Carl von Ossietzky
Universität
Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Environment Design for Learning the Optimal Power Flow With Reinforcement Learning

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften
der Carl von Ossietzky Universität Oldenburg
zur Erlangung des Grades und Titels

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation

von Herrn Thomas Wolgast

geboren am 11.07.1993 in Dannenberg (Elbe)

1. Gutachterin:	Prof. Dr.-Ing. Astrid Nieße Department für Informatik Carl von Ossietzky Universität Oldenburg
2. Gutachter:	Prof. Minghua Chen, Ph.D. School of Data Science The Chinese University of Hong Kong (Shenzhen)
Tag der Disputation:	23. Juni 2025

Abstract

The Optimal Power Flow (OPF) problem is one of energy research's most important constrained optimization problems. In recent years, neural networks and deep learning algorithms have emerged as a promising new approach to approximate the OPF, especially regarding solution speed and real-time capability. Deep Reinforcement Learning (RL) seems especially promising, considering that RL does not require ground-truth data nor domain-knowledge. Instead, the problem-specific domain knowledge is encapsulated in the RL *environment*, which serves as problem representation. However, the existing literature shows a lack of OPF benchmark environments and established methods to design optimal RL environments. To lay the foundation for reproducible and comparable RL-OPF research, this thesis first introduces *OPF-Gym*,[a] a Python framework for OPF environments, including five different OPF benchmark environments. Second, as a general methodology for RL environment design, this thesis proposes to consider environment design as an optimization problem by utilizing the Hyperparameter Optimization (HPO) framework and its existing optimization algorithms. When applied to the five benchmark environments, the HPO-based approach consistently outperforms a manually derived design from a pre-study regarding constraint satisfaction and optimization performance. Additionally, statistical tests show that multiple specific environment design decisions consistently outperform their respective alternatives across the five benchmark problems, suggesting some generality of the results on how to optimally design RL-OPF environments. Altogether, the proposed *OPF-Gym* framework is the most advanced benchmark for reproducible RL-OPF research, and the proposed HPO-based environment design methodology is the first general approach to automated RL environment design.

[a] `https://github.com/Digitalized-Energy-Systems/opfgym`

Zusammenfassung

Das Problem des Optimal Power Flow (OPF) ist eines der wichtigsten beschränkten Optimierungsprobleme in der Energieforschung. In den letzten Jahren haben sich neuronale Netze und Deep-Learning-Algorithmen als vielversprechender neuer Ansatz zur Annäherung des OPF herausgestellt, insbesondere hinsichtlich der Lösungsgeschwindigkeit und Echtzeitfähigkeit. Deep Reinforcement Learning (RL) scheint besonders vielversprechend zu sein, da RL weder Ground-Truth-Daten noch Domänenwissen erfordert. Stattdessen wird das problemspezifische Fachwissen in der RL-*Umgebung* gekapselt, die als Problemdarstellung dient. Die vorhandene Literatur zeigt jedoch einen Mangel an OPF-Benchmark-Umgebungen und etablierten Methoden zur Gestaltung optimaler RL-Umgebungen. Um die Grundlage für reproduzierbare und vergleichbare RL-OPF-Forschung zu schaffen, stellt diese Arbeit zunächst *OPF-Gym*[a] vor, ein Python-Framework für OPF-Umgebungen, das fünf verschiedene OPF-Benchmark-Umgebungen umfasst. Zweitens schlägt diese Arbeit als allgemeine Methodik für das Design von RL-Umgebungen vor, das Umgebungsdesign als Optimierungsproblem zu betrachten, indem das Hyperparameter-Optimierungs-Framework (HPO) und seine bestehenden Optimierungsalgorithmen genutzt werden. Bei der Anwendung auf die fünf Benchmark-Umgebungen übertrifft der HPO-basierte Ansatz durchweg ein manuell aus einer Vorstudie abgeleitetes Design hinsichtlich der Erfüllung von Beschränkungen und der Optimierungsleistung. Darüber hinaus zeigen statistische Tests, dass mehrere spezifische Design-Entscheidungen bei den fünf Benchmark-Problemen durchweg besser abschneiden als ihre jeweiligen Alternativen, was auf eine gewisse Allgemeingültigkeit der Ergebnisse hinsichtlich des optimal Designs von RL-OPF-Umgebungen hindeutet. Insgesamt ist das vorgeschlagene *OPF-Gym*-Framework der fortschrittlichste Benchmark für reproduzierbare RL-OPF-Forschung, und die vorgeschlagene HPO-basierte Methodik für das Umgebungsdesign ist der erste allgemeine Ansatz für das automatisierte RL-Umgebungsdesign.

[a] `https://github.com/Digitalized-Energy-Systems/opfgym`

Acknowledgement

I want to thank everyone who has supported me over the last six years and helped me to bring this doctoral project to a successful conclusion.
I am especially grateful to Prof. Astrid Nieße for her support and supervision throughout the whole project. She has always been supportive of new ideas, even allowing me to pursue a topic outside of her primary research focus, for which I am truly grateful.
Many thanks also to my second examiner, Prof. Minghua Chen, especially for traveling all the way to participate in my disputation.
Thanks to Prof. Oliver Kramer for chairing the examination board and for all the fruitful discussions in our office.
So many thanks to all my colleagues at EI/DES and PSI who have accompanied me over the years in Hannover and Oldenburg. Thank you for the discussions, your support, and the great time together, especially to Rico, Stephan F., Paul Hendrik, Emilie, Julia, Jens, Torge, Stephan B., Lasse, Sasha, Malin, and Nils.
Special thanks to Prof. Richard Hanke-Rauschenbach, who took the time to discuss the possibility of pursuing a doctorate with me one evening in Hannover. In the aftermath of that conversation, he introduced me to Astrid, which started my whole doctoral journey.
Thanks to Nika for her essential role in my disputation.
My biggest thanks go to my partner Verena. She always supported me with lots of love, but also gentle reminders to get the thing done. I do not know if I would have made it without you. I love you!
Thank you all so very much!

Contents

Acronyms

DDPG Deep Deterministic Policy Gradient. 14–17, 19–21, 30, 60, 70, 72, 83, 87, 89

DNN Deep Neural Network. xv, 1, 2, 10–12, 14, 15, 22, 29, 30, 32, 47, 56

DRL Deep Reinforcement Learning. 14–17, 20

EHV Extra-High-Voltage. 37

GNN Graph Neural Network. 20–22

HPO Hyperparameter Optimization. iii, v, xix, 17, 18, 24, 58–60, 63–65, 71, 76–78, 84, 85, 87, 88

HV High-Voltage. 53, 54, 56

L2RPN Learning to run a power network. 23

LV Low-Voltage. 37

MDP Markov Decision Process. xv, 12, 13, 20, 36

MINLP Mixed Integer Non-Linear Programming. 7

ML Machine Learning. xvi, xx, 1, 3, 4, 10, 17, 21–23, 26, 29–31, 33, 44, 56, 78

MLP Multi-Layer Perceptron. 10, 11, 87

MV Medium-Voltage. 52, 53, 56

OPF Optimal Power Flow. iii, v, xvii, xix–xxi, 1–4, 7–9, 19–23, 26, 27, 29, 31–35, 37–44, 47–53, 55–60, 65, 66, 68, 75–78, 81, 82, 89

POMDP Partially-Observable MDP. 31

PPO Proximal Policy Optimization. 19–21, 60

ReLU Rectified Linear Unit. 10

RES Renewable Energy Systems. 1, 9, 10, 21, 37, 54

RL Reinforcement Learning. iii, v, xv–xvii, xix–xxi, 2–4, 12–16, 19–27, 29–49, 51, 52, 55–60, 62, 64, 66, 69–73, 75–79, 81, 82, 89

RQ Research Question. 2–5, 29, 35, 37, 57, 75–77, 87

SAC Soft Actor-Critic. 14, 16–18, 20, 21, 30, 60, 71, 72, 83, 87, 89

TD Temporal Difference. 15

TD3 Twin-Delayed DDPG. 19–21

List of symbols

α	Temperature parameter in SAC. 16–18
β	Weight of the penalty in the reward function. 43
γ	Discount factor in RL. 12, 13, 15, 16
θ	Parameters of a DNN. 15–17
λ	One hyperparameter configuration. 17
Λ	Hyperparameter search space. 17, 18
μ	Mean of a Normal distribution. 45
π	The policy of an RL algorithm. 12–18
ρ	Initial state distribution in MDPs and RL. 12
σ	Standard deviation of Normal distribution. 45
τ	Soft target update parameter. 15
ψ	Weight of the objective function in the invalid case.. 43
Ω	Share of invalid solutions in test dataset. 59
A	Action space in RL. 12–14
a	Action in RL. 12–18, 42–44, 48, 49
B	Branches (lines and transformers). 51
$\mathbf{b}$	Bias vector. 11
b	Bias of a Neuron. 10
b	Brach index (lines and transformers). 51
$\sim c$	Estimated generalization error. 17, 18
$\mathcal{D}$	Finite dataset. 45
D	Replay buffer in RL. 16, 18
g	Generator index. 52–54

$\mathcal{H}$ Entropy. 16, 17
h Output of a single Neuron. 10, 11

I Buses. 50
i Bus index. 50

ΔJ Mean error of valid objective values compared to a baseline. 59
J Objective function. xvi, 13, 16, 42–44, 52–55, 59

l Load index. 55
l Loss in ML. 15, 17

N Number of samples. 59
N Noise function. 16
$\mathcal{N}$ Normal distribution. 45

$\mathbf{P}$ Active power vector. 52–55
P Active power. 48, 49, 51–56
P Penalty function. 42–44
P Transition probabilities in RL. 12
$\mathbf{p}$ Market price vector. 53–55
p Market price. 53, 55

$\mathbf{Q}$ Reactive power vector. 52–55
Q Reactive power. 51–54, 56
Q Action-value or Q-value in RL. 13–17

R Reward Function in RL. 12, 13
r Reward in RL. 12, 13, 15, 16, 18, 42–44

S Apparent power. 51, 52
S State space in RL. 12–14
s Environment state in RL. 12–18, 42–45, 49
s Storage system index. 54, 55

T Episode length in RL. 13, 16, 18
t Current step. 13–18, 55

$\mathcal{U}$ Uniform distribution. 45
U Voltage in a power grid. 50
u Control variable in the OPF. 7

V State-value in RL. 13

$\mathbf{W}$ Weight matrix of a neural network. 11
$\mathbf{w}$ Weight vector of a neural network layer. 10

$\mathbf{X}$ Input matrix. 11
X Observation space in RL. 52–55
$\mathbf{x}$ Input vector of a neural network. 10
x Observation in RL. 14
x System state variable in the OPF. 7

y Target value in RL. 15, 16

$\mathbb{E}$ Expectation of a random variable. 13, 15–17
$*$ Indicates optimality. 13, 14, 17, 59
$'$ Indicates a target network. 15

Related Publications

Within the Scope of this Work

The following publications directly contributed to this thesis and are either early results and findings or peer-reviewed dissemination of the final results of this work.

Learning the Optimal Power Flow: Environment Design Matters, 2024, first author [1]: This work investigates the importance of environment design for learning the OPF with RL. Various environment design options from the literature are discussed, implemented, and compared. The results demonstrate that the environment design can have a drastic impact on RL training performance. However, the experiments were performed for only two RL-OPF environments, from which we cannot derive general conclusions. Further, the approach is not suited to derive general recommendations on how high-performing environments should be designed. The results of this work mainly served as a pre-study for the proposed automated environment design in chapter 6 of this thesis.

Environment Design for Reinforcement Learning: A Practical Guide and Overview, 2024, first author (Whitepaper, not peer-reviewed) [2]: This work provides an overview of RL environment design, for which the literature is very limited. In the whitepaper, a first definition of RL environment design is given, the design categories reward, observation, action, episode, and data are identified, and various practical guidelines are provided on what to consider during RL environment design. Most of the recommendations are lessons learned during the work on this thesis.

A General Approach of Automated Environment Design for Learning the Optimal Power Flow, 2025, first author [3]: This work extends the pre-study in [1] and proposes automated RL environment design by applying the HPO framework to the problem. By the example of five different RL-OPF environments, the publication shows that the HPO-based approach outperforms the manually derived environment design from [1] and identifies various design decisions as especially relevant for overall performance, some of which were not discussed in literature before. The methodology and results have mainly been processed in chapter 6 of this thesis.

***OPF-Gym* Environment Framework + Software Paper, 2025, first author (version 1.0 published, paper submission planned) [4]:** This planned software paper intends to present the open-source *OPF-Gym* benchmark framework that was developed for this thesis. It demonstrates how *OPF-Gym* can serve as a RL-OPF performance benchmark, allows for almost arbitrary OPF formulations, and enables systematic environment design studies. This work will disseminate and supplement chapter 5 of this thesis.

Related to this work

While the previous publications directly contribute to this thesis, multiple related publications also resulted in valuable insights for this work.

Towards modular composition of agent-based voltage control concepts, 2019, first author [5]: This publication re-implements and combines multiple agent-based voltage control approaches from the literature to achieve modularity and plug-and-play capability. Two main lessons were derived from this work: First, instead of handcrafting voltage control rules, voltage control can be seen as a special case of the OPF problem, enabling us to utilize various existing algorithms and literature. Second, the re-implementation of existing strategies from the literature proved very challenging, emphasizing the importance of reproducibility of research outcomes. This way, the work on this publication motivated the step to the OPF problem and the creation of open-source benchmarks to improve reproducibility and comparability of research.

Reactive Power Markets: A Review, 2022, first author [6]: This work provides a comprehensive literature overview on reactive power markets. Reactive power markets are relevant for this work since they are commonly modeled as OPF problems. One of the benchmark RL-OPF problems presented in this work is a reactive power market problem (see 5.5.3), which was created based on the knowledge gained from the research done for this review paper.

Design and evaluation of a multi-level reactive power market, 2022, co-author [7]: This work originated from the supervision of a student's master's thesis. We propose a hierarchical reactive power market that enables multiple grid operators to solve the required OPF together without exchanging grid information. The general approach is transferable to non-reactive-power-market OPF problems and compatible with ML-based OPF approximations, as done in this work. It also demonstrates how conventional OPF solvers reach their limits for large-scale systems.

Towards reinforcement learning for vulnerability analysis in power-economic systems, 2021, first author [8]: In this work, an RL agent is trained to maximize profit on a reactive power market. The main purpose was to show how RL can be used to find harmful but profitable strategies in energy markets. However, since the market is modeled with an OPF, this scenario also provided a good example of a use case where an OPF was required to be solved millions of times. With conventional approaches, this is computationally very heavy and motivates this thesis in its attempt to create faster OPF solvers.

Approximating energy market clearing and bidding with model-based reinforcement learning, 2024, first author [9]: Building on the previous work, multi-agent RL was used to train multiple participants for optimal bidding behavior in an energy market environment. The energy market is represented by an OPF, which would be computationally heavy with conventional OPF approaches, as discussed before. Following the same approach as in this thesis, the OPF was solved with RL-trained neural networks. Further, domain knowledge was used to improve training performance. However, the OPF solving was not the research priority and was only used as a means to an end to speed up training of the multi-agent RL approach.

Ten Recommendations for Engineering Research Software in Energy Research, 2025, co-author [10]: This publication focuses on software engineering for energy research software and provides guidelines for energy research software developers. The *OPF-Gym* framework created for this thesis was developed and published with these guidelines in mind. Further, the lessons learned from the *OPF-Gym* development contributed to the writing of the paper, and vice versa.

1 Introduction

1.1 Motivation

Due to increasing penetration with Renewable Energy Systems (RES) and slow grid expansion, electric power grids are more and more operated close to their physical capabilities [11]. To compensate for a lack of grid expansion and save potentially billions of euros [12], the power system must be operated closer to its mathematical optimum. The general framework for optimizing power grid states is the Optimal Power Flow (OPF), which aims to determine the cost-minimal power flows, subject to various constraints. However, especially for more complex variants of the OPF, conventional solvers are very slow [13], which becomes a problem if OPF solutions are required in high frequency, for various different situations [8], or in real-time [14]. In practice, slow OPF solvers become an issue in two different scenarios: First, when operating a real-world power grid by repeatedly solving the OPF in real-time [11, 13]. Second, when the OPF needs to be solved in-the-loop over a long time horizon and/or various use cases [1, 8], for example, when researching long-term future energy scenarios with the help of large-scale simulations. Consequently, an increasing share of literature focuses on the development of fast and real-time capable OPF solvers, which can replace or augment the conventional approaches (see related work section 3).

One emerging approach to speed up the OPF is to use Machine Learning (ML) to train Deep Neural Networks (DNNs) to solve the OPF. DNNs are promising because, after successful training, only a series of matrix multiplications are required to map from the unoptimized grid state to the optimal actuator setpoints. That is computationally fast, easy to parallelize, and deterministic without any convergence issues, which are good preconditions for real-time capability. Additionally, large DNNs can deal very well with local minima [15], which remain a big problem for conventional solvers [11]. That trait makes it possible that DNN-based approaches may even outperform conventional solvers in the long-term.

DNNs can be trained to solve the OPF by using any of the three general ML paradigms: supervised learning, unsupervised learning, or reinforcement learning [16]. Supervised learning is the most frequently used literature approach for the OPF [17]. However, supervised

learning requires ground-truth data, which is a big disadvantage because it requires massive usage of conventional solvers for training data generation [13, 18]. Further, Huang et al. [16] demonstrated that non-globally-optimal training data may deteriorate training performance disproportionally. However, to generate training data for supervised learning, conventional solvers are required, which cannot guarantee globally optimal solutions in most cases [11].

Unsupervised learning usually aims to reveal structure in the training data, which is why it is not the obvious choice for solving an optimization problem. The existing approaches [16, 19, 20] utilize lots of domain knowledge, which can improve performance but limits the generality of the approaches.

Reinforcement Learning (RL) aims to learn an optimal mapping from state representations to actions by maximizing a reward signal from trial-and-error actions. Since the actuator setpoints in the power grid can be modeled as actions and the reward signal can represent the objective function and the constraints of the OPF, RL is a natural choice for the OPF [14, 21, 22].

The RL framework consists of two components: the agent and the environment. The environment is a representation of the problem, in this case, the OPF problem. Most domain knowledge is encapsulated in the environment. The agent, on the other hand, aims to find the optimal strategy to solve the given problem by interacting with its environment. Therefore, both components need to be explored to train DNNs to solve OPF problems. While there are several advanced approaches for the agent side of learning the OPF with RL, e.g. [23, 24, 25], the environment representation of the OPF problem remains completely unexplored. That is not only true for the RL-OPF literature but also for the RL literature in general [26], although it was shown several times that the environment design significantly impacts overall training performance [26, 27, 28, 29, 30], indicating an important research gap. Hence, the focus of this doctoral thesis will be on RL environment design for learning the OPF.

1.2 Research Questions

The goal of this thesis is to explore the importance of environment design for solving the OPF with RL. Hence the overall guiding Research Question (RQ) is:

General RQ: What is the importance of RL environment design for solving the OPF with RL, regarding both optimization performance and constraint satisfaction?

This general research question can be split into three subordinate research questions, as discussed in the following:

RQ1: OPF as an RL Problem Considering that the environment serves as an OPF problem representation, the first step is to understand the intricacies of the OPF as an RL problem. This is also necessary to understand in which use cases RL can, should, and should not be applied to the OPF problem:

> **RQ1:** What are the characteristics, difficulties, and chances of the OPF as an RL problem formulation?

Here, it is important to differentiate between different OPF variants and their specific requirements. Further, a comparison with established OPF solvers is required to understand where RL can become a useful tool in the toolbox. Finally, it is important to determine the particularities of the OPF as an RL problem since this is an important requirement for designing the RL environments and also choosing suitable RL algorithms later on.

RQ2: Benchmark Environments As discussed before, the OPF problem formulation is completely encapsulated in the RL environment. A common approach in ML research to make research reusable, reproducible, and comparable is to establish open benchmarks. For learning the OPF, such benchmarks are still mostly missing [17]. However, considering that the environment representation of the OPF is the main focus of this thesis, such benchmarks are strictly required. Hence RQ2:

> **RQ2:** How should a benchmark framework for OPF environments look like that ensures reproducible and comparable research with fixed benchmarks but also degrees of freedom for systematic environment design?

Here, it is important to deal with two conflicting goals. On the one hand, we need a static benchmark so that research publications using that benchmark are comparable with each other over various use cases and possibly multiple years. On the other hand, it is unclear in advance what the optimal environment design should look like. In other words, the benchmark environments are not only a remedy for research but also a research subject itself. Hence, when creating an OPF benchmark environment, it is important to differentiate between the OPF problem to solve, which needs to stay unchanged for comparability, and the specific RL environment design, which is subject to research and a degree of freedom for environment design, which will be focus of the next RQ.

RQ3: Environment Design As discussed before, we need to find an optimal environment representation of the OPF problem that is beneficial for overall performance.

> **RQ3:** How to formulate OPF problems as RL environments for maximum learning performance, including constraint satisfaction, optimization performance, and learning speed?

Considering that environment design is not only an unsolved problem for the OPF but in RL research in general, it is important to develop a general methodology that is broadly applicable to all kinds of RL environment design problems. Further, automated optimization-based approaches should be preferred over manual ones to reduce engineering effort, but also to achieve better performance and for better integration with existing RL workflows.

1.3 Thesis Structure

This thesis is structured as follows. Chapter 2 provides the fundamentals to the OPF problem and the RL framework. Chapter 3 identifies multiple research gaps in the literature by discussing related works that apply RL and ML to the OPF, relevant RL environment frameworks and benchmarks, and general literature on RL environment design. Afterward, the thesis is structured along the previously presented Research Questions. Chapter 4 investigates RQ1 by discussing the overall characteristics of the OPF as an RL problem. Chapter 5 tackles RQ2 by presenting the developed RL-OPF framework *OPF-Gym*, which contains fixed benchmarks but also degrees of freedom for systematic environment design. Chapter 6 considers RQ3 by presenting and verifying an automated optimization-based approach to RL environment design. Finally, chapter 7 provides a discussion of this thesis' approach, its overall contribution, and the answers to the earlier RQs. Chapter 8 ends the thesis with an overarching conclusion.

Figure 1.1 shows the resulting thesis structure. It demonstrates how chapters 4 to 6 directly tackle the RQs 1-3, respectively, how chapters 2 to 5 all serve as a foundation to create the *OPF-Gym* framework, and how *OPF-Gym* is used to implement and evaluate the environment design methodology in chapter 6.

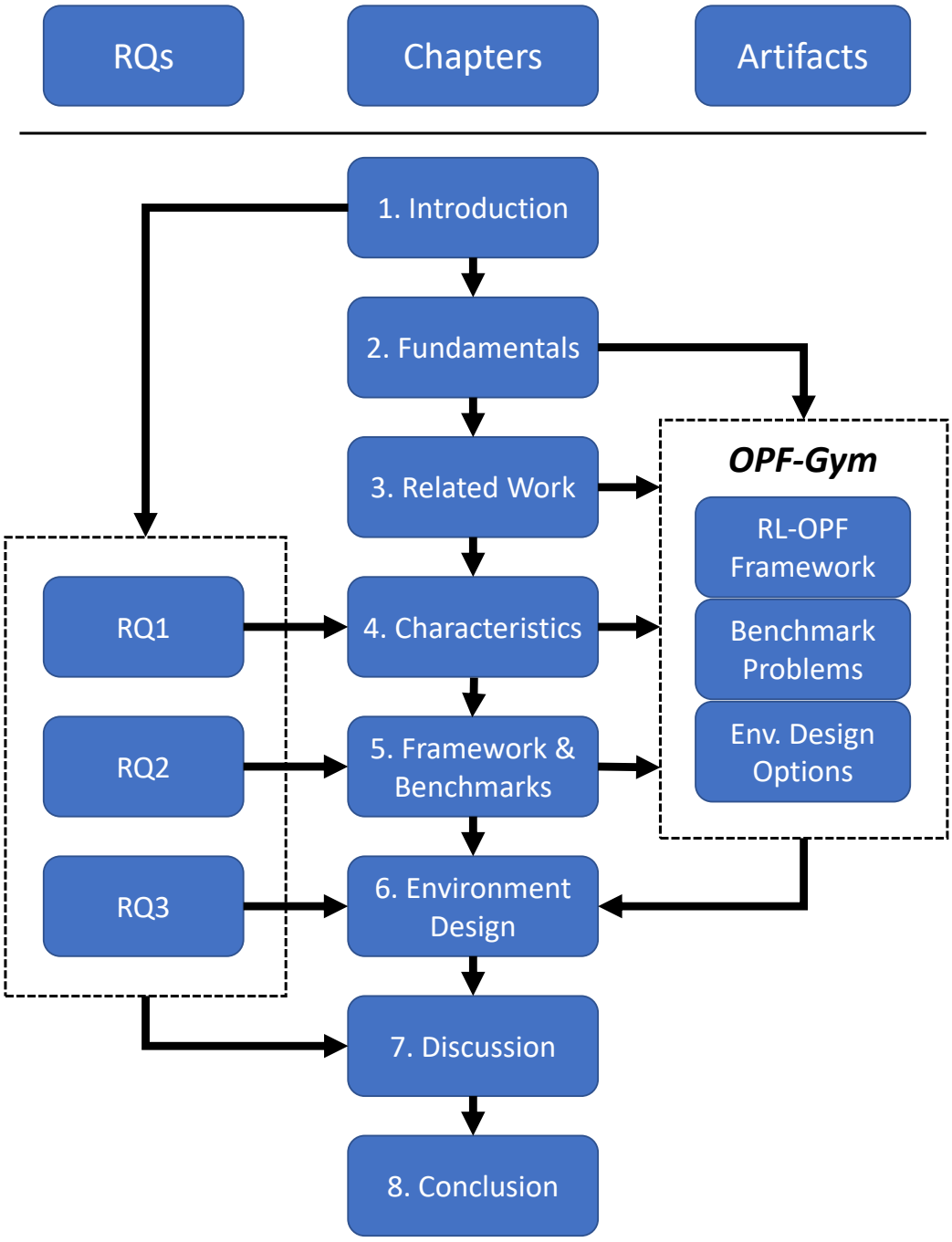


Figure 1.1: Overall thesis structure, including RQs, chapter structure, and resulting research artifacts.

2 Fundamentals

2.1 Optimal Power Flow

The OPF is an umbrella term for optimization problems that include the power system equations in the constraints. In its general form, the OPF can be represented in the following way: [11]

$$\begin{aligned} \min \quad & f(u, x) \\ \text{s.t.} \quad & g(u, x) = 0 \\ & h(u, x) \leq 0 \end{aligned} \tag{2.1}$$

with the objective function f, the equality constraints g, the inequality constraints h, the control variables u, and the non-controllable state variables x. In its general form, the OPF is a large-scale, non-linear, non-convex optimization problem that often contains both discrete and continuous control variables [11]. Therefore, without simplifying assumptions, it requires an Mixed Integer Non-Linear Programming (MINLP) formulation. One very common simplification is the DC-OPF, which is a linearization of the general AC-OPF case [11]. However, this work will only focus on the general AC-OPF since overly simplified OPF formulations are considered to be harmful to grid efficiency nowadays [12].

Because of its very general definition, the OPF can be used to model and solve a variety of problems in power system optimization, for example:

- **Constrained Economic Dispatch** to meet the active power demand at minimal costs without inducing any constraint violations [11] .
- **Optimal Voltage Control** to minimize voltage deviations and active power losses in the system [31]. Usually formulated as an optimal reactive power dispatch [32].
- **Reactive Power Markets** to find the cost-minimal reactive power procurement, usually to perform voltage control [6].
- **Topology Control** to optimize power flows by adjusting the grid topology with binary switches as actuators [32].

- **Unit Commitment** to schedule future generation subject to system constraints and operating constraints of generators [11].

Additionally, all these problems can also be combined to be solved at the same time. However, it is more common to treat them as separate problems [33]. Comprehensive lists of typical objectives, constraints, and control variables can be found in [11].

In addition to the various kinds of possible objective functions, constraints, and control variables, an increasing amount of requirements is placed upon OPF solvers, which results in more advanced and difficult-to-solve OPF variants:

- **Stochastic OPF**: The stochastic OPF considers the increasing uncertainty of the power system state by incorporating random variables, for example, to consider (potentially erroneous) generation forecasts and stochastic loads. [34]

- **Security-Constrained OPF**: In addition to the standard constraints, the security-constrained OPF aims to ensure constraint satisfaction in case of contingencies. That is often done using the $n-1$ criterion, which means that all constraints must be satisfied for all potential single contingencies in addition to the base case. The objective is still to minimize the objective function of the base case. [32]

- **Risk-Based OPF**: The strict enforcement of security constraints may lead to significant additional costs, even though some contingencies are very unlikely to happen. The risk-based OPF incorporates contingency probabilities to find a trade-off between system security and efficiency, which is also common practice for transmission system operators. [32]

- **Multi-Stage OPF**: Consideration of ramp constraints, line switching, or storage system actuators often requires the OPF to be solved over multiple time steps. In that case, the constraints must be satisfied in all time steps. [34]

- **Distributed OPF**: Usually, the OPF is seen as a centralized planning tool [11]. However, an increasing amount of literature aims to solve the OPF in a distributed fashion for incorporation of multiple players (e.g. multiple grid operators), for improved resilience, or better computational feasibility in large-scale problems [34].

- **Real-Time OPF**: The real-time OPF aims to ensure real-time capability, which is required for intraday markets and when the OPF solution is intended to be used to directly control and optimize the power system. Important requirements for the real-time OPF are fast convergence and that all intermediate solutions are valid. [11]

Again, all these more difficult cases can be combined, which makes the OPF problem arbitrarily difficult to solve, especially when applied to large-scale systems. Stott and Alsac [33] even conclude that useful theories of global optimality will likely never happen for non-trivial and realistic OPF problems.

Solution methods to the OPF can be classified into three general categories:

- **Classical (deterministic) optimization** like the interior point method, Newton's method, or the Simplex Method are used conventionally. They are fast and can provide optimality guarantees in case of convexity. However, they have difficulties dealing with discrete variables and non-convex problems, require continuity and differentiability of the objective function, and are sensitive to initial conditions. [11]
- **Meta-heuristics** like evolutionary algorithms or particle swarm optimization are able to deal with most of these problems. They can theoretically converge to globally optimal solutions (given infinite time), can easily deal with discrete variables, and do not rely on continuity or differentiability of the objective function. However, they are significantly slower, often struggle with local optima, and have difficulties enforcing constraints. [35]
- **Data-driven** methods are the most recent development for solving the OPF. They have similar characteristics to meta-heuristics. They are well suited for stochastic problems and are often able to solve problems where the exact problem definition is unknown. They require a single computationally heavy training phase before deployment. After that, they are significantly faster than both previously mentioned approaches. [36]

Further, different categories can be combined into hybrid approaches, which aim to combine the advantages of multiple methods [35].

Overall, there is no single solution yet existing to solve the entirety of OPF variants all at once. Instead, OPF algorithms are usually tailored for specific use cases [33, 11]. Solving the OPF became even more difficult in recent years for various reasons. The increasing share of RES results in significantly more degrees of freedom and more stochasticity. The same is true for the increasing demand response capabilities [36]. Additionally, RES and demand response require stronger incorporation of the distribution systems, which were often neglected for OPF research in the past [36]. Further, market clearing happens at increasingly shorter time scales [11]. Finally, new devices like FACTS devices are more difficult to incorporate and complicate the OPF application [11].

Additionally, the OPF becomes more and more important for grid operation and simulation. The OPF is increasingly central to the secure operation of energy markets [33].

Further, the expansion of RES and other distributed energy resources outpaces the required grid expansion in most countries, which necessitates more close-to-optimal operation of the existing infrastructure [11].

2.2 Machine Learning and Deep Learning

The following overview on ML is mainly based on the work of Géron [37], except where explicitly referenced otherwise.

ML is the science of building machines that can learn from data to fulfill some task. ML is usually classified into three general paradigms [38]: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the training dataset is labeled, containing inputs and desired outputs. The goal is to find a function that maps from inputs to outputs so that it generalizes to unseen data. In unsupervised learning, the dataset consists of only inputs, i.e., it is unlabeled. The goal is usually to uncover structure and relationships within the data, for example, correlations, outliers, or lower-dimensional representations of the input data. In contrast to supervised and unsupervised learning, reinforcement learning does not have a pre-defined dataset but actively collects data by trial-and-error interaction with an environment that represents the problem to solve. The goal is to maximize performance regarding the given problem. Reinforcement learning will be described in more detail in section 2.3.

For complex problems with high-dimensional input and output data, modern ML algorithms often utilize DNNs for function approximation. DNNs are inspired by the human brain and work as general function approximators [39]. The most basic DNN architecture is the Multi-Layer Perceptron (MLP), which will be presented in the following.

The MLP consists of multiple sequential layers of so-called *Neurons*. A Neuron receives an input vector, performs a non-linear differentiable operation on that input, and outputs a scalar value. The Neuron computes a weighted sum of the inputs $\mathbf{x}$, adds a bias b, and applies an activation function to the output.

$$h(\mathbf{x}) = \text{activation}(\mathbf{x}^\intercal \mathbf{w} + b) \tag{2.2}$$

A Neuron can be trained by tweaking the weights $\mathbf{w}$ and the bias b to move the output in the desired direction by using gradient descent. The activation function is required to learn non-linear functions. It needs to be differentiable. Common activation functions are the sigmoid function, the hyperbolic tangent function *tanh*, the Rectified Linear Unit (ReLU), and variations of it [40].

A full MLP consists of multiple sequential layers of Neurons, as visualized in Figure 2.1.

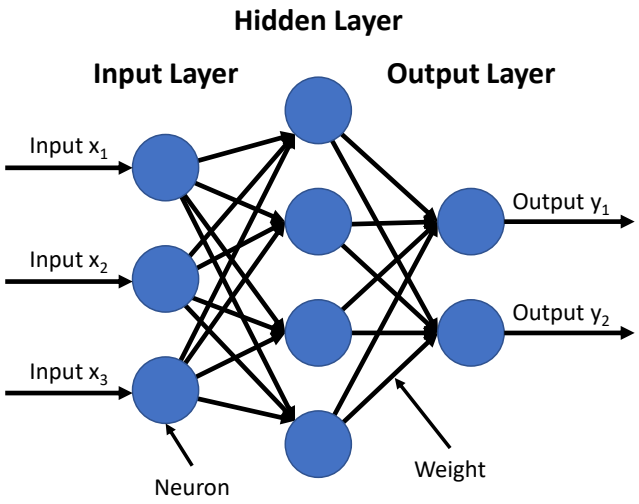


Figure 2.1: Visualization of an MLP neural network.

The first layer is called the *input layer*, followed by multiple *hidden layers*, and a single *output layer*. This stacked architecture with potentially billions of weights is fully differentiable and can serve as a general, continuous, non-linear function approximator. Computing the output of an MLP works equivalently to that of the Neurons it consists of, only replacing vector notation with matrix notation:

$$h\left(\mathbf{X}\right) = \text{activation}\left(\mathbf{XW} + \mathbf{b}\right) \tag{2.3}$$

This calculation can be performed sequentially layer by layer to compute the output of the DNN, always using the output of the previous layer as input to the next one. Since every single operation is differentiable, the whole network is differentiable and can be trained with gradient descent and backpropagation.

Backpropagation is the algorithm to compute the gradients of a multi-layered DNN. It consists of a Forward-Pass, a Backward-Pass, and a gradient descent step: [37]

1. Forward-Pass: The output is computed for the given input data and all intermediate results of all Neurons are stored.

2. Backward-Pass: The error of the output relative to some target output is computed by using a problem-specific loss function. Then, a derivation is performed to compute the gradients of the weights with regard to the loss. That happens by going through the layers backwards using the *chain rule* of derivatives. In other words, the error is backpropagated through the network, hence the name backpropagation.

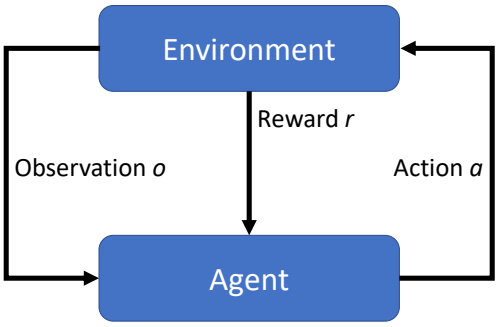


Figure 2.2: The Reinforcement Learning framework.

3. Gradient-Descent: Finally, a gradient descent step is performed by using the computed gradients to optimize all the weights for loss minimization.

These three steps together form one backpropagation training step. Training steps are usually performed mini-batch-wise by performing all operations for n inputs together and averaging the loss. This enables the efficient use of matrix operations and parallelization and also smoothes the training process with regard to outliers. Due to the many matrix operations, DNN training profits significantly from GPU usage instead of CPUs.

2.3 Reinforcement Learning

The following overview on RL is mainly based on the standard work of Sutton and Barto [41], except where explicitly referenced otherwise.

In RL, an agent observes its environment, manipulates that environment's state s with actions a, and receives a scalar reward r that indicates the quality of the action in the given state. This happens sequentially, and the goal is to learn a policy that maximizes the total sum of rewards. The interaction process is visualized in Figure 2.2.

The following paragraphs introduce multiple key components of modern RL algorithms and environments, which will often be referenced later in this thesis.

Markov Decision Process RL problems are usually formalized as incompletely known Markov Decision Processes (MDPs). MDPs are defined by a tuple $(S, A, \gamma, R, P, \rho)$, with the state space S, the action space A, the discount factor $0 \leq \gamma \leq 1$, the reward function R, the state transition probabilities P, and the distribution of initial states ρ. The goal is to find an optimal policy $\pi(a|s)$ that maps from states s to probability distributions over

actions a in a way that maximizes the expected sum of discounted rewards, called *return*.

$$J(\pi) = \mathbb{E}\left[\sum_{t}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right] \tag{2.4}$$

where $\mathbb{E}[\cdot]$ denotes the expectation of a random variable. The discount factor γ is required here to prioritize short-term rewards over long-term rewards and also to prevent an infinite expected return for infinite-horizon problems.[1]

However, in RL, the agent does not have full access to all information in the MDP definition. Usually, we assume that the agent knows the action space A, the state space S, and the discount factor γ but does not know the environment dynamics or the reward function. Therefore, the agent must collect data about rewards and dynamics by trial-and-error interaction with the environment. In other words, the agent *explores* its environment. Since exploration requires selecting non-optimal actions, RL inherently faces an exploration-exploitation trade-off. The agent aims to maximize reward (exploitation) but needs to perform non-optimal actions (exploration) to understand the environment dynamics required for reward maximization.

Value Function An important concept in RL is the value function, which is used to estimate the expected return of a given state or action. This way, value functions can be used for decision-making and learning. The *state-value* function V is the expected return when applying policy π from state s onwards.

$$V_\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s\right] \;\forall\, s \in S \tag{2.5}$$

Closely related, the *action-value* function or Q-value function denotes the expected return of taking action a in state s and following policy π thereafter.

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s, a\right] \;\forall\, s \in S \text{ and } \forall\, a \in A \tag{2.6}$$

Both kinds of value functions can be estimated from experience following some existing policy. The optimal policy π^* is defined such that it results in optimal value functions V^* and Q^*.

$$V^*(s) = \max_\pi V_\pi(s) \;\forall\, s \in S \tag{2.7}$$

[1] It is common practice to use a discount factor even in episodic problems with finite episode lengths of T.

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a) \; \forall \; s \in S \text{ and } \forall \; a \in A \tag{2.8}$$

States and Observations A common assumption for RL environments is that the environment's state fulfills the *Markov* property, which means that the current state s_t provides all information required for optimal decision making and value estimation.

Until now, we assumed that the agent receives the environment's state for decision making. However, in practice, the agent often only receives an observation x, which is a non-perfect representation of the environment's state. For example, the observation can be redundant, incomplete, or noisy [2]. In the ideal case, the observation fulfills the Markov property as well to allow for optimal decision-making. If that is not the case, the RL problem is partially observable and requires special RL algorithms since most standard RL algorithms assume full observability.

On-Policy vs. Off-Policy In RL, we can differentiate between two different classes of algorithms: On-policy and off-policy. On-policy algorithms can only be trained with data that was created by the current policy, which means that data needs to be deleted as soon as the policy changes. In contrast, off-policy algorithms can be trained with data resulting from any policy. This way, data can be stored and re-used even if the policy changes. In general, off-policy algorithms are considered to be more sample-efficient, i.e., they require fewer environment interactions [42]. In contrast, on-policy algorithms are usually computationally cheaper on the algorithm side. Therefore, off-policy algorithms should be preferred if the environment is computationally heavy, and on-policy algorithms if vice versa.

Deep Reinforcement Learning Conventional RL algorithms like Q-learning [43] can deal with discrete observation and action spaces but not with continuous or high-dimensional spaces. Modern RL algorithms utilize DNNs for function approximation, for example, of the value function or the policy. These algorithms are then called Deep Reinforcement Learning (DRL) algorithms and can deal with high-dimensional and continuous observation spaces [44] and action spaces [45].

2.4 Standard Reinforcement Learning Algorithms

The following section will present two standard DRL algorithms used in this work, namely, the DDPG and the SAC algorithm. Both can learn policies with continuous actions and

are sample-efficient off-policy RL algorithms.

2.4.1 Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm by Lillicrap et al. [45] is the base DRL off-policy algorithm for continuous action spaces. DDPG utilizes the actor-critic architecture, where the actor represents the policy and the critic approximates the Q-function. Both are usually represented by DNNs.

The critic $Q(s_t, a_t)$ with parameters θ^Q serves as a tool to train the actor $\pi(a)$. The critic uses Temporal Difference (TD)-learning to predict the action-value Q by minimizing the loss $l\left(\theta^Q\right)$:

$$l\left(\theta^Q\right) = \mathbb{E}\left[\left(Q\left(s_t, a_t|\theta\right) - y_t\right)^2\right] \tag{2.9}$$

with the target value y_t:

$$y_t = r(s_t, a_t) + \gamma Q\left(s_{t+1}, \pi(s_{t+1}) \mid \theta^Q\right) \tag{2.10}$$

This way, the critic is recursively trained to estimate how good an action is in a given state. The procedure of using the critic itself for target prediction is called *bootstrapping*.

The actor training utilizes the estimation capability and the differentiability of the critic network by maximizing the expected Q-value with the following loss function:

$$l(\theta^\pi) = -Q\left(s_t, \pi(s_t)\right) \tag{2.11}$$

However, this training procedure is only stable when utilizing two ideas from Mnih et al. [44]. First, note that the targets for Q in (2.10) depend on Q itself, which results in unstable Q updates. As in [44], DDPG uses target networks Q' and π' of the critic and actor for target prediction. The target networks are constrained to move slowly by using soft updates from the main networks, which stabilizes optimization:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{2.12}$$

with $\tau << 1$. Second, since DDPG is an off-policy algorithm, it can utilize a replay buffer to store and re-use old training samples. Each new environment transition tuple (s_t, a_t, r_t, s_{t+1}) gets stored in the buffer. In each training step, a mini-batch gets sampled from the buffer to perform the previously described training updates. The replay buffer improves hardware utilization, training stability, and overall performance.

Combining these ideas results in a deterministic policy that maximizes the expected

return given the states. To achieve exploration despite having a deterministic policy, noise is added to the actions when interacting with the environment. Lillicrap et al. proposed an Ornstein-Uhlenbeck process, however, using a normal distribution is more common nowadays [46]. Algorithm 1 shows the DDPG algorithm in pseudo-code.

Algorithm 1 Deep Deterministic Policy Gradient (DDPG) [45].

```
Randomly initialize actor and critic networks.
Initialize actor and critic target networks as copies, respectively.
Initialize replay buffer D.
while Training do
    Receive initial environment state s_0
    for t = 0, ..., T do
        Select noisy action a_t = π(s_t) + N_t based on state s_t.
        Perform action a_t: Observe reward r_t and new state s_{t+1}.
        Store transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D.
        Sample a random mini-batch of transitions from D.
        Calculate target values with equation (2.10).
        Update the critic network by minimizing the mean loss of equation (2.9).
        Update the actor network by minimizing the mean loss of equation (2.11).
        Update both target networks according to equation (2.12).
    end for
end while
```

2.4.2 Soft Actor-Critic (SAC)

The Soft Actor-Critic (SAC) algorithm by Haarnoja et al. [42, 47] is an off-policy DRL algorithm that builds upon DDPG. In contrast to DDPG, SAC uses a stochastic policy. The idea is to change the RL objective to maximize not only the reward but also the entropy of actions:

$$J = \sum_{t=0}^{T} \mathbb{E}\left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right] \tag{2.13}$$

In other words, the goal is to maximize the return while acting as randomly as possible. The temperature parameter α serves as a weighting of reward maximization vs. entropy maximization. Adding the entropy term to the objective function affects the training of both actor and critic. The target value y of the critic changes to:

$$y_t = r(s_t, a_t) + \gamma(Q\left(s_{t+1}, \pi(s_{t+1}) \mid \theta^Q\right) + \alpha \mathcal{H}(\pi(\cdot|s_{t+1}))) \tag{2.14}$$

The stochastic actor has two output vectors now, which are interpreted as the mean and standard deviation of a Normal distribution. The loss function changes to:

$$l(\theta^{\pi}) = -Q\left(s_t, \pi(s_t)\right) - \alpha\mathcal{H}(\pi(\cdot|s_{t+1})) \tag{2.15}$$

It turned out that a fixed temperature parameter α results in brittleness regarding the scaling of the reward function [47]. A relatively high reward magnitude neglects entropy maximization, while a small reward neglects reward maximization. Additionally, the mean reward magnitude is expected to increase during training, which complicates the choice of α even more. Therefore, Haarnoja et al. added a way to update the α parameter consistently during training to achieve a minimum expected entropy using gradient descent [47]:

$$l(\alpha) = \mathbb{E}\left[-\alpha \log \pi_t((s_t|a_t)) - \alpha\mathcal{H}^{\text{target}}\right], \tag{2.16}$$

which increases the weighting in function (2.13) if the current expected entropy is too low and vice versa.

In addition to entropy maximization, SAC uses the double critic idea [48] to prevent a positive bias of the critic prediction, which is considered harmful for training performance [49]. SAC trains two critics independently and uses the minimum of both predictions for actor and critic training (equations (2.14) and (2.15)), as proposed by [48].

Overall, the discussed changes of SAC relative to DDPG result in improved exploration, more stable training, robustness regarding hyperparameter choices, and overall significantly better training performance, which makes it one of the state-of-the-art off-policy DRL algorithms. Algorithm 2 shows the full SAC algorithm.

2.5 Hyperparameter Optimization

The following section provides an overview on Hyperparameter Optimization (HPO). It mainly summarizes the overview of Bischl et al. [50].

Most ML algorithms have various hyperparameters that substantially influence the resulting performance, convergence, and learning speed. For example, the learning rate or the batch size are typical hyperparameters. The goal of Hyperparameter Optimization (HPO) is to find the optimal hyperparameter setting λ^* that minimizes the estimated generalization error $\sim c(\lambda)$ of the algorithm on unseen data:

$$\lambda^* \in \text{argmin}_{\lambda\in\Lambda} \sim c(\lambda) \tag{2.17}$$

The hyperparameters λ can be selected from the hyperparameter search space Λ, which

Algorithm 2 Soft Actor-Critic (SAC) [42, 47].

Randomly initialize one actor and two critic networks.
Initialize critic target networks as copies respectively (no actor target network in SAC).
Initialize replay buffer D.
while Training **do**
  Receive initial environment state s_0
  for $t = 0, \ldots, T$ **do**
    Sample stochastic action $a_t = \pi(s_t)$
    Perform action a_t: Observe reward r_t and new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D.
    Sample a random mini-batch of transitions from D.
    Calculate target values with equation (2.14).
    Update both critic networks by minimizing the mean loss of equation (2.9).
    Update the actor network by minimizing the mean loss of equation (2.15).
    Update temperature parameter α according to equation (2.16).
    Update critic target networks according to equation (2.12).
  end for
end while

contains all n hyperparameters and their ranges:

$$\Lambda = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_n \tag{2.18}$$

HPO is usually a blackbox optimization problem and is solved by iteratively proposing a hyperparameter setting, training the algorithm on a training dataset, estimating the generalization error $\sim c$ on a validation dataset, which gets repeated until convergence. The estimation of the generalization error is often noisy, which requires sophisticated resampling methods like cross-validation [50] to prevent a biased performance estimation.

This results in a computationally expensive blackbox optimization problem. However, it is consensus that automated HPO should be preferred over manual search. Various algorithms can be utilized for HPO, for example, simple algorithms like grid search and random search, or more sophisticated optimization algorithms like Bayesian optimization or evolutionary algorithms.

3 Related Work

The following chapter presents the related work relevant to this thesis. First, we discuss various publications that apply RL to the OPF problem, focusing on the current state of the art and the utilization of standardized benchmarks. The second section presents selected non-RL approaches to the OPF, focusing on aspects that are transferable to RL and again reviews the usage of benchmarks. The third section discusses multiple open-source environment frameworks and standardized datasets that are related to the OPF and are potentially relevant to this work. Finally, we discuss the state of the art in RL environment design literature.

3.1 Optimal Power Flow With Reinforcement Learning

The following section presents an overview of recent approaches to solve the OPF with RL. Whenever the authors deviate from using a standard RL algorithm and add advanced or problem-specific features, it is made explicit to depict the development of the RL-OPF field. Table 3.1 summarizes the discussed RL-OPF literature.

Yan and Xu [21] use DDPG to learn the OPF in the IEEE 118-bus system. They replace the DDPG critic directly with the OPF cost function plus lagrangian terms to incorporate constraints in the learning process, utilizing domain knowledge of the underlying problem.

Woo et al. [22] apply Twin-Delayed DDPG (TD3) to solve the same general OPF problem as in Yan and Xu. They incorporate a more advanced load data sampling by combining time-series data with random noise to create unique data points.

Zhou et al. [18] combine the Proximal Policy Optimization (PPO) algorithm with supervised pre-training to solve a security-constrained OPF problem in the IEEE 14-bus and the Illinois 200-bus systems. They demonstrate that pre-trained PPO outperforms base PPO regarding optimization performance and outperforms pure supervised training regarding constraint satisfaction.

Zhen et al. [51] combine supervised pre-training with the TD3 algorithm and are the first to model the OPF problem as a one-step RL environment to simplify training.

Nie et al. [52] use extensive time-series data to train a TD3 algorithm to perform voltage

control with reactive power in a Microgrid environment, utilizing a modified IEEE 30-bus system.

Cao et al. [53] apply PPO to a stochastic multi-stage OPF that considers stochastic wind turbine feed-in and storage systems with time-dependent constraints in a modified IEEE 33-bus system. Their approach outperforms a particle swarm optimization used for reference. Further, they demonstrate the effectiveness of reward normalization and reward clipping to stabilize training.

Zhou et al. [54] use the PPO with supervised pre-training and convolutional neural networks to solve a stochastic economic dispatch in the Illinois 200-bus system. Additionally, they consider N-1 topology changes and propose a classifier that predicts the feasibility of the OPF problem.

Liu et al. [55] apply DDPG to a multi-stage OPF problem in a modified IEEE 33-bus distribution network, considering storage system constraints over multiple time steps.

Yizhi Wu et al. [24] argue that constraint satisfaction becomes especially challenging in stochastic OPF problems and propose chance-constrained MDPs to consider constraints in stochastic use cases. They utilize Bayesian neural networks for function approximation.

Tong Wu et al. [25] developed a constrained version of TD3 and applied it to a stochastic multi-stage economic dispatch with dynamically changing constraints. For function approximation, they use complex-valued Graph Neural Networks (GNNs). Interestingly, their DRL approach outperformed the baseline stochastic OPF solver regarding computation time, even with the training time included.

Yi et al. [23] combine the SAC algorithm with various advanced concepts, including a pre-training based on domain knowledge, a linear safety layer, and automatic updates of penalty factors for constraint satisfaction. They are also the first to consider real-world online training instead of simulation-based offline training.

Overall, an increasing amount of literature aims to solve the OPF with RL. The earlier publications [22, 18, 51, 52] provide proof-of-concepts by applying standard DRL algorithms to basic OPF problems. More recent research [53, 54, 55, 24, 25, 23] focuses on more challenging OPF formulations, especially focusing on stochastic and multi-stage OPF. Regarding performance, the overall consensus is that RL can approximate the optimal AC-OPF solutions quite well and can outperform the DC-OPF [22], meta-heuristics [53], or pure supervised learning [18]. Algorithm-wise, mainly off-policy algorithms like DDPG, TD3, or SAC are used due to their sample efficiency except for the state-of-the-art on-policy algorithm PPO in [18, 53, 54].

The more recent publications often incorporate domain knowledge into some base RL algorithm. While this is beneficial for performance, especially regarding constraint satisfaction, incorporating domain knowledge can also result in additional engineering effort.

Table 3.1: Overview on RL-OPF literature.

	OPF Variant	RL Algorithm	Benchmark System(s)
[21]	Base OPF	DDPG + Lagrange	IEEE 118-bus
[22]	Base OPF	TD3	IEEE 118-bus
[18]	Security-Constrained	Pre-trained PPO	IEEE 14-bus, Illinois 200-bus
[51]	Base OPF	Pre-trained TD3	IEEE 39-bus
[52]	Base OPF	TD3	Mod. IEEE 30-bus
[53]	Stochastic, Multi-Stage	PPO + Clipping	Mod. IEEE 33-bus
[54]	Stochastic	Pre-trained CNN-PPO	Illinois 200-bus
[55]	Multi-Stage	DDPG	Mod. IEEE 33-bus
[24]	Stochastic	PPO + BNN	IEEE 33-bus
[25]	Stochastic, Multi-Stage	Constr. GNN-TD3	IEEE 14-bus, IEEE 30-bus
[23]	Multi-Stage, Security-Cons.	SAC + Various	Mod. IEEE 33-bus

The RL algorithm is now fine-tuned for a specific problem, which jeopardizes the modularity of RL that usually allows us to apply a wide range of algorithms to a wide range of environments. The second drawback of the literature is that the OPF environment design is usually barely discussed, although there is a wide range of possible design decisions [1]. Finally, although all use benchmark power systems, the actual OPF problem to solve is not standardized in any way. Further, the benchmark power systems are often modified to incorporate additional aspects like RES or storages into the problem. This way, the publications are not comparable with each other, which makes it very difficult to evaluate the true performance of the approaches. For the same reason, it remains unclear if the proposed algorithms can perform well on a wider range of OPF problems.

3.2 Optimal Power Flow With Supervised and Unsupervised Learning

The previous section focused on the literature on how to apply RL to the OPF problem. However, many aspects from non-RL-OPF literature are directly transferable to RL, like the modeling of the OPF, the utilization of shared benchmarks, or advanced neural network architectures. Hence, the following section presents some selected publications that apply supervised or unsupervised training to the OPF. For a more comprehensive review of the topic, refer to recent surveys, focusing mainly on non-RL approaches [17, 56].

Liu et al. [57] train a physics-informed convolutional neural network in a supervised fashion by utilizing domain knowledge for the loss function and the network architecture. They argue that utilizing domain knowledge reduces training data demand, which is one of the main hindrances of solving the OPF with ML.

In their seminal work, Park et al. [58] observed that for real-world large-scale OPF problems, the input and output spaces of end-to-end-trained DNN become extremely large, which requires larger models and impedes training. They propose a principal component analysis of the input and output spaces for supervised training. Compared to standard supervised approaches, their approach results in the smallest models that achieve the overall best performance, especially regarding constraint satisfaction. With 30,000 buses, they also use the biggest test system in the available literature so far.

Zhou et al. [59] identify the limiting factor of most existing ML-OPF approaches, which is that they assume a fixed network topology and fixed admittances, which is not transferable to real-world scenarios. They propose to use the admittance matrix as additional input and use supervised learning to train a single DNN for a range of possible network topologies and line admittances.

Huang et al. [16] argue that supervised learning is not suited for learning the OPF because ground-truth data for advanced problems is usually not available. Instead, they train a DNN in unsupervised fashion by directly encoding the objective function and penalties for constraint satisfaction into the loss function.

Similarly, Owerko et al. [19] train a GNN by using the objective function and multiple penalty functions for constraint satisfaction as a loss function for unsupervised training.

Wang and Srikantha [20] propose a very sophisticated unsupervised combination of generative adversarial networks and an autoencoder to generate and fine-tune candidate solutions to the OPF. The overall architecture heavily builds upon OPF domain knowledge as well.

Overall, the non-RL-OPF literature provides various interesting approaches that are directly transferable to RL. For example, dimension reduction, GNNs, and using the admittance matrix as an input can be expected to be beneficial for performance, training speed, and generality of results. However, as in the previous section, we can observe a strong trend towards utilizing domain knowledge in the training process, especially in the unsupervised approaches where the objective function and the constraints are often directly encoded into the loss function of the neural networks or even the network architecture itself. Further, the overview demonstrates the same lack of benchmarking that was identified in the RL-OPF literature, which makes a comparison of results very difficult. This way, it remains unclear what the current state-of-the-art approaches are regarding overall performance.

3.3 Related Benchmark Environments and Frameworks

In the previous two sections, we identified the problem that most ML-OPF works do not benchmark their results in a way that they are comparable to other research. Therefore,

the following section will discuss various open-source frameworks and libraries that are related to the OPF, focusing on RL but also presenting some selected non-RL libraries.

Closest to the focus of this work, Henry and Ernst [60] present *Gym-ANM*, an open-source RL environment framework for active network management tasks in distribution systems, built on top of *OpenAI Gym*. Along with the general framework, they present a single multi-stage OPF toy environment.

Cui and Zhang [61] published *Andes_gym*, an open-source framework to create environment representations of control tasks in the energy system. They use *OpenAI Gym* for the environment API and *ANDES* [62] for the dynamic control simulation

Together with the 2020 Learning to run a power network (L2RPN) challenge, Marot et al. [63] published the *Grid2Op* framework, which enables the representation of sequential network operation actions as RL *OpenAI Gym* environments. While they also allow continuous redispatching actions, they focus mainly on discrete topological actions to satisfy safety constraints over longer time periods.

Coffrin et al. [64] created the *PGLib-OPF* benchmark library, which is a collection of existing OPF problems in the *MATPOWER* format that were standardized and updated to newer standards. They discussed measures to artificially make the OPF problems more difficult to make them more suitable for benchmarking because algorithmic improvements become more visible with high optimality gaps.

Joswig-Jones et al. [65] argue that many ML-OPF approaches are trained on too limited datasets. They present *OPF-Learn*, a framework to generate the broadest feasible dataset for supervised training. Their approach requires some conventional OPF solver for dataset creation, which hinders applicability to RL-based approaches.

Additionally, various RL environment frameworks and benchmarks focus on related use cases like microgrid control [66, 67], storage system dispatch in distribution networks [68], or sustainability tasks like CO2 reduction [69].

In conclusion, several open-source frameworks enable the creation of datasets or environments for OPF-related tasks in the power system. Only Henry and Ernst [60] enable the creation of RL environments for the OPF problem. However, they focus on the multi-stage OPF, which is a special case, and do not provide fixed benchmark environments that enable and encourage comparability of ML-OPF research progress. Finally, none of the mentioned RL environment frameworks discussed their environment design decisions in detail.

3.4 Environment Design for Reinforcement Learning

After the previous section identified a lack of benchmark environments in the ML-OPF domain and also the absence of systematic environment design considerations, the following

section will provide an overview of the RL environment design literature. Refer to [2] for a more comprehensive overview of the topic.

Environment design in RL aims to find the easiest-to-solve environment representation for a given problem. More specifically, the environment design should support convergence to the optimal policy, learning speed, policy robustness, generality, and be compatible with standard RL algorithms and frameworks. That definition is in stark contrast to how most open-source RL benchmark environments are designed. They are designed to be difficult but solvable. That is to support the development and testing of new RL algorithms and methods from an academic perspective. However, when applying RL to real-world and practical problems, we aim for environments that support and simplify the learning instead of challenging it. See [2] for an in-depth discussion.

The following section presents a short list of relevant publications that fit with our more practical notion of RL environment design. Most of them focus on specific aspects of environment design like the action space definition [26, 28, 70], the observation space [26, 27, 71], the reward function [26, 29], the episode definition [26, 30], or the utilized data and state distributions [26, 72]. These five categories, i.e., action, observation, reward, episode, and data, also make up the core design decision categories in RL environment design, apart from problem-specific aspects [2]. In all of them, the literature is quite limited.

Peng and van de Panne [70] compare four different action space representations in robotics/locomotion tasks. They observe significant performance differences regarding learning speed, final performance, and robustness. Further, they found advantages of high-level action representations over low-level representations, e.g., controlling the target angle of a PD controller instead of controlling torque. Finally, they note that some design decisions are continuous and require parameter tuning, similar to HPO.

Kanervisto et al. [28] investigate action space shaping[1] in computer game environments with a focus on discrete actions and removing unnecessary actions. Most noteworthy, they found that squashing multi-discrete actions into a single high-dimensional discrete action space can jeopardize learning success. Further, they point out the great difficulty of environment design that it is initially unknown if the problem is solvable with RL.

Kim and Ha [27] note the lack of established principles for the observation space definition. They investigated different observation space variants in robotics, like binary contact flags, observation histories, and global observations, and found significant performance differences. They also found that redundant observations usually do not hurt performance. In an attempt towards automated environment design, they propose a search algorithm to

[1] The authors assume the existence of some original action space with the goal to shape an improved action space representation. Note that the assumption of a given action space deviates from our environment design definition above, which considers environment design from scratch, making it more general.

automatically select the best-performing observation channels.

Yang and Nachum et al. [71] aim to optimize the observation space representation as well but with a different approach. They compare multiple unsupervised pretraining objectives with existing offline data to find lower-dimensional representations. They demonstrate that such pretraining can be beneficial for subsequent online learning.

Ng et al. [29] investigate reward shaping, i.e., adjusting the reward function to provide a more useful learning signal. They show multiple examples of how bad reward design can result in faulty policies that exploit the reward function without solving the underlying problem. They demonstrate theoretically and empirically how potential-based reward shaping can increase reward density and improve learning speed.

Pardo et al. [30] focus on the episode definition and its termination. They demonstrate the importance of differentiating between terminating and truncating the environment, where termination represents an actual win/lose situation from which zero future reward is expected onwards, while truncation is an artificial interruption of the episode, for example, to achieve a better mixing of the training data. Their findings have found their way into the *Gymnasium* API [73], which now differentiates termination and truncation, in contrast to its now outdated predecessor *OpenAI Gym*. They also show that if time limits are used to terminate the episode, the remaining time must be part of the observation space. Otherwise, the problem becomes partially observable.

Zhang et al. [72] demonstrate how overfitting can become a problem in RL if the learning agent can solve the given problem by memorizing high-performing action sequences. If that happens, a well-performing policy in the training environment fails to transfer well to a testing environment. They propose to strictly separate train and test datasets, as it is common practice in supervised learning but not yet common in RL.

Reda et al. [26] investigate all kinds of environment design decisions in locomotion/robotics tasks, including the state distribution, the control frequency, episode termination, action space, reward function, and observation space. For example, they confirm the influence of the episode termination (see [30]) and the importance of separate test datasets to test for policy generality (see [72]). On the other hand, they challenge the effectiveness of unsupervised observation representation learning (compare [71]) and the superiority of high-level action representations (compare [70]). Overall, they conclude that many different environment design decisions significantly influence training performance and that environment design is a highly neglected field of research that often remains undocumented.

To summarize, Table 3.2 provides an overview of the previously discussed literature, including their respective domains and touched design categories. All publications focus on very specific domains, mostly robotics. No publication considers environment design in the energy domain. All except Reda et al. [26] investigate only a single design category. None of

Table 3.2: Overview on RL environment design literature.

	Domain	Observation	Action	Reward	Data	Episode
[70]	Robotics[1]		✓			
[28]	Games		✓			
[27]	Robotics[1]	✓				
[71]	Robotics[1]	✓				
[29]	Grid-World Nav.			✓		
[30]	Robotics[1]					✓
[72]	Maze Navigation				✓	
[26]	Robotics[1]	✓	✓	✓	✓	✓

[1]Using *MuJoCo* benchmark environments or their *PyBullet* equivalents.

the publications investigated potential interdependencies of design decisions, although Peng and van de Panne [70] suspect such relationships between action representation and reward. Some works propose first approaches to automated environment design, e.g., observation space optimization [27, 71]. However, these are designed specifically for the observation space and not readily transferable to other aspects of environment design.

Overall, due to the domain-specific and limited literature, it is difficult to draw general conclusions and best practices about environment design, except the utilization of test datasets [72] and Pardo et al.'s findings about episode termination [30]. The lack of general rules may indicate that environment design should happen problem-specific, which suggests the usage of automated approaches to reduce engineering effort as proposed by [1, 27, 71].

3.5 Summary and Research Gaps

The previous sections presented the current state of the art of solving the OPF with ML, relevant benchmarks and datasets, and an overview of RL environment design literature. The main discovery and research gap is the absence of rigorous benchmarking in ML-OPF research, independent of the utilized ML paradigm. This might reduce comparability by disguising the performance of the approaches, can jeopardize independent evaluation of new algorithms, and slows down overall research progress this way. Section 3.3 confirms this by showing that no such benchmark environments are available. This lack of benchmarks becomes increasingly relevant considering that ML-OPF research shows a strong trend towards incorporating domain knowledge into the learning process, which may limit the generality of the approaches. Without benchmarking, it remains unclear if such approaches generalize to other use cases outside the ones created by the authors for this specific approach.

The second general research gap is the neglect of environment design aspects in the RL-OPF literature [1] and also in the general RL literature [26]. The existing literature shows no clear rules to follow for designing RL environments for a given problem, which suggests that automated and problem-specific approaches should be used to approach environment design rigorously. This is especially true for environment design with parameters in the continuous space, where manual design is not realistically possible.

4 Characteristics of the OPF as RL Problem

The following chapter aims to answer RQ1 with an in-depth analysis of the OPF as an RL problem, built on the state-of-the-art presented earlier.

> **RQ1:** What are the characteristics, difficulties, and chances of the OPF as an RL problem formulation?

First, we analyse the characteristics of the RL-OPF approach in comparison with more conventional approaches to identify in which situation RL can and should be preferred. Afterward, we discuss the difficulties and chances of formulating the OPF as an RL problem, which lays the foundation for designing the environment framework proposed in the next chapter 5.

4.1 Reinforcement Learning in Comparison to Conventional Solvers and Meta-Heuristics

With conventional solvers, meta-heuristics, and data-driven methods, we discussed three overall approaches to solving the OPF in section 2.1. In the following, we extend the analysis of Frank et al. [11, 35], which compares meta-heuristics with conventional solvers for the OPF, by adding RL as a third category. Note that most RL characteristics apply to other data-driven methods as well, e.g., supervised learning. However, to limit the scope of this work, we will focus on RL-trained DNNs in the comparison. Also, note that the assessments apply to the overall characteristics of the general approaches. There are exceptions in almost all cases, in particular, to remedy specific weaknesses. That, however, does not disprove the pros and cons of the three categories on a general level.

The biggest difference of any ML-based approach to conventional solvers or meta-heuristics is that they require a **preliminary training phase**. While the other approaches solve the OPF per instance, ML-based approaches train a model (usually a DNN) that serves as a solver. This way, lots of computation is required before deployment. However,

after training, the resulting ML model can produce solutions for a given problem very quickly. Overall, the computation is shifted from online inference to offline training.

The very fast inference time makes DNNs advantageous regarding **real-capability**. While conventional solvers are strongly dependent on starting points and meta-heuristics on stochastic exploration, trained DNNs only perform a series of matrix multiplications to map from inputs to outputs. These operations are fast, easy to parallelize, and have predictable computation times, which provides a strong basis for real-time capability.

However, the big drawback of any ML model is that it is only meaningful for the data distribution it was trained and tested on. One consequence is the difficulty of making any **guarantees regarding optimality or constraint satisfaction**. Especially the lack of guaranteed constraint satisfaction can become a big problem in critical infrastructure like power systems where constraint satisfaction is often a strict requirement [33]. However, there are advances regarding the feasibility of DNN-generated solutions. Some Safe RL algorithms can guarantee constraint satisfaction if a perfect oracle exists [74]. Further, an increasing amount of literature aims to provide worst-case guarantees of DNNs' performance [75, 76].

While constraint satisfaction guarantees are a problem, RL can deal better with **qualitative constraints** like non-numeric, categorical, or descriptive conditions than conventional methods. These often result in non-differentiability and non-continuity, which most conventional solvers have difficulties with, while the RL reward does not strictly require differentiability or continuity.

In general, similar to meta-heuristics, RL can deal well with **non-differentiability** and **black-box problems**. RL makes only little assumptions about the nature of the problem, which makes it modular to some extent. In the standard RL framework of agent and environment, the RL algorithm can easily be exchanged for a given problem if it can deal with the action/observation space of the environment. For example, the DDPG algorithm can always be exchanged with the more state-of-the-art SAC algorithm (compare section 2.4).

Further, when using DNNs for function approximation, RL can easily deal with **discrete variables** in the state, action, or reward space, similarly to meta-heuristics, which remains a problem for many conventional solvers [11].

While conventional solvers require special algorithms and significant additional solving time for **sequential actions** or **stochastic problems**, RL algorithms consider both inherently. Sequential actions are considered by the standard RL objective to maximize the sum of rewards over the full episode. Stochasticity is represented by the training data distribution and by maximizing the expected sum of rewards.

Finally, RL provides advantages if the **power system model is imperfect** and for

Table 4.1: Characteristics of solving the OPF with conventional solvers, meta-heuristics, and RL in comparison.

	Conventional	Meta-Heuristics	Reinforcement Learning
Training Required	No	No	Yes
Solving/Inference Speed	Medium	Slow [35]	Fast (after training)
Optimality Guarantees	Partly [11]	No*	No
Constraint Guarantees	Yes	No [35]	Partly
Qualitative Constraints	Partly [35]	Yes [35]	Yes
Non-Differentiability	No [35]	Yes [35]	Yes
Black-Box Problems	No	Yes [35]	Yes
Discrete Variables	Difficult [11]	Yes [35]	Yes
Sequential Actions	Difficult [33]	Yes	Yes
Stochastic Problems	Difficult	Yes	Yes
Partial Observability	No	No	Yes
Imperfect System Model	No	No	Partly

*Assuming finite computation time [35].

partial observability where the system state is only partly known. If the system model is expected to be imperfect, the RL agent can be trained on data from real-world measurements instead of simulation data, for example, as proposed by [23]. While conventional solvers require a preliminary state estimation to deal with partial observability, the RL problem can be represented as Partially-Observable MDP (POMDP) [41] instead of the standard MDP framework to consider imperfect observations of the system's state.

Table 4.1 summarizes the pros and cons of RL for the OPF in comparison to the other two. Altogether, using RL over conventional solvers trades few disadvantages, like the lack of guarantees or the expensive training, for various advantages, like the improved real-time capability or the inherent consideration of sequential actions and stochasticity.

In conclusion, RL – as any ML method – is no generally superior approach to conventional solvers or meta-heuristics. However, it provides specific advantages and can become an additional tool in the toolbox of OPF researchers and practitioners when specific requirements need to be fulfilled. RL seems to be especially well-suited for real-time application of the multi-stage and the stochastic OPF, both described in section 2.1. RL can be used to solve the OPF end-to-end [13] or as part of hybrid OPF methods [35, 17], for example, by providing good starting points for warm-start of conventional methods [36, 17]. Both last points match well with the discussed RL-OPF literature, where pre-training, stochastic OPF, and multi-stage OPF are recurring themes (compare section 3.1).

4.2 Challenges and Chances of the RL-OPF

While the previous section discussed the advantages and disadvantages of using RL to solve the OPF in comparison to conventional solvers and meta-heuristics, the following section will focus on the challenges and chances when formulating and solving the OPF as an RL problem. Note that a short variant of this list is already published in [1].

High-dimensional state and action spaces The OPF can become a very large-scale optimization problem with extremely high-dimensional state and action spaces [36]. While high-dimensional state spaces are less problematic due to the emergence of DNNs [44], large-scale action spaces remain a challenge in RL due to the curse of dimensionality [45]. Large-scale OPF problems may contain thousands of generators, switches, shunts, etc. In RL, such high-dimensional action spaces are very challenging and not often investigated. For example, the widely used MuJoCo[1] RL benchmark environments have only six actions on average.

Dynamic constraints Many constraints in power system optimization, especially of the actuators, are state-dependent in their nature. For example, the maximum active power feed-in of wind turbines or solar systems depends on the current weather situation. The reactive power range of a generator often depends on its active power feed-in (or vice versa). The maximum power flow over a transmission line is temperature- and weather-dependent. Consequently, these dynamic state-dependent constraints need to be considered in the observation space of the RL agent to make optimal decision-making possible.

Continuous and discrete actuators Large-scale real-world OPF problems usually consider continuous and discrete actuators as degrees of freedom for optimization [32], for example, discrete transformer taps and continuous generator setpoints. While there are standard RL algorithms for discrete and continuous action spaces respectively, considering both at the same time as hybrid action space usually requires special algorithms [77], limiting the usage of standard algorithms.

Expensive action evaluation To evaluate the utility of, e.g., generator setpoints, a power flow calculation needs to be performed to compute voltage levels and power flows. Since power flow calculations are computationally expensive, especially for large-scale grids, the number of computations should be kept minimal, which requires sample-efficient RL

[1] `https://gymnasium.farama.org/environments/mujoco/`, last access: 2024-06-26

algorithms. This aligns well with the RL-OPF literature where mostly sample efficient off-policy algorithms are used (compare section 3.1).

Limited realistic data The existing open datasets of realistic grid states are very limited. To train an RL agent to solve the OPF in realistic grid states, such datasets are required for training and testing. A random sampling of states can not be expected to be useful since most random grid states are very unrealistic [1]. However, most open datasets contain a maximum of a few 10,000 data points, which is too little for most ML applications.

Safety-critical system The power grid is a safety-critical system where constraint satisfaction is non-negotiable and strong performance guarantees are desirable. However, as discussed in the previous section, standard RL algorithms are not able to provide performance guarantees and even specialized safe RL algorithms only to some extent.

Models availability Grid operators usually have sufficiently good models and simulators of their power grids available.[2] Therefore, it is usually possible to train RL agents in simulation. Further, the model knowledge can be utilized to improve training speed by providing domain knowledge to the agent instead of learning from scratch, as done, for example, in [23, 25, 9].

Strong OPF baselines The OPF has been subject to research for multiple decades now. We have a wide range of existing sophisticated algorithms that have been verified by scientific literature and put into practice for real-world grid operation. This way, we have strong baselines available for the evaluation of novel approaches like RL, and it can be easily evaluated if the resulting performance is competitive or not.

Modular framework As already mentioned in section 3.1, RL results in some modularity regarding the environment (OPF problem) and the agent (RL algorithm). In consequence, the RL agent can be replaced by more advanced algorithms quite easily, and vice versa, which results in several advantages. It allows us to apply base RL algorithms without utilizing any domain knowledge. This way, the performance of RL-OPF algorithms will automatically improve with general RL progress. Further, this modularity makes it possible to apply the same algorithm to a wide range of OPF problems, for example, stochastic or multi-stage OPF variants.

[2] This is mainly true for higher-voltage systems. For low and medium-voltage grids, models are often uncertain or even completely missing [36]. However, since the OPF is mainly applied to higher-voltage systems, we can assume the existence of sufficiently good models.

Dense reward function The objective function of the OPF problem is very dense, which transfers to the RL reward function. We can compute non-zero and information-rich rewards for every action within the power grid. Dense rewards are not strictly required for RL but helpful for training speed [29].

Summary and Conclusion The previous paragraphs identified multiple challenges and chances in formulating the OPF as an RL problem. While the availability of models, strong algorithmic baselines, the dense utility function, and the modular RL framework can be expected to prove beneficial for solving the problem, the high-dimensional hybrid action spaces, the dynamic constraints, the limited datasets, and the strict safety requirements complicate solving the OPF problem with RL. All these points have to be considered when formulating the OPF as an RL environment, which lays the foundation for the next chapter of this thesis.

5 *OPF-Gym* Environment Framework and Benchmarks

The following sections present *OPF-Gym*,[1] which is the RL-OPF benchmark environment framework developed for this thesis to answer RQ2:

> **RQ2:** How should a benchmark framework for OPF environments look like that ensures reproducible and comparable research with fixed benchmarks but also degrees of freedom for systematic environment design?

OPF-Gym consists of a general framework to define RL-OPF environments, various pre-implemented environment design options, and five basic OPF problems for benchmarking. To maximize broad usability and applicability, *OPF-Gym* focuses on the single-step, deterministic OPF base case with continuous actuators, in contrast to advanced use cases like multi-stage OPF or stochastic OPF. However, more advanced OPF cases are still possible with *OPF-Gym*, which will be discussed later. Technical details about *OPF-Gym* will not be discussed in depth and can be found in its documentation.[2]

This chapter is structured as follows: *OPF-Gym* builds upon three open-source frameworks, which will be presented in the first section. In section 5.2, the core of the framework is presented, including its approach to handling the dichotomy between fixed benchmark problems and degrees of freedom for environment design optimization. Section 5.3 summarizes the resulting advantages and capabilities but also the limitations of the framework. Section 5.4 lists all design decisions implemented in *OPF-Gym* for environment design (degrees of freedom), while section 5.5 defines the (fixed) OPF benchmark problems for this thesis.

[1] `https://github.com/Digitalized-Energy-Systems/opfgym`
[2] `https://opf-gym.readthedocs.io/en/latest/`

5.1 Utilized Open-Source Frameworks

The *OPF-Gym* framework builds upon three main open-source frameworks, which are *Gymnasium* for the RL environment API, *pandapower* for power grid modeling, and *SimBench* for power grid benchmark data. All three will be presented in the following.

Environment API: *Gymnasium* *Gymnasium*[3] [73] is the quasi-standard API to build single-agent RL environments in python. It defines a simple API standard that can represent an MDP. *Gymnasium*'s core is the `Env` class, from which can be inherited to create custom environments. To create a custom *Gymnasium* environment, two core methods need to be implemented: the `reset()` and the `step()` method. The `reset()` method is called to start a new episode. It resets the environment to some (usually random) initial state and returns the respective observation as state representation. Based on the observation, the agent performs some action, which is given to the environment via the `step()` method. The `step()` method returns the reward signal, two binary flags whether the episode terminated or truncated,[4] and the next observation. This observation is used to select the next action, which results in a sequential loop that ends when the environment terminates or truncates. In addition to the observations, the environment always also provides an `info` dictionary that can be used to communicate non-standard information to the agent. The whole procedure is shown in Algorithm 3.

Algorithm 3 High-level interaction with a *Gymnasium* environment.

```
Initialize environment and agent
while Training do
    Call env.reset() method
    Receive initial observation and info tuple (obs, info)
    Set terminated = truncated = False
    while terminated is False & truncated is False do
        Select action act = policy(obs)
        Call environment.step(act) method
        Receive tuple (obs, reward, terminated, truncated, info)
        Train the agent with the received data
    end while
end while
```

In addition to the described methods, a custom *Gymnasium* environment requires defined action and observation spaces, which define its dimensionalities and whether they are continuous or discrete.

[3] `https://gymnasium.farama.org/index.html`, last access: 2024-08-05.

[4] The difference of terminated vs. truncated is as follows: A termination is the natural end of the episode, for example, by winning or losing a game or achieving the intended goal. In contrast, truncation describes a premature ending, for example, to mix up training data or because of a failed simulator. [2, 30]

Finally, *Gymnasium* provides so-called *Wrappers*, which allow to perform general modifications to environments, for example, truncating observations, re-scaling actions, or similar.

Grid modeling and power flow calculation: *pandapower* The *pandapower*[5] [78] library is a combination of the data analysis library *pandas* and the power flow solver *PYPOWER*. It models complete power grids and all kinds of grid components like loads, generators, transformers, lines, etc. Further, various benchmark systems are pre-implemented as pandapower networks like the CIGRE or IEEE systems. To analyze power systems, *pandapower* provides computational tools like power flow solvers, an OPF solver, short-circuit calculation, state estimation, etc. For this thesis, mainly the network modeling, the power flow calculation, and the OPF calculation are used. Additionally, the import feature of *pandapower* allows for easy import from other standards, for example, from the mentioned *PGLib-OPF* library, which uses the supported *MATPOWER* format.

Benchmark power systems and time-series data: *SimBench* *SimBench*[6] [79] is a dataset of benchmark power grids. Additionally, each grid is equipped with realistic time-series data of one full year in 15-minute intervals for all generators, loads, and storage systems. These accompanying datasets of overall 35k steps are the main difference to other benchmark grids like IEEE or CIGRE and the main reason why *SimBench* was chosen for this work. *SimBench* provides power grids of all voltage levels from Low-Voltage (LV) to Extra-High-Voltage (EHV) and in three different expansion levels: *present*, *future*, and *far-future* with increasing expansion levels of RES and storage systems. Further, the *SimBench* grids can be combined to create multi-level networks over multiple voltage levels. Finally, *SimBench* provides full integration with *pandapower*, which allows for easy loading and manipulation of the grids.

5.2 The *OPF-Gym* Framework

The following section provides an overview of the *OPF-Gym* framework[7] developed for this thesis and to answer RQ2. The main challenge of RQ2 was the dichotomy between fixed problem definitions for benchmarking on the one hand and degrees of freedom for performance optimization in the form of environment design.

We can see an RL environment as a combination of two parts. First, a fixed underlying problem definition, which defines the goal, the available knowledge, and the actuators of

[5] `https://pandapower.readthedocs.io/en/latest/index.html`, last access: 2024-08-05.
[6] `https://simbench.de/en/`, last access: 2024-08-06.
[7] `https://github.com/Digitalized-Energy-Systems/opfgym`

Figure 5.1: Class structure of the *OPF-Gym* framework.

the RL agent. It cannot be changed. Second, an open-for-optimization environment design in the form of the reward function, the observation and action space, the episode definition, and the data distribution, which serve as a representation of the problem to solve [2]. This problem/representation-split is explicitly considered by the class structure of *OPF-Gym*. We distinguish the parameterizable and customizable `OpfEnv` base class and its problem-defining subclasses. The `OpfEnv` base class inherits from the *Gymnasium* `Env` class and implements its API. The class structure is visualized in Figure 5.1.

The `OpfEnv` Superclass The underlying `OpfEnv` class has three purposes. First, it defines the environment design space in an extensible way. In other words, the user can define custom variants of the typical environment design decisions, for example, reward functions, data sampling variants, etc. Additionally, multiple environment design variants are pre-implemented for easy usage. These pre-implemented design options will be presented later in section 5.4. Second, the observation and action spaces are automatically generated for the given OPF problem. Third, the `OpfEnv` class defines the `reset()` and `step()` methods according to the *Gymnasium* API, which will be discussed in the following.

The `reset()` method is called to start a new episode. Its high-level implementation in the `OpfEnv` class is shown in Algorithm 4. First, the `reset()` method of *Gymnasium*'s `Env` superclass is called to handle seeding. Second, a random power grid state is sampled for which the OPF needs to be solved, usually, the load active and reactive power values. This way, one episode represents a specific OPF problem, which remains unchanged during the episode. The exact sampling method is an open environment design decision. Third, for some environment design variants, an initial power flow calculation (using *pandapower*) is performed to compute voltage levels and power flows. If that initial power flow is required, also an initial action is required (usually power setpoints). Finally, the `reset()` method returns the initial observation, which serves as a state representation of the previously sampled OPF problem. How to transform grid states into observations is another open design decision discussed later on.

After the RL agent chooses an action based on the initial observation, the `step()` method is called repeatedly for the agent-environment interaction. Its procedure is shown in Algo-

Algorithm 4 The `reset()` method of *OPF-Gym*.

Call superclass `Env.reset()` method
Randomly sample current grid state (see section 5.4.2)
if Initial power flow required (see Table 5.1) **then**
    Apply some initial action (see section 5.4.3)
    Compute power flow calculation
end if
Get initial observation (see section 5.4.3)
Return (observation, info)

rithm 5. First, the agent action is applied to the current grid state, which means translating the agent's action to actuator setpoints like generator power setpoints or transformer tap changer setpoints. The exact action representation is another open design decision. After the actuators are set, the *pandapower* power flow is used to calculate the new grid state, including power flows and voltage values resulting from the agent's action. Next, a reward is calculated that represents the goodness of the new grid state, representing the objective function and the constraint satisfaction, which is again an open design decision. Then, it is determined whether the episode is terminated or truncated, which is another design decision. Finally, the next observation is created, and all data is returned to the agent for learning and further decision-making. Additionally, both `reset()` and `step()` return an info dictionary that provides information about the current state of the environment.

Algorithm 5 The `step()` method of *OPF-Gym*.

Receive agent action as input
Apply action to the current grid state (see section 5.4.5)
Perform power calculation to compute new grid state
Calculate reward (see section 5.4.1)
Episode terminated or truncated? (see section 5.4.4)
Get next observation (see section 5.4.3)
Return (observation, reward, terminated, truncated, info)

The Benchmark Subclasses The problem-specific benchmark subclasses inherit from the `OpfEnv` class and define a specific OPF problem, for example, an economic dispatch in a specific grid. That is done by creating a *pandapower* grid and defining an OPF by setting controllable units, prices for active/reactive power, and constraints as it is done in *pandapower*.[8] This way, all OPF variants possible in *pandapower* can automatically be transferred to *OPF-Gym* environments. Each subclass represents one fixed RL-OPF benchmark problem. The implemented benchmark problems for this thesis will be presented in

[8] See `https://pandapower.readthedocs.io/en/latest/opf.html`, last access 2024-10-28

section 5.5.

Further, the observation and action spaces need to be defined in the subclass because they are problem-specific. That is done by creating a mapping from the *pandapower* tables structure to a *Gymnasium* space. This way, arbitrary parts of a *pandapower* grid can be defined as actions and/or observations. The conversion to *Gymnasium* action and observation spaces happens automatically.

5.3 Features and Limitations of *OPF-Gym*

Various features and limitations result from the previously described framework. *OPF-Gym* allows for the creation of almost arbitrary OPF problem representations. As one main feature, all OPF problems that are solvable with *pandapower* are also possible with *OPF-Gym*. This includes continuous actuators like load, generator, and storage active and/or reactive setpoints. It also includes linear, quadratic, and piece-wise linear prices for active and reactive power setpoints. Since all *pandapower* OPFs can be represented as an RL environment, this allows for easy evaluation of the RL algorithm's performance compared to established conventional solvers.

Additionally, *OPF-Gym* allows for the creation of more advanced OPF variants that are not solvable with *pandapower*.[9] For example, discrete variables can be used as actuators without additional effort. From the agent's perspective, all actions are defined as continuous (compare [2]), which makes special RL algorithms with hybrid discrete/continuous action spaces unnecessary. Instead, standard RL algorithms are always directly applicable. *OPF-Gym* also allows for adding arbitrary constraints. This way, for example, the security-constrained OPF is possible by adding additional constraints for the N-1 case. The stochastic OPF can be implemented by adding stochastic effects. For example, *Gymnasium* wrappers can be used to create noisy observations, which might represent measurement or prediction errors, without changing the actual grid state. A partially observed OPF is possible by providing only a subset of observations required for optimal decision-making. The multi-stage OPF can be implemented by extending the `step()` method so that the grid state gets updated to the next state in the *SimBench* time-series dataset.

In general, most parts of the environment can be customized by the user. For example, custom power flow and OPF calculations can be provided, if the default *pandapower* solver is not sufficient. Further, arbitrary state variables can be introduced by adding columns to the *pandapower* tables.

Finally, while focusing on RL, *OPF-Gym* provides some support for supervised learning

[9] See `https://opf-gym.readthedocs.io/en/latest/advanced_features.html` for various example implementations of advanced OPF problems.

as well. To allow for easy performance comparisons with supervised approaches from literature, a convenience function automatically generates labeled datasets for a given *OPF-Gym* environment. This way, RL and supervised learning approaches can be evaluated on the exact same benchmark problems, which is required to compare performance of the two paradigms.

However, the design of *OPF-Gym* also results in some drawbacks, limitations, and required assumptions. The most important limitation is that *OPF-Gym* inherits most of *pandapower*'s limitations. For example, ground-truth evaluation is only possible if the OPF is solvable with *pandapower*. More advanced OPF variants like stochastic or multi-stage OPFs are not solvable with *pandapower* and require custom OPF solvers. Additionally, the power grid and all its units must be modelable in *pandapower* to be usable. The second big limitation is the dependency on *SimBench* time-series data. While non-*SimBench* systems are generally possible, time-series data is often required to make the environment meaningful. For example, completely randomly sampled training data is mostly unrealistic [1], and the multi-stage OPF is meaningless without underlying time-series data. Another minor limitation is that the power flow of the current grid state needs to be solvable. Otherwise, the reward function and observation creation will fail. Finally, since all actuators are represented by continuous action spaces, RL algorithms that only work on discrete action spaces are not applicable, especially DQN [44] and all its derivatives.

5.4 Environment Design Space

While the OPF problem is clearly defined, it remains unclear how to represent it as RL environment. The following sections will present all the pre-implemented options for environment design categorized by the choice of the reward function, the data distribution, the observation space, the episode definition, and the action space. The implemented design variants were chosen based on two aspects. First, typical design variants from the literature were implemented as identified in [1]. However, these environment designs from literature are usually fixed points in a continuous space, often neglecting the space in between. Therefore, the second aspect is to implement parameterizable design options, whose parameters can be used as degrees of freedom for environment design optimization and analysis as shown in [3]. Overall, 22 different environment design variables were implemented, from which 15 will be presented here in detail since they turned out to be especially relevant [3].

The content of this section was mainly derived from the existing environment designs in the RL-OPF literature in [1] and later extended in [3].

5.4.1 Reward Function

The reward functions for the RL-OPF problem used in literature can be roughly grouped into two categories: The *Summation* and the *Replacement* reward [1]. The *Summation* reward used in [9, 14, 22, 52, 55] is very straightforward by simply adding the objective function and some penalty function to consider optimization and constraint satisfaction at the same time:

$$r_{\text{sum}}(s,a) = -J(s,a) - P(s,a) \tag{5.1}$$

This way, the learning agent always has an incentive to improve its policy regarding both goals, which makes the reward very dense. However, in practice, constraint satisfaction is often far more important than optimization, and the *Summation* method has difficulties representing that because it is challenging to ensure that rewards for valid states are strictly higher than those for invalid states. However, if that is not the case, the agent might choose to sacrifice constraint satisfaction for better optimization performance [1].

The *Replacement* reward used in [18, 51, 54] tackles that problem with a case distinction that prioritizes constraint satisfaction over optimization. The idea is to yield strictly negative penalties for invalid states and to provide positive optimization rewards only for valid states:

$$r_{\text{replace}}(s,a) = \begin{cases} -J(s,a) + r_{\text{valid}} & \text{if valid} \\ -P(s,a) & \text{else} \end{cases} \tag{5.2}$$

The constant offset reward $r_{\text{valid}} > 0$ is required to ensure that the optimization reward is always positive. This forces the agent to first ensure constraint satisfaction befor optimization regarding the objective function is even possible. However, compared to *Summation*, the reward signal becomes less dense by removing the optimization signal in invalid states. Further, the offset reward is difficult to tune. Using the worst-case objective value would be a natural choice. However, that might result in overly conservative behavior [1]. Finally, too strong offsets might result in extreme gradients at the valid/invalid switching point.

In addition to the previously discussed drawbacks, both reward variants require extensive tuning to balance the magnitudes of the objective function and the penalty function because both can have arbitrary scales. For example, the costs of high-voltage systems might be several orders of magnitude higher than those of a low-voltage system, even when using the exact same cost function. This will inherently create an over-prioritization of one of the goals. Also, too high or too low reward magnitudes might jeopardize training performance [80].

Summation and *Replacement* represent two distinct cases in a continuous range of potential reward functions. For systematic environment design, the range between these distinct cases needs to be considered as well. Therefore, the following parameterisable reward function r_{design} is created that combines these two and also adds some more benefits for experiment design as discussed in the following:

$$r_{\text{design}}(s,a) = (1-\beta)\hat{J}_{\text{norm}}(s,a) + \beta\hat{P}_{\text{norm}}(s,a) \tag{5.3}$$

The function consists of an adapted objective function $\hat{J}_{\text{norm}}(s,a)$ and an adapted penalty function $\hat{P}_{\text{norm}}(s,a)$, which represent the optimization and constraint satisfaction goals respectively, and will be discussed later. To ensure that both functions are in the same order of magnitude, both are independently normalized.[10] The separated normalization allows for controlled balancing of optimization and constraint satisfaction by using a *Penalty Weight* factor $0 \leq \beta \leq 1$, where high values prioritize constraint satisfaction, and vice versa. The *Penalty Weight* parameter is one parameter for automated environment design in chapter 6.

The adapted objective function is defined as follows:

$$\hat{J}_{\text{norm}}(s,a) = \begin{cases} -J_{\text{norm}}(s,a) & \text{if valid} \\ -\psi J_{\text{norm}}(s,a) & \text{else} \end{cases} \tag{5.4}$$

To consider the full range between *Summation* and *Replacement*, the *Invalid Objective Share* $0 \leq \psi \leq 1$ is introduced. Here, $\psi = 1$ represents the *Summation* method, while $\psi = 0$ represents the *Replacement* method. The intermediate range can be used to explore the space in between and is a parameter for environment design.

Until now, we treated the objective function J for reward calculation as identical to the objective function of the underlying OPF problem. However, most often, the objective function consists of two parts, where one can be influenced by the control actions while the other cannot. For example, the system losses can be influenced to some extent but cannot be minimized to exactly zero, which results in some fixed offset. As mentioned before, the reward scale influences RL training performance significantly. Therefore, an uncontrollable offset in the reward might negatively influence learning performance by superimposing the useful reward signal. Hence, we investigate two different objective functions for reward

[10] Since we do not know the reward distribution and range beforehand, the normalization is done by sampling sufficient amounts (multiple thousand) of random state-action pairs. The resulting distribution of objective values and penalties is used to estimate the mean, variance, minimum, and maximum of both functions, which are used for normalization. In this work, both functions are normalized to zero mean and a variance of one. However, min-max scaling to the range [-1, 1] is possible as well.

calculation:

$$J(s,a) = \begin{cases} J_{\text{OPF}}(s,a) - J_{\text{init}}(s) & \text{if } \textit{Diff-Objective} \\ J_{\text{OPF}}(s,a) & \text{else} \end{cases} \tag{5.5}$$

In the *Diff-Objective* variant is used, we subtract the estimated uncontrollable part from the objective function. It is estimated by calculating J_{init} of the initial state before acting. The choice between the two options results in another binary parameter for environment design.

The adjusted penalty function is defined as follows:

$$\hat{P}_{\text{norm}}(s,a) = \begin{cases} r_{\text{valid}} & \text{if valid} \\ -P_{\text{norm}}(s,a) - r_{\text{invalid}} & \text{else} \end{cases} \tag{5.6}$$

To represent the *Summation* method, the offset reward *Valid Reward* r_{valid} can be set to zero, whereas all cases $r_{\text{valid}} > 0$ represent the *Replacement* reward. Additionally, a constant *Invalid Penalty* $r_{\text{invalid}} \geq 0$ is introduced to investigate if valid states should be rewarded, invalid states should be punished, or both. Both offset rewards are parameters for environment design. As discussed before, choosing the right offset reward is non-trivial. However, due to the normalization, its optimum can be expected in the low single-digit range, which is useful for defining the search space later on.

In summary, the parameterized reward function $r_{\text{design}}(s,a)$ ensures normalization and contains five parameters for environment design to balance optimization and constraint satisfaction, to find the optimal middle ground of *Summation* and *Replacement*. The five parameters are the *Penalty Weight*, the *Valid Reward*, the *Invalid Penalty*, the *Invalid Objective Share*, and the *Diff-Objective* flag. All derived degrees of freedom for environment design are collected and summarized later in Table 5.1.

5.4.2 Training Data Distribution

A core question in ML is the choice of the training data distribution. In RL, the data distribution is often neglected [2], which can result in policies that are not transferable to the general case, i.e., over-fitting [72]. In the case of the OPF, to be useful for grid operators or researchers, it is strictly required that the learned policy performs well in real-world scenarios. Therefore, the testing dataset should be as close to reality as possible. And the training dataset should be selected such that it allows for maximal performance on the test dataset [2].

If possible, the natural choice for the train dataset is to take the same approach as for

the test dataset and sample the environment state s from some realistic dataset $\mathcal{D}$ of load and generator time-series data [9, 14, 52, 55]:

$$s \sim \mathcal{D} \tag{5.7}$$

Ideally, the train dataset is a subset of the same dataset as the test data. However, real-world or at least realistic time-series data of power grid states is still very limited. Therefore, in an RL setting, we can expect repetition of data at some point. Further, such close-to-reality time-series datasets will contain very limited numbers of edge cases, like holidays, extreme weather events, etc. Overall, that might limit the generality of the learned policy. [1]

To counteract the limitedness and the lack of generality of realistic data, data can be sampled from some random distribution, for example, using a Normal distribution [21]:

$$s \sim \mathcal{N}(\mu, \sigma^2) \tag{5.8}$$

with mean μ and variance σ^2, or a Uniform distribution [18, 22, 54]:

$$s \sim \mathcal{U}(s_{\min}, s_{\max}) \tag{5.9}$$

with the data range $[s_{\min}, s_{\max}]$. Such random data can be created infinitely to create large datasets. However, most of these randomly sampled grid states will be highly unrealistic [1]. Therefore, when training with randomly sampled data, it might happen that the RL agent learns a policy for situations that will never happen.

As discussed before, using realistic datasets or sampling random data both have their respective drawbacks and benefits. Realistic datasets are closest to actual real-world scenarios but are limited and barely consider edge cases. Random data can be created infinitely but will mostly be unrealistic. The straightforward approach is to combine both, to use some realistic data for transferability and some random artificial data for generality. However, the balancing of randomness and realism in the dataset is not obvious. Therefore, for environment design, a parameterized sampling method is introduced that combines all three approaches:

$$s \sim \begin{cases} \mathcal{D} & \text{with probability } x \\ \mathcal{N}(\mu, \sigma^2) & \text{with probability } y \\ \mathcal{U}(s_{\min}, s_{\max}) & \text{with probability } z \end{cases} \qquad \text{s.t. } x + y + z = 1.0 \tag{5.10}$$

with the respective probabilities *Realistic Data* share x, *Normal Data* share y, and *Uniform Data* share z to sample from each distribution respectively. These probabilities are three

more continuous parameters for environment design.

In summary, we discussed three data sampling methods and how they can be used as options for environment design. Overall, this results in three parameters for environment design: the *Realistic Data* share, the *Uniform Data* share, and the *Uniform Data* share for data sampling. For the *Realistic Data*, the *SimBench* time-series datasets are used.

5.4.3 Observation Space

Another important environment design decision is the choice of the observation space. Regarding the state of the environment, usually, the *Markov* property is used to describe whether the state contains all the information required to make optimal decisions [41]. As argued in [2], the Markov property should be applied to the observation space, too, since we want to design an environment that enables the agent to learn optimal policies.[11]

In power grid calculation, each node in the power system has four state properties: active power, reactive power, voltage magnitude, and voltage angle. If we assume that the topology and electrical characteristics of the grid itself stay constant, only two of the four properties are required for full knowledge about the grid state. For example, if the active and reactive power of all nodes are known, a power flow calculation can be performed to calculate all node voltages. And if the node voltages are known, all power flows in the system can be computed as well. In other words, half of the state properties of electrical power grids are redundant.

Based on the Markov property and the redundancy of power grid states, we define the first observation space variant, which we call *Markov* in the following [1]. For example, it was used by [9, 18, 21, 51, 52, 55]. The idea is to only provide the bare minimum of observations that are required to make optimal decisions. Those are all non-controllable active and reactive power values, all economic variables like active or reactive power prices, and dynamic constraints that depend on the current state, e.g., maximum wind or solar feed-in. While the *Markov* observation variant should theoretically be sufficient to learn optimal policies, additional observations might improve overall performance or learning speed. For example, the constraints are mostly related to state variables like voltages, line loads, etc., which were identified as redundant before. To deal with these constraints in the *Markov* variant, the agent needs to learn some kind of implicit power flow calculation to predict and prevent such constraint violations, which might complicate training. Because of that, the second observation space variant called *Redundant* is to provide all state

[11] Note that we are discussing here which observations should be given to the agent. However, in real-world grid operation, some data is not available, which might result in a partially-observable RL problem. In that case, the Markovian observation space can provide information on which measurements would be important for decision-making and where to place additional sensors.

information to the agent, e.g. voltage magnitudes or line loading, as used by [14, 22, 54].

However, similar to the reward function definition, only looking at the *Markov* and the *Redundant* observation space neglects the space in between. Some redundant observations might prove helpful, while others could even be harmful for learning [27]. In theory, we could test every single observation channel to determine if it is helpful for training or not, similar to the approach by Kim and Ha [27]. However, since we are dealing with potentially thousands of observation channels, we do this on a category basis instead. For example, does adding all line loads to the observation space improve performance? This results in five boolean parameters for environment design to define whether to *Add Line Loading*, *Add Trafo Loading*, *Add Voltage Magnitude*, *Add Voltage Angle*, and *Add Slack Power* values, respectively. The expected advantage is that more information enables the agent to predict and prevent constraint violations better. However, the significantly bigger observation space may slow down training. Further, we need to perform an additional expensive power flow calculation to compute these state variables initially.

In summary, we discussed the Markov property of the observation space and how it applies to the OPF problem. We discussed five boolean environment design parameters to add redundant observations to the observation space, which allows us to investigate the whole space from providing the full *Redundant* observation space (all five True) to only providing the *Markov* observations (all five False).

5.4.4 Episode Definition

Regarding the episode definition, we have the least amount of options. The main question is whether the OPF should be formulated as a 1-step or n-step environment. Both options can be found in the literature [1].

In the *1-Step* variant used in [9, 14, 51], the OPF problem is formulated in a one-shot fashion. The agent observes the grid state, acts upon it, and receives its reward. After each step, the environment terminates and is reset to a new state. Therefore, the agent must learn a direct mapping from state to optimal action without any possibility of correcting its action if it turns out to be bad. However, the potential advantage is that the problem is simpler to learn. Most RL algorithms learn by predicting the value function (see eq. 2.6). In a 1-step environment, learning the value function is reduced to a purely supervised learning problem, which simplifies training. Note that this *1-Step* variant is rather unusual in RL, where sequential environments are more common.[12]

[12] Strictly speaking, the 1-step RL problem is reduced to a contextual bandit problem [81], for which specialized algorithms can be found in literature. However, since the contextual bandit literature focuses mostly on non-DNN-based function approximation, discrete actions, and low dimensional observation spaces, we neglect it for this work.

The *n-Step* variant is more common for RL environments and is also more closely related to how conventional OPF solvers work. The idea is to let the agent sequentially improve its action by observing the next states. In other words, the agent observes a grid state, acts upon it, receives its reward, observes the updated state, and so on. The potential advantage is that the agent can correct mistakes. For example, the initial action might result in a voltage violation, which the agent can then correct by altering its initial action. The drawback is that the agent needs to make predictions over multiple steps, which complicates training, as discussed before. An additional drawback is that the agent interacts with the same grid state multiple times, which requires additional power flow calculations. It also results in less variance in the observed states.

Overall, the decision between the *1-Step* and the *n-Step* variant results in a single parameter for environment design, which are the *Steps Per Episode* to interact with the environment. If it is one, it is the *1-Step* variant, while all other values result in the *n-Step* variant.

5.4.5 Action Space

For the OPF, the actions are usually continuous active or reactive power setpoints of generators, loads, or storage systems. Discrete actuators like transformer taps or switches are possible as well. However, due to their ordinal nature, they can be represented as continuous actions, too, plus a rounding operator [2]. Therefore, in this work, we will assume continuous actuators. We define the action space in the continuous range [0, 1], where zero is interpreted as the lowest possible setpoint and one as the maximum setpoint, demonstrated here by the example of an active power setpoint P_{set}:

$$P_{\text{set}} = a \cdot (P_{\max} - P_{\min}) + P_{\min} \text{ with } a \in [0, 1] \tag{5.11}$$

This implies that the action space is bounded, which is common in RL and fits well with power system actuators, which mostly have clearly defined nominal power ranges.

However, one property of power system actuators is that their setpoint range can be limited by dynamic constraints. For example, wind turbines and photovoltaic systems have limited maximum feed-in depending on the weather situation. Storage systems have a limited power range when they are close to being full or empty. This can become problematic when the agent can perform impossible actions like setting photovoltaic feed-in to 100% at night time. Considering this domain knowledge, the RL environment should be designed to prevent such actions because they might sabotage training or can even be exploited by the RL agent [2]. To deal with such situations, we introduce the binary variable *Autoscaling* to choose between two different action representations for the environment design:

If *Autoscaling* is True, we use the current state-dependent range:

$$P_{\text{set}} = a \cdot (P_{\max}(s) - P_{\min}(s)) + P_{\min}(s) \text{ with } a \in [0, 1] \tag{5.12}$$

The advantage is that it is impossible for the RL agent to pick an out-of-range action. However, the disadvantage is that the same agent action is interpreted differently depending on the environment state [2]. For example, $a = 1$ for a wind turbine can be interpreted as 3 MW in a strong-wind state, while it is interpreted as 1 MW in a low-wind state. Therefore, the agent needs to learn how a change in constraints maps to setpoints.

In contrast, without autoscaling, out-of-range actions are prevented by using the fixed nominal setpoint range $[P^{\text{nom}}_{\min}, P^{\text{nom}}_{\max}]$ and clipping invalid actions to the current state-dependent range.

$$P_{\text{set}} = \text{clip}(a \cdot (P^{\text{nom}}_{\max} - P^{\text{nom}}_{\min}) + P^{\text{nom}}_{\min}, P_{\min}(s), P_{\max}(s)) \text{ with } a \in [0, 1] \tag{5.13}$$

The advantage is that the same action always represents the same setpoint. However, the drawback is that many actions are essentially meaningless because they get clipped. This way, the agent practically utilizes only part of the action space, which might result in exploration problems. For example, if we consider a wind turbine that can only operate at a maximum of 50% power due to low wind, all setpoints $a >= 0.5$ will be interpreted the same and yield the same reward. This way, the agent does not receive any feedback on whether, for example, $a = 0.6$ or $a = 0.8$ is superior, which might result in wasted agent-environment interactions. The question of whether to use *Autoscaling* or not is another degree of freedom for environment design. Also note that this design decision is relevant not only for the OPF but for all kinds of dynamically constrained action spaces, e.g., dynamically changing maximum motor torque in robotics [26]. However, it has not been investigated deeper in the literature yet.

5.4.6 Overview of the Design Space

In the previous sections, we discussed the *OPF-Gym* pre-implemented environment design options and derived tunable parameters for environment design. All relevant environment design parameters are summarized in Table 5.1, including the respective potential sampling space and its unit type. Most degrees of freedom exist in the data sampling, the reward function, and the observation space definition, while the episode and action space definitions do not leave much room for optimization. *OPF-Gym* contains seven more design variables that were neglected here for brevity since they did not turn out to be relevant in the later experiments [3].

Table 5.1: Implemented environment design space.

	Design Decision	Type	Design Space
Reward	*Valid Reward*	float	$[0, \infty]$
	Invalid Penalty	float	$[0, \infty]$
	Invalid Objective Share	float	[0.0, 1.0]
	Penalty Weight	float	[0.0, 1.0]
	Diff-Objective	boolean	{True, False}
Data	*Normal Data*	float	[0%, 100%][1]
	Uniform Data	float	[0%, 100%][1]
	Realistic Data	float	[0%, 100%][1]
Obs	*Add Voltage Magnitude*	boolean	{True, False}
	Add Voltage Angle	boolean	{True, False}
	Add Line Loading	boolean	{True, False}
	Add Trafo Loading	boolean	{True, False}
	Add Slack Power	boolean	{True, False}
Episode	*Steps Per Episode*	integer	{1, 2, 3, ...}
Action	*Autoscaling*	boolean	{True, False}

[1] Constrained to a total of 100%.

5.5 Implemented Benchmark OPF Environments

The following section presents the five open-source benchmark environments created for this thesis. As discussed earlier, they represent the actual fixed OPF problem to solve. The benchmarks all represent different OPF variants and use different power grid models. All power grid models and the accompanying time-series data were taken from the *SimBench* dataset [79]. Further, all five OPFs are defined to be solvable with the *pandapower* OPF solver to enable comparisons with ground-truth results. This implies that we focus on the base case OPF, without discrete actuators, no multi-stage OPF, no stochastic OPF, and no security constraints.

All five OPF problems share some characteristics, which we will discuss in the following. All environments have the same system-level constraints, which are a voltage band of ±5 pu of all buses I

$$0.95 \text{ pu} \leq U_i \leq 1.05 \text{ pu} \quad \forall\, i \in I \tag{5.14}$$

and a maximum branch loading of 80% for all branches B (lines and transformers):

$$\frac{S_b}{S_{\max,b}} \leq 80\% \quad \forall\, b \,\in\, B \tag{5.15}$$

Additionally, some environments have constrained active or reactive power flows from the external grid,

$$P^{\text{ext}}_{\min} \leq P^{\text{ext}} \leq P^{\text{ext}}_{\max} \tag{5.16}$$

$$Q^{\text{ext}}_{\min} \leq Q^{\text{ext}} \leq Q^{\text{ext}}_{\max} \tag{5.17}$$

which will be specified for each OPF use case separately. If not stated otherwise, the default values are $-\infty$ and ∞, respectively (unconstrained). All five used power grids are connected to one external grid, respectively, which also serves as a slack node.

The *SimBench* dataset provides time-series profile data of power setpoints of the loads (active and reactive power), generators (only active power), and storages (active power) in the system. Since the technical limits of the units are not explicitly provided, we assume that the maximum setpoints in the time-series data are also the technical limit of the respective unit. These maximum technical limits are used to define the RL action space, if the respective unit is an actuator, and for data sampling, if the grid states are not sampled from the *SimBench* profiles (see section 5.4.2).

Some *SimBench* grids have very high numbers of generators, loads, and storage systems, for example, the `1-MV-semiurb-1-sw` grid with 123 generators, 118 loads, and 87 storage systems. Since most of these units are relatively small and have little influence on the overall grid state, we focus on the biggest n units as degrees of freedom for the OPF. The exact numbers for n will be specified separately for each use case in the next sections.

If not explicitly mentioned, the reactive power setpoints of generators and storage systems are assumed to be zero. In other words, we do not assume any automatic reactive power control strategy in addition to the OPF.

Note that for consistency with *pandapower*, we use the *pandapower* signing system in the following, which means we use the consumer system for loads and storages and the generator system for generators and external grids (slack nodes).[13] For example, positive storage power represents a power flow from the grid to the storage system (load), while positive external grid power means a flow from the external grid to the considered grid (generation).

[13] Compare `https://pandapower.readthedocs.io/en/latest/about/units.html`, last access 2025-03-14.

5.5.1 Voltage Control (`VoltageControl`)

The first RL-OPF benchmark environment is a voltage control problem formulated as OPF. The objective is to minimize system-wide active power losses:

$$\min J = \sum P_{\text{gen}} - \sum P_{\text{load}} \tag{5.18}$$

That is subject to voltage band, line load, and slack power flow constraints, which are especially relevant for this environment (empirically tested).

The environment uses the `1-MV-semiurb-1-sw` *SimBench* system, which is a 122-bus Medium-Voltage (MV) system. To make the problem more challenging, all generators are upscaled by factor 1.3 and all loads by factor 1.5. Further, the reactive power flow over the slack bus is constrained to a maximum of 0.5 Mvar, which prevents the OPF from exploiting its neighbor grid extensively.

The actuators for control are the reactive power setpoints of ten generators and four storage systems in the system. We assume that the agent is allowed to control the full reactive power range $[Q_{\min}, Q_{\max}]$ that is possible for a given active power setpoint P_g of generator g:

$$Q_{\max,g} = -Q_{\min,g} = \sqrt{S_{\max,g}^2 - P_g^2} \tag{5.19}$$

The apparent power $S_{\max,g}$ of generator g is computed from the maximum active power from the respective *SimBench* time-series data $P_{\max,g}$ and an assumed $\cos(\varphi) = 0.95$:

$$S_{\max,g} = \frac{P_{\max,g}}{\cos(\varphi)} \tag{5.20}$$

In other words, we assume that all controllable generators are obliged to provide reactive power for voltage control.

The Markov observation space X (see section 5.4.3) consists of all load active and reactive power values, generator active power, and storage active power setpoints:

$$X = \{\mathbf{P}_{\text{Load}}, \mathbf{Q}_{\text{Load}}, \mathbf{P}_{\text{Gen}}, \mathbf{P}_{\text{Storage}}\} \tag{5.21}$$

The active power setpoints of the controllable units are required as observations because they imply the available reactive power range. Alternatively, we could provide $Q_{\max,g}$ as observations, respectively, which are equivalent by providing the same information about the environment state.

5.5.2 Economic Dispatch (`EcoDispatch`)

The `EcoDispatch` environment represents an economic dispatch problem with the objective of minimizing overall active power generation costs in the system:

$$\min J = \sum_g p_g^P \cdot P_g + p_{\text{slack}}^P \cdot P_{\text{slack}} \tag{5.22}$$

with the assumption of linear market prices for all generators and the slack bus, which are sampled randomly for each grid state. The slack bus is considered as another generator in the system. For that, we constrain the slack active power flow to only positive values to prevent it from being used as a controllable load. The maximum slack active power is set identically to the biggest generator in the system.

The `EcoDispatch` environment uses the `1-HV-urban-0-sw` High-Voltage (HV) system with 372 buses. The loads are upscaled by 1.5 to make the problem more challenging.

The actuators are the active power setpoints in the range $[0,\, P_{\text{max},g}]$ of all 42 generators in the system. The maximum active power setpoints P_{max} of the generators g are taken from the accompanying *SimBench* time-series data, respectively:

$$P_{\text{max},g} = \max(\mathbf{P}_g^{\text{simbench}}) \tag{5.23}$$

The required observations to fulfill the Markov property are the load active and reactive power values and the active power prices of the generators:

$$X = \{\mathbf{P}_{\text{Load}}, \mathbf{Q}_{\text{Load}}, \mathbf{p}_{\text{Gen}}^P\} \tag{5.24}$$

5.5.3 Reactive Power Market (`QMarket`)

The reactive power benchmark environment `QMarket` is very similar to the `VoltageControl` environment with the difference that the reactive power is now priced, which makes it a reactive power market problem, for which market clearing is often formulated as OPF [6]. The objective is to find the optimal trade-off of reactive power market costs and active power loss costs:

$$\min J = p^P \left(\sum P_{\text{gen}} - \sum P_{\text{load}}\right) + \sum_g Q_g^2 \cdot p_g^Q \tag{5.25}$$

We assume quadratic reactive power prices p^Q according to Samimi et al. [82]. The reactive power prices are sampled randomly for each state to represent different bidding behaviors.

The environment uses the `1-MV-rural-0-sw` *SimBench* MV system with 97 buses.

The loads are upscaled by 1.5 to make the problem more challenging. Similar to the `VoltageControl` environment, the reactive power flow over the slack is constrained to a maximum of 0.1 Mvar to prevent exploitation of the neighbor grid.

The actuators are the reactive power setpoints of ten generators in the system, which are modeled as described for the `VoltageControl` environment in section 5.5.1. To consider the market context, the reactive prices of all generators are added to the observation space:

$$X = \{\mathbf{P}_{\text{Load}}, \mathbf{Q}_{\text{Load}}, \mathbf{P}_{\text{Gen}}, \mathbf{p}^{Q}_{\text{Gen}}\} \tag{5.26}$$

Note that the utilized *SimBench* system does not contain storage systems, which is why there are no storage power setpoints in the observation space definition.

5.5.4 Maximize Renewable Feed-in (`MaxRenewable`)

The `MaxRenewable` environment aims to maximize the total renewable active power feed-in of all renewable generators g the system:

$$\min J = -\sum_{g} P_{g}^{\text{RES}} \tag{5.27}$$

The relevant constraints are the maximum line load, transformer load, and the voltage band. The `MaxRenewable` environment uses the `1-HV-mixed-1-sw` HV *SimBench* system with 355 buses. All loads and generators are downscaled by 0.8.

As actuators, the active power setpoints of 15 generators and three storage systems are controllable. Since the generators are RES like wind turbines, we assume that the current feed-in can only be decreased but not increased from a given initial setpoint $P_{\text{init},g}$, which results in the action range $P_{\text{max},g}$ for generator g:

$$P_{\text{max},g} = P_{\text{init},g} \tag{5.28}$$

The storage systems are not part of the objective function but can be used as an actuator by the agent to artificially increase load to allow for more renewable feed-in. For every storage system s, we assume a fixed setpoint range of $[-P_{\text{max},s}, P_{\text{max},s}]$. That also means we neglect state-of-charge and other limiting factors here.

The Markov observations are the load active and reactive power values, the active power setpoints of all non-controllable (nc) storage systems, and the initial active power setpoints

of all generators:

$$X = \{\mathbf{P}_{\text{Load}}, \mathbf{Q}_{\text{Load}}, \mathbf{P}^{\text{nc}}_{\text{Storage}}, \mathbf{P}_{\text{Gen}}\} \tag{5.29}$$

Note that the active power setpoints of the non-controllable generators are required for estimating the overall power flows in the system, while the initial setpoints of the controllable generators are required to know the agent's own action range.

5.5.5 Load Shedding (`LoadShedding`)

The `LoadShedding` environment aims to find the cost-minimal strategy for load shedding of controllable loads, where each load is assigned a different linear load shedding price p^P_l, which is sampled randomly. The same is true for storage systems with price p^P_s:

$$\min J = -\sum_l p^P_l P_l - \sum_s p^P_s P_s \tag{5.30}$$

The `LoadShedding` env uses the `1-MV-comm-2-sw` commercial area medium voltage *SimBench* system with 111 buses. To create load shedding situations, the slack active power feed-in is constrained to 8 MW, the loads are upscaled by 2.2 and the generators by 1.6. As actuators, 15 controllable loads and one big storage system are available. Similar to the `MaxRenewable` environment, we assume that the load can only be decreased from an initial setpoint, which results in action setpoints of range $[0,\ P_{\text{max},l}(t)]$:

$$P_{\text{max},l}(t) = P_{\text{init},l}(t) \tag{5.31}$$

The action space of the storage system is $[-P_{\text{max},s},\ P_{\text{max},s}]$, where $P_{\text{max},s}$ is the maximum value of the simbench time-series again.

The Markov observations are the initial load active and reactive power values, the active power of the generators, the active power of the non-controllable storage systems, and all active power prices:

$$X = \{\mathbf{P}_{\text{Load}}, \mathbf{Q}_{\text{Load}}, \mathbf{P}^{\text{non-act}}_{\text{Storage}}, \mathbf{P}_{\text{Gen}}, \mathbf{p}^P\} \tag{5.32}$$

5.5.6 Overview of Benchmark Environments

The five RL-OPF benchmark environments are designed to represent a wide variety of OPF use cases but also to show the capabilities of the *OPF-Gym* environment framework. Table 5.2 summarizes some key characteristics of the benchmark problems. Three use cases consider prices for a market-based optimization, while the two others represent purely

Table 5.2: Overview of pre-defined benchmark environments in the *OPF-Gym* framework.

Environment	Grid Level	Actuators	Market	N_{bus}	N_{act}	N_{obs}[1]
`VoltageControl`	MV	Gens, Storages (Q)	No	122	14	442
`EcoDispatch`	HV	Gens (P)	Yes	372	42	201
`QMarket`	MV	Gens (Q)	Yes	97	10	305
`MaxRenewable`	HV	Gens, Storages (P)	No	355	18	172
`LoadShedding`	MV	Loads, Storages (P)	Yes	111	16	386

[1]Using the the minimal *Markov* observation space variant.

technical optimizations of the energy system. In three cases, active power is used for optimization, and two times, an optimal reactive power flow is performed.[14] In four cases, generators are used to optimize the power flows, three times storage systems, and in one case, controllable loads. Two times, the OPF is performed in a high-voltage system and three times in a MV system.

The utilized power systems range from 97 to 372 buses in size. The observation spaces range from length 172 to 442, which is very high-dimensional and shows why usually DNN-based approaches are preferred over conventional ML approaches. The action spaces range from length 10 to 42, which is unusually large for RL environments, as already discussed in section 4.2.

The characteristics of environments, like the utilized *SimBench* system, the constraints, the number of actuators, etc., were determined by trial-and-error experiments to create OPF problems that are neither impossible nor too easy to solve. In theory, all five presented use cases are highly parameterizable. For example, the power grid can be exchanged, constraints can be changed or added, actuators can be changed, and so on (compare section 5.3). However, the environments were presented *as is* since they are intended as static benchmark problems for future research. To ensure comparability, they should not be changed if not strictly required.

[14]Note that in no case a shared active and reactive power optimization is performed. While the *OPF-Gym* framework allows for such shared optimization, there are two reasons why it is not considered in the benchmark environments. First, a shared active/reactive power optimization is not considered to be useful because it can result in setpoints that are optimal in theory but not economically reasonable in practice [33]. Second, the *pandapower* OPF, which is used for evaluation later on, cannot consider apparent power constraints, which would be required to solve such shared optimizations in a meaningful way.

6 Automated Design of RL-OPF Environments

After the previous chapter described the *OPF-Gym* framework to create RL-OPF environments of different environment designs, the following chapter will investigate RQ3:

> **RQ3:** How to formulate OPF problems as RL environments for maximum learning performance, including constraint satisfaction, optimization performance, and learning speed?

The chapter is structured as follows: Section 6.1 contains the overall approach for automated environment design, including the utilized class of optimization algorithms, the design search space, and the OPF-specific metrics. Section 6.2 investigates the performance of the automated approach compared to a manually derived baseline. Section 6.3 shows how statistically significant environment design decisions can be derived and which design decisions are statistically significant over all five benchmark environments. Section 6.4 verifies the earlier results by repeating the training runs with the optimized environment designs, demonstrating that the performance improvements are reproducible.

The content of the chapter largely corresponds to the content of [3]. It also uses and expands the results of the pre-study performed in [1].

6.1 Approach

Two approach RQ3, two steps are necessary. The first is to develop a methodology to find high-performing environment representations of OPF problem regarding the above metrics. For that, we will formulate environment design as a multi-objective optimization problem. Second, after optimization, the derived RL-OPF environment designs can be analyzed to determine which environment design decisions are especially important for performance and how they deviate between different OPF problems.

In the pre-study [1], we demonstrated that environment design has a significant influence on RL-OPF performance and derived first results on which design variants perform well and which do not. Further, the study showed a significant trade-off between constraint

satisfaction and optimization performance. However, in the publication, only two different OPF problems were considered, the investigated environment design space was quite limited, and the performance differences were derived from 1-to-1 comparisons, neglecting interdependencies and continuous design decisions. In the following sections, we will discuss a more systematic approach to RL-OPF environment design. For that, we will use [1] as a baseline for performance evaluation.

6.1.1 Environment Design as a Multi-Objective Hyperparameter-Optimization Problem

To determine the best-performing environment designs and to enable automation of the process, we will define the RL-OPF environment design process as an optimization problem. To consider the two main performance metrics of the RL-OPF – constraint satisfaction and optimization performance – we will utilize the multi-objective optimization framework. For that, we will utilize an idea first developed in [2]. In section 5.2, we already discussed the dichotomy of RL environments, which consist of an immutable problem to solve and tunable environment design options. The idea here is to consider the environment design options as environment hyperparameters, similar to hyperparameters on the algorithm side, like the batch size or the learning rate. This allows us to directly apply all algorithms and techniques from HPO (see section 2.5), which is a well-studied field of research [50], including approaches for multi-objective HPO.

Figure 6.1 visualizes the general approach, which consists of an inner and an outer loop. We start with some random initialization of the environment design. In the inner loop, an RL algorithm learns by interacting with the current environment design. After training, we can evaluate the performance regarding one or multiple performance metrics. Based on the performance in the inner loop, some HPO algorithm proposes a new environment design in the outer loop. This can be repeated till convergence or until some termination criterion is reached. The RL algorithm and its hyperparameters stay unchanged during the whole process. Note that this general procedure is compatible with almost arbitrary combinations of RL algorithm, RL problem, and HPO algorithm.

6.1.2 Evaluation Metrics

To formulate the automated environment design as an optimization problem, we need to define one or multiple metrics to optimize. In the case of the OPF, we need metrics for two main objectives: optimization performance and constraint satisfaction. For this, we assume the existence of a baseline conventional OPF solver for comparison to evaluate the performance of our approach later on. This is not strictly necessary for evaluation but it is helpful to put the RL agent's performance in relation to that of a conventional solver.

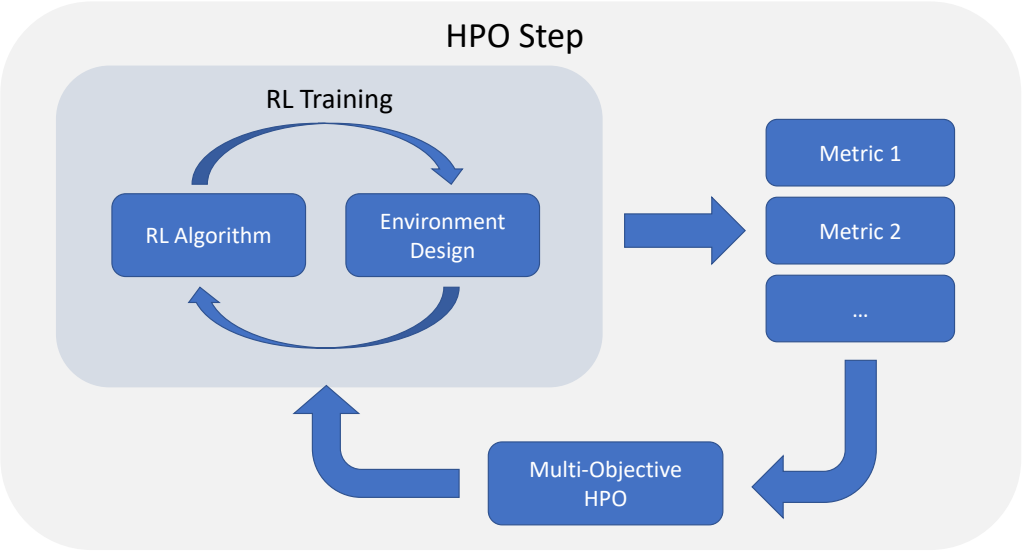


Figure 6.1: Inner and outer loop of the multi-objective HPO for the automated environment design.

For the constraint satisfaction performance, we compute the share of invalid solutions when testing on the test (or validation) dataset. Here, two things have to be considered. First, it should not be evaluated negatively when the agent does not find a valid solution when no valid solution exists. Second, it should be evaluated positively if the agent finds a valid solution where a conventional OPF solver fails:

$$\Omega = 1 - \frac{N_{\text{valid}}}{N_{\text{valid,base}}} \tag{6.1}$$

with the number of valid solutions N_{valid}. The resulting *invalid share* metric Ω needs to be minimized and becomes negative if the RL agent outperforms the conventional OPF solver regarding constraint satisfaction.

The optimization performance, or the ability to find the global cost minimum, can be represented by the *mean error* ΔJ of the objective values of all valid solutions in comparison with the ground-truth optimal values:

$$\Delta J_{\text{valid}} = \frac{1}{N_{\text{valid}}} \sum_{i=1}^{N_{\text{valid}}} (J_i - J_i^*) \tag{6.2}$$

with the objective function J. Only valid solutions are considered here because constraint satisfaction is mandatory, and good optimization performance in invalid states is meaningless. Again, the metric needs to be minimized and becomes negative when the RL

agent outperforms the conventional OPF solver. However, while the *invalid share* has an upper bound of one, the *mean error* metric has no upper limit.

With the two metrics *invalid share* and *mean error*, the goal is to minimize both metrics, with the point [0, 0] representing equal performance of the RL agent and the conventional solver.

6.1.3 Experimentation

The experiments for this work are performed as follows. For multi-objective HPO in the outer loop, the open-source framework *Optuna* [83] is used. As an optimization algorithm, the *NSGAIIISampler* [84] with its default parameters is chosen. It was chosen because it outperformed other optimizers in undocumented pre-studies and is capable of multi-objective optimization. Overall, 100 outer loop HPO steps are performed, where each optimization step represents one environment design setting.

Regarding the inner RL loop, one important aspect to consider is the stochasticity of RL experiments, which might distort evaluation with positive or negative outliers [85]. To counteract stochasticity, each sample consists of three training runs with different seeds to achieve a robust performance estimation. The metrics are averaged over the three runs, respectively. Each single RL training run is performed with the basic RL algorithm DDPG [86] for 40k training steps. The short training time of 40k steps[1] was chosen to favor fast-converging environments and to make the problem computationally tractable. We will later test the validity of that decision. The off-policy DDPG algorithm was chosen over more state-of-the-art PPO [87] or SAC [42] because it converged faster than both algorithms in an undocumented pre-study. That aspect is especially important here considering the short training times discussed before. The DDPG hyperparameters can be found in Table A.1 in the Appendix. We will test later if the results are transferable to other RL algorithms.

The accompanying environment's time-series datasets with 35k data points are split into training, validation, and testing datasets using randomized nested resampling as described by Bischl et al. [50]. First, the data is split deterministically into 80% training data and 20% testing data. The testing data is neither used for training nor for environment design evaluation during optimization. That is to prevent a positive bias by picking environment designs that perform best on the test dataset by chance [50]. Instead, they will only be used for the verification of results later on. After the test split, the remaining 80% are randomly split into training and validation datasets. For this work, we use 7k samples for training. This small amount is explicitly chosen to create an artificial shortage of *Realistic Data* to determine if randomly sampled data can compensate for that lack of data, as discussed

[1] For comparison, in the pre-study [1], the training times were 1-2 million steps for very similar environments.

in section 5.4.2. This allows us to evaluate if random data sampling method can replace realistic data without a performance drop.

The performance evaluation during environment design happens on the validation data. Again, we have to consider stochasticity. To achieve a robust performance estimation, four separate evaluations on the validation data are performed, after 25k, 30k, 35k, and 40k training steps, respectively. This procedure dampens outliers and prefers quick learning over slow learning.

6.1.4 Environment Design Space

Table 6.1: Utilized environment design search and sampling space.

	Design Decision	Type	Design Space
Reward	*Valid Reward*	float	[0, 2.0]
	Invalid Penalty	float	[0, 2.0]
	Invalid Objective Share	float	[0.0, 1.0]
	Penalty Weight	float	[0.01, 0.99]
	Diff-Objective	boolean	{True, False}
Data	*Normal Data*	float	[0%, 100%][1]
	Uniform Data	float	[0%, 100%][1]
	Realistic Data	float	[0%, 100%][1]
Obs	*Add Voltage Magnitude*	boolean	{True, False}
	Add Voltage Angle	boolean	{True, False}
	Add Line Loading	boolean	{True, False}
	Add Trafo Loading	boolean	{True, False}
	Add Slack Power	boolean	{True, False}
Episode	*Steps Per Episode*	integer	~~{1, 3, 5}~~ $\rightarrow$ {1}[2]
Action	*Autoscaling*	boolean	{True, False}

[1] Constrained to a total of 100%.
[2] Subsequently modified after early experiments.

The environment design search space implemented in *Optuna* is summarized in Table 6.1. For some cases, the search space was restricted compared to the full design space shown in Table 5.1. The *Valid Reward* and *Invalid Penalty* ranges are defined as [0.0, 2.0], where 2.0 can be interpreted as two times the standard deviation of the normalized penalties (compare section 5.4.1). The *Penalty Weight* range is slightly restricted to [0.01, 0.99] to prevent complete neglect of one of the two objectives. The three data shares are constrained to 100% by sampling them in the range [0%, 100%] and then dividing each

data share by the sum of shares, which approximates a Dirichlet distribution.

The *Steps Per Episode* space was originally chosen as $\{1, 3, 5\}$ to represent the *1-Step* variant ($N = 1$), the *n-Step* variant with few steps ($N = 3$), and the *n-Step* variant with more steps ($N = 5$) to also investigate the influence of different episode lengths. However, early experiments showed that the *1-Step* variant statistically significantly outperformed the other two options so dramatically that they had to be removed from the search space to prevent that 2/3 of the search space becomes meaningless. This was true for all five environments and aligns well with the same general result in [1]. Hence, only the *1-Step* variant was used for the automated environment design experiments.

6.1.5 Baseline Environment Design

The previously described approach of multi-objective optimization yields a Pareto front of non-dominated environment designs. To determine if the optimized environment designs result in higher performance than a manual design, we consider a manually derived environment design as a baseline for comparison. For that, we use the environment design that was derived manually in the pre-study [1] by A/B-testing different design options. The exact baseline environment design can be found in Table 6.2. It uses no offset rewards, no random training data, no redundant observations, 1-step episodes, and action autoscaling. However, the baseline design deviates for one aspect from that environment design by using a normalized reward instead of an unscaled reward to ensure comparability (compare section 5.4.1). Further, [1] does not provide any information regarding the *Penalty Weight*, which is why we consider multiple values, i.e., the values $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The training runs with the manual design are performed the same as was described in the previous section, except using ten different seeds to be more robust against outliers.

To compare the performances of the baseline environment designs with the designs from multi-objective optimization, the performance metrics of the baseline designs can be placed in relation to the Pareto front. If they are left below the Pareto front, the baseline outperforms the optimized designs; if they are right above the Pareto front, they get outperformed; and if they are directly on the front, the overall performance is roughly identical. However, considering that the performance of RL experiments is always stochastic to some extent, we have to consider the variance of results in the process, as will be shown later.

Table 6.2: Manually derived baseline environment design from the pre-study [1].

	Design Decision	**Type**	**Search Space**
Reward	*Valid Reward*	float	0.0
	Invalid Penalty	float	0.0
	Invalid Objective Share	float	1.0
	Penalty Weight	float	{0.1, 0.3, 0.5, 0.7, 0.9}
Data	*Normal Data*	float	0%
	Uniform Data	float	0%
	Realistic Data	float	100%
Obs	*Add Voltage Magnitude*	boolean	False
	Add Voltage Angle	boolean	False
	Add Line Loading	boolean	False
	Add Trafo Loading	boolean	False
	Add Slack Power	boolean	False
Episode	*Steps Per Episode*	integer	1
Action	*Autoscaling*	boolean	True

6.2 Performance Evaluation

The following sections compare the resulting performance of the proposed HPO-based automated design with the manually derived baseline design from [1]. The results for the `EcoDispatch` and `LoadShedding` environments are especially noteworthy and are discussed in detail, while the other results get summarized.

6.2.1 Economic Dispatch

Figure 6.2 shows the results for the `EcoDispatch` environment. The figure contains all non-dominated and dominated solutions from the multi-objective optimization in red and blue, respectively. Additionally, we added the baseline runs with the manual design in green. The cross in the upper Figure shows the mean standard deviation calculated over all 100 samples of the HPO.

We can observe a strong tradeoff between the two metrics, which results in the typical curved Pareto front. All points from the manual design are placed right above the Pareto front with a distance of multiple standard deviations. That shows that the manual designs get strictly dominated by the solutions from automated design. This is especially true for the samples with less focus on constraint satisfaction (lower *Penalty Weight*). Altogether, the solutions from the automated design significantly outperform the manual design.

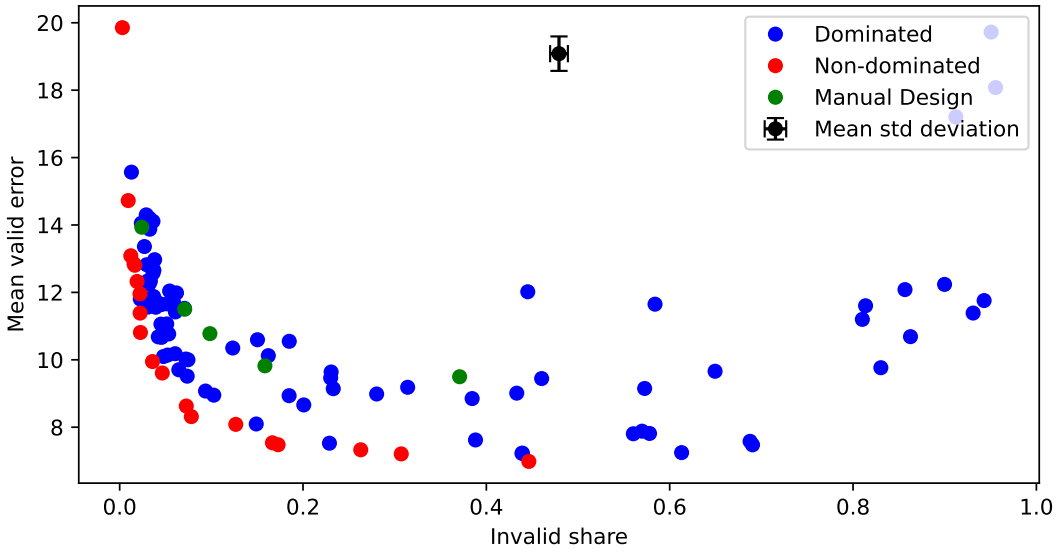


Figure 6.2: `EcoDispatch`: Dominated and non-dominated samples from the HPO in comparison with the baseline solutions. Note: Singular extreme outliers were clipped for the plot.

6.2.2 Load Shedding Environment

Figure 6.3 shows the resulting distribution of training performances for the `LoadShedding` environment. Again, the non-dominated solutions of the automated design result in a curved Pareto front that visualizes the trade-off between constraint satisfaction and optimization performance. However, in this case, the automated designs do not strictly outperform the manual design. Instead, all manual designs are located on the far right end of the Pareto front, almost independent of the chosen penalty weight. Therefore, the baseline design is generally competitive here. However, it also shows that even with varying penalty weights, only a very small part of the search space was considered. While the manual design consistently failed to achieve constraint satisfaction, the left-most solutions from the automated design demonstrate that the RL agent can even outperform the conventional solver by 7% regarding constraint satisfaction, which means that it can find valid solutions where the conventional solver fails. Without the explorative character of the multi-objective environment design, that would not have been possible.

6.2.3 Remaining Environments

For conciseness and to prevent repetitions, this section summarizes the results for the remaining three environments, `VoltageControl`, `MaxRenewable`, and `QMarket`. Their re-

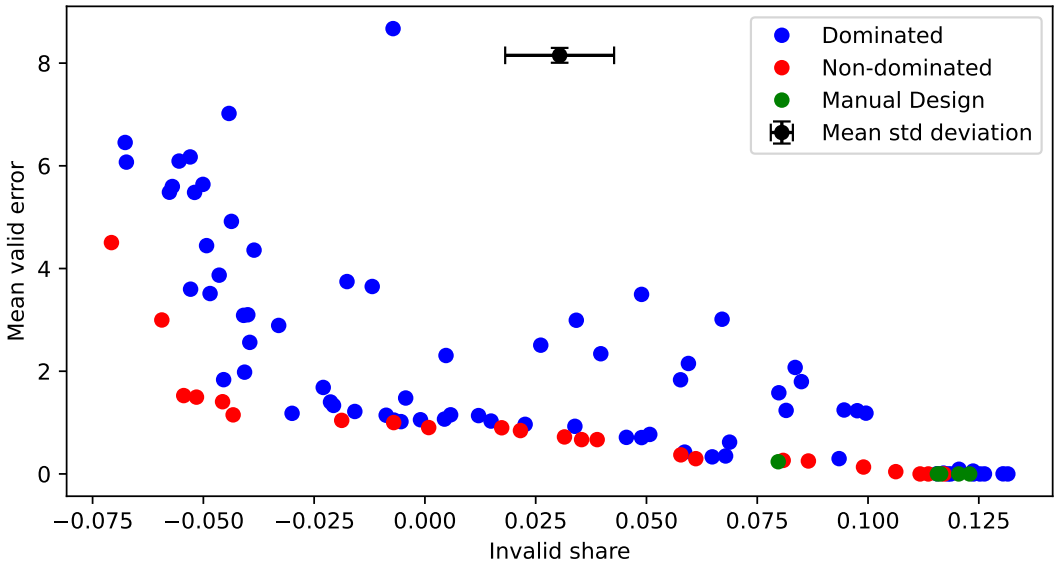


Figure 6.3: `LoadShedding`: Dominated and non-dominated samples from the HPO in comparison with the baseline solutions. Note: Singular extreme outliers were clipped for the plot.

spective Pareto fronts can be found in Appendix A.2. For all three, the non-dominated solutions from the automated design either outperform the solutions from the manual design or achieve the same performance. In the `VoltageControl` environment, the automated design even outperforms the manual one by multiple standard deviations. For the `MaxRenewable` environment, we can observe again that the manual design achieves consistently good optimization performance but fails regarding constraint satisfaction, similar to the `LoadShedding` environment discussed before.

In summary, the automated design consistently outperforms the manual design. Over all five OPF environments, there is not a single case where the manual design resulted in a solution left-below the Pareto front of the automated design.

6.3 Environment Design Evaluation

One characteristic of the performed multi-objective optimization is that we cannot extract a single best environment design from the previous results. Instead, we receive a set of non-dominated solutions as shown before. However, we can draw general conclusions about which environment design decisions are relevant for which metric and potentially also which environment design decisions are generally better than others.

To do this, we can split the generated solutions into two groups – dominated and non-

dominated solutions – and then test if there are statistically significant differences regarding the environment design. For example, if all non-dominated solutions contain the voltage magnitude in the observation space, while the distribution in the dominated set is 50/50, there is a high probability that voltage magnitude observations are required for generating non-dominated environment designs.

While splitting regarding dominated/non-dominated is natural for multi-objective optimization, other criteria are possible as well. In the following, we will split the generated solutions regarding four criteria:

1. *Pareto*: Non-dominated vs. dominated
2. *Validity*: Good *invalid share* vs. bad *invalid share*
3. *Optimization*: Good *mean error* vs. bad *mean error*
4. *Utopia*: Sum of both normalized metrics.

Regarding the latter three, we will perform the split by putting the top 20% solutions into one group and the bottom 80% into the other group.[2] Then, we determine the p-value by using Welch's t-test [88] for the continuous parameters and the chi-squared test [89] for the discrete ones. We reject the null hypothesis that a design decision does not impact performance if $p < 0.05$. This way, we can test the performance differences of all environment design decisions for statistical significance regarding the four evaluation criteria.

This general procedure can be performed for individual environments but also for all environments together. The following section 6.3.1 shows the results over all five OPF problems to yield general insights for RL-OPF environment design, while section 6.3.2 discusses specific individual environments and their results.

6.3.1 General OPF Environment Design

In this section, we will test if there are design decisions that have a statistically significant influence over all 500 samples. For that, we extract the top 20% designs from all five environments, respectively, as described before. Additionally, for the *Pareto* criterion, we have to consider that the environments have different numbers of non-dominated solutions. We use Fisher's method [90] to combine the respective p-values without favoring the environments with more non-dominated solutions.

[2] The 20/80-split was chosen based on an undocumented sensitivity analysis. The analysis showed that the choice of the cut-off point influences the general results only marginally.

Figure 6.4 shows the environment design decisions with statistically significant influence on the performance regarding the four evaluation criteria over all five environments. For the discrete design decisions, the figure shows which discrete variant is prevalent in the high-performing group. For the continuous variables, the mean of the top group is shown, with a comment if it is high or low relative to the search space. Note that the actual data points were removed for a better overview. The dotted lines hint at the respective areas of the evaluation criteria in a stylized manner.

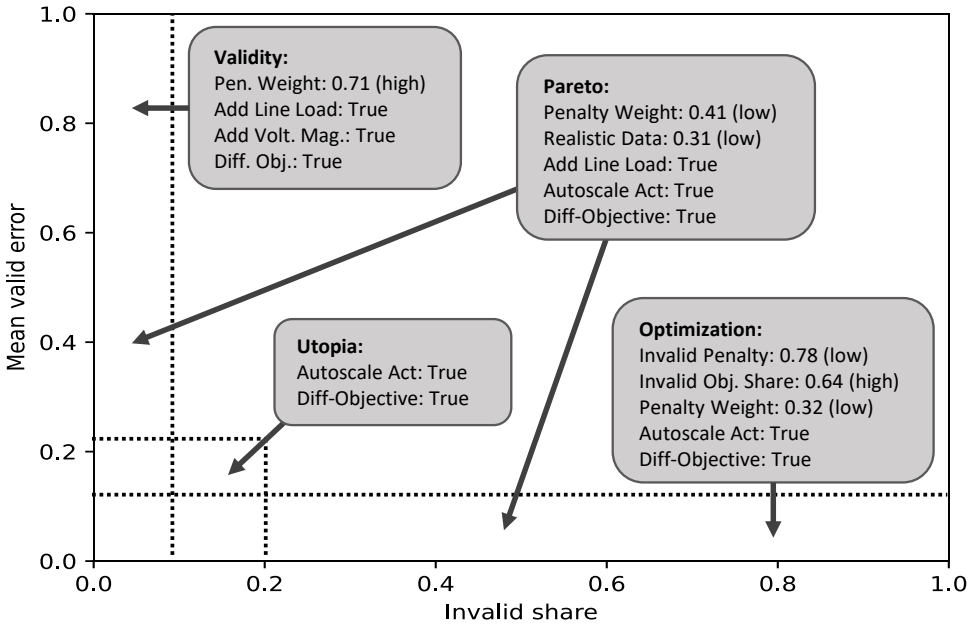


Figure 6.4: Statistically significant environment design decisions over all five environments for all four evaluation criteria, respectively.

The first noteworthy result is that while we investigated 15 different design decisions, only 2-5 of them resulted in statistically significant performance impacts over all five environments. In conclusion, while the earlier results demonstrated that environment design impacts performance on a general level, the same is not true for each individual design decision. Some are far more important than others.

Over all five environments and all evaluation criteria, only some design decisions are noteworthy to discuss. A high *Penalty Weight* benefits constraint satisfaction but harms optimization performance. However, we cannot derive novel insights from that observation because the general trade-off is already known [1, 74, 17]. Instead, this observation can be seen as a confirmation that the results align with the existing knowledge.

Using the *Diff-Objective* and *Autoscaling* options seems to consistently benefit perfor-

mance, almost independent of the evaluation criterion. Considering that these results were generated over five different environments and 1500 training runs, we can conclude that both should be considered for designing OPF environments.

Also noteworthy is that a relatively low *Realistic Data* share resulted in significantly more non-dominated Pareto front solutions, while the *Normal Data* and *Uniform Data* shares have not resulted in any statistically significant results. This confirms that time-series data can indeed be supplemented by randomly sampled data to some extent. That is in contrast to the results from the pre-study [1], which suggested that randomly sampled data cannot result in competitive performance. Instead, the results show that adding randomly generated data can improve performance if the *Realistic Data* is limited. However, the results do not provide information if a uniform or a normal distribution is better for random sampling.

Again in contrast to [1], the results indicate that adding redundant observations can improve performance, especially regarding constraint satisfaction. That is the case for the line loading observations and the voltage magnitudes. However, considering that in different OPF variants, different constraints are more important than others, we can assume again that it is use-case-specific.

Altogether, this section demonstrated how statistical tests can be used to determine design decisions that impact performance in statistically significant ways. However, note that while we could not find statistical significance for some other design decisions, that does not mean that they do not influence performance at all. It is also possible that more samples would have been required to show the statistical significance of their influence.

6.3.2 Specific Environment Design

While the previous section discussed design decisions that yield statistically significant results over all five benchmark environments, this section discusses the specifics of singular environments by the example of the `EcoDispatch` and the `LoadShedding` environments. To prevent redundancy, we will focus on the design decisions that deviate from Figure 6.4. Figure 6.5 visualizes the statistically significant environment design decisions in the `EcoDispatch` environment for the four evaluation criteria.

Most `EcoDispatch`-specific results align well with Figure 6.4. For example, to achieve good *Validity*, a high *Penalty Weight* and a low *Diff-Objective* seem to be advantageous, while the opposite is true for good *Optimization* performance. However, there are also some deviations. Especially noteworthy is that for good *Pareto* and *Optimization* performance, a high *Uniform Data* data share is advantageous in this environment. Further, the results regarding *Add Line Loading* and *Diff-Objective* are not as consistent as in the general case,

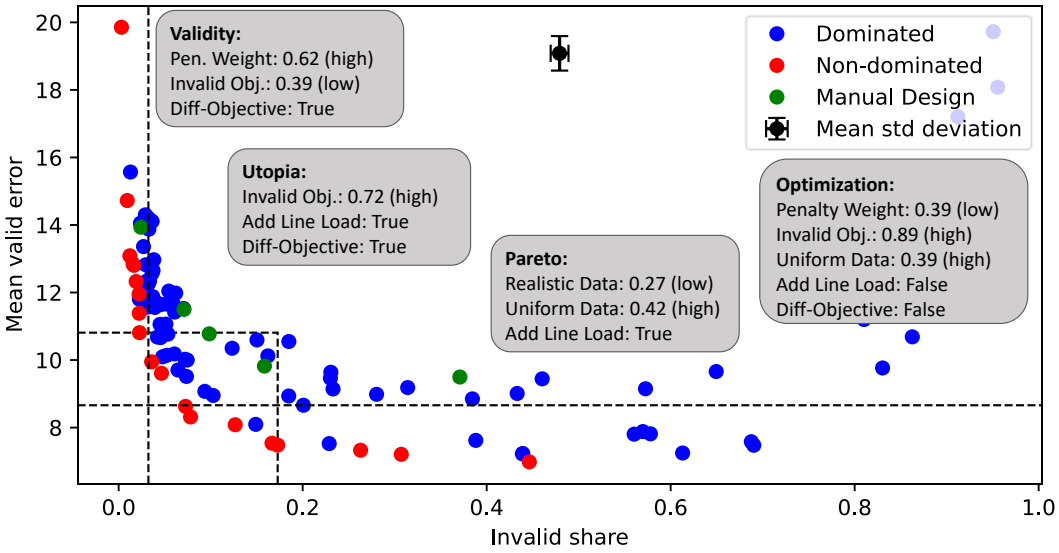


Figure 6.5: `EcoDispatch`: Statistically significant environment design decisions for all four evaluation criteria, respectively. Note: Singular extreme outliers were clipped for the plot.

resulting in better or worse performance depending on the evaluation criterion.

Figure 6.6 shows the statistically significant design decisions for the `LoadShedding` environment. Again, the results align well with Figure 6.4 on a general level but with some exceptions. In this case, it is especially noteworthy that a very small share of *Realistic Data* data is helpful for performance (*Validity* and *Pareto*), which means higher shares of randomly sampled data. Further, additional observations seem to be useful for performance in this environment, i.e., *Add Voltage Angle* for *Pareto* and *Add Slack Power* for *Optimization* performance.

For brevity, the results for the `VoltageControl`, `QMarket`, and `MaxRenewable` environments are not discussed further here. They can be found in the Appendix A.3. Overall, they all show the same general message that they align well with the general results from the previous section, with always some exceptions or deviations. Altogether, this indicates that the RL environments should be designed problem-specific for maximum performance. However, considering that some design decisions are very consistent over all five environments, e.g., *Autoscaling* or *Diff-Objective*, the general results from the previous section can be expected to provide good results if a problem-specific environment optimization is too computationally expensive.

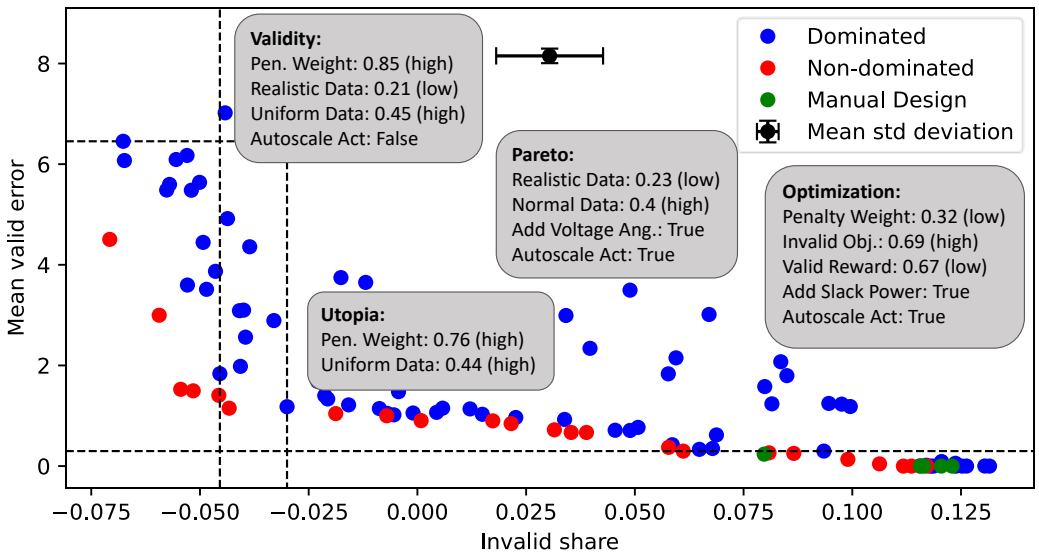


Figure 6.6: `LoadShedding`: Statistically significant environment design decisions for all four evaluation criteria, respectively. Note: Singular extreme outliers were clipped for the plot.

6.4 Verification of Optimized Designs

In the previous experiments, we made multiple decisions that might have resulted in a positive bias. To make the automated design computationally tractable, we chose short training times of 40k steps, which implicitly assumes that the resulting environment designs also work for longer training times. We chose DDPG for environment design because it performed well in trial runs, but it remains unclear if the environments that were optimized for DDPG also perform well with other RL algorithms. Further, we have not yet extracted specific environment designs from the samples. Can we extract environment designs from the results that reproducibly dominate the baseline?

To verify the performance gains from the automated environment design, we will perform verification training runs. For that, we will focus on the two environments where the automated design clearly outperformed the baseline design regarding both metrics: `EcoDispatch` and `VoltageControl`. The hypothesis is that we can extract environment designs from the previous optimization that will result in reproducibly superior performance compared to the baseline design.

1. We increase the training time from 40k steps to 500k. Further, we now use the full available training set instead of the reduced dataset used before.

2. We perform the experiments for DDPG, which was used for the optimization, and

SAC [42], which is considered to be the state-of-the-art off-policy RL algorithm.

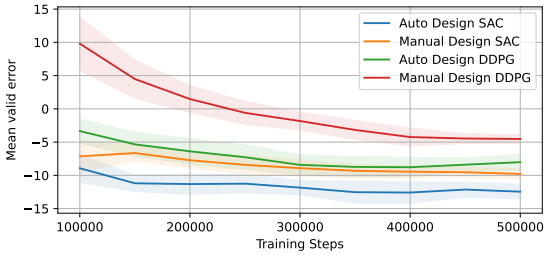
3. For both environments, we pick a combination of the best five samples regarding the *Utopia* criterion to prevent outliers. For the continuous variables, we use the mean; for the discrete ones, we take the most-used setting. We do that for all design decisions, regardless of statistical significance. The resulting exact environment designs can be found in Table 6.3.

Table 6.3: The two verification environment designs in comparison to the manual design.

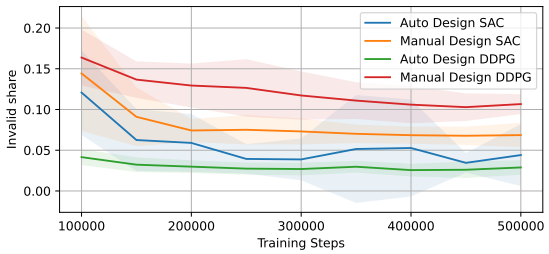
	Design Decision	Manual	Best Eco	Best Voltage
Reward	*Valid Reward*	0.0	0.88	0.97
	Invalid Penalty	0.0	1.11	0.57
	Invalid Obj. Share	1.0	0.8	0.47
	Penalty Weight	0.1/0.5	0.54	0.16
	Diff-Objective	False	True	True
Data	*Normal Data*	0%	23.79%	35.51%
	Uniform Data	0%	41.24%	28.69%
	Realistic Data	100%	34.97%	35.8%
Obs	*Add Voltage Mag.*	False	False	False
	Add Voltage Angle	False	True	False
	Add Line Loading	False	True	True
	Add Trafo Loading	False	True	True
	Add Slack Power	False	False	False
Episode	*Steps Per Episode*	1	1	1
Action	*Autoscaling*	True	True	True

Further, we now compute the performance metrics on the test datasets instead of the validation datasets. They were not used for automated design before, as it is good practice in HPO [50]. For the baseline design, we picked the penalty weight that resulted in the best *Utopia* performance in the previous training runs. That is a penalty weight of 0.5 for `EcoDispatch` and 0.1 for `VoltageControl`. All metrics are calculated as the mean of 10 different random seeds to consider the stochasticity of results [85, 91]. The resulting training curves are shown in Figure 6.7. The 2x2 subfigures show the performance regarding the two metrics *mean error* and *invalid share* for the two environments, respectively.

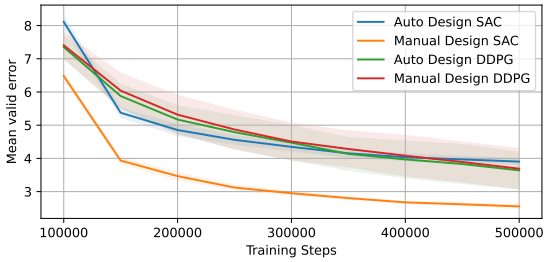
Regarding the *mean error* metric in the `VoltageControl` environment, the automated design outperforms the manual design for both RL algorithms. Regarding the *invalid share* metric, again, the automated design generally outperforms the baseline for both



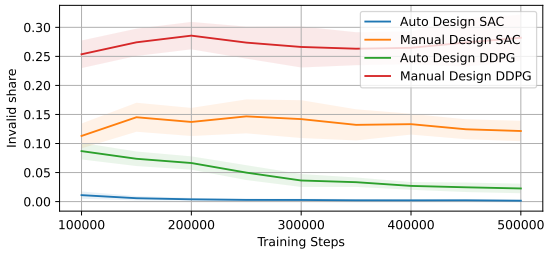
(a) `VoltageControl` with the *mean error* metric.



(b) `VoltageControl` with the *invalid share* metric.



(c) `EcoDispatch` with the *mean error* metric.



(d) `EcoDispatch` with the *invalid share* metric.

Figure 6.7: Learning curves for the two verification environments, `EcoDispatch` and `VoltageControl`, for both considered performance metrics and for both the DDPG and the SAC algorithm. The colored areas mark one standard deviation. All plots are created with a rolling average over two steps.

algorithms. However, the combination of automated design plus SAC resulted in multiple outliers, which can be deduced from the high standard deviation.

For the `EcoDispatch` environment, again, the automated design outperforms the manual design, however, with one exception. While the automated design plus SAC resulted in almost perfect constraint satisfaction, the optimization performance was the worst out of the four combinations. Hence, this is the only case where the automated design does not dominate the manual design regarding both metrics. However, it also does not get dominated by the baseline. The switch from DDPG to SAC resulted in a stronger focus on constraint satisfaction for the `EcoDispatch` environment. The exact reason for this behavior is out-of-scope for this work. Therefore, for this combination, we cannot draw a general conclusion that one environment design is superior to the other.

Altogether, we can conclude that the optimized environment designs from the multi-objective optimization reproducibly outperform and dominate the baseline design, if we use the same RL algorithm. However, the results also suggest that the optimized environment design is not fully transferable from one RL algorithm to the other, which indicates some

kind of overadjustment of the environment to the RL algorithm, similar to overfitting [72].

7 Discussion

This doctoral thesis investigated the importance of environment design for learning the OPF with RL. The following section will summarize this thesis' main findings, provide answers to its research questions, and discuss its contributions and limitations.

7.1 Research Questions and Findings

In section 1.2, we derived a general RQ regarding the overall importance of RL environment design for learning the OPF and subdivided it into three sub-RQs, which were used to structure this thesis. In the following, we will first discuss the three sub-RQs to then answer the general RQ of this thesis:

> **RQ1:** What are the characteristics, difficulties, and chances of the OPF as an RL problem formulation?

Chapter 4 discussed the general characteristics, difficulties, and chances of the RL-OPF problem. The main findings are twofold: First, while RL has some drawbacks compared to conventional solvers, like a lack of guarantees regarding constraint satisfaction or convergence, it holds noteworthy advantages regarding real-time capability, non-differentiable problems, sequential problems, stochastic problems, and others. Altogether, that makes RL another promising tool in the toolbox for solving future OPF problems in research or the real world. Second, when formulating the OPF as an RL problem, we have to consider multiple noteworthy special characteristics. For example, the high-dimensional state and action spaces, the strict constraints, and the lack of large-scale realistic datasets make the OPF challenging to solve. On the other hand, the dense feedback signal for RL, the availability of power system models, and the existing OPF baselines can prove helpful when solving the OPF with RL. These general findings provide an important basis for building OPF environments but also for solving the OPF with RL on a general level.

RQ2: How should a benchmark framework for OPF environments look like that ensures reproducible and comparable research with fixed benchmarks but also degrees of freedom for systematic environment design?

Chapter 5 presents the *OPF-Gym* framework and benchmarks developed for this thesis. To ensure reproducible and comparable research, *OPF-Gym* is open-source and provides five different fixed OPF benchmark problems for multiple common OPF use cases like the economic dispatch or voltage control. To enable the usage of the benchmarks and still allow for RL environment design, the developed framework strictly distinguishes between the fixed underlying OPF problem and its environment representation, which has various degrees of freedom for systematic environment design. By strictly separating the two parts of the RL environments, *OPF-Gym* makes it possible to modularly test new environment design representations without changing the underlying OPF problem to solve. To also provide a basis for extensive environment design analyses, *OPF-Gym* contains ready-to-use implementations of various environment design options, like different reward functions and data sampling methods. For example, multiple data sampling methods are implemented to investigate how to deal with the lack of realistic time-series data or the *Autoscaling* option to deal with dynamic constraints (compare section 4).

RQ3: How to formulate OPF problems as RL environments for maximum learning performance, including constraint satisfaction, optimization performance, and learning speed?

Considering that no systematic environment design methods exist in the RL literature (see section 3.4), chapter 6 proposes a general methodology for RL environment design, which is applicable to the OPF but also for the general case of RL environment design. Considering the large amount of potential OPF environment design options (see section 5.4), an automated optimization-based approach is preferred over manual environment design. Hence, the RL-OPF environment design problem is formulated as a multi-objective HPO problem. Utilizing the HPO framework allows for reusing the vast existing HPO algorithm and knowledge base, which results in simplicity and generality of the methodology. The utilization of multi-objective optimization allows for considering both constraint satisfaction and optimization performance explicitly. Learning speed was given lower priority by implicitly considering it with short training times.

To verify the proposed environment design methodology, the HPO-based optimization was applied to the five benchmark problems and the 22-dimensional environment design

space of *OPF-Gym*, in comparison to a manually derived design from the pre-study to this thesis in [1]. The experiments demonstrated that the HPO-derived environment designs outperform the manual design. Further, it is shown how statistical analyses can be used to derive general findings on which RL-OPF environment design decisions are statistically significant for these performance differences. For example, the results indicate that *Autoscaling* actions and the *Diff-Objective* reward consistently improve performance. Finally, the derived environment designs, their performance gains, and their reproducibility were verified by separate training runs and testing on the unseen test dataset.

In conclusion, chapter 6.1 approached RQ3 by successfully developing a general methodology for RL environment design, verifying its performance, and providing statistically significant findings on how OPF environments should look like.

General RQ: What is the importance of RL environment design for solving the OPF with RL, regarding both optimization performance and constraint satisfaction?

On a general level, this thesis' results clearly indicate that environment design plays a significant role when solving the OPF as an RL problem, as it was already shown for other domains like locomotion [26]. Using the same RL algorithm with the same hyperparameters, the HPO-designed environments strictly outperform the manual baseline design. This is true for both constraint satisfaction and optimization performance by achieving non-dominated solutions. The results also suggest that only a few design variables are mainly responsible for the performance differences, which aligns well with similar observations in the HPO literature where usually few hyperparameters influence performance significantly, too [92]. In conclusion, while environment design is important for RL training performance on a general level, singular design variables can potentially be neglected when creating RL environments.

7.2 Limitations and Outlook

While this thesis resulted in important finding on RL environment design and solving the OPF with RL, it also left multiple questions unanswered and open for future work.

OPF-Gym was developed for this thesis to establish benchmark RL-OPF environments for comparability and reproducibility and as a general framework for other researchers to create their own OPF environments. While *OPF-Gym* allows for almost arbitrary OPF formulations, this thesis focused completely on standard OPF cases and neglected advanced problems like the stochastic OPF or the security-constrained OPF. In future work, to enable

real-world application in the future, these more advanced problems need to be considered explicitly, which can be expected to result in further research problems to be solved. Hence, the set of benchmark environments should be extended with examples of advanced OPF problems to provide a universal set of RL-OPF environments.

Chapter 6 proposed to use HPO as a general framework for RL environment design. While utilizing the HPO framework and algorithms results in reusability of existing algorithms and methods, it also inherits the drawbacks of HPO. The most relevant one is the required computational effort [50]. Overall, 300 RL training runs per environment were performed for the automated environment design. That is especially problematic considering that RL in itself is considered to be computationally costly [42]. To reduce computation times, there are three noteworthy options: First, the OPF use case required multi-objective optimization. However, in most practical cases, we care about a single performance metric and can use single-objective optimization, which simplifies the problem. Second, while this work considered overall 22 different design variables, we found that only a few of them impact performance in significant ways. That suggests that the computation times can be reduced by using a smaller search space with only the most relevant design decisions. Third, since we utilized the HPO framework, we can perform the optimization of the agent's hyperparameters together with the environment design variables. That can be expected to be more efficient than performing both subsequently. In summary, while the HPO-based environment design is computationally expensive, there are multiple potential countermeasures, which can be investigated in future work.

The verification results in section 6.4 suggest that an optimized environment design for one RL algorithm is not necessarily transferable to other algorithms. One important advantage of the RL framework is its modularity, which comes from the agent/environment split. If the performance drops by switching to another RL algorithm, as the results indicate, that advantage no longer exists. Therefore, making the environment design robust to subsequent algorithm changes is an important step for future work.

The main goal of this thesis was to investigate the importance of environment design for learning the OPF with RL, which inherently restricts the scope and significance of the results to RL as the OPF solver. However, as discussed in chapter 4, RL is not strictly superior to other approaches like conventional solvers, meta-heuristics, or other ML approaches (see section 3.2). Hence, this thesis provided important results on how to apply the potential tool RL for solving the OPF, but discussed only theoretically when to apply it. A large-scale empirical comparison of conventional solvers vs. meta-heuristics vs. ML-based approaches is still missing in the OPF literature and would be an important basis to make justified decisions on when to use which algorithmic tool.

Similarly, the focus of this thesis was mainly on the energy domain. While the automated

environment design methodology was designed to be generally applicable to all kinds of RL problems, this was not further verified. The same is true for the specific environment design decisions investigated in this thesis. For example, it was shown that the action *Autoscaling* achieves significantly better performance than simply clipping invalid actions (see section 6.3). Do these results transfer to the general case of dynamic constraints and, therefore, to other domains? The same is true for the other results. Should we generally use *Diff-Objective* when solving an optimization problem with RL? Should we generally add some share of randomly sampled environment states to our training, when the available dataset of realistic states is limited? To answer these questions on a general level, large-scale experiments over multiple domains and various RL problems are required. Potentially, this can yield general findings on how to design RL environments.

8 Conclusion

The OPF is an important optimization problem to enable more efficient and stable power grid operation and planning. Training deep neural networks with RL is a promising approach to solving even advanced and large-scale OPF problems in real-time. However, the design of the OPF problem representation as an RL environment has not been investigated in the literature yet. Hence, this thesis explores the importance of RL environment design for learning the OPF with RL.

The main contributions are as follows:

- This thesis provides **theoretical insights** on when to apply RL to the OPF, what its advantages and disadvantages are compared to conventional approaches, and what to consider when formulating the OPF problem as an RL environment.
- It provides a **research software foundation** for the RL-OPF by presenting the open-source *OPF-Gym* benchmark framework. *OPF-Gym* allows for easy creation of RL-OPF environments, offers five fixed benchmark environments for better comparability and reproducibility of research, and offers various implementations of environment design options to choose from.
- It proposes a **general RL environment design methodology** and applies it to the five benchmark problems. The results show the superiority of the automated design vs. a manually derived baseline environment, demonstrate the importance of environment design for overall performance, and identify multiple design decisions that are especially important for the RL-OPF problem, thus providing **important insights on OPF environment design**.

Considering that the proposed methodology is generally applicable to all kinds of environment design problems, this work also has implications for RL research outside the energy domain, e.g., locomotion [26]. The same is true for the identified significant design decisions in this work, which might be transferable to similar problems in other domains, hence motivating further research.

In conclusion, this thesis establishes a foundation for reproducible and comparable RL-based OPF research by introducing *OPF-Gym*, the first benchmark framework for RL-

OPF environments. Furthermore, the proposed environment design methodology presents the first general approach to automated RL environment design and optimization. By demonstrating its effectiveness across multiple benchmark problems, this work paves the way for more efficient, scalable, and systematically designed RL environments in energy system optimization and for RL research in general.

A Appendix

A.1 Hyperparameter choices

Table A.1: DDPG and SAC hyperparameter choices for chapter 6.

Hyperparameter	DDPG	SAC
Actor layers/neurons	(256, 256, 256)	(256, 256, 256)
Critic layers/neurons	(256, 256, 256)	(256, 256, 256)
Actor Learning rate	0.0001	0.0001
Critic Learning rate	0.0005	0.0005
Batch size	256	256
Gamma	0.9	0.9
Memory size	1000000	1000000
Noise standard deviation	0.1	N.A.
Start train at step	2000	2000
Tau	0.001	0.001
Entropy learning rate	N.A.	0.0001

A.2 Detailed Pareto fronts from the automated design

The following Figures A.1, A.2, and A.3 show the Pareto front plots of the three environments that have not been explicitly discussed in section 6.2.

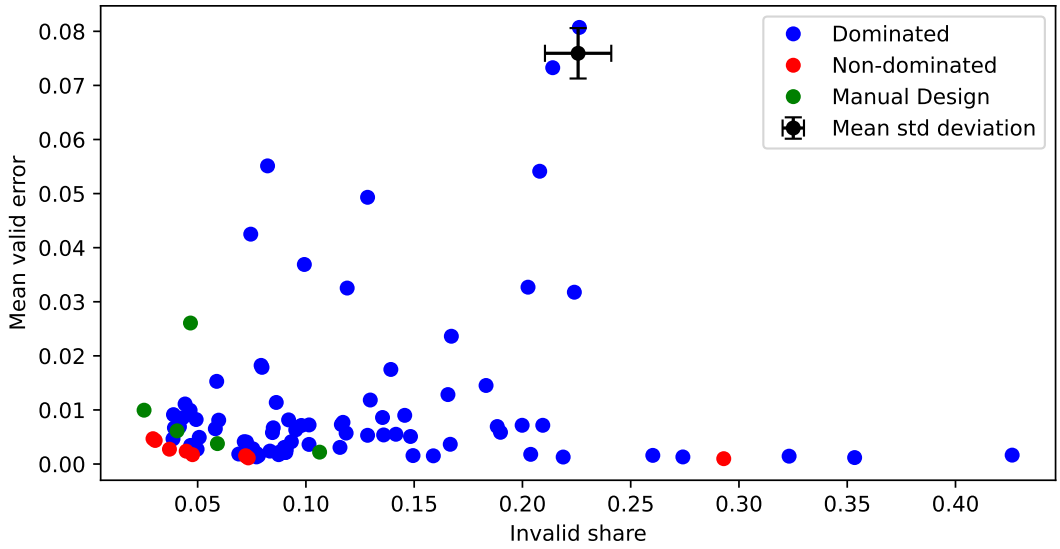


Figure A.1: `QMarket`: Dominated and non-dominated samples from the HPO in comparison with the baseline solutions. Note: Singular extreme outliers were clipped for the plot.

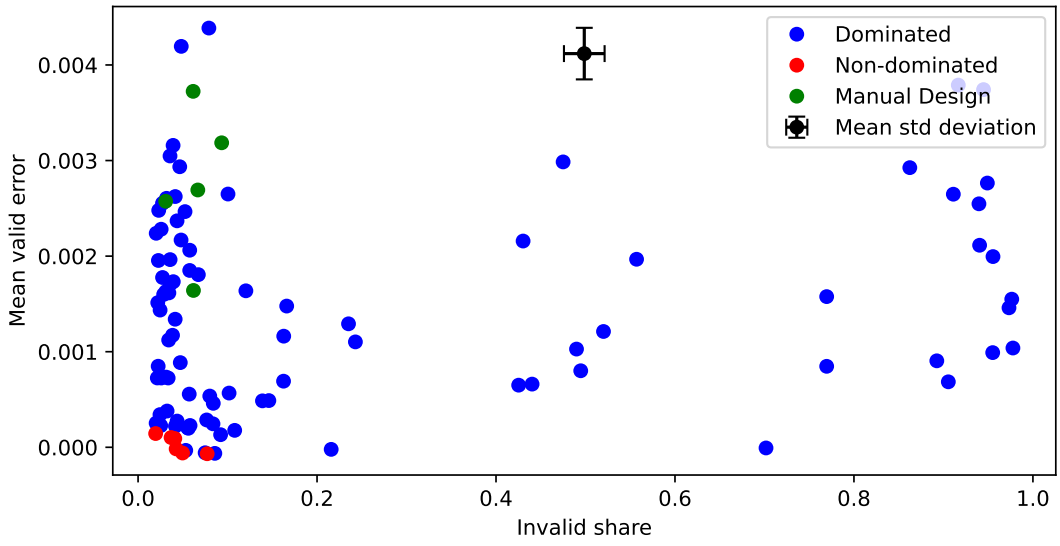


Figure A.2: `VoltageControl`: Dominated and non-dominated samples from the HPO in comparison with the baseline solutions.

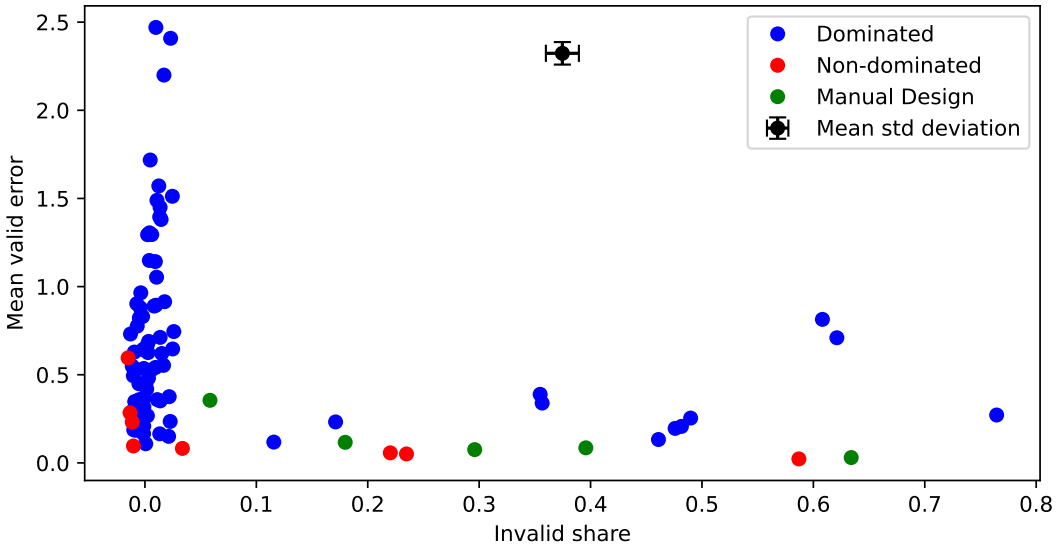


Figure A.3: `MaxRenewable`: Dominated and non-dominated samples from the HPO in comparison with the baseline solutions.

A.3 Detailed statistically significant design decisions

The following Figures A.4, A.5, and A.6 show the annotated Pareto front plots of the three environments that have not been explicitly discussed in section 6.3.2.

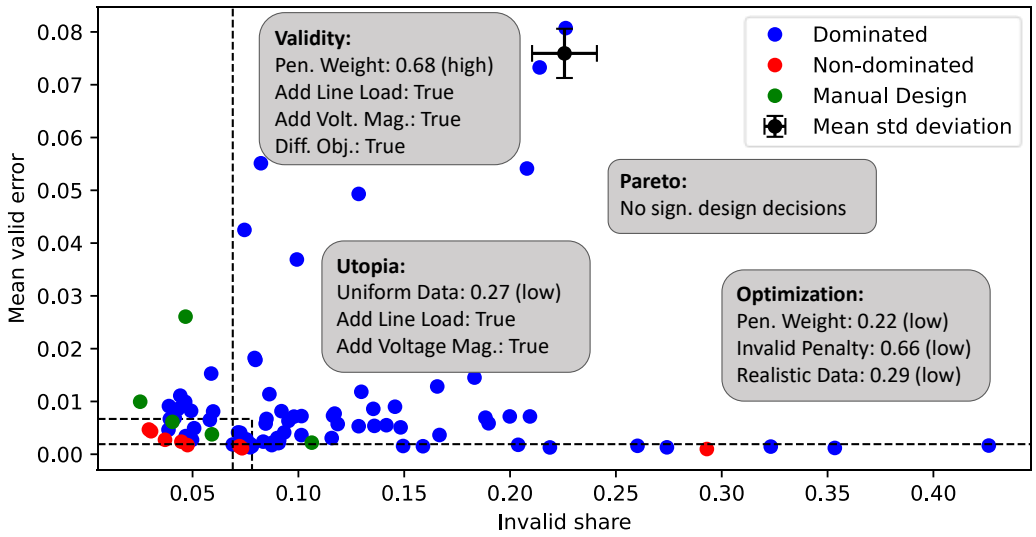


Figure A.4: `QMarket`: Statistically significant environment design decisions for all four evaluation criteria, respectively. Note: Singular extreme outliers were clipped for the plot.

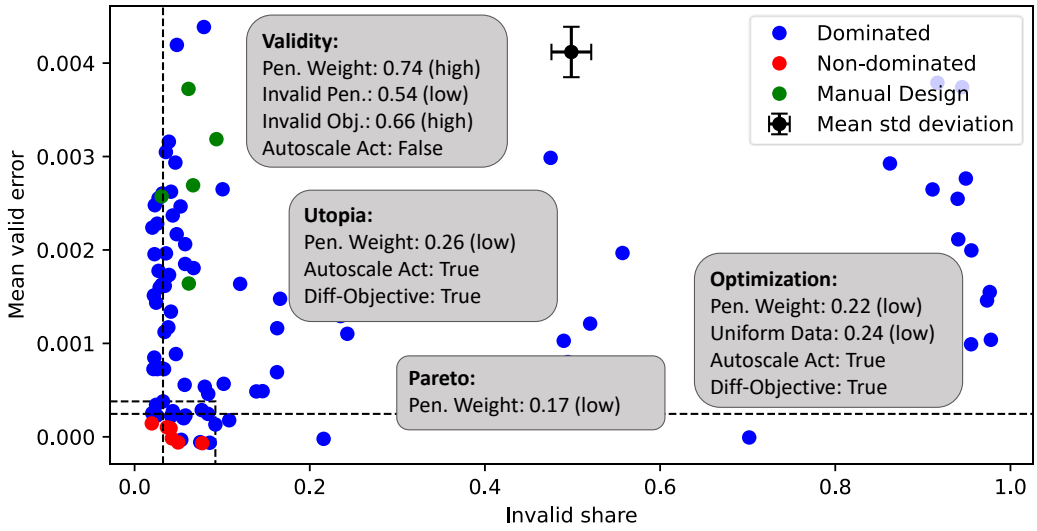


Figure A.5: `VoltageControl`: Statistically significant environment design decisions for all four evaluation criteria, respectively.

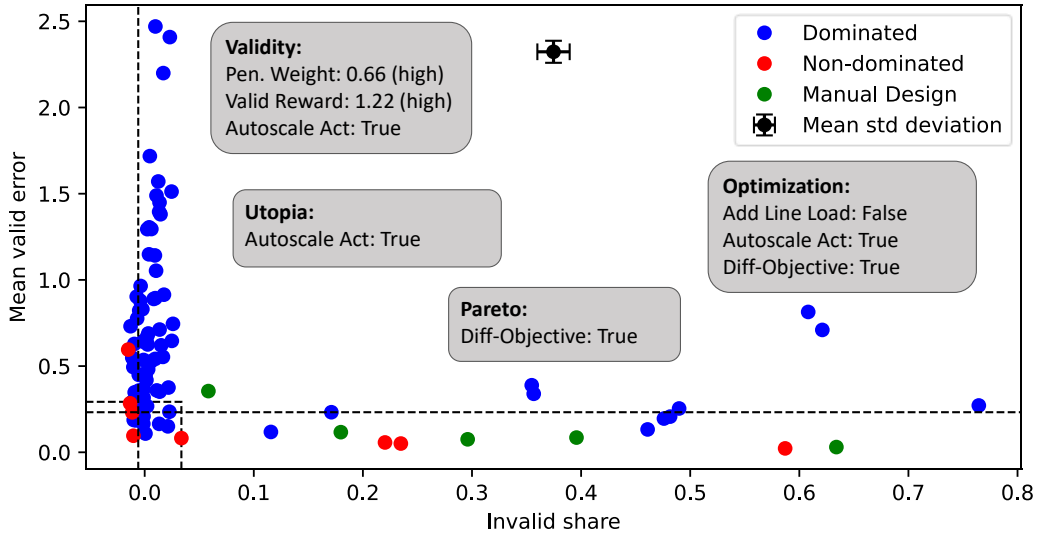


Figure A.6: `MaxRenewable`: Statistically significant environment design decisions for all four evaluation criteria, respectively.

List of Figures

List of Tables

Bibliography

[1] Wolgast, T., Nieße, A.: Learning the optimal power flow: Environment design matters. Energy and AI, 100410 (2024). doi:10.1016/j.egyai.2024.100410

[2] Wolgast, T.: Whitepaper: Environment Design for Reinforcement Learning: A Practical Guide and Overview (2024). doi:10.13140/RG.2.2.28673.77925

[3] Wolgast, T., Nieße, A.: A general approach of automated environment design for learning the optimal power flow. In: Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems. E-Energy '25, pp. 108–121. Association for Computing Machinery, New York, NY, USA (2025). doi:10.1145/3679240.3734626

[4] Wolgast, T.: OPF-Gym: A benchmark environment framework for learning the optimal power flow (2025). `https://github.com/Digitalized-Energy-Systems/opfgym`

[5] Wolgast, T., Nieße, A.: Towards modular composition of agent-based voltage control concepts. Energy Informatics **2**(S1), 26 (2019). doi:10.1186/s42162-019-0079-x

[6] Wolgast, T., Ferenz, S., Nieße, A.: Reactive Power Markets: A Review. IEEE Access **10**, 28397–28410 (2022). doi:10.1109/ACCESS.2022.3141235

[7] Bozionek, J., Wolgast, T., Nieße, A.: Design and evaluation of a multi-level reactive power market. Energy Informatics **5**(1), 6 (2022). doi:10.1186/s42162-022-00191-x

[8] Wolgast, T., Veith, E.M., Nieße, A.: Towards reinforcement learning for vulnerability analysis in power-economic systems. Energy Informatics **4**(3), 21 (2021). doi:10.1186/s42162-021-00181-5

[9] Wolgast, T., Nieße, A.: Approximating Energy Market Clearing and Bidding With Model-Based Reinforcement Learning. IEEE Access **12**, 145106–145117 (2024). doi:10.1109/ACCESS.2024.3472480

[10] Ferenz, S., Frost, E., Schrage, R., Wolgast, T., Beyers, I., Karras, O., Werth, O., Nieße, A.: Ten recommendations for engineering research software in energy research. In: Proceedings of the 16th ACM International Conference on Future and Sustainable

Energy Systems. E-Energy '25, pp. 446–459. Association for Computing Machinery, New York, NY, USA (2025). doi:10.1145/3679240.3734606

[11] Frank, S., Steponavice, I., Rebennack, S.: Optimal power flow: A bibliographic survey I. Energy Systems **3**(3), 221–258 (2012). doi:10.1007/s12667-012-0056-y

[12] Cain, M.B., O'neill, R.P., Castillo, A.: History of optimal power flow and formulations. Federal Energy Regulatory Commission **1**, 1–36 (2012)

[13] Kotary, J., Fioretto, F., Van Hentenryck, P., Wilder, B.: End-to-End Constrained Optimization Learning: A Survey. arXiv (2021). 2103.16378. doi:10.48550/arXiv.2103.16378

[14] Nie, H., Chen, Y., Song, Y., Huang, S.: A General Real-time OPF Algorithm Using DDPG with Multiple Simulation Platforms. In: 2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia), pp. 3713–3718 (2019). doi:10.1109/ISGT-Asia.2019.8881174

[15] Kawaguchi, K., Huang, J., Kaelbling, L.P.: Effect of Depth and Width on Local Minima in Deep Learning. Neural Computation **31**(7), 1462–1498 (2019). doi:10.1162/neco_a_01195

[16] Huang, W., Chen, M., Low, S.H.: Unsupervised Learning for Solving AC Optimal Power Flows: Design, Analysis, and Experiment. IEEE Transactions on Power Systems, 1–13 (2024). doi:10.1109/TPWRS.2024.3373399

[17] Khaloie, H., Dolanyi, M., Toubeau, J.-F., Vallée, F.: Review of machine learning techniques for optimal power flow. Applied Energy **388**, 125637 (2025). doi:10.1016/j.apenergy.2025.125637

[18] Zhou, Y., Zhang, B., Xu, C., Lan, T., Diao, R., Shi, D., Wang, Z., Lee, W.-J.: A Data-driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning. Journal of Modern Power Systems and Clean Energy **8**(6), 1128–1139 (2020). doi:10.35833/MPCE.2020.000522

[19] Owerko, D., Gama, F., Ribeiro, A.: Unsupervised Optimal Power Flow Using Graph Neural Networks. In: ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6885–6889 (2024). doi:10.1109/ICASSP48485.2024.10446827

[20] Wang, J., Srikantha, P.: Fast Optimal Power Flow With Guarantees via an Unsupervised Generative Model. IEEE Transactions on Power Systems **38**(5), 4593–4604 (2023). doi:10.1109/TPWRS.2022.3212925

[21] Yan, Z., Xu, Y.: Real-Time Optimal Power Flow: A Lagrangian Based Deep Reinforcement Learning Approach. IEEE Transactions on Power Systems **35**(4), 3270–3273 (2020). doi:10.1109/TPWRS.2020.2987292

[22] Woo, J.H., Wu, L., Park, J.-B., Roh, J.H.: Real-Time Optimal Power Flow Using Twin Delayed Deep Deterministic Policy Gradient Algorithm. IEEE Access **8**, 213611–213618 (2020). doi:10.1109/ACCESS.2020.3041007

[23] Yi, Z., Wang, X., Yang, C., Yang, C., Niu, M., Yin, W.: Real-Time Sequential Security-Constrained Optimal Power Flow: A Hybrid Knowledge-Data-Driven Reinforcement Learning Approach. IEEE Transactions on Power Systems **39**(1), 1664–1680 (2024). doi:10.1109/TPWRS.2023.3262843

[24] Wu, Y., Ye, Y., Hu, J., Zhao, P., Liu, L., Strbac, G., Kang, C.: Chance Constrained MDP Formulation and Bayesian Advantage Policy Optimization for Stochastic Dynamic Optimal Power Flow. IEEE Transactions on Power Systems **39**(5), 6788–6791 (2024). doi:10.1109/TPWRS.2024.3430650

[25] Wu, T., Scaglione, A., Arnold, D.: Constrained Reinforcement Learning for Predictive Control in Real-Time Stochastic Dynamic Optimal Power Flow. IEEE Transactions on Power Systems **39**(3), 5077–5090 (2024). doi:10.1109/TPWRS.2023.3326121

[26] Reda, D., Tao, T., van de Panne, M.: Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning. In: Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games. MIG '20, pp. 1–10. Association for Computing Machinery, New York, NY, USA (2020). doi:10.1145/3424636.3426907. https://dl.acm.org/doi/10.1145/3424636.3426907

[27] Kim, J.T., Ha, S.: Observation Space Matters: Benchmark and Optimization Algorithm. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 1527–1534 (2021). doi:10.1109/ICRA48506.2021.9561019

[28] Kanervisto, A., Scheller, C., Hautamäki, V.: Action Space Shaping in Deep Reinforcement Learning. In: 2020 IEEE Conference on Games (CoG), pp. 479–486 (2020). doi:10.1109/CoG47356.2020.9231687

[29] Ng, A.Y., Harada, D., Russell, S.J.: Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning. ICML '99, pp. 278–287. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)

[30] Pardo, F., Tavakoli, A., Levdik, V., Kormushev, P.: Time Limits in Reinforcement Learning. In: Proceedings of the 35th International Conference on Machine Learning, pp. 4045–4054. PMLR, (2018). https://proceedings.mlr.press/v80/pardo18a.html

[31] Sun, H., Guo, Q., Qi, J., Ajjarapu, V., Bravo, R., Chow, J., Li, Z., Moghe, R., Nasr-Azadani, E., Tamrakar, U., Taranto, G.N., Tonkoski, R., Valverde, G., Wu, Q., Yang, G.: Review of Challenges and Research Opportunities for Voltage Control in Smart Grids. IEEE Transactions on Power Systems **34**(4), 2790–2801 (2019). doi:10.1109/TPWRS.2019.2897948

[32] Capitanescu, F.: Critical review of recent advances and further developments needed in AC optimal power flow. Electric Power Systems Research **136**, 57–68 (2016). doi:10.1016/j.epsr.2016.02.008

[33] Stott, B., Alsac, O.: Optimal Power Flow - Basic Requirements for Real-Life Problems and Their Solutions. In: SEPOPE XII Symposium, Rio de Janeiro, Brazil (2012). https://www.ieee.hr/_download/repository/Stott-Alsac-OPF-White-Paper.pdf

[34] Faulwasser, T., Engelmann, A., Mühlpfordt, T., Hagenmeyer, V.: Optimal power flow: an introduction to predictive, distributed and stochastic control challenges. at - Automatisierungstechnik **66**(7), 573–589 (2018). doi:10.1515/auto-2018-0040

[35] Frank, S., Steponavice, I., Rebennack, S.: Optimal power flow: A bibliographic survey II. Energy Systems **3**(3), 259–289 (2012). doi:10.1007/s12667-012-0057-x

[36] Pandey, A., Almassalkhi, M.R., Chevalier, S.: Large-Scale Grid Optimization: The Workhorse of Future Grid Computations. Current Sustainable/Renewable Energy Reports **10**(3), 139–153 (2023). doi:10.1007/s40518-023-00213-6

[37] Géron, A.: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2. edition edn. " O'Reilly Media, Inc.", (2019). https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

[38] Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015). doi:10.1126/science.aaa8415. https://www.science.org/doi/pdf/10.1126/science.aaa8415

[39] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems **2**(4), 303–314 (1989). doi:10.1007/BF02551274

[40] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, (2016). http://www.deeplearningbook.org

[41] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, Second edition edn. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts (2018)

[42] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: Proceedings of the 35th International Conference on Machine Learning, pp. 1861–1870. PMLR, (2018). https://proceedings.mlr.press/v80/haarnoja18b.html

[43] Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3-4), 279–292 (1992). doi:10.1007/BF00992698

[44] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. arXiv (2013). 1312.5602. doi:10.48550/arXiv.1312.5602

[45] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous Control with Deep Reinforcement Learning. arXiv (2016). 1509.02971. doi:10.48550/arXiv.1509.02971. http://arxiv.org/abs/1509.02971

[46] OpenAI: Spinning Up in Deep RL - Deep Deterministic Policy Gradient. https://spinningup.openai.com/en/latest/algorithms/ddpg.html. [Online; accessed 2024-06-20]

[47] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S.: Soft Actor-Critic Algorithms and Applications. arXiv (2019). 1812.05905. doi:10.48550/arXiv.1812.05905

[48] Fujimoto, S., Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. In: Proceedings of the 35th International Conference on Machine Learning, pp. 1587–1596. PMLR, (2018). https://proceedings.mlr.press/v80/fujimoto18a.html

[49] Hasselt, H.: Double Q-learning. In: Advances in Neural Information Processing Systems, vol. 23. Curran Associates, Inc., (2010).

https://proceedings.neurips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html

[50] Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., Lindauer, M.: Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. WIREs Data Mining and Knowledge Discovery **13**(2), 1484 (2023). doi:10.1002/widm.1484

[51] Zhen, H., Hefeng, Z., Weizhe, M., Zhao, L., Yixuan, W., Yuan, X., Jun, S., Xiaofeng, H.: Design and tests of reinforcement-learning-based optimal power flow solution generator. Energy Reports (2021). doi:10.1016/j.egyr.2021.11.126

[52] Nie, J., Liu, Y., Zhou, L., Jiang, X., Preindl, M.: Deep Reinforcement Learning Based Approach for Optimal Power Flow of Microgrid with Grid Services Implementation. In: 2022 IEEE Transportation Electrification Conference & Expo (ITEC), pp. 1148–1153. IEEE, Anaheim, CA, USA (2022). doi:10.1109/ITEC53557.2022.9813862

[53] Cao, D., Hu, W., Xu, X., Wu, Q., Huang, Q., Chen, Z., Blaabjerg, F.: Deep Reinforcement Learning Based Approach for Optimal Power Flow of Distribution Networks Embedded with Renewable Energy and Storage Devices. Journal of Modern Power Systems and Clean Energy **9**(5), 1101–1110 (2021). doi:10.35833/MPCE.2020.000557

[54] Zhou, Y., Lee, W.-J., Diao, R., Shi, D.: Deep Reinforcement Learning Based Real-time AC Optimal Power Flow Considering Uncertainties. Journal of Modern Power Systems and Clean Energy **10**(5), 1098–1109 (2022). doi:10.35833/MPCE.2020.000885

[55] Liu, X., Fan, B., Tian, J.: Deep Reinforcement Learning Based Approach for Dynamic Optimal Power Flow in Active Distribution Network. In: 2022 41st Chinese Control Conference (CCC), pp. 1951–1956 (2022). doi:10.23919/CCC55666.2022.9902611

[56] Jiang, B., Wang, Q., Wu, S., Wang, Y., Lu, G.: Advancements and Future Directions in the Application of Machine Learning to AC Optimal Power Flow: A Critical Review. Energies **17**(6), 1381 (2024). doi:10.3390/en17061381

[57] Liu, C., Li, Y., Xu, T.: A Neural Network Approach to Physical Information Embedding for Optimal Power Flow. Sustainability **16**(17), 7498 (2024). doi:10.3390/su16177498

[58] Park, S., Chen, W., Mak, T.W.K., Van Hentenryck, P.: Compact Optimization Learning for AC Optimal Power Flow. IEEE Transactions on Power Systems **39**(2), 4350–4359 (2024). doi:10.1109/TPWRS.2023.3313438

[59] Zhou, M., Chen, M., Low, S.H.: DeepOPF-FT: One Deep Neural Network for Multiple AC-OPF Problems With Flexible Topology. IEEE Transactions on Power Systems **38**(1), 964–967 (2023). doi:10.1109/TPWRS.2022.3217407

[60] Henry, R., Ernst, D.: Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems. Energy and AI **5**, 100092 (2021). doi:10.1016/j.egyai.2021.100092

[61] Cui, H., Zhang, Y.: Andes_gym: A Versatile Environment for Deep Reinforcement Learning in Power Systems. In: 2022 IEEE Power & Energy Society General Meeting (PESGM), pp. 01–05 (2022). doi:10.1109/PESGM48719.2022.9916967

[62] Cui, H., Li, F., Tomsovic, K.: Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis. IEEE Transactions on Power Systems **36**(2), 1373–1384 (2021). doi:10.1109/TPWRS.2020.3017019

[63] Marot, A., Donnot, B., Dulac-Arnold, G., Kelly, A., O'Sullivan, A., Viebahn, J., Awad, M., Guyon, I., Panciatici, P., Romero, C.: Learning to run a Power Network Challenge: A Retrospective Analysis. In: Proceedings of the NeurIPS 2020 Competition and Demonstration Track, pp. 112–132. PMLR, (2021). https://proceedings.mlr.press/v133/marot21a.html

[64] Babaeinejadsarookolaee, S., Birchfield, A., Christie, R.D., Coffrin, C., DeMarco, C., Diao, R., Ferris, M., Fliscounakis, S., Greene, S., Huang, R., Josz, C., Korab, R., Lesieutre, B., Maeght, J., Mak, T.W.K., Molzahn, D.K., Overbye, T.J., Panciatici, P., Park, B., Snodgrass, J., Tbaileh, A., Van Hentenryck, P., Zimmerman, R.: The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms. arXiv (2021). 1908.02788. doi:10.48550/arXiv.1908.02788

[65] Joswig-Jones, T., Baker, K., Zamzam, A.S.: OPF-learn: An open-source framework for creating representative AC optimal power flow datasets. In: 2022 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), pp. 1–5. IEEE, (2022)

[66] Heid, S., Weber, D., Bode, H., Hüllermeier, E., Wallscheid, O.: OMG: A Scalable and Flexible Simulation and Testing Environment Toolbox for Intelligent Microgrid Control. Journal of Open Source Software **5**(54), 2435 (2020). doi:10.21105/joss.02435

[67] Henri, G., Levent, T., Halev, A., Alami, R., Cordier, P.: Pymgrid: An Open-Source Python Microgrid Simulator for Applied Artificial Intelligence Research. arXiv (2020). 2011.08004. doi:10.48550/arXiv.2011.08004

[68] Hou, S., Gao, S., Xia, W., Duque, E.M.S., Palensky, P., Vergara, P.P.: RL-ADN: A High-Performance Deep Reinforcement Learning Environment for Optimal Energy Storage Systems Dispatch in Active Distribution Networks. arXiv (2024). 2408.03685. doi:10.48550/arXiv.2408.03685

[69] Yeh, C., Li, V., Datta, R., Arroyo, J., Christianson, N., Zhang, C., Chen, Y., Hosseini, M.M., Golmohammadi, A., Shi, Y., Yue, Y., Wierman, A.: SustainGym: Reinforcement Learning Environments for Sustainable Energy Systems. Advances in Neural Information Processing Systems **36**, 59464–59476 (2023)

[70] Peng, X.B., van de Panne, M.: Learning locomotion skills using DeepRL: Does the choice of action space matter? In: Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation. SCA '17, pp. 1–13. Association for Computing Machinery, New York, NY, USA (2017). doi:10.1145/3099564.3099567

[71] Yang, M., Nachum, O.: Representation Matters: Offline Pretraining for Sequential Decision Making. In: Proceedings of the 38th International Conference on Machine Learning, pp. 11784–11794. PMLR, (2021). https://proceedings.mlr.press/v139/yang21h.html

[72] Zhang, C., Vinyals, O., Munos, R., Bengio, S.: A Study on Overfitting in Deep Reinforcement Learning. arXiv (2018). 1804.06893. doi:10.48550/arXiv.1804.06893. http://arxiv.org/abs/1804.06893

[73] Towers, M., Terry, J.K., Kwiatkowski, A., Balis, J.U., Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Tan, A.J.S., Younis, O.G.: Gymnasium. Zenodo (2024). doi:10.5281/zenodo.10655021

[74] Zhang, L., Zhang, Q., Shen, L., Yuan, B., Wang, X., Tao, D.: Evaluating Model-Free Reinforcement Learning toward Safety-Critical Tasks. Proceedings of the AAAI Conference on Artificial Intelligence **37**(12), 15313–15321 (2023). doi:10.1609/aaai.v37i12.26786

[75] Venzke, A., Qu, G., Low, S., Chatzivasileiadis, S.: Learning Optimal Power Flow: Worst-Case Guarantees for Neural Networks. In: 2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), pp. 1–7 (2020). doi:10.1109/SmartGridComm47815.2020.9302963

[76] Nellikkath, R., Tanneau, M., Hentenryck, P.V., Chatzivasileiadis, S.: Scalable Exact Verification of Optimization Proxies for Large-Scale Optimal Power Flow. arXiv (2024). 2405.06109. doi:10.48550/arXiv.2405.06109

[77] Li, B., Tang, H., Zheng, Y., Hao, J., Li, P., Wang, Z., Meng, Z., Wang, L.I.: HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation. In: International Conference on Learning Representations (2021). https://openreview.net/forum?id=64trBbOhdGU

[78] Thurner, L., Scheidler, A., Schäfer, F., Menke, J.-H., Dollichon, J., Meier, F., Meinecke, S., Braun, M.: Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. IEEE Transactions on Power Systems **33**(6), 6510–6521 (2018). doi:10.1109/TPWRS.2018.2829021

[79] Meinecke, S., Sarajlić, D., Drauz, S.R., Klettke, A., Lauven, L.-P., Rehtanz, C., Moser, A., Braun, M.: SimBench—A Benchmark Dataset of Electric Power Systems to Compare Innovative Solutions Based on Power Flow Analysis. Energies **13**(12), 3290 (2020). doi:10.3390/en13123290

[80] van Hasselt, H.P., Guez, A., Guez, A., Hessel, M., Mnih, V., Silver, D.: Learning values across many orders of magnitude. In: Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc., (2016). https://proceedings.neurips.cc/paper/2016/hash/5227b6aaf294f5f027273aebf16015f2-Abstract.html

[81] Bietti, A., Agarwal, A., Langford, J.: A Contextual Bandit Bake-off. Journal of Machine Learning Research **22**(133), 1–49 (2021)

[82] Samimi, A., Kazemi, A., Siano, P.: Economic-environmental active and reactive power scheduling of modern distribution systems in presence of wind generations: A distribution market-based approach. Energy Conversion and Management **106**, 495–509 (2015). doi:10.1016/j.enconman.2015.09.070

[83] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19, pp. 2623–2631. Association for Computing Machinery, New York, NY, USA (2019). doi:10.1145/3292500.3330701

[84] Deb, K., Jain, H.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems

With Box Constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014). doi:10.1109/TEVC.2013.2281535

[85] Eimer, T., Lindauer, M., Raileanu, R.: Hyperparameters in Reinforcement Learning and How To Tune Them. In: Proceedings of the 40th International Conference on Machine Learning, pp. 9104–9149. PMLR, (2023). https://proceedings.mlr.press/v202/eimer23a.html

[86] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous Control with Deep Reinforcement Learning. arXiv (2016). 1509.02971. doi:10.48550/arXiv.1509.02971

[87] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv (2017). 1707.06347. doi:10.48550/arXiv.1707.06347

[88] Welch, B.L.: The generalization of 'STUDENT'S'problem when several different population varlances are involved. Biometrika **34**(1-2), 28–35 (1947)

[89] Pearson, K.: X. *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling*. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **50**(302), 157–175 (1900). doi:10.1080/14786440009463897

[90] Fisher, R.A.: Statistical Methods for Research Workers. In: Kotz, S., Johnson, N.L. (eds.) Breakthroughs in Statistics: Methodology and Distribution, pp. 66–70. Springer, New York, NY (1992). doi:10.1007/978-1-4612-4380-9_6

[91] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI'18/IAAI'18/EAAI'18, pp. 3207–3214. AAAI Press, New Orleans, Louisiana, USA (2018)

[92] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)

Statement of Originality

I hereby confirm in lieu of oath, that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgments. Furthermore, the work was not carried out using unrecognisable generative AI. I certify that I have followed the general principles of scientific work and publications as written in the guidelines of good research of the Carl von Ossietzky Universität Oldenburg. This work has not yet been submitted to any examination office in the same or similar form.

On the usage of AI: To some extent, the work on this thesis was supported by AI tools like *ChatGPT*, *GitHub Copilot*, and *Grammarly*. Mainly, AI was used during programming to autocomplete and to create first drafts of new implementations. For the writing of this document, AI tools were used to propose the structuring of chapters and for grammar checking.

Oldenburg (Germany), 10.09.2025 Thomas Wolgast