Table A.1: Verbose description of each element of the environment state vector

Element	Description	Range	Unit
s_0	Fossile active power generation	[0, 200]	MW
s_1	Residential active power consumption	[-10, 0]	MW
s_2	Solar farm active power production	[0, 30]	MW
s_3	Industrial active power consumtion	[-30, 0]	MW
s_4	Wind farm active power production	[0, 50]	MW
s_5	EV park active power consumtino	[-30, 0]	MW
s_6	Energy storage active power flow	[-50, 50]	MW
s_7	Fossile reactive power	[-200, 200]	Mvar
s_8	Residential reactive power	[-2, 2]	Mvar
s_9	Solar farm reactive power	[-30, 30]	Mvar
s_{10}	Industrial reactive power	[-6, 6]	Mvar
s_{11}	Wind farm active power	[-50, 50]	Mvar
s_{12}	EV park reactive power	[-6, 6]	Mvar
s_{13}	Energy storage reactive power	[-50, 50]	Mvar
s_{14}	Energy storage state of charge	[0, 100]	MWh
s_{15}	Solar farm available active power	[0, 30]	MW
s_{16}	Wind farm available active power	[0, 50]	MW
s_{17}	Time of day	[0, 95]	-

A.2 Action Vector

Table A.2: Verbose description of each element of the environment action vector.

Element	Description	Range	Unit
a_0	Wind farm active power set point	[0, 50]	MW
a_1	Solar farm active power set point	[0,30]	MW
a_2	Wind farm reactive power set point	[-50, 50]	Mvar
a_3	Solar farm reactive power set point	[-30, 30]	Mvar
a_4	Energy storage active power setpoint	[-50, 50]	MW
a_5	Energy storage reactive power setpoint	[-50, 50]	Mvar