

# Safer AI via Exploiting the Structure of Learned Systems for Monitoring, Verification, Abstraction, Representations, and Explainability

**Stefanie Mohr**

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

**Doktors der Naturwissenschaften (Dr. rer. nat.)**

genehmigten Dissertation.

**Vorsitz:** Prof. Dr.-Ing. Matthias Althoff

**Prüfende der Dissertation:**

1. Prof. Dr. Jan Křetínský
2. Assoc. Prof. Dr. Guillermo Pérez
3. Prof. Dr. Nils Jansen

Die Dissertation wurde am 15.07.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 18.03.2025 angenommen.



Für Ben und Christoph





# Abstract

With the rise of machine learning and artificial intelligence (AI), ensuring their safety becomes paramount. One can do so by applying verification, monitoring, and explainability, each contributing to increase the trust and safety in AI systems. This thesis presents solution approaches addressing these aspects.

**Safe Neural Networks (NNs).** Verification of NNs is typically conducted offline on a fixed set of inputs, providing a reliability certificate for these inputs. However, standard verification methods do not scale to real-world problems. To address this issue, we demonstrate how the *abstraction* of NNs reduces the size of the verification problem. Abstraction creates a more compact representation of the NN while maintaining the verification problem. It does so by providing an error bound on the difference. Additionally, since verification is limited to specific inputs, we also focus on *monitoring*. This monitoring observes the NN’s behavior during runtime and detects when it encounters novel inputs. We introduce the *Gaussian* monitor as a lightweight and compact monitoring method. Additionally, we introduce our tool, MONITIZER, which facilitates the evaluation and optimization of monitors. Lastly, we showcase the need for NN verification tools through a real-world use case of temperature prediction, introducing three intuitive methods to enhance trust in NN behavior.

**Strategy Representation.** In AI, partially observable Markov decision processes (POMDPs) serve as a standard model for representing real-world scenarios. They model both decision-making and limited observability, e.g. a robot can only perceive its direct environment through a camera, not what happens in the next room. A method for making good decisions in such a system is called a *strategy* (controller, policy). Typically, they are given in the form of a table. We introduce a learning algorithm that transforms such tables into concise automata, effectively representing the same strategy more efficiently.

# Übersicht

Die weitverbreitete Nutzung von Künstlicher Intelligenz (KI) und maschinellern Lernen verlangt nach einer strukturellen Untersuchung und bestenfalls Garantie ihrer Sicherheit. Methoden wie Verifikation, Überwachung und Erklärbarkeit tragen stark dazu bei, das Vertrauen in KI basierte Systeme zu stärken und ihre Sicherheit zu garantieren. In dieser Arbeit werden mehrere Ansätze präsentiert, die sich mit diesen Aspekten befassen.

**Sicherheit Neuronaler Netze (NN).** Die Verifikation von NN ist normalerweise ein Prozess, der vor dem Einsatz des Netzes stattfindet und auf einer begrenzten Anzahl an Eingaben durchgeführt wird. Er liefert schließlich ein Zertifikat für die Zuverlässigkeit der Aussagen des NN auf diesen spezifischen Eingaben. Allerdings skalieren diese Verifikationsansätze schlecht und sind für reale Anwendungen bisher nicht zu verwenden. Um dieses Problem anzugehen, wird in dieser Arbeit die *Abstraktion* von NN eingeführt, die die Größe des Problems verkleinert, indem es eine kompaktere, aber äquivalente Repräsentation des NN erzeugt und damit erlaubt, eine Verifikation auf dem kleineren Netz durchzuführen. Falls es nicht möglich ist, eine äquivalente Darstellung zu generieren, liefert die Abstraktion eine Fehlerberechnung, die die Unterschiede des originalen und des abstrakten NN begrenzt. Da die Verifikation nur auf spezifischen, definierten Eingaben durchgeführt wird, wird in dieser Arbeit zusätzlich die Überwachung von NN eingeführt. Diese überprüft während der Laufzeit des NN sämtliche Eingaben und die Reaktion des Netzes und meldet ein Problem, sobald eine Eingabe erkannt wird, die nicht genug Ähnlichkeit mit den bisher bekannten Trainingsdaten aufweist. Ein Teil dieser Arbeit ist ein solcher Ansatz, der auf Normalverteilungen basiert und daher ein sehr schlanker und kompakter Überwachungsmechanismus ist. Für eine Erleichterung der Auswertung und Optimierung dieser Methoden wird unser Tool MONITIZER eingeführt. Außerdem wird an der konkreten Anwendung der Vorhersage von Wassertemperatur von Flüssen die Notwendigkeit von neuen, intuitiven Analyse-Methoden von NN demonstriert.

**Repräsentation von Strategien.** Im Bereich der KI werden teilweise beobachtbare Markov-Entscheidungsprozesse (POMDPs) als realitätsnahes Modell verwendet, da sie Entscheidungsfindung mit begrenzter Beobachtbarkeit der Realität kombinieren. Man kann sich das anhand eines Roboters vorstellen, der zwar seine direkte Umgebung mittels einer Kamera wahrnehmen kann, aber nicht weiß, was im nächsten Raum passiert. Methoden, die die Entscheidungen in POMDPs beschreiben, werden Strategien genannt, die normalerweise als Tabelle dargestellt werden. In dieser Arbeit wird ein neuer Ansatz präsentiert, der aktives Automaten-Lernen verwendet, um solche Tabellen in kompakte Automaten zu überführen, die dieselbe Strategie repräsentieren, allerdings in einem deutlich kleineren und lesbarerem Format.



# Acknowledgements

First and foremost, I want to thank all my co-authors, without whom this thesis would not exist in the way it is: Muqsit Azeem, Alexander Bork, Debraj Chakraborty, Calvin Chau, Konstantina Drainas, Jürgen Geist, Marta Grobelna, Kush Grover, Vahid Hashemi, Sudeep Kanav, Lisa Kaule, Jan Křetínský, Sabine Rieder, Emmanouil Seferis, Bhumika Uniyal, and Romy Wild.

In particular, I would like to thank my advisor, Jan, for all his support and guidance. I am grateful for your belief and trust in me and for the discussions about work and life. Furthermore, I am thankful to all of my (former) colleagues. You made my time at university so much more joyful. I am really happy to have been at our chair and to meet so many great people. Our lunches and coffee breaks were a very welcome distraction. And I loved being part of a great group. We had some really fun times together. Special thanks to my colleagues

- Marta, for sharing a room with me, sharing funny and stressful times at work, and for sharing chocolate.
- Max, for your positive attitude that just made me always enjoy your company.
- Maxi, for the great scientific and private discussion, the fun and the hard work, and all the guidance and support for a beginning PhD student.
- Muqsit, for the fun discussions, coffee breaks, and the joint trips.
- Pranav, for the supervision in my Master's thesis and for showing me the fun in research.
- Sabine, for sharing great discussions about Neural Networks and the difficulties of being an industrial PhD.
- Sudeep, who pushed me to (over?) my limits but helped me improve far more than I expected.

On a more private note, I want to thank my family: my parents for making me the

person I am today and for believing in me; Tobias, Sebastian, and Torben for their support whenever I needed help; and Maxi, the dog, for endless walks and cuddles during the lockdowns.

Last but by far not the least, I want to thank my husband, Christoph. I am and will forever be grateful for your support: you asked almost annoyingly smart questions that improved my work, your L<sup>A</sup>T<sub>E</sub>Xwisdom and your template saved me a lot of time and nerves in the preparation of this thesis, your cooking kept me alive when deadlines were approaching, and finally, you always cheered me up and believed in me when I thought this project would never end. You and Ben are everything to me.

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Contributions of this Thesis . . . . .	4
1.2. Publication Summary . . . . .	5
1.3. Outline . . . . .	6
<b>2. Preliminaries</b>	<b>7</b>
2.1. Neural Networks . . . . .	7
2.2. Partially Observable Markov Decision Processes . . . . .	12
<b>3. Neural Network Abstraction</b>	<b>15</b>
3.1. State of the Art . . . . .	16
3.2. Contribution . . . . .	17
3.2.1. Framework . . . . .	18
3.2.2. Error Bound . . . . .	21
3.2.3. Refinement . . . . .	21
3.2.4. Experimental Results . . . . .	22
3.3. Future Work . . . . .	23
<b>4. Neural Network Monitoring</b>	<b>25</b>
4.1. State of the Art . . . . .	25
4.2. Contribution: Gaussian Monitoring . . . . .	28
4.2.1. Approach . . . . .	29
4.2.2. Experimental Results . . . . .	31
4.3. Contribution: MONITIZER . . . . .	33
4.3.1. Framework Description . . . . .	34
4.3.2. Evaluation . . . . .	36

4.4. Future Work . . . . .	37
<b>5. Use Case - River Temperature Prediction</b>	<b>39</b>
5.1. State of the Art . . . . .	39
5.2. Contribution . . . . .	40
5.2.1. Water Temperature Prediction . . . . .	40
5.2.2. Model Analysis . . . . .	41
5.2.3. Statistical and Hydrological Evaluation . . . . .	44
5.3. Future Work . . . . .	45
<b>6. POMDP Strategy Representation via Automata Learning</b>	<b>47</b>
6.1. State of the Art . . . . .	48
6.2. Contribution . . . . .	49
6.2.1. Automaton Learning . . . . .	49
6.2.2. Heuristics . . . . .	52
6.2.3. Experiments . . . . .	52
6.3. Future Work . . . . .	53
<b>7. Conclusion</b>	<b>55</b>
<b>I. Publications</b>	<b>71</b>
A. Syntactic vs Semantic Linear Abstraction and Refinement of Neural Net- works . . . . .	72
B. Assessment of Neural Networks for Stream-Water-Temperature Prediction	94
C. Learning Explainable and Better Performing Representations of POMDP Strategies . . . . .	101
D. MONITIZER: Automating Design and Evaluation of Neural Network Mon- itors . . . . .	123
E. Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neu- ral Networks . . . . .	139
F. Predicting stream water temperature with artificial neural networks based on open-access data . . . . .	151



## Definitions

Neural Network (NN) . . . . .	9
Neural Network Semantics . . . . .	10
Activation Values . . . . .	11
Markov Decision Process (MDP) . . . . .	12
Partially Observable Markov Decision Process (PODMP) . . . . .	13
Finite State Controller (FSC) . . . . .	14
Gaussian Monitor . . . . .	30

## Examples

Neuron and Neural Network . . . . .	8
Feed Forward NN . . . . .	10
Markov Decision Process (MDP) . . . . .	12
Partially Observable Markov Decision Process (PODMP) . . . . .	13
Finite State Controller (FSC) . . . . .	14
Replacement of a Neuron by a Linear Combination . . . . .	19
Gaussian Monitor . . . . .	29
FSC Generation for POMDP strategies . . . . .	51

## Figures

LiNNA Framwork . . . . .	18
Replacement of a Neuron by a Linear Combination . . . . .	20
Threshold Selection for Monitors . . . . .	25
Exemplary Depiction of the Gaussian Monitor . . . . .	30
Histogram of Activation Values . . . . .	32
Framework of MONITIZER . . . . .	34
Sample Output of MONITIZER . . . . .	37
Overview of the FSC Learning Framework . . . . .	50



# Abbreviations

**AI** Artificial Intelligence

**AUROC** Area Under the Receiver Operating Characteristic Curve

**FSC** Finite State Controller

**ID** In-Distribution

**MAE** Mean Absolute Error

**MDP** Markov Decision Process

**ML** Machine Learning

**MSE** Mean Squared Error

**NN** Neural Network

**OOD** Out-of-Distribution

**POMDP** Partially Observable Markov Decision Process

**RL** Reinforcement Learning

**RMSE** Root Mean Squared Error

**RNN** Recurrent Neural Network

**ROC** Receiver Operating Characteristic



# 1 Introduction

In recent years, we have seen a rapid development of Artificial Intelligence (AI) and Machine Learning (ML) based systems with increased efficiency and performance, leading to them being more frequently used in various applications. Some examples of these are autonomous driving [Che+17], financial forecasting [SMR22], chatbots [Ope24], autonomous drones [Gu+20], medical imaging [Anw+18], and chess engines [Aut24; Rom+24]. However, alongside these advancements, there are significant challenges, especially in the safety and reliability of AI systems. While these aspects are less critical in areas like gaming or chatbots, they are crucial for autonomous driving, where failures can lead to severe and even fatal consequences [Hen24]. Therefore, there is a growing interest in improving trust and proving the safety of AI systems. This can be achieved through verification, which involves proving or disproving properties of the system, monitoring through surveillance during runtime, or finding more robust representations of these systems. In this thesis, we address these three aspects in two different ways: improving the safety of Neural Networks (NNs) using verification and monitoring and finding good representations for strategies of Partially Observable Markov Decision Processes (POMDPs).

## Safety of Neural Networks

Especially since the invention of *ChatGPT* [Ope24], a huge model that can parse and interpret natural language, NNs have gained much attention. By nature, they are *black-box systems* because they are obtained by using a set of data from which the systems “learn” the task. However, it is not a priori fixed how this information is retained in the system. Only the structure is fixed, but how it is filled with the relevant connections and weights that finally define it is up to the learning. This flexibility allows a wide range of applications with limited domain knowledge, where they excel especially in

## 1. Introduction

image processing, sometimes even outperforming humans [Bue+19; HKW11]. However, their deployment in critical applications like autonomous driving [Che+17] or medical imaging [Anw+18] necessitates certifying their safety or at least building trust in their reliability. This certification issue became even more evident when research discovered that NNs are susceptible to *adversarial attacks* [Sze+14; GSS15], specifically modified inputs to trick the NN.

These insights gave rise to the **verification** of NNs, where most techniques focus on the *robustness property* [Zha+18; Bri+23; Ehl17; CNR17; Kat+19; Kat+17]. This property describes that an NN should be robust to perturbations, i.e. it predicts the same output for a slightly perturbed input. While this property has received most of the scientific interest because it is modeled simply in mathematical terms, other properties have been found interesting, like reachability [Hua+19] or monotonicity [Liu+20b]. Various techniques for verifying NNs are evaluated yearly in a competition [Bri+23]. For an in-depth overview of those techniques, refer to the handbook on NN verification [Alb21].

There has been much progress in the development of verification techniques. For instance, in 2020, the best verification tool could only verify 180 instances in five minutes [JL24], whereas in 2023, the best tool verified 2000 instances in the same time [Bri+23]. Still, it took five days to answer 10,000 verification queries for the biggest NN in the competition (with  $\sim 140$ M parameters). Large models, like ChatGPT [Ope24], have more than 1.7B parameters and a much more complex structure, which requires many more verification queries and renders its verification entirely out of scope.

With the reliability of NNs, we face another issue: even if the NN is well trained and works well on similar data, so-called In-Distribution (ID) data, its behavior on a significantly different input, so-called *Out-of-Distribution (OOD)*, is unclear. Imagine a perception system of an autonomous drone that was trained on images from sunny weather suddenly facing images with heavy rain. It is impossible to know upfront whether the NN would still deliver correct and reliable results. This issue was first raised by [HG17] but is closely related to the existence of the adversarial examples since both are related to an NN misclassifying the input.

To tackle this problem, we need a system that monitors an NN and raises an alarm when it is about to process an unknown and unseen input. For this reason, such systems are called **(runtime) monitors**. They oversee the network during runtime and prevent it from predicting potentially unknown inputs. This research area has especially found

interest in industry [Has+23; CNY19; HS22] since it is a much more lightweight approach than verification and builds an additional layer of control.

With this research area just emerging, there is still much potential for improvements and new developments of monitoring ideas. More importantly, the current evaluation of monitors lacks a structural approach. Most published results define their own OOD datasets without (much) justification [Liu+20a; SL22]. Those datasets might contain inputs that should be considered ID. For example, the NN is trained to detect cars, and some of the OOD images contain cars in the background. Is it expected that the monitor raises an alarm? Until today, it is unclear which datasets to use as OOD, and there is not much discussion on whether the ones in use make sense. Apart from that, setting up a monitor is tricky for a user since this involves setting one or several hyper-parameters. Depending on the application, this can be a challenging problem and needs a structural approach to solving it without requiring domain experts.

## **Strategy Representation**

In AI, POMDPs are a standard model for representing real-world scenarios. They combine non-determinism, probability, and partial observability, making them more reflective of real-world conditions where agents do not have complete information about the state of the environment. While originating from the AI area [KLC98], they have also gained attention in the formal methods community [MHC03a; BKQ22; CCT16; KLC98]. While the AI community uses methods to get an approximate solution without guarantees [Hau00], the formal methods community focuses on more precise solutions with guarantees [Lov91].

Still, the analysis of POMDPs is challenging. Solving for typical objectives like quantitative reachability or total reward is undecidable [MHC03a]. Therefore, finding a strategy (aka policy, controller) for resolving the non-determinism is impossible to do optimally. This is why many works focus on **finding and representing strategies** that perform well in practice without them being provably optimal [Win+21; Bor+20; BKQ22; And+22; And+23].

In general, optimal strategies require *infinite memory* [MHC03b], which can be overcome by using NNs, typically Recurrent Neural Network (RNN), to represent the strategy [CJT21; DKT08] or alternatively, using randomized controllers found via gradient descent [Hec+22] or convex optimization [ABZ10]. However, we face the demand for un-

derstandable and explainable controllers [Wei+23], where both NNs and randomized controllers struggle. Therefore, it is reasonable to look at Finite State Controllers (FSCs), which provide a balance between performance and explainability [KLC98; Meu+99b; Meu+99a; Bon02; SSJ23].

### 1.1 Contributions of this Thesis

This thesis contains four contributions to the areas mentioned above and problems:

**Abstraction** Since verification of NNs does not scale to real-world systems, we introduce a framework for abstraction using linear combinations. In Chapter 3, we introduce our novel abstraction method for NNs, using semantic information and linear combinations of neurons. We reduce the size of the NN while giving error bounds on the difference and show its empirical improvement on related approaches.

**Monitoring** Our contribution to this topic is twofold: On one hand, we introduce a lightweight monitoring method using Gaussian models in Section 4.2. This straightforward method scales well to NNs of various sizes and outperforms related work in their predictive accuracy. Secondly, since the community still needs to tackle the evaluation and optimization of runtime monitors, we present MONITIZER, a modular tool for developing monitors, their optimization, and evaluation, making it applicable to both industry and science.

**Applied Verification** In Chapter 5, we introduce the use case of river temperature prediction. We show that NNs achieve better performance in this task than standard methods. Since they are to be applied by users with less expertise in ML, we introduce three methods for checking their reliability, specifically designed for this application.

**Strategy Representation** Since FSCs are considered more explainable, we introduce a method for learning an FSC to represent a POMDP strategy in Chapter 6. Our approach uses our adaptation of active automata learning to create FSCs. This approach is built modularly as a postprocessing method, making it universally helpful for converting controllers into a more concise FSC.



## 1.2 Publication Summary

This is a publication-based dissertation containing four core papers and two additional papers. The original publications can be found in the Appendix.

### Core Publications:

- (A) Calvin Chau, Jan Křetínský, and Stefanie Mohr: Syntactic vs Semantic Linear Abstraction and Refinement of Neural Networks. ATVA 2023: 401-421. (see Appendix A)
- (B) Stefanie Mohr, Konstantina Drainas, and Jürgen Geist: Assessment of Neural Networks for Stream-Water-Temperature Prediction. ICMLA 2021: 891-896. (see Appendix B)
- (C) Alexander Bork, Debraj Chakraborty, Kush Grover, Jan Křetínský, and Stefanie Mohr: Learning Explainable and Better Performing Representations of POMDP Strategies. TACAS 2024: 299-319. (see Appendix C)
- (D) Muqsit Azeem, Marta Grobelna, Sudeep Kanav, Jan Křetínský, Stefanie Mohr, and Sabine Rieder: Monitizer: Automating Design and Evaluation of Neural Network Monitors. CAV 2024: 265-279. (see Appendix D)

### Other Publications:

- (E) Vahid Hashemi, Jan Křetínský, Stefanie Mohr, and Emmanouil Seferis: Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neural Networks. RV 2021: 254-264. (see Appendix E)
- (F) Konstantina Drainas, Lisa Kaule, Stefanie Mohr, Bhumika Uniyal, Romy Wild, and Jürgen Geist: Predicting stream water temperature with artificial neural networks based on open-access data. Hydrological Processes 2023: Volume 37, Issue 10. (see Appendix F)

All papers have been published in peer-reviewed conference proceedings or scientific journals and are self-contained. Each paper is prefaced with a brief summary and a list of the thesis author's contributions.

## 1. Introduction

Further, the thesis author has co-authored three peer-reviewed publications not included in this thesis.

### Other Co-Authored Publications of the Author:

- (G) Pranav Ashok, Vahid Hashemi, Jan Křetínský, and Stefanie Mohr: Deep-Abstract: Neural Network Abstraction for Accelerating Verification. ATVA 2020.
- (H) Krishnendu Chatterjee, Joost-Pieter Katoen, Stefanie Mohr, Maximilian Weininger, and Tobias Winkler: Stochastic games with lexicographic objectives. Formal Methods in System Design 2023.
- (I) Thomas Brihaye, Krishnendu Chatterjee, Stefanie Mohr, Maximilian Weininger: Risk-aware Markov Decision Processes Using Cumulative Prospect Theory. LICS 2025.

## 1.3 Outline

This thesis contains six chapters and an appendix with all included publications. Chapter 2 presents the mathematical background required to understand the remainder of the thesis. Chapter 3 contains our approach to the abstraction of NNs from Publication (A). We continue in Chapter 4 with the monitoring of NNs, introducing the Gaussian monitor from Publication (E) and our tool MONITIZER from Publication (D). Chapter 5 introduces the use case of river temperature prediction, including Publications (B) and (F). Chapter 6 introduces our work on strategy representation for POMDPs from Publication (C). Finally, we conclude the thesis in Chapter 7.

## 2 Preliminaries

This thesis explores two relevant models in AI: Neural Networks (NNs) and Partially Observable Markov Decision Processes (POMDPs), which we will introduce in the upcoming chapter.

NNs are frequently, though inaccurately, used as synonym for the terms AI or ML. In reality, ML includes numerous more models, including decision trees, support vector machines, logistic regression, kernels, and more. However, NNs stand out as the most prominent ML model due to their ability to outperform other approaches. This is attributed to their intricate structure and capacity to learn complex patterns. Section 2.1 provides a comprehensive introduction to their basic architecture and semantics, laying a solid foundation for understanding Chapters 3 to 5.

On the other hand, POMDPs weave non-determinism and probabilities with partial observability, offering a robust framework for modeling real-world scenarios. This practical application has solidified their position as one of the most longstanding models in AI. While NNs showcase the power of AI by learning and approximating complex functions, POMDPs are used because of their ability to mimic reality, for example, to train NNs. As a foundation for Chapter 6, this thesis introduces POMDPs in Section 2.2 with their formal definition and presents Finite State Controllers (FSCs) as an efficient method for how to represent their strategies.

As usual,  $\mathbb{N}$  refers to the natural numbers,  $\mathbb{Z}$  describes the integers, and  $\mathbb{R}$  refers to the real numbers. We write  $\mathbb{Z}^d$  and  $\mathbb{R}^d$  to refer to the d-dimensional vector space over the integers and real numbers.

### 2.1 Neural Networks

At their core, NNs are mathematical functions mapping inputs to outputs. Their distinctive feature lies in how they are structured and their ability to “learn” from data.

## 2. Preliminaries

Traditional programming involves an explicit definition of the program or function by the user, whereas, for a learned system, we only need its general structure. The learning process then fills it with concrete values derived from given data.

We can distinguish *supervised* and *unsupervised* learning: Supervised learning requires an output for any given input, whereas unsupervised learning derives an output automatically and does not require a given output. NNs are more common in the supervised setting, where they are presented with input-output pairs, which they try to mimic by adjusting their parameters. This process is called the *learning process*. It is typically done by using optimization procedures like gradient descent, where the model's parameters are iteratively changed towards optimal values. For a detailed explanation of learning, and methods like gradient descent, refer to [RN20, Chapter 19]. In other words, learning describes a process of optimization based on empirical data.

The name “Neural Network” originates from efforts to model the human brain via creating artificial “neurons” to model biological neurons [MP43; MMR55]. Mathematically speaking, neurons are the smallest computation units in an NN. They are interconnected to form a network that then globally computes an output. An NN is built up from *layers* that sequentially compute a value for their input. Each of these layers is made from several neurons. Example 2.1 illustrates a *neuron* and a *feed-forward NN*.

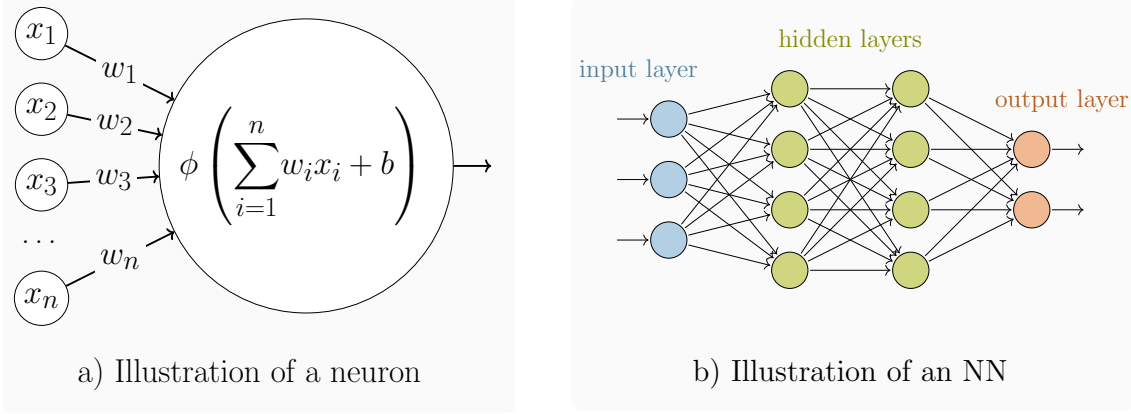
Formally, an NN is a function  $f: \mathcal{X} \rightarrow Y$  from the input set  $\mathcal{X} \subseteq \mathbb{R}^d$  to the output set  $Y$ . Examples of input spaces include  $\mathcal{X} = \mathbb{R}^2$ , describing the x- and y-location on a map, or  $\mathcal{X} = [0, 255]^{128 \times 128 \times 3}$  for RGB-colored images. The output set can be continuous,  $Y \subseteq \mathbb{R}^m$ , or discrete,  $Y \subseteq \mathbb{Z}^n$ . Examples for the output include  $Y = [-360, 360]$ , describing the steering angles of a car, or  $Y = \{\text{“cat”}, \text{“dog”}\}$  for the distinct classes cats and dogs.

### **Example 2.1:** *Neuron and Neural Network.*

On the left, we see an illustration of a neuron: It receives  $n$  input values,  $x_1, \dots, x_n$ . After computing the weighted sum and adding a bias, it applies a non-linear function  $\phi$  on the result and passes this value on.

The illustration on the right contains a picture of a feed-forward NN. Intuitively, a feed-forward NN receives input within the *input layer*. Each neuron in the first

*hidden layer* receives a weighted sum of all neurons in the input layer as input. The second hidden layer receives a weighted sum of all outputs from the first hidden layer. In this way, the information is processed layer by layer. The computed values of the NN are given by the output of the *output layer*.



From Example 2.1, we know that an NN processes the information by computing the weighted sum of the inputs from the previous layer. However, one more crucial part is necessary for NNs to be so powerful: the activation functions. They serve as a non-linear break between the layers. Otherwise, a single matrix multiplication could represent the computation of an NN. In this work, we use the ReLU activation function, defined as  $\text{ReLU}(x) = \max\{0, x\}$ . However, there are many more with different strengths and purposes [TES24].

**Definition 2.1:** *Neural Network (NN).*

A *feed-forward Neural Network* is a tuple  $N = (\mathcal{W}, \mathcal{B}, \Phi, L)$ , where  $L$  is the number of layers,  $\mathcal{W} = (W^{(1)}, \dots, W^{(L)})$  are the weight matrices,  $\mathcal{B} = (\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L)})$  the biases, and  $\Phi = (\phi^{(1)}, \dots, \phi^{(L)})$  the activation functions. For some  $n \in [1, \dots, L]$ , we have  $W^{(n)} \in \mathbb{R}^{d_n \times d_{n-1}}$ ,  $\mathbf{b}^{(n)} \in \mathbb{R}^{d_n}$ , and  $\phi^{(n)}: \mathbb{R}^{d_n} \rightarrow \mathbb{R}^{d_n}$  a non-linear function. Here,  $d_n$  describes the *size* of layer  $n$ .

In a feed-forward NN, information flows strictly in one direction: from layer  $m$  to layer  $n$ , where  $m < n$ . At each step, the output is computed as a weighted sum of the previous layer's output plus an additional *bias* vector and the application of an activation function.

**Definition 2.2:** *Neural Network Semantics.*

The function computed by the NN,  $f_N : \mathcal{X} \rightarrow Y$ , is defined by the following semantics. For an input  $\mathbf{x} \in \mathcal{X}$ , the output  $f_N(\mathbf{x}) = \mathbf{y}$  is iteratively computed as:

$$\begin{aligned}\mathbf{h}^{(0)}(\mathbf{x}) &= \mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x} \\ \mathbf{h}^{(n+1)}(\mathbf{x}) &= W^{(n)}\mathbf{z}^{(n)}(\mathbf{x}) + \mathbf{b}^{(n+1)} \\ \mathbf{z}^{(n+1)}(\mathbf{x}) &= \phi^{(n+1)}(\mathbf{h}^{(n+1)}(\mathbf{x})) \\ \mathbf{y} &= \mathbf{z}^{(L)}(\mathbf{x})\end{aligned}$$

The NN semantics are an iterative computation layer by layer. We present a concrete example of a feed-forward NN for a better and more intuitive understanding. Intuitively, there is a single matrix multiplication and the application of the non-linear activation function in each step. This process is repeated for each layer in the network.

**Example 2.2:** *Feed Forward NN.*

We look again at Example 2.1 with a concrete instantiation. The values of the first layer,  $\mathbf{h}^{(1)}(\mathbf{x}) = W^{(0)}\mathbf{z}^{(0)}(\mathbf{x}) + \mathbf{b}^{(0)} \in \mathbb{R}^4$ . The same holds for the second layer. We have  $\mathbf{y} \in \mathbb{R}^2$ , i.e. two output neurons.

Assume

$$W^{(0)} = \begin{pmatrix} 1 & -1 & 3 \\ -1 & 2 & 1 \\ -1 & -1 & -1 \\ 1 & 2 & 3 \end{pmatrix} \quad W^{(1)} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & 2 & -1 & -2 \\ -2 & 1 & 1 & -1 \\ 1 & 2 & 3 & 1 \end{pmatrix}$$

$$W^{(2)} = \mathbf{I} \quad W^{(3)} = \begin{pmatrix} 1 & 1 & -1 & 1 \\ 2 & -6 & 2 & 1 \end{pmatrix}$$

$\mathbf{b}^{(1)} = \mathbf{b}^{(2)} = \mathbf{b}^{(3)} = \mathbf{b}^{(4)} = \mathbf{0}$ ,  $\phi^{(1)} = \phi^{(2)} = \phi^{(3)} = \text{ReLU}$ , and  $\phi^{(4)} = \text{Id}$

For an input  $\mathbf{x} = [1, 1, 1]^T$ , we get

- |  |  |
|--|--|
| 1. $\mathbf{h}^{(1)}(\mathbf{x}) = [3, 2, -3, 6]^T$              | 6. $\mathbf{z}^{(3)}(\mathbf{x}) = \mathbf{h}^{(3)}(\mathbf{x})$ |
| 2. $\mathbf{z}^{(1)}(\mathbf{x}) = [3, 2, 0, 6]^T$               | 7. $\mathbf{h}^{(4)}(\mathbf{x}) = [14, 7]^T$                    |
| 3. $\mathbf{h}^{(2)}(\mathbf{x}) = [-7, 1, -10, 13]^T$           | 8. $\mathbf{z}^{(4)}(\mathbf{x}) = \mathbf{h}^{(4)}(\mathbf{x})$ |
| 4. $\mathbf{z}^{(2)}(\mathbf{x}) = [0, 1, 0, 13]^T$              | 9. $\mathbf{y}(\mathbf{x}) = \mathbf{z}^{(4)}(\mathbf{x})$       |
| 5. $\mathbf{h}^{(3)}(\mathbf{x}) = \mathbf{z}^{(2)}(\mathbf{x})$ |  |

This means, the NN computes  $f_N(\mathbf{x}) = [14, 7]^T$ .

Example 2.2 demonstrates that the weight matrices  $\mathcal{W}$  and the biases  $\mathcal{B}$  are significant in the computation. Therefore, during learning, we optimize for good values of  $\mathcal{W}$  and  $\mathcal{B}$ . Usually, we are given a finite set of example values to learn from: the *training set*  $\mathcal{X}_{\text{train}} \subseteq \mathcal{X}$  with a matching set of outputs  $Y_{\text{train}} \subseteq Y$ . The learning process defines a *loss function* that describes the performance of the NN in a singular value. For example, we are given an input pair  $(x, y) \in \mathcal{X}_{\text{train}} \times Y_{\text{train}}$ , where  $y$  is the true value the NN should predict for  $x$ . The loss function could be the absolute difference between the true value  $y$  and the prediction of the NN, i.e.  $|y - f_N(x)|$ , but there are many more [BB24], and their use depends on the task to be learned and the concrete application.

For abstraction and monitoring, we also need to define the behavior of the NN. For this, we use *activation values* [Ash+20; CNY19] as a description of the behavior of the NN for some input. The idea is to collect the inner values of the NN by passing values to the NN and tracking the output of each neuron. They are also called the “semantic” [Publication (A)] information of an NN since they are based on input values.

**Definition 2.3:** *Activation Values.*

For some finite set  $X \subseteq \mathcal{X}$ , we define the *activation values* of a layer  $n$  of the NN as a matrix  $\mathbf{Z}^{(n)}(X) = [\mathbf{z}^{(n)}(\mathbf{x})]_{\mathbf{x} \in X} \in \mathbb{R}^{|X| \times d_n}$ . Then, the activation values of one neuron  $i$  of layer  $n$  are  $\mathbf{z}_i^{(n)}(X) := \mathbf{Z}^{(n)}(X)_{*,i}$  the  $i$ th column of  $\mathbf{Z}^{(n)}(X)$ .

We use the activation values for monitoring in Section 4.2 and for creating an abstraction of NNs in Chapter 3.

## 2.2 Partially Observable Markov Decision Processes

This section establishes all required concepts necessary to understand Chapter 6. Initially, we formally define Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs), accompanied by illustrative examples. Then, we introduce FSCs as a means to represent POMDP strategies.

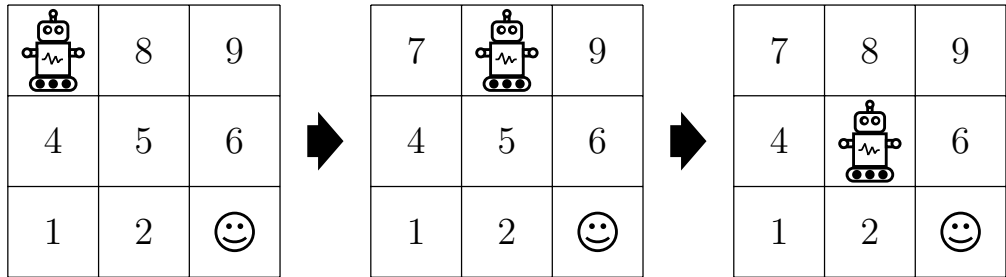
A (discrete) probability distribution on a countable set  $S$  is a function  $d : S \rightarrow [0, 1]$  such that  $\sum_{s \in S} d(s) = 1$ . We denote the set of all probability distributions on the set  $S$  as  $\text{Dist}(S)$ .

**Definition 2.4:** *Markov Decision Process (MDP).*

An MDP is a tuple  $\mathcal{M} = (S, A, P, s_0)$  where  $S$  is a countable set of states,  $A$  is a finite set of actions,  $P : S \times A \rightarrow \text{Dist}(S)$  is a partial transition function, and  $s_0 \in S$  is the initial state.

An MDP is a model that involves non-determinism and probabilities. Starting in an initial state, an *agent* moves on the set of states. Whenever it chooses an action, the probability distribution of the current state and the chosen action determine where it will move next. In such a model, we can discuss the probability of reaching a set of target states (reachability), or, assuming that states have an assigned reward, we can compute the collected reward until reaching a target state (total reward). [BK08, Chapter 10.6] provides a more detailed introduction to MDPs and their properties. For a more intuitive understanding of MDPs, we introduce an example below.

**Example 2.3:** *Markov Decision Process (MDP).*



A robot is randomly placed on a 3x3 grid. Its goal is to move to the lower-



right corner, marked by a smiling face. The states of this MDP are defined by the cells in the grid, i.e.,  $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ . The actions are  $A = \{left, right, up, down\}$  and define which action the robot wants to perform. For each action, there is a 25% chance that the robot fails to move and stays in place.

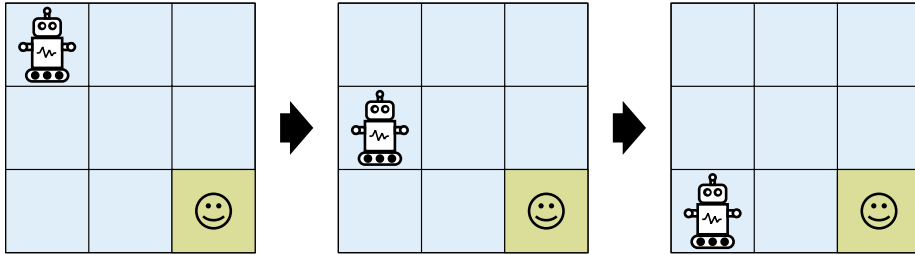
A POMDP is an MDP where the agent does not have the complete information about its current state, which is called partial observability. Partial observations could mean that it only perceives a subset of the state variables (e.g., only the temperature in one room instead of for every room in a house) or a complete transformation of the state (e.g., the color “blue” for state 1).

**Definition 2.5:** *Partially Observable Markov Decision Process (PODMP).*

A POMDP is a tuple  $\mathcal{P} = (\mathcal{M}, Z, \mathcal{O})$  where  $\mathcal{M} = (S, A, P, s_0)$  is the underlying MDP with a finite number of states,  $Z$  is a finite set of observations, and  $\mathcal{O} : S \rightarrow Z$  is an observation function that maps each state to an observation.

We extend Example 2.3 to the partial observable setting in Example 2.4.

**Example 2.4:** *Partially Observable Markov Decision Process (PODMP).*



In Example 2.3, the robot knows exactly which state it is in. In the partially observable setting, it can only distinguish whether it has already reached the goal (**g**) or not (**b**). One can imagine this as a robot in a house with rooms that all look identical, except for the room with the charger, which is highlighted in green. The underlying MDP is the one from before, where we have two observations  $Z = \{\mathbf{b}, \mathbf{g}\}$ . All states, except the goal state 3, have observation **b**, therefore  $\mathcal{O} = \{(1 \rightarrow \mathbf{b}), (2 \rightarrow \mathbf{b}), (3 \rightarrow \mathbf{g}), (4 \rightarrow \mathbf{b}), (\rightarrow \mathbf{b}), (6 \rightarrow \mathbf{b}), (7 \rightarrow \mathbf{b}), (8 \rightarrow \mathbf{b}), (9 \rightarrow \mathbf{b})\}$ .

## 2. Preliminaries

To describe the principle of a *strategy*, we introduce the notation of a *finite path*. For an MDP  $\mathcal{M}$ , a finite path  $\rho = s_0 a_0 s_1 \dots s_i$  is a sequence of states and actions such that for all  $t \in [0, i - 1]$ ,  $a_t$  is an available action in  $s_t$  and  $s_{t+1}$  is a possible successor of  $s_t$ . A strategy for an MDP is then a mapping of paths to states. A *memoryless* strategy only has access to the last state in the path. In other words, a memoryless strategy decides which action to play based on the state the agent is in.

For MDPs, strategies have access to the *full* state information, which is unsuitable for POMDPs. Therefore, we need another notion of strategies only based on partial observations. For a POMDP, we call a strategy *observation-based* if for any two paths  $\rho_1, \rho_2$  with states  $s_i(\rho_1), s_i(\rho_2)$ , and if all the observations are the same, i.e.,  $\mathcal{O}(s_i(\rho_1)) = \mathcal{O}(s_i(\rho_2))$ , then the suggested action by the strategy for both paths is the same. In other words, the strategy must output the same action whenever we have the same sequence of observations. Therefore, a strategy for a POMDP can be viewed as mapping observation sequences to available actions.

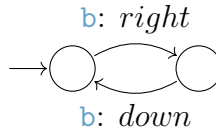
Finding a good representation for such strategies is one of the critical tasks, which we will discuss in Chapter 6. For our representation, we use an automaton, precisely an FSC, which can be seen as a finite Mealy machine that maps observation sequences to actions.

**Definition 2.6:** *Finite State Controller (FSC).*

An FSC is a tuple  $\mathcal{F} = (N, \gamma, \delta, n_0)$  where  $N$  is a finite set of nodes,  $\gamma : N \times Z \rightarrow \text{Dist}(A)$  is an action mapping,  $\delta : N \times Z \rightarrow N$  is the transition function, and  $n_0$  is the initial node.

**Example 2.5:** *Finite State Controller (FSC).*

Consider an FSC for Example 2.4, shown below. The simplest (and optimal) strategy is to move down and right in turns. We see the two states and the transitions in between. When having the observation **b**, the FSC outputs “right” and changes state. When we see the same observation again, the output is “down” and we change to the left state.



## 3 Neural Network Abstraction

As highlighted in the introduction, the increasing use of NNs in safety-critical applications has sparked a significant interest in their verification. Formal verification, a process of proving or disproving the correctness of a system concerning a specific formal specification or property, is a crucial step in ensuring the safety and reliability of NNs. However, formal verification still needs to achieve the necessary scalability to be applied to real-world scenarios. *Abstraction*, a powerful tool from standard model-checking, is used to reduce the size of a verification problem [CGL92; Cla+00]. In general, abstraction is a method that strives to remove unnecessary, redundant, or irrelevant information and gain a smaller but equivalent version of the problem. Similarly, we can look at an NN and wonder whether it contains redundant or irrelevant information and, if so, whether we can remove the redundancy and gain a smaller, but equivalent NN. In the following, we define the problem statement.

**Problem 3.1:** *Abstraction Problem.*

Given a neural network  $N$ , a set  $X$  and an allowed error margin  $\varepsilon$ , find an abstraction  $\tilde{N}$ , such that  $|f_N(x) - f_{\tilde{N}}(x)| < \varepsilon$  for all  $x \in X$ .

**Remark 3.1:**

Our experiments indicate that exact abstractions (i.e.,  $\varepsilon = 0$ ) are almost impossible. Breaking it down to the level of neurons, we found that not even *one* neuron that contributes to the computation of the NN function can be removed while keeping the NN function the same. Thus, we focus on approximations that minimize the distance between the original and the abstract network.

In the following, we will present the current state of NN abstraction and then introduce our abstraction method from Publication (A).

## 3.1 State of the Art

The closely related area of research, called *compression* aims to reduce the NN’s size. This includes methods like pruning [LDS89; Han+15; SB15; ZYZ18] and knowledge distillation [BCN06; HVD15]. However, these methods do not strive for equivalence and, most importantly, do not provide error bounds or guarantees, making them impractical for verification. On the other hand, abstraction provides guarantees and estimates of the closeness of the smaller (abstract) NN to the original.

The use of *abstractions* started with an abstraction on the weights by introducing intervals [PT10], however, this approach was limited to a one-layer NN. Building upon this approach, Prabhakar and Afzal [PA19] use abstractions after representing an NN as an *interval NN*, which was further improved by using more complex abstract domains [ST20]. Although these works are theoretically intriguing, their practicality has not been investigated.

Several works utilize *abstract domains* to verify NNs [Geh+18; Sin+19a; Sin+19b], representing inputs as elements of said abstract domains. However, they do not generate a specific abstract network that could be reused or inspected for further insights.

There are three works that we consider closest to our work and which we will introduce in more detail. Elboher, Gottschlich, and Katz [EGK20] propose a *classification of neurons* reflecting how they change the NN’s output and a merging of neurons of the same class. Thereby, they can guarantee that the abstraction over-approximates the original network. However, the property to be verified needs to be integrated directly into the network, making the approach significantly less flexible. Additionally, this tight entanglement of specification and NN makes it impossible to retrieve and reuse the abstraction for anything other than verifying that specific property.

The tool DEEPABSTRACT [Ash+20] by the same author as this thesis is a predecessor of the work that we will present later in the chapter. It uses the *activation values* of the neurons as semantic information to retrieve “similar” neurons that can be merged. This way, the approach generates a smaller NN that is still close to the original.

One of the works closest to ours is the *bisimulation* of NNs [Pra22]. Their idea is similar to the other abstraction methods [EGK20; Ash+20] in that it clusters similar

neurons and merges them. In contrast to DEEPABSTRACT, however, it uses the syntactic information of the weights and biases rather than the semantic information of the activation values. In more detail, the syntax of a neuron is determined by its incoming weights and bias. If this information is the same or close enough for two neurons, they are merged to create a bisimilar network. This process is repeated for all identical or a certain amount of similar neurons.

## 3.2 Contribution

This section summarizes our results from Publication (A), where all results are taken from the paper without any additional citation. We improve on the state of the art in several ways:

### Summary of the contributions:

- We propose a **flexible, novel abstraction framework** that allows a replacement of neurons by a *linear combination* of other neurons. This improvement on neuron replacements enables a reduction of the size of the NN of up to 60% without a significant decrease in test accuracy.
- We provide a theoretical **error bound** for the abstraction.
- We develop an **abstraction-refinement procedure** to balance between abstraction precision and size, showing that strategic refinement can yield better results than direct moderate abstraction.
- We introduce a classification of abstraction techniques into **syntactic** and **semantic** abstractions, showing that semantic abstractions can reduce the network size twice as much as syntactic abstraction.

### Central Idea

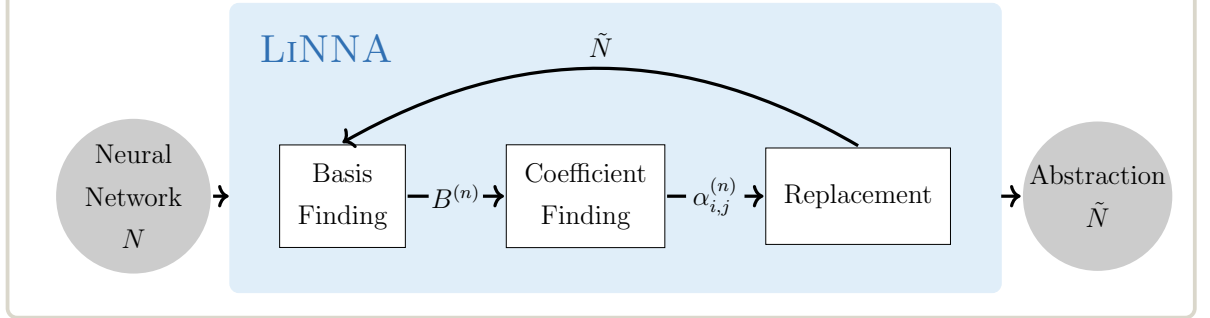
In Publication (A), we present an improved and richer abstraction scheme than previous work. In contrast to DEEPABSTRACT [Ash+20] and the bisimulation [Pra22], we replace a group of similar neurons not by a single representative but by a linear combination. As an intuition, take the vectors  $[1, 0]$ ,  $[0, 1]$ , and  $[2, 1]$ . While they are not close in Euclidean distance, we can produce the last one by a linear combination of the first two:  $[2, 1] = 2 \cdot [1, 0] + [0, 1]$ . With our tool LiNNA (**L**inear **N**eural **N**etwork **A**bstraction) from Publication (A), we can find such combinations and replace such neurons.

### 3.2.1 Framework

Our abstraction process, detailed in Figure 3.1, using linear combinations of neurons involves three steps: finding a set of neurons to replace (basis finding), finding coefficients for the linear combinations (coefficient finding), and replacing the neurons (replacement). While these steps can be performed one after the other, they are interdependent and are thus performed in a loop. This loop is crucial as it allows for multiple iterations, ensuring that the best possible basis and coefficients are found.

**Figure 3.1:** LiNNA Framework.

We show the process of LiNNA in three steps: basis finding, coefficient finding and replacement. We start with an NN and try to find a good basis  $B^{(n)}$  for some layer  $n$ . Using this, we find good coefficients for replacing a neuron  $i$ ,  $\alpha_{i,j}^{(n)}$  of said layer. Then, we replace neuron  $i$  with a linear combination using the coefficients. We evaluate the performance of the abstraction and, if it is not good enough, start the loop again to search for a better basis.



**Basis Finding** In this step, we find neurons that should be replaced, or dually, which neurons should remain in a *basis* for some layer. In other words, which neurons carry so much information that they are best apt to model the behavior of other neurons? We demonstrate two techniques for finding them: The *greedy* method uses a look-ahead and evaluates the induced error after replacement for each neuron in a layer. The *heuristic* method uses the variance of the activation vectors as an indicator, where we assume that a higher variance correlates with a higher computational power of the neuron and, thus, its importance.

**Coefficient Finding** Assuming we have a basis for some layer, we find the coefficients necessary to replace the remaining neurons not part of the basis. We propose two methods for solving this problem: linear programming (minimizing the Euclidean distance) and orthogonal projection (minimizing the absolute distance). Our empirical results demonstrate that orthogonal projection is much more scalable. Note that this typically yields one optimal solution, so this step is only repeated if the basis changes.

**Replacement** In this step, we replace a neuron with a linear combination of basis neurons from the same layer. The replacement is done by summing up the respective outgoing weights scaled by the coefficients, as demonstrated in Example 3.1. Assuming that the linear combination is an exact representation of the behavior of the replaced neuron, this would yield a smaller NN that computes the same function.

**Example 3.1:** *Replacement of a Neuron by a Linear Combination.*

Consider the NN in Figure 3.2. It has two input neurons  $n_1^{(0)}, n_2^{(0)}$  and two output neurons  $n_1^{(2)}, n_2^{(2)}$ . In between, there is one hidden layer, layer 1, with three neurons  $n_1^{(1)}, n_2^{(1)}, n_3^{(1)}$ , i.e.  $d_1 = 3$ .

We are given the basis  $B^{(1)} = \{n_2^{(1)}, n_3^{(1)}\}$ . This means, we want to replace neuron  $n_1^{(1)}$ , and we have the linear combination  $\alpha_{1,1}^{(1)} \cdot n_2^{(1)} + \alpha_{1,2}^{(1)} \cdot n_3^{(1)}$ .

We remove neuron  $n_1^{(1)}$ , and add additional weight to the outgoing weights of the basis neurons. In particular, we add to  $W_{1,2}^{(1)} = -1$  the portion of  $W_{1,1}^{(1)}$  that  $n_2^{(1)}$  has to take over, i.e.  $\tilde{W}_{1,2}^{(1)} = W_{1,2}^{(1)} + W_{1,1}^{(1)} \cdot \alpha_{1,1}^{(1)} = -1 + \alpha_{1,1}^{(1)}$ . Analogously, we change the other outgoing weights of the basis neurons:

$$\begin{aligned}\tilde{W}_{1,3}^{(1)} &= W_{1,3}^{(1)} + W_{1,1}^{(1)} \cdot \alpha_{1,2}^{(1)} = -2 + \alpha_{1,2}^{(1)} \\ \tilde{W}_{2,2}^{(1)} &= W_{2,2}^{(1)} + W_{2,1}^{(1)} \cdot \alpha_{1,1}^{(1)} = 3 + 2 \cdot \alpha_{1,1}^{(1)} \\ \tilde{W}_{2,3}^{(1)} &= W_{2,3}^{(1)} + W_{2,1}^{(1)} \cdot \alpha_{1,2}^{(1)} = 1 + 2 \cdot \alpha_{1,2}^{(1)}\end{aligned}$$

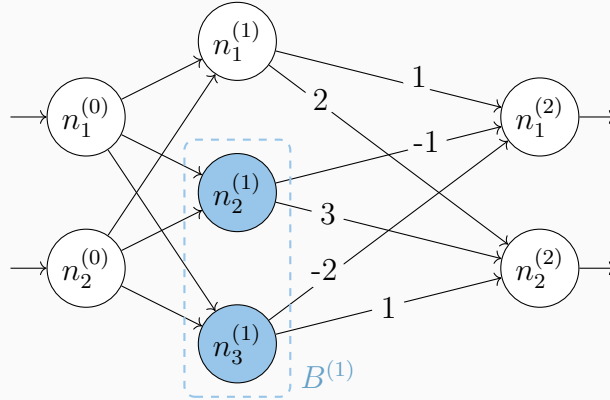
The resulting NN has one neuron less. Under the assumption that for all inputs  $\mathbf{x}$ ,  $\mathbf{z}_1^{(1)}(\mathbf{x}) = \alpha_{1,2}^{(1)} \mathbf{z}_2^{(1)}(\mathbf{x}) + \alpha_{1,3}^{(1)} \mathbf{z}_3^{(1)}(\mathbf{x})$ , the abstraction and the original produce the same output.

**Figure 3.2:** *Replacement of a Neuron by a Linear Combination.*

The upper illustration shows the original network with the basis  $B^{(1)}$  in blue. The bottom illustration contains the abstraction where we replace neuron  $n_1^{(1)}$  with the linear combination  $\alpha_{1,1}^{(1)} \cdot n_2^{(1)} + \alpha_{1,2}^{(1)} \cdot n_3^{(1)}$ .

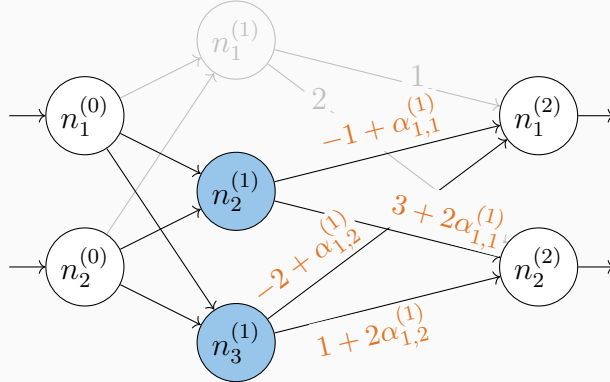
**Original**

$$W^{(1)} = \begin{pmatrix} 1 & -1 & -2 \\ 2 & 3 & 1 \end{pmatrix}$$



**Abstraction**

$$\tilde{W}^{(1)} = \begin{pmatrix} 0 & -1 + \alpha_{1,1}^{(1)} & -2 + \alpha_{1,2}^{(1)} \\ 0 & 3 + \alpha_{1,1}^{(1)} & 1 + \alpha_{1,2}^{(1)} \end{pmatrix}$$





### 3.2.2 Error Bound

We set our goal at reducing the difference between the activation values of the original neuron  $\mathbf{z}_i^{(n)}(\mathbf{x})$  and the activation values of the replacement  $\sum_{j \in B^{(n)}} \alpha_{i,j}^{(n)} \mathbf{z}_j^{(n)}(\mathbf{x})$ . We can quantify the error of the abstraction, i.e., the difference of the output of the original network and the reduced one, where we refer to [Gra95] for an easier read.

**Theorem 3.1:** *Abstraction Error Bound.*

Let  $N$  be an NN with  $L$  layers. For each layer  $n \in \{1, \dots, L\}$ , we have a basis of neurons  $B^{(n)}$  and a set of replaced neurons  $R^{(n)}$ . Let  $\tilde{N}$  be the network after replacing neurons in  $R^{(n)}$  with the process described above. If for all layers  $n \in \{1, \dots, L\}$  and for all inputs  $\mathbf{x} \in X \subset \mathcal{X}$ , we have

- for  $i \in R^{(n)} : \|\mathbf{z}_i^{(n)}(\mathbf{x}) - \sum_{j \in B^{(n)}} \alpha_{i,j}^{(n)} \mathbf{z}_j^{(n)}(\mathbf{x})\| \leq \varepsilon^{(n)}$
- $|\sum_{i \in R^{(n)}} W_{*,i}^{(n)} \sum_{t \in B^{(n)}} \alpha_{i,j}^{(n)}| \leq \eta^{(n)}$

then we have for  $\mathbf{x} \in X$ :

$$\|f_{\tilde{N}}(\mathbf{x}) - f_N(\mathbf{x})\| \leq b \frac{1 - a^{L-1}}{1 - a}$$

where  $a = \lambda(\|W\| + \eta)$  and  $b = \lambda\|W\|\varepsilon$  with

- $\lambda = \max_n \lambda^{(n)}$  where  $\lambda^{(n)}$  is the Lipschitz-constant of  $\phi^{(n)}$
- $\|W\| = \max_n \|W^{(n)}\|_1$
- $\eta = \max_n \eta^{(n)}$
- $\varepsilon = \max_n \varepsilon^{(n)}$

### 3.2.3 Refinement

The abstraction may fail to capture a few critical behavioral aspects of the original network. This can happen either because it predicts inputs differently than the original network or because some inputs cannot be verified, although they are safe. We can *refine* the abstraction for these inputs, so-called *counterexamples*, rather than start the abstraction again from the complete original network. Therefore, we propose three different methods for finding suitable candidate neurons that shall be restored:

**Difference-guided** The difference-guided refinement considers the difference between a neuron  $\mathbf{z}_i^{(n)}(\mathbf{x})$  in the original network and the linear combination  $\sum_{j \in B^{(n)}} \alpha_j \mathbf{z}_j^{(n)}(\mathbf{x})$  with which it was replaced in the abstracted network. We evaluate this for all neurons on the counter example  $\mathbf{x}$  and restore the neuron with the largest difference.

**Gradient-guided** The gradient-guided refinement follows a similar approach as the difference-guided, but additionally, we use the gradient of the loss-function, which was already used during training. Using the gradient gives a better insight into which neurons “need to be repaired” for a better performance.

**Look-ahead** We simulate the restoration of each replaced neuron and observe how it changes the difference between the output  $\tilde{\mathbf{y}}$  of the abstraction and  $\mathbf{y}$  of the original network. We then choose to restore the neuron that minimizes this difference. Again, we can quantify the difference with an appropriate loss function, as in the gradient-guided refinement.

#### 3.2.4 Experimental Results

We implemented our abstraction technique in the tool LINNA (Linear Neural Network Abstraction) (available at <https://github.com/cxlvinchau/LINNA>). Since there are few approaches available, our experimental comparison was limited. We compared our implementation of LINNA to the predecessor DEEPABSTRACT [Ash+20] and our own implementation of the Bisimulation [Pra22], for which no implementation was readily available. We reduce the size of the NNs and evaluate their accuracy on the test set  $\mathcal{X}_{\text{test}}$ . Our results show that LINNA outperforms both other approaches: LINNA, which is based on semantic information, can reduce the number of neurons up to 60% without a significant change in the accuracy, whereas the accuracy already decreases after a reduction of 30% of the neurons when using the bisimulation, which relies on syntactic information. DEEPABSTRACT, also based on semantic information, manages to get to a 40% reduction before having a performance loss.

### 3.3 Future Work

#### Our work

We have significantly improved the abstraction of NNs: Our approach outperforms similar works [Ash+20; Pra22] in the case of a reusable general abstraction. We show that semantic information can significantly improve the precision of the abstraction. However, a formal connection between the abstraction and the original network is still too broad, especially compared to empirical evidence. Therefore, there is still a gap to be closed: finding a formal description that captures what we already see in the experiments.

Additionally, we can improve the abstraction by employing similar methods to [EGK20] and classifying the neurons. We can do so by marking them as increasing or decreasing or looking at their influence on special output classes. Lastly, we want to explore how far abstraction methods are also sensible compression methods. We already know that they decrease the size of the NNs. However, we need to find out how this compares to existing compression methods. Vice versa, it could be interesting to look at compression methods and if it is possible to give guarantees on the closeness of the compressed network to the original.

#### Other work

The advances after Publication (A) are as follows: A *survey* on the abstraction of NNs [Bou+23a], published soon after and contains an overview of most of the works that we included in the State of the Art. An *abstraction refinement process* that refines the over-approximation of the output of an NN [LA23] but focuses on reachability analysis and not general NNs. There is some improvement in using Interval NNs [Bou+23b], similar to [PA19], including some experimental results, yet, this approach is solely based on syntactic information in the NN.

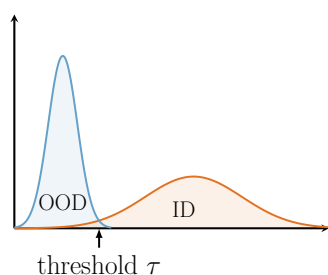


## 4 Neural Network Monitoring

As NNs continue infiltrating various areas of technology and industry, ensuring their safety becomes imperative. While there is a large body of work on their verification (refer to Chapter 3), there is also the need for runtime monitoring, which is the focus of this chapter. Monitoring can support the robust performance of NN systems in real-world applications. It involves continuously observing the NN’s behavior during runtime to detect any potential erroneous inputs, so-called Out-of-Distribution (OOD) inputs. In contrast, verification is done offline and requires concrete specifications for the NN that can be proven.

### 4.1 State of the Art

**Figure 4.1:** *Threshold Selection for Monitors.*



A NN monitor should generally distinguish between In-Distribution (ID) data, e.g., data similar to the training data, and OOD data. This classification typically relies on a “threshold”, as illustrated in Fig. 4.1. The monitor generates a value for each input to the monitored NN and determines whether the input is ID or OOD based

on that value. Selecting a threshold to distinguish between these classes depends on the concrete application since there is rarely one optimal value for all settings.

Additionally, a monitor might contain several adjustable parameters that impact its performance. The process of finding a good set of parameters is called *hyperparameter tuning*. The research community has not investigated this aspect of monitors extensively,

#### 4. Neural Network Monitoring

and the values are often selected manually or after a short testing phase. To facilitate the practical application of runtime monitoring in industry, this process requires thorough investigation and the development of methods to find a good set of parameters without requiring expert user input.

Apart from the interest in finding good monitors, comparing them is also necessary. Without a comparative analysis, determining the “best” monitor is impossible. Evaluating a monitor on a single dataset considered OOD does not necessarily have any indication of its overall performance. Instead, the evaluation should be tailored to the specific application to evaluate a monitor transparently.

##### Summary of the challenges:

**Need for monitoring methods** With many methods already available, there is yet no optimal solution. Many existing methods fail to detect certain special cases of OOD, highlighting the need for other and more involved monitoring approaches.

**Hyperparameter selection** Monitors contain at least one, and often several, hyperparameters that need to be configured before the monitor is deployed. This selection is currently done only heuristically without optimizing for the concrete application.

**Transparent Evaluation** Monitors are usually evaluated on some set, not from the ID data. However, there is often insufficient justification on whether this set reasonably represents an OOD set. Additionally, monitors can perform differently across various datasets. For a specific application, it is crucial to carefully choose the set on which the monitor will be evaluated.

**Support for development** The lack of adequate tool support complicates the development of new monitoring ideas and the development of a concrete monitor for an application. It makes the process time-consuming and requires significant domain knowledge, making it harder for the industry to use NN monitors.

In this chapter, we focus on a twofold contribution: first, we introduce a novel monitoring technique, the GAUSSIAN monitor [Publication (E)], tackling the first challenge;

secondly, we introduce our tool MONITIZER [Publication (D)], which allows for optimization and transparent evaluation of NN monitors tackling the latter three challenges.

## Monitoring Methods

There is extensive research on monitoring methods for NN, with a comprehensive overview in two surveys [Yan+21] and [Sal+22]. This section summarizes the most relevant monitors that we also used in Publication (D).

Most OOD detection methods use the output of the last layer, the *logits*. Among them, MaxLogit [ZX23] considers the highest valued logit from the NN output and assumes that a higher value corresponds to a higher certainty of the NN’s decision. Similarly, MDS [Lee+18] analyzes the confidence scores of the NN’s predictions, and Softmax [HG17] uses the maximum softmax probability (MSP) from an NN’s output to predict whether an input is OOD. Other works compute scores based on the logits, like the Energy score [Liu+20a], a scaled Energy score [Guo+17], the entropy [Mac+21], or the Kullback-Leibler divergence [Hen+22a].

Other than the last layer, some works consider additionally the *gradients* of the loss function to have a better OOD detection [HGL21] or use the gradient of the inputs for a distortion to create a sharper distinction between OOD and ID [LLS18].

Some works consider information from the *intermediate layers* for the prediction by computing the relative Mahalanobis distance [Ren+21], the Simplified Hopfield Energy [Zha+23b], using k-nearest-neighbor classification [Sun+22], or by creating virtual logits [Wan+22]. Some add to that scaling of the activations [Dju+23], clipping of the activations [SGL21], or altogether dropping some by sparsification [SL22].

The most similar work to our Gaussian monitor is [HLS20]. Similar to our approach, the authors consider the *activation values* of the neurons. For each class in the dataset, they collect the activation vectors of the class samples and cluster them. They construct a box abstraction for each cluster, such that each class corresponds to a set of boxes. Finally, during testing, they check whether the activation vector of a new sample is contained in one of the boxes of its predicted class; if not, they raise an alarm.

## Tooling for Monitor Generation

OPENOOD [Zha+23a] offers task-specific OOD detection benchmarks, each containing an ID set and multiple OOD datasets for a task (e.g., Open Set Recognition and

Anomaly Detection). It contains several monitors from the literature, including training methods for NNs for better OOD detection and data augmentation methods. However, it lacks the functionality to tune the monitors for a given objective, which is essential for industrial users. Additionally, there is no comprehensive and transparent evaluation of the monitors and no justification for choosing the OOD sets.

PYTORCH-OOD [KFO22] is a library for OOD detection, yet despite its name, it is not part of the official PyTorch library. It includes several monitors and datasets and supports the evaluation of the integrated monitors. However, it does not support the optimization of the monitors for a given objective and lacks a transparent evaluation.

While a framework exists to optimize an NN monitor during runtime [Kat+22], it is meant for active learning. That is, after generating a monitor, it can be updated during runtime. Additionally, this framework is not built for extensibility and reusability, as it already lacks an executable.

### 4.2 Contribution: Gaussian Monitoring

This section describes our approach from Publication (E) without further citations. It introduces a **novel method** for detecting OOD inputs in NNs during runtime using **Gaussian models**. The introduced method is **simple** and **scalable** while performing better than existing methods<sup>1</sup>.

Gaussian models are typically used to describe the likeliness of a behavior. We apply this concept to model the “behavior” of a neuron by a Gaussian distribution. We build upon prior work [CNY19], proposing to use “activation patterns” of neurons. The authors monitor which subsets of neurons are activated for known inputs; whenever a very different subset is activated, they raise an alarm. This idea was further refined in [HLS20], where activation values of neurons are enveloped into hyper-boxes (multidimensional intervals); whenever a very different value is observed (outside the boxes), an alarm is raised.

Our method advances the box abstraction concept by replacing the discrete and exact enveloping with a more continuous and fuzzy representation. We achieve this by learning a Gaussian model of each monitored neuron, providing a more adaptable representation of the neuron’s behavior and allowing for rare outliers.

---

<sup>1</sup>At the time. Note that most of the monitors mentioned in the section before were published after our publication.



### 4.2.1 Approach

The Gaussian monitor is created by finding Gaussian models for each neuron. We first extract activation values from the NN for a given set of input values and model each neuron as a normal distribution. We demonstrate this principle in Example 4.1 and Figure 4.2.

#### Example 4.1: Gaussian Monitor.

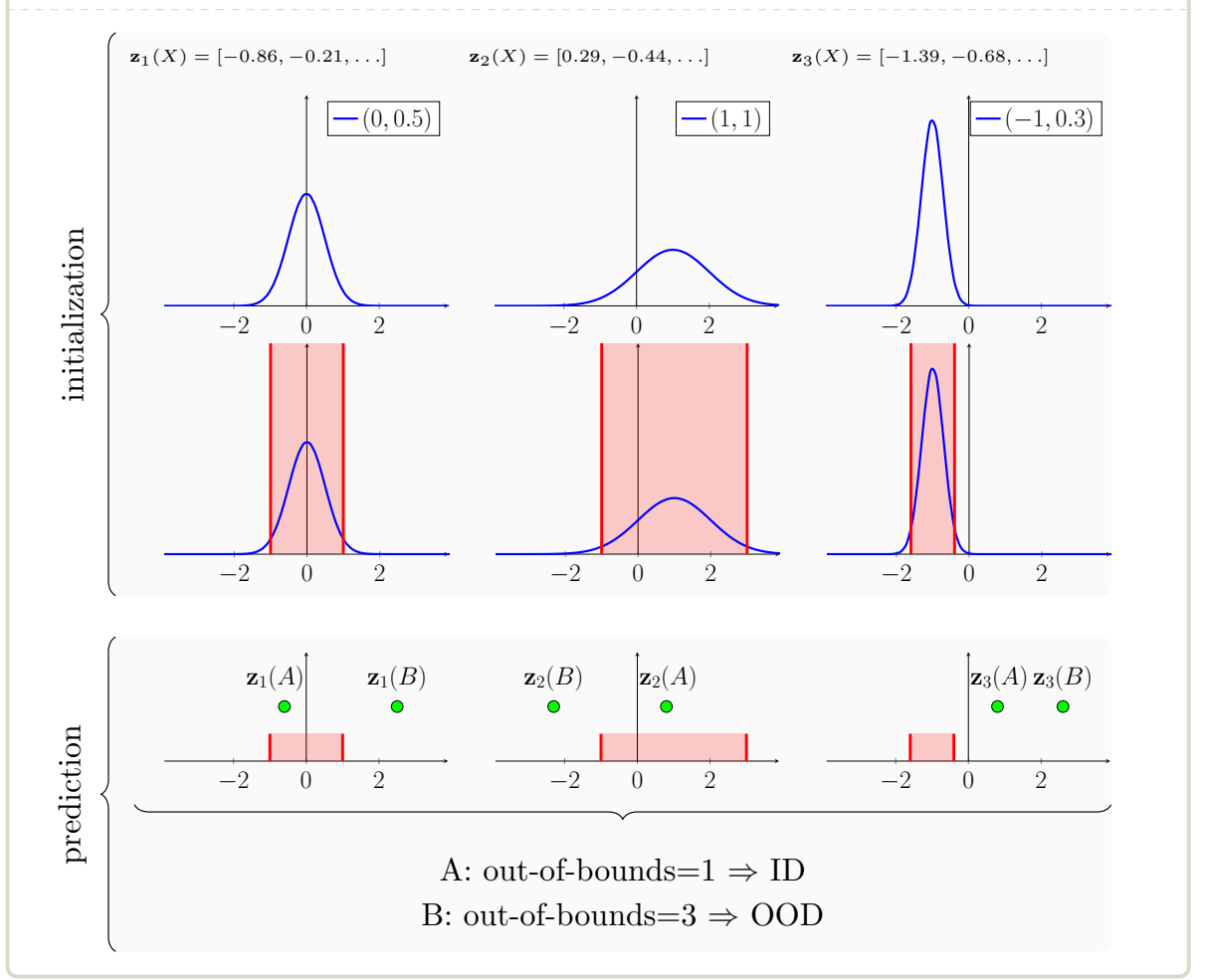
We describe the idea of the Gaussian monitor using the illustration in Figure 4.2. While the monitor usually tracks the behavior of all neurons in an NN, in this example, we assume that we are only tracking the behavior of *three* neurons for demonstration purposes. We gather the activation values (Definition 2.3) for those three neurons in  $\mathbf{z}_1(X), \mathbf{z}_2(X), \mathbf{z}_3(X)$  for some finite set  $X$ , shown in the first row of the Figure.

Based on these values, we fit a Gaussian model to each neuron. In the example, we find the best configurations  $(0, 0.5), (1, 1), (-1, 0.3)^a$ , where a configuration is (mean  $\mu$ , standard deviation  $\sigma$ ). We plot the normal distributions in blue in the second row.

We create an interval that contains the 95th percentile of the Gaussian. For normal distributions, these intervals can be computed by taking  $[\mu - 2\sigma, \mu + 2\sigma]$ . Afterward, each neuron is assigned its interval and the initialization phase is completed. We show these intervals in the third row as red boxes.

In the prediction phase, i.e. during runtime, we get two new inputs to the NN. We call them A and B and compute  $\mathbf{z}_1(A), \mathbf{z}_2(A), \mathbf{z}_3(A)$  and  $\mathbf{z}_1(B), \mathbf{z}_2(B), \mathbf{z}_3(B)$  (as plotted in green in the last row of plots). Then, we count how often these values are within the intervals of the neurons. Input A is contained in the intervals of neurons one and two, but B is in no interval. We can decide whether to consider an input OOD based on these *votes*. In this example, we want the activation value of a new input to be within the interval of at least two neurons. Therefore, A is considered ID, and B is considered OOD.

<sup>a</sup>We all know that the example was chosen in this way that the Gaussians look nice.

**Figure 4.2:** *Exemplary Depiction of the Gaussian Monitor.*

After collecting the activation values for the neurons, i.e.,  $\mathbf{z}_i(X)$ , we fit a normal distribution to these vectors, which consists of finding the mean and the standard deviation. Additionally, we fit a separate Gaussian distribution to each neuron for each possible *output class* of the NN, which we have not demonstrated in Example 4.1. Since we assume a normal distribution of activation values, we expect the majority of samples to fall within the range  $[\mu_i^{(n)}(c) - k\sigma_i^{(n)}(c), \mu_i^{(n)}(c) + k\sigma_i^{(n)}(c)]$ , where  $k$  is typically a value close to 2. This interval then contains 95% of the samples, i.e., the 95th percentile.

**Definition 4.1:** *Gaussian Monitor.*

Let  $N$  be a classification NN that distinguishes between  $k$  different classes. For all layers  $n \in \{1, \dots, L - 1\}$ , we get the activation values  $\mathbf{Z}^{(n)}(X)$  for some set

$X$ . For each class  $c \in \{1, \dots, k\}$ , each layer  $n$ , and each neuron  $i$  in this layer, we define the mean  $\mu_i^{(n)}(c)$  and standard-deviation  $\sigma_i^{(n)}(c)$  of  $\mathbf{z}_i^{(n)}(X_c)$ , where  $X_c = \{\mathbf{x} \in X \mid f(\mathbf{x}) = c\}$ , i.e. all activation values of the neuron that are classified with label  $c$ .

The Gaussian Monitor is then defined as the tuple  $(M, \Sigma, T)$ , where  $M = \{\mu_i^{(n)}(c)\}_{i,n,c}$ ,  $\Sigma = \{\sigma_i^{(n)}(c)\}_{i,n,c}$  for each neuron  $i$ , layer  $n$ , and class  $c$ , and  $T = \{\tau_1, \dots, \tau_n\}$  the thresholds for each layer  $n$ .

Given a Gaussian monitor, we can predict whether a new input  $\mathbf{x}$  would be considered OOD or ID. The input  $\mathbf{x}$  is fed to the NN, and we record the predicted class  $c$ . We also retrieve the activation values  $\mathbf{Z}^{(n)}(\mathbf{x})$ . For each neuron  $i$ , we check if the activation value for  $\mathbf{x}$  lies within the interval  $[\mu_i^{(n)}(c) - 2\sigma_i^{(n)}(c), \mu_i^{(n)}(c) + 2\sigma_i^{(n)}(c)]$ . This information is collected for all neurons in all monitored layers. For each layer  $n$ , we determine whether the number of neurons with the activation value outside the interval exceeds the threshold  $\tau_n$ . The results from all layers are then combined using a boolean AND, i.e., the sample  $\mathbf{x}$  is considered OOD if, in *all* layers, the number of neurons where the activation value is outside the interval exceeds the respective threshold.

## 4.2.2 Experimental Results

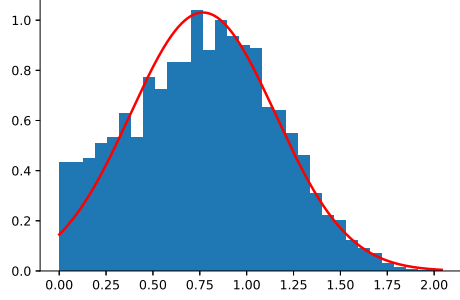
In our experiments, we evaluate the Gaussian monitor on four NNs across four different datasets: MNIST[LCB10], FashionMNIST[XRV17], CIFAR-10[KH+09], and GT-SRB[Sta+11]. The evaluation has two primary objectives: first, to verify our assumption that the activation values of the neurons are normally distributed, and second, to compare the performance of our monitor with an existing approach [HLS20].

### Gaussian Assumption

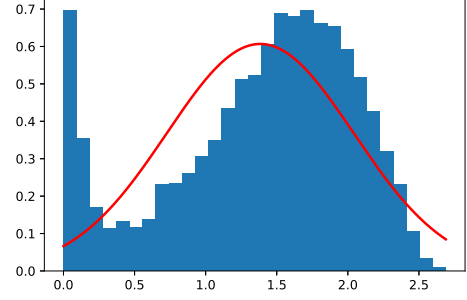
Our findings indicate that our assumption that the activation values of the neurons are normally distributed is generally valid. Figure 4.3 presents some of the results of Publication (D) from different datasets. Although the distributions are not always perfectly normal, they are typically close to a Gaussian distribution. We can also see the influence of the ReLU in Fig. 4.3 (b), as there is a spike at value 0.

**Figure 4.3:** *Histogram of Activation Values.*

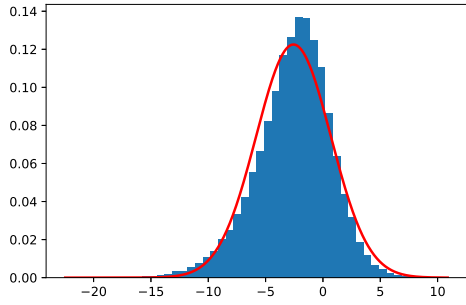
We show the histogram of the activation values on the training set for one randomly chosen neuron in a random layer of the NN for each of the four different datasets. The values are shown in blue, and the fitted Gaussian in red.



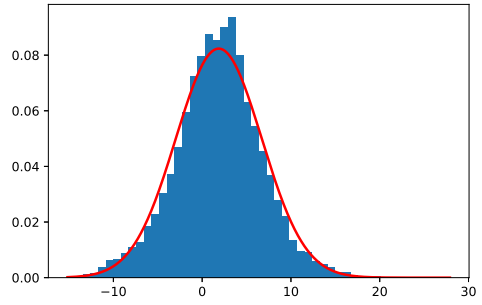
(a) MNIST



(b) FashionMNIST



(c) CIFAR-10



(d) GTSRB

### Comparison to the Box-monitor

In our evaluation, we compare the performance of the Box-monitor [HLS20] to our approach on one NN per dataset. We defined a specific case of OOD by training the NN only on a subset of the classes of the dataset. For instance, the MNIST dataset [LCB10] contains images of digits from zero to nine; we train the network on the digits zero to five and consider six to nine as OOD. We choose this set of OOD to ensure that the OOD images are close to the original images, but the NN is not trained to recognize them explicitly. Our findings indicate that the performance of our approach is generally comparable to or better than the Box-monitor [HLS20]. Notably, on CIFAR-10, our

approach significantly outperforms the Box-monitor in accurately detecting OOD inputs.

Overall, our approach is promising and achieves good results. However, we observe challenges in balancing between the correct detection of OOD inputs while minimizing the false alarms, i.e., wrongly predicting ID as OOD. This issue highlights the challenge already mentioned in Figure 4.1, for which we propose a solution approach in the next section.

### 4.3 Contribution: Monitizer

This section introduces the results from Publication (D) without further citations. Let us summarize the three challenges we addressed in developing MONITIZER: selection of hyperparameters, transparent evaluation and tool support for monitor development. MONITIZER effectively addresses all these challenges since it is a new modular tool for the **optimization and evaluation** of NN monitors.

#### Summary of Contributions:

- A **library** of 19 monitoring methods, 9 datasets, and 15 pretrained NNs.
- A **one-click solution** to develop a monitor for a specific application, making it highly suitable for industrial applications.
- It contains a well-developed **classification of OOD**, some of which are directly and automatically generated for any given ID dataset, allowing for a transparent evaluation.
- It allows for an **optimization of the hyperparameters** of a monitor for a specific objective.

**Improvement on State of the Art.** All existing tools for NN monitoring, including OPENOOD, PYTORCH-OOD, and MONITIZER, contain several monitors and datasets, but there are notable differences. Unlike OPENOOD and PYTORCH-OOD, MONITIZER provides functionality to tune monitors for a given objective and supports a transparent evaluation of monitors on a specific ID dataset by automatically providing generated OOD inputs. Apart from that, MONITIZER provides a *one-click* solution that evaluates all monitors to return the best available option for a given objective, making

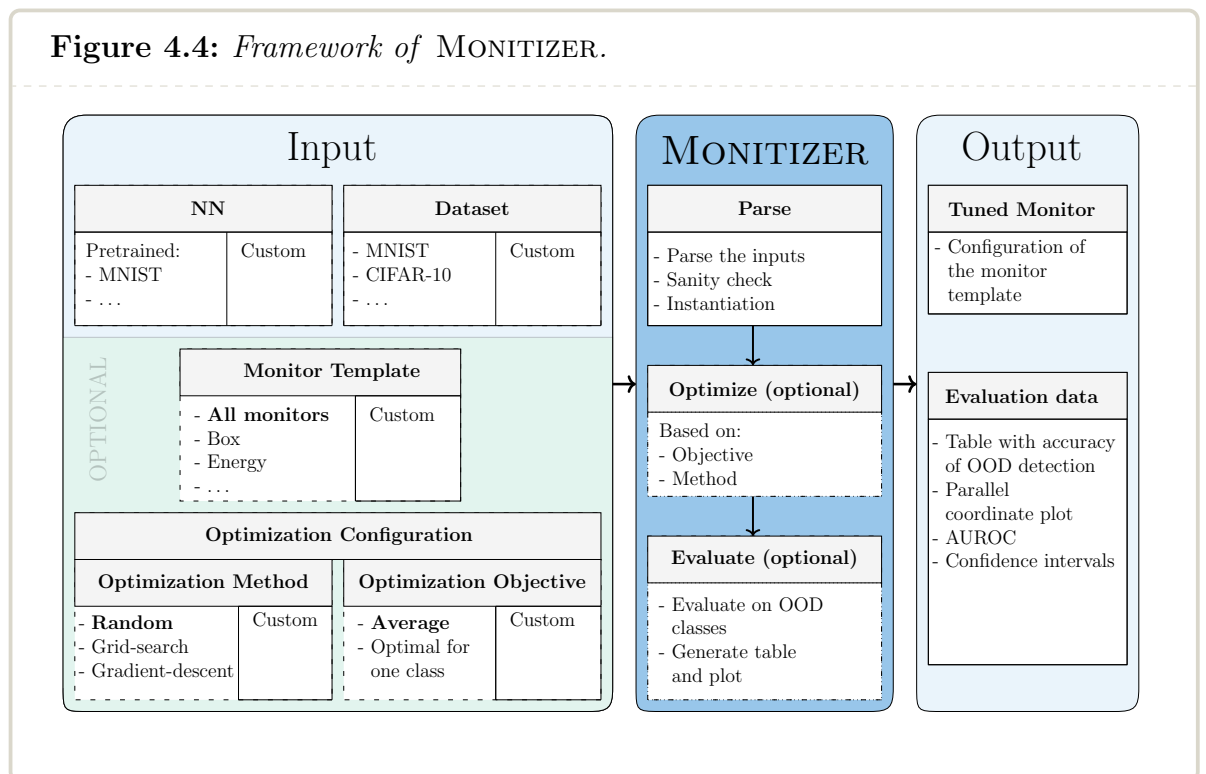
## 4. Neural Network Monitoring

it intuitive to use for non-experts. Built with extensibility in mind, MONITIZER allows to change the NNs, the ID dataset, the monitor, the objective, the OOD datasets, and the optimization methods with very little effort, making it a versatile and user-friendly tool.

### 4.3.1 Framework Description

The key components of MONITIZER are the optimization and the evaluation of NN monitors. Monitors contain parameters that must be specified before use, e.g., which layers the Gaussian Monitor should monitor or the thresholds. These values are often set manually. Some parameters can also be set by specifying a desired performance on the ID set since they directly affect this performance. For specific applications, it is preferable to optimize them for this particular setting. MONITIZER allows the user to define their objective and optimizes the parameters accordingly.

Figure 4.4 illustrates the framework of MONITIZER, which is described in more detail later on.



## Input

MONITIZER accommodates various use cases by allowing a customization of all inputs. As input, it requires an NN that should be monitored, and the corresponding ID dataset, as a monitor can only be created for a specific NN, and can only be evaluated on a given ID dataset. If no additional configuration is provided, MONITIZER automatically executes the one-click method, which evaluates the Area Under the Receiver Operating Characteristic Curve (AUROC)<sup>2</sup> of all integrated monitors on a set of automatically generated OOD inputs.

Alternatively, MONITIZER allows users to select a monitor template, which is a monitor without pre-configured parameters. This template can be one of 19 implemented monitors from the literature (see also the related work above) or an implementation provided by the user.

Additionally, the user can specify the optimization configuration, which is the crucial difference between MONITIZER and other available tools: the possibility to optimize a monitor. First, the user must define the *optimization objective*, i.e., what to optimize for. For this, MONITIZER provides implementations for standard functions and allows the user to define their own. The user can also define the *optimization method*, which allows them to choose from *random*, *grid-search* or *gradient-descent*. This way, the user can receive a monitor perfectly adapted to their use case.

## Phases of the Tool

MONITIZER operates in several phases: parsing, optimization, and evaluation, where the latter two are optional.

**Parsing** MONITIZER parses the inputs, loads the NN and dataset, and instantiates the monitor(s) with default values if available.

**Optimization** During this phase, MONITIZER optimizes the parameters of the monitor template(s) to maximize a specified objective. This process can vary depending on the optimization method and objective. The optimization method searches through the search space of candidates for the parameters of the monitor. Depending on the chosen method, this happens randomly, by specifying a search grid,

---

<sup>2</sup>The Receiver Operating Characteristic (ROC) curve shows the performance of a binary classifier with different decision thresholds. The AUROC computes the area under this curve. The best possible value is 1, indicating perfect prediction.

or using gradient descent on the optimization objective value. Once the method chooses a candidate, it evaluates the optimization objective on it. Based on the evaluation outcome, the search may stop or continue for another iteration. MONITIZER also supports multi-objective optimization. In this case, however, there is no singular optimal solution. Instead, MONITIZER provides the user with a Pareto curve of optimized solutions.

**Evaluation** In this phase, MONITIZER evaluates a (possibly optimized) monitor on the OOD classes. In our work, we introduce a classification of OOD data. At the highest level, it distinguishes between *generated* and *collected* OOD inputs. Where the first is automatically produced by MONITIZER (e.g., by adding noise or perturbation), the second must be manually selected. This classification ensures a transparent evaluation, as the evaluated datasets are clearly defined.

The user can choose to opt out of optimization or evaluation, which is useful if they already possess a well-tuned monitor and wish to utilize the evaluation framework of MONITIZER, or because they want to tune a monitor but not evaluate it (yet).

### Output

The output of MONITIZER includes a tuned monitor optimized for a given objective (if selected) and the corresponding evaluation results. The latter consists of a table detailing the prediction accuracy for OOD detection and the accuracy for predicting ID inputs. Additionally, MONITIZER presents this table as a parallel coordinate plot for a better graphical understanding. If the monitor was not optimized, MONITIZER computes the AUROC, which does not require defining the threshold. On request, MONITIZER provides confidence intervals for all computed values.

#### **4.3.2 Evaluation**

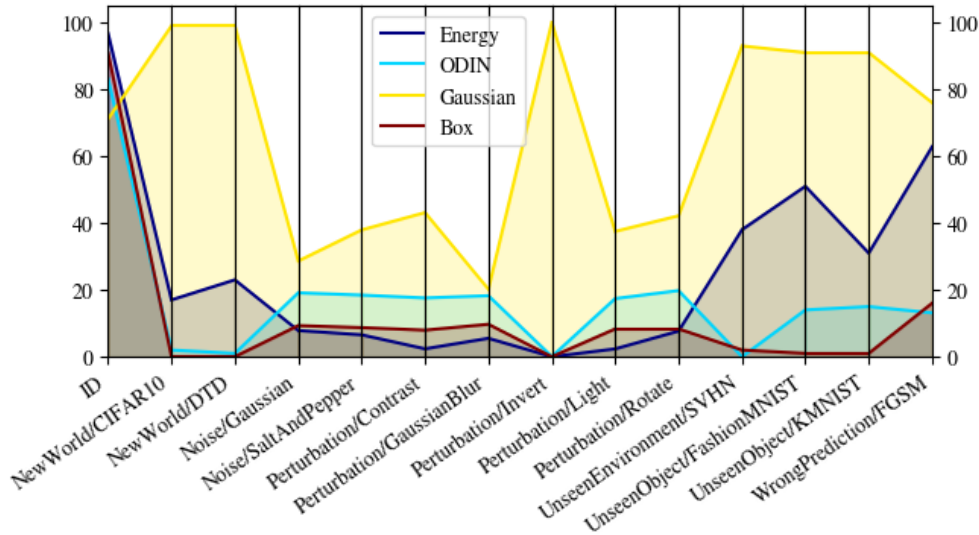
Figure 4.5 shows an exemplary output of MONITIZER. In this case, we evaluate four different monitors (Energy [Liu+20a], ODIN [LLS18], Box-monitor [HLS20], and Gaussian monitor [Publication (E)]) on all available OOD datasets. We can see that three monitors have a similar performance on the ID dataset but varying performances on the OOD datasets. Mainly, Energy performs noticeably better on the collected OOD, namely UnseenObject, UnseenEnvironment, and WrongPrediction. We can also see that



the Gaussian monitor performs better on all OOD samples with a worse performance on ID. This shows the practical appearance of the issue raised in Figure 4.1: Having a better performance on ID leads to a worse performance on OOD and vice versa.

**Figure 4.5:** *Sample Output of MONITIZER.*

This figure shows a parallel coordinate plot as it is output by MONITIZER. There are four monitors: Energy [Liu+20a], ODIN [LLS18], Box-monitor [HLS20], and Gaussian monitor [Publication (E)]. We tuned them on an NN that was trained on MNIST [LCB10] and optimized randomly for the detection of CIFAR-10 [KH+09] as OOD while keeping a 70% accuracy on predicting ID.



For further examples of how MONITIZER works, we refer to Publication (D) and the repository of the tool (<https://gitlab.com/live-lab/software/monitizer>).

## 4.4 Future Work

MONITIZER facilitates the comparison of monitoring methods, clearly highlighting their strengths and weaknesses. We can leverage this information to build an ensemble of monitors, balancing their weaknesses and strengths. Moreover, it allows us to conduct a study on existing monitoring techniques using MONITIZER's transparent evaluation. Such studies can provide valuable insights into patterns and methodologies that show

#### *4. Neural Network Monitoring*

similar behaviors and spark new ideas for monitoring methods. The Gaussian monitor was adapted for object detection [Has+23]; therefore, we also plan to extend MONITIZER for NNs that are not restricted to image classification, such as object detection.

## **5 Use Case - River Temperature Prediction**

In this chapter, we examine a use case of NNs and emphasize the need for deeper insights and understandability beyond simple metrics such as accuracy. The efficacy and versatility of NNs have enabled their application across a wide range of research areas: mechanical engineering [Ren+20], building construction [PM21; Con+21], physics [SBV21; Cai+21], chemistry [Sch+21; MC21], human resources [WJ21], biology [CA21], and many more. In our work, we use NNs to predict the temperature of streams.

Given the critical influence of temperature on abiotic and biotic processes [Ans+20], expected changes in stream water temperatures from global warming and other human activities are likely to significantly impact aquatic ecosystems [SMI81]. With the rapid progression of global warming [Pör+22], there is an urgent need for detailed and reliable prediction of water temperature in streams to allow timely and effective countermeasures. In Publication (B), we focus on predicting stream water temperature using NNs and analyzing their behavior for a more understandable evaluation. We extend this with further data in Publication (F).

### **5.1 State of the Art**

There are various methods to predict the water temperature of streams. The most straightforward approach is linear regression, which assumes a direct correlation between air and water temperatures [Cai06; Kri+13; PFS98; RHŠ15]. However, physical effects lead to a non-linear relationship between the two parameters [MS99], rendering these approaches unrealistic. Stochastic modeling offers more complexity through techniques like multiple regression analysis [Cai06; CES98; Ahm+07], second-order Markov processes [CES98], Box and Jenkins time-series models [CES98; Ahm+07], and second-

## 5. Use Case - River Temperature Prediction

order autoregressive models [Ahm+07]. Additionally, ML methods, including Gaussian process regression and decision trees, have been employed [Zhu+19], and, recently, NNs have also been utilized for this purpose.

Rabi, Hadzima-Nyarko, and Šperac [RHŠ15] evaluate many different models for temperature prediction, among which NNs perform best. However, their study is limited to one specific river, and they evaluate the models solely based on correlation coefficients, which is insufficient to assess the actual performance of a complicated model like an NN. Similarly, in [Zha+18], NNs outperform other ML models but they are only evaluated based on the Root Mean Squared Error (RMSE)<sup>1</sup>. While this gives a general indication of the NN’s performance, it does not offer insight into the robustness or reliability of the model.

Numerous verification methods exist for NNs [Kat+17; Kat+19; Geh+18; Sin+19a; Sin+19b; Zha+18], but they naturally focus on proving a particular property of the NN. Therefore, domain experts must define the desirable property to be verified. Additionally, many of these methods are tailored to classification NNs, making their application to regression tasks more challenging.

## 5.2 Contribution

The contributions of this chapter are threefold: First, we **develop NNs** for predicting the water temperature and evaluate them on existing metrics, improving their precision compared to related work. Second, we introduce **novel analysis methods** for regression NNs, allowing for a more comprehensive investigation of the developed models. Lastly, we employ our analysis methods for a **hydrological study** to evaluate more NNs, focusing on different features provided from hydrology experts.

### 5.2.1 Water Temperature Prediction

In this section, we outline the development of the NNs employed for temperature prediction from Publication (B) without any further citation. We begin with a description of the data, followed by an analysis of the models and their performance.

We use three inputs for the training of the NNs, which are all publicly available:

---

<sup>1</sup>The root of the average Euclidean distance of the true and predicted values

- **Air temperature:** Daily mean temperature at the closest available measurement site that is between 3 and 50km of the stream [Deu]
- **Runoff:** The part of the precipitation that flows into the streams and rivers [Bay]
- **Date:** The date represented as a value in the interval of 0 to 366

In this work, we use a one- to three-day time window as input for the NNs, with the output being the predicted river temperature for the following day. This allows for randomly splitting of the dataset into training and test sets.

The NNs used in this work are relatively small compared to more advanced applications like image processing. Since previous approaches displayed good performance using simpler models [Zha+18], we conclude that this size is adequate, and we opt for smaller models with a modest number of layers and neurons. Specifically, we use fully connected NNs with up to three hidden layers and a total of 90 neurons. Each of the six rivers in the dataset is modeled with a separate NN, as our experiments indicate that the current input features are insufficient to train one general model for all rivers, given their significantly different performance.

Our models achieve a RMSE ranging from 0.47 to 1.57, which is better than most related works. Interestingly, the RMSE varies on the rivers, although all hyper-parameters (architecture of the NN, training method, etc.) were the same. Additional input features, such as the amount of sunlight the river receives and information about the entire course of the stream up to the measurement site, may enhance the performance of the NNs.

### 5.2.2 Model Analysis

ML models are typically evaluated using specific metrics. We use the accuracy or classification tasks, which denotes the percentage of correctly classified inputs. Regression models are usually evaluated by metrics that compare measured values with predicted values, such as RMSE, Mean Absolute Error (MAE), Mean Squared Error (MSE), or  $R^2$  [Ark23, Chapter 2.5]. Simple models, such as linear regression, show straightforward and well-defined behavior and do not require any additional analysis. However, the behavior of NNs is much more complex and poorly understood, underlined by extensive research on explainability [Sam+21]. To enhance the reliability and trustworthiness of NNs, we introduce three additional metrics to analyze their behavior.

## 5. Use Case - River Temperature Prediction

**Idea.** Functional analysis inspired the analysis methods since it typically involves tasks such as computing maximal and minimal points, examining the gradients, and determining zero points. We want to use them to inspect the black-box of NNs.

### Analysis methods from Publication (B):

**Robustness Analysis** This method computes empirical robustness values of the NN, which can be seen as finding the maximum value of the derivative.

**Min-Max Analysis** As the name suggests, we evaluate the NN empirically to find its maximal and minimal values for a specified range of input values.

**Impact Analysis** In this method, we evaluate the importance of each of the input features of the NN, similar to computing partial derivatives.

### Robustness Analysis

In classification tasks, local robustness refers to the model remaining “robust” against small perturbations in the input and still classifying it correctly [Kat+17]. However, for regression models, which do not involve classes, we redefine local robustness to a continuity criterium: Given an input and its neighborhood, how much does NN’s prediction change at most within this area?

Evaluating the robustness provides insight into how sensitive the NN is to its input. Low robustness indicates that even slight perturbations in the input can lead to significant differences in the output. To assess the robustness of our NNs, we use DEEPPOLY [Sin+19a] originally designed to evaluate the local robustness of classification models. With minor modifications, we can make it work for our setting. For a given  $\varepsilon$ , we empirically compute the minimum, maximum, and mean absolute derivation  $\delta = |f(y) - f(x)|$  for inputs  $x, y$  to the NN and  $|x - y| < \varepsilon$ .

**Results.** The average value of  $\delta$  typically increases with the number of neurons per layer, indicating that larger NNs tend to be less robust. In particular, in some instances, for  $\varepsilon < 1$ ,  $\delta$  exceeds one, which means that an input change of at most one degree Celcius in air temperature could result in the NN predicting a river temperature change of up to three degrees Celcius. This discrepancy is undesirable because, intuitively, the temperature of a river should not vary by that much with minor air temperature change.

This suggests that the NN lacks sufficient robustness or needs more input features to account for such variations.

### **Min-Max Analysis**

We can quickly determine the minimum or maximum output value for simple models, such as linear regression or trigonometric functions. While this is not straightforward for NNs, these values can provide insight into how realistic the computed function is. By using gradient descent [RN20, Chapter 4.2], we optimize the inputs of the NN to maximize or minimize the output. Unlike NN training, this approach optimizes the input values instead of the internal parameters. To ensure that the inputs remain realistic, we constrain them within plausible ranges, such as  $-45^{\circ}\text{C}$  to  $60^{\circ}\text{C}$  for temperature, and  $-1$  to  $40\text{ m}^3/\text{s}$  for runoff, which slightly exaggerates typical real-world conditions. Due to the complexity of NN, finding a global optimum is challenging [HBH02]. Therefore, we start the optimization from several random starting points and choose the overall found maximum and minimum, respectively.

**Results.** The output values of the NN vary between  $-19^{\circ}\text{C}$  and  $61^{\circ}\text{C}$ . The lower temperature of  $-19^{\circ}\text{C}$  could be an almost plausible water temperature in a very shallow river that completely freezes during an exceptionally harsh winter. However,  $61^{\circ}\text{C}$  is not realistic under any circumstances. On the other hand, we achieve these extreme values only with unrealistic input scenarios, such as  $-45^{\circ}\text{C}$  on one day and  $60^{\circ}\text{C}$  on the next, indicating that these extreme values are only outliers. For a more accurate and realistic evaluation, it is necessary to define realistic inputs clearly, also in relation to each other, and to determine the corresponding maximum and minimum values.

### **Impact Analysis**

In introducing the *impact analysis*, we adopt the concept of sensitivity analysis [ZMC94]. This approach identifies the input features to which the NN is most sensitive. Specifically, it measures how much an input feature of the NN contributes to the final output value. The simplest method for computing this involves computing the gradient of the output with respect to each input feature. This value indicates how much a change in the input feature affects the output.

By comparing these gradients across different input features, we can assess their rel-

## 5. Use Case - River Temperature Prediction

ative importance in the overall computation. This analysis also allows us to investigate the difference between the streams.

**Results.** Our results indicate that this evaluation effectively identifies the most crucial parameters for good-performing models. These findings align with our other experiments, where we trained the NN using a subset of the input features. Less relevant features can be excluded from the input set without diminishing the performance of the NN. Additionally, we discover that the different streams have varying important features, suggesting that we still need some distinguishing information between the rivers to explain this behavior.

### 5.2.3 Statistical and Hydrological Evaluation

We extend our work by including a more hydrological perspective in Publication (F) in this section without further citations. We explore a question left open after the previous approach: Why is there a difference in the performance of the NN for varying rivers?

#### Summary of Contributions:

**Input Features** We evaluate various combinations of input parameters, including air temperature and discharge (as in Publication (B)), and additionally, water level and sunshine duration. We confirm the results from the previous work that the different streams need a different set of input features for optimal performance.

**Influence of Environmental Factors** We include a hydrological inspection, such as land use in the riparian strip and the size of the data set. We discover that forested land around the stream correlates with a better performance of the NN, while grasslands have a negative impact. We also find that the predictions are more accurate for streams closer to their source.

**Model Performance Metrics** Next to the RMSE and the known analysis methods from before, we also compute the Pearson's correlation coefficient (R) and the Percent Bias (PBIAS) [GSY99]. However, the RMSE is the most critical indicator, validating its use in our earlier work.



Applying the analysis method from Section 5.2.2 yields results consistent with our previous findings.

We normalize all input features to a range between zero and one, with perturbations set at 0.01 (approximately  $0.8^{\circ}\text{C}$ ). The **robustness analysis** reveals an average change in the output of  $2^{\circ}\text{C}$ , with the highest change observed in the stream Otterbach at  $10^{\circ}\text{C}$ . The NNs in our second work incorporate more input features. We believe this higher number of features accounts for the increased impact of the perturbations. Additionally, we identified a significant statistical correlation between the R-value and the perturbation. This indicates that the R-value can serve as a preliminary indicator, potentially obviating the need for costly robustness analysis.

The **Min-Max analysis** reveals a significant divergence between the observed minimal and maximal values in the data ( $0^{\circ}\text{C}$  to  $23^{\circ}\text{C}$ ) and the output values computed by the NN ( $-24^{\circ}\text{C}$  to  $101^{\circ}\text{C}$ ). Furthermore, we detect a significant statistical correlation between the RMSE and the maximum values. This indicates that a poorer-performing NN tends to produce unrealistically high temperature values.

The **impact analysis** supports our earlier findings that the NNs for different rivers assign varying importance to input features. While some NNs primarily rely on air temperature, others find water level the most computationally relevant feature. We trained various NNs with different sets of input features and compared the results to the impact analysis. They are primarily consistent: if a feature has a high impact, the NN's performance worsens when the feature is omitted. However, we observed that the sunshine duration, which had the lowest measured impact (often near zero), still contributed to better performance when included as an input feature. This indicates a limitation in the insight of the impact computation and underscores the need for alternative measures.

## 5.3 Future Work

### Our work

A promising direction for future research involves exploring additional input parameters to enhance the performance of the NNs. Our findings suggest that the NNs perform best

## 5. Use Case - River Temperature Prediction

on headwater streams close to their source. Therefore, incorporating data on vegetation around the stream and the measurement site or accumulating such information along the river course prior to the measurement site could be beneficial. To develop a general model applicable to various rivers, it is essential to identify the distinguishing input features for the rivers.

Additionally, we have yet to evaluate the performance of the NNs over time. Investigating whether the performance degrades over time represents an intriguing and relevant direction for future research and toward the goal of a robust prediction model. Furthermore, examining the impact of the current influence of climate change on model performance could provide valuable insights into how to make the model more robust to future changes.

### **Other work**

Since the publication of our works (B) and (F), there have been some advancements in the prediction of water temperatures of rivers using NARX-based models (Nonlinear AutoRegressive model with eXogenous inputs) [Zhu+24], which is a type of NN that receives as input external values and retains information about its prior performance. The use of ML models was extended to graph NN for the prediction of water flow forecasting [Rou+23]. Our recognition of the need for additional evaluation criteria for NNs has led to new metrics for assessing Reinforcement Learning (RL) controllers [Cha+22]. Furthermore, in [Pan+24], the authors study how the stream temperature varies across different locations and times of the day, building on our finding that riparian vegetation significantly influences the river temperature.

## 6 POMDP Strategy Representation via Automata Learning

In this chapter, we dive into POMDPs and how to represent their strategies. POMDPs are typically used as a computational model of reality whenever we want to model a situation where the agent does not have complete information about the state of the world. This makes them very useful for applications that need a more accurate representation of reality, like robotics or autonomous cars, where not everything can be perceived or predicted.

In Publication (C), we focus on *representing strategies* that perform well in practice without being provably optimal. In the following, we present our results without further citations to the original publication. There are three core aspects of strategy synthesis procedures:

1. *quality* of the synthesized strategies,
2. *size* and *explainability* of the representation of the synthesized strategies,
3. *scalability* of the computation method.

These objectives do not always align. For example, a bigger strategy could produce a better value, or it might take longer to produce a better strategy. Therefore, we must find a balance in optimizing for these aspects.

We use active automata learning as a core technique. It helps us to turn complex strategies into simpler forms, i.e., FSCs. Their compact structure makes them easier to understand and use without making them less precise. This balance between performance and simplicity is essential, especially where trust and understandability are crucial.

## 6.1 State of the Art

There are numerous approaches to solving planning problems in POMDPs, as evidenced by the literature [SS73; Hau00; SPK13]. Many state-of-the-art solvers employ point-based methods such as *PBVI* [PGT03], *Perseus* [SV05], and *SARSOP* [KHL08], which address both bounded and unbounded discounted properties. These methods typically utilize  $\alpha$ -vectors [KLC98] to represent the strategy, which is useful for analysis but heavily lacks explainability and involves a significant computational overhead. Notably, while *SARSOP* allows an export in automaton format, there is no explanation of how it is generated or what it exactly means. In the AI area, strategies for POMDPs are also represented as *Recurrent Neural Networks (RNNs)* [Car+19; Car+23; CJT21; DKT08; HS15]. However, all of these approaches suffer from the lack of interpretability of NNs (see also Chapters 3 and 4).

Further, some methods focus on exploring and resolving the belief MDP underlying a POMDP, particularly for optimizing infinite-horizon objectives without discounting [NPZ17; Bor+20; BKQ22]. However, the resulting strategies are typically large and contain redundant information.

Additionally, there are direct approaches to strategy synthesis, where strategies are created from a set of potential candidates [Han98; Meu+99a] and then applied to a POMDP to form a Markov chain. Search methods for these strategies include *inductive synthesis* [And+22], *gradient descent* [Hec+22; Meu+99b], and *convex optimization* [ABZ10; Jun+18; Cub+21].

There is a long history of representing strategies for POMDPs as FSCs [Meu+99b; Meu+99a; Han98; And+22]. The closest approach to our work is using inductive synthesis of FSCs for POMDP strategies [And+22]. This approach relies on the tool PAYNT [And+21], whose original purpose is automatically generating probabilistic programs. It was later adapted to generate families of candidate FSCs that are evaluated to find the optimal choice. This approach iteratively increases the size of the FSCs once it finds that the current size is insufficient. While this promises tiny representations, it also comes with a computational overhead of searching the whole design space.

In SAYNT [And+23], the idea of inductive synthesis is tightly coupled with STORM [Hen+22b]. While STORM performs *belief exploration* to gather information, SAYNT generates a candidate FSC that can improve the exploration. This tandem approach improves on the previous results of only using PAYNT [And+22].

## 6.2 Contribution

Publication (C) proposes a highly scalable **postprocessing methodology** for POMDP strategies that significantly enhances both the quality and the size of the representation. Our technique is **universally compatible** with existing frameworks that output a strategy, only requiring the ability to query the strategy function (i.e., to determine which action corresponds to a given sequence of observations). Notably, tools like STORM benefit significantly from our improvements in strategy quality and representation.

Our approach employs *active automata-learning* techniques to derive a compact FSC representation of a strategy. On one hand, we acquire a fully equivalent automaton representation, thereby preserving the original strategy’s value. On the other hand, we introduce *heuristics* that subtly modify the strategy and can improve the value and size.

### Summary of Contributions:

- A novel application of automaton learning for POMDP strategies.
- Transforming any given POMDP strategy into a similar compact FSC improving size and explainability, e.g., with at most ten nodes in more than 80% of the benchmarks.
- Enhancing strategy quality compared to the given input on 12 of 25 benchmarks and better than PAYNT on 19 benchmarks.
- Demonstrating scalability in challenging benchmarks, outperforming PAYNT on all but two benchmarks.

### 6.2.1 Automaton Learning

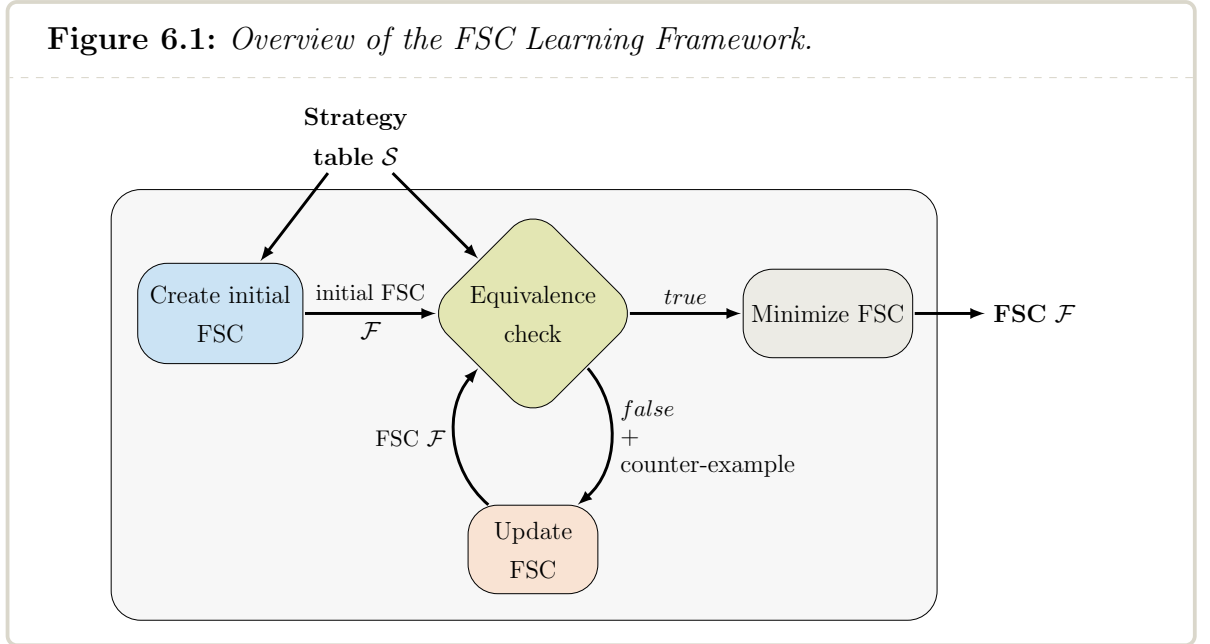
Since an FSC can be represented as a Mealy machine, our framework mimics an extension of the L\*-automaton-learning approach [Ang87] for learning Mealy machines [SG09]. The core difference between our approach and the standard algorithm lies in the nature of our sparse learning space: not all observations in a POMDP are reachable from all states. Consequently, numerous sequences of observations are inherently impossible within the POMDP. To mark such situations, we introduce a new symbol (a “don’t-care”

## 6. POMDP Strategy Representation via Automata Learning

symbol,  $\dagger$ ) denoting that the FSC has complete freedom to decide what to do.

Figure 6.1 provides an overview of the learning process. The input is expected to be a (partially defined) strategy in the form of a table mapping observation sequences in the POMDP to a distribution over actions. We create an initial FSC from this input, which serves as a first guess based on the following immediate observation. Depending on the complexity of the given strategy, this initial guess requires updates. The *equivalence check* then verifies whether the hypothesis FSC accurately represents the given strategy or identifies inconsistencies, which it returns as a counterexample. In the latter case, the FSC is updated. This process iterates until the strategy table and the hypothesis FSC coincide. Note that the algorithm is guaranteed to converge if the given strategy is finite and consistent.

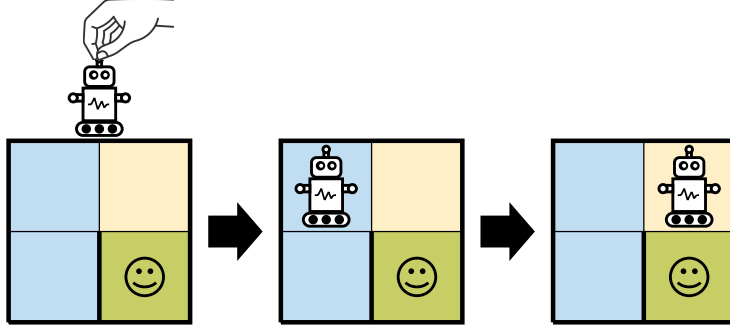
The presence of “don’t-care” symbols leads to a bigger FSC than necessary since the learning will try to distinguish outputs from actual actions. This leads to more states than required, and we need to perform an additional minimization step, which involves replacing “don’t-care” entries with actual observations to reduce the size of the automaton.



We demonstrate the automaton learning and minimization in Example 6.1.

**Example 6.1:** *FSC Generation for POMDP strategies.*

We consider a robot that is randomly dropped onto a  $2 \times 2$  grid, where it can land in any position (uniformly at random). The robot can move within the grid, not crossing any wall. However, its actions may fail with a certain probability. Its goal is to reach the goal state (marked in green with a smiley). The key problem is that the robot can only see whether it is still hanging (*init*) or the indicated colors: blue (*b*), yellow (*y*), goal (*g*).

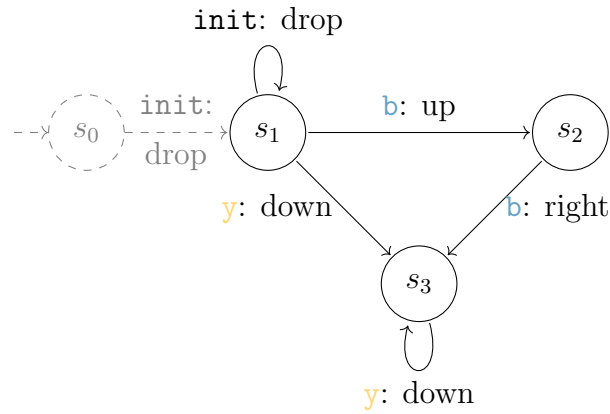


On the left side, we can see an example of a strategy table that our approach would receive as input. For some observation sequences, we have a given action in the table. For example, after seeing *init* and *b*, the robot is supposed to move up.

In this example, the initial hypothesis FSC is already equivalent to the table. On the right, we show the resulting FSC, where the dashed state is the one that is removed in the minimization. The solid states represent the final FSC. Note that the minimization removes only one state and creates a self-loop in state  $s_1$  instead.

Strategy Table  $\mathcal{S}$ :

Observation sequence	action
[ <i>init</i> ]	drop
[ <i>init</i> , <i>b</i> ]	up
[ <i>init</i> , <i>y</i> ]	down
[ <i>init</i> , <i>b</i> , <i>b</i> ]	right
[ <i>init</i> , <i>y</i> , <i>y</i> ]	down
[ <i>init</i> , <i>b</i> , <i>b</i> , <i>y</i> ]	down



### 6.2.2 Heuristics

In our application, using STORM, we encounter an additional situation beyond the “don’t care” scenario: STORM performs belief-exploration [Bor+20] to find a strategy that is inherently incomplete (in most cases). Therefore, in some states, we “don’t-know” what an optimal action could be. To overcome this, we introduce another symbol,  $\chi$ , representing states with reachable observations, in contrast to “don’t-care” but in which the best action remains unknown.

Such a situation with unknown optimal actions can also arise in other applications. Depending on the computation of the strategy, it may be incomplete, necessitating the use of the “don’t-know” symbol. While these symbols do not directly influence the learning process, they cannot be entirely disregarded. We must replace them by actual actions using some heuristics for the final FSC to yield a complete strategy (see Section 3.4 of Publication (C)).

We propose two heuristics for this case:

**Minimization** In this case, we replace every  $\chi$  with a  $\dagger$ . The minimization produces the smallest possible automaton by ignoring all “don’t-know” and “don’t-care” entries. Ideally, this results in a strategy that uses existing actions as candidates for the “don’t-know” situations.

**Distribution** In this case, we replace every  $\chi$  by a random distribution over all actions that have appeared before for the given observation. This results in an FSC where some output actions are randomized.

Our experiments show that none of the heuristics outperforms the other on all benchmarks. Since our learning is typically very fast, we propose to use a portfolio approach containing both heuristics.

### 6.2.3 Experiments

Let us repeat the three objectives for POMDP strategies: quality, size, and scalability. In our experiments, we evaluate the performance of our approach in light of each objective.

**Quality** To verify the good quality of our FSCs, we compare the value of our strategy to the value of the given strategy table and PAYNT [And+22]. Our FSCs always



perform equal and often better than the given table from STORM [Hen+22b]. Compared to PAYNT, our FSCs provide a better or equal strategy in 19 of 29 benchmarks.

**Size** To verify that our FSCs are compact, we compare the size of the FSC to the size of the given table and the size of the FSCs of PAYNT. Our FSCs are always smaller than the given table, often by magnitudes. Compared to PAYNT, we have smaller or equally-sized FSCs in 15 of 29 benchmarks, which is surprising given that PAYNT inherently generates the smallest possible FSCs.

**Scalability** To check the scalability of our approach, we compare the runtime of STORM plus learning with the runtime of PAYNT. We outperform PAYNT in 24 cases and are equal in 3 cases, but never worse. Note that the learning alone usually takes less than one second if there is no timeout; only in three cases does it take several seconds, proving that this approach is easily scalable even for larger instances.

Overall, our approach is considerably faster than PAYNT, with often better performance in terms of the resulting FSCs' size and quality.

## 6.3 Future Work

We can continue our line of research by integrating our learning framework in a tandem approach similar to [And+23], which promises even more improvement in the quality of the strategies. Additionally, we can find other heuristics to improve the learned automaton and get more information from STORM [Hen+22b] to increase their performance.

Lastly, there is research on learning automata with incomplete information [ST94; GKS03], which could be interesting in our case to mitigate the minimization in the end. It could allow our approach to be even more scalable since there are six benchmarks where STORM [Hen+22b] still provides a result, but the learning times out.



## 7 Conclusion

This thesis presents significant advancements in improving the safety of AI-based systems. These cover the safety and reliability of NNs and efficient representations of POMDP strategies.

For NNs, this thesis introduces a novel *abstraction* framework that creates a smaller version of the network while providing guarantees on the difference. We achieve this by replacing neurons with linear combinations based on semantic information, which yields better results than relying on syntactic information. Although the current abstraction techniques do not sufficiently reduce the problem to scale the verification of huge NNs to a realistic setting yet, this work makes an essential step in this direction. Especially the insight that semantic information is more valuable for abstraction offers a promising avenue for future research and improvement.

Additionally, this thesis introduces a novel, light-weight *monitoring method* for NNs using Gaussian models and shows its improved performance compared to existing approaches. To address the problem of developing and efficiently evaluating runtime monitors, we introduce MONITIZER, an extensible framework designed for both monitor developers and industrial users. With this, we provide urgently necessary tooling for developing runtime monitors in industrial settings and provide an environment for transparent and comparable evaluation of monitors. This tool significantly enhances the application of runtime monitors in real applications and eases their development by the scientific community.

Furthermore, we present a straightforward *automaton learning* technique for the strategy representation of POMDPs, enabling the transformation of any strategy into an FSC. This approach can be applied to less understandable representations, like RNNs or  $\alpha$ -vectors, provided they can be transformed into a table or directly integrated into the  $L^*$  learning framework. With this, we pave the way for converting less understandable strategies into the more human-readable format of an FSC.

## 7. Conclusion

Overall, this thesis contains solution approaches for problems related to the safety and reliability of AI systems. Towards a more scalable verification, we provide an approach for the abstraction of NNs, a monitoring method for NNs based on Gaussian models, tooling for monitor generation for more reliable NN-based systems, and a scalable method for transforming POMDP strategies to FSCs for more explainability. All of them contribute to a safer application of AI systems.

# Bibliography

- [ABZ10] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. “Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs”. In: *Auton. Agents Multi Agent Syst.* 21.3 (2010), pp. 293–320 (cit. on pp. 3, 48).
- [Ahm+07] Behrouz Ahmadi-Nedushan, André St-Hilaire, Taha BMJ Ouarda, Laurent Bilodeau, Elaine Robichaud, Nathalie Thiémonge, and Bernard Bobée. “Predicting river water temperatures using stochastic models: case study of the Moisie River (Québec, Canada)”. In: *Hydrological Processes: An International Journal* 21.1 (2007), pp. 21–34 (cit. on pp. 39, 40).
- [Alb21] Aws Albarghouthi. “Introduction to Neural Network Verification”. In: *Found. Trends Program. Lang.* 7.1-2 (2021), pp. 1–157 (cit. on p. 2).
- [And+21] Roman Andriushchenko, Milan Ceska, Sebastian Junges, Joost-Pieter Katoen, and Simon Stupinský. “PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs”. In: *CAV (1)*. Vol. 12759. Lecture Notes in Computer Science. Springer, 2021, pp. 856–869 (cit. on p. 48).
- [And+22] Roman Andriushchenko, Milan Ceska, Sebastian Junges, and Joost-Pieter Katoen. “Inductive synthesis of finite-state controllers for POMDPs”. In: *UAI*. Vol. 180. Proceedings of Machine Learning Research. PMLR, 2022, pp. 85–95 (cit. on pp. 3, 48, 52, 101).
- [And+23] Roman Andriushchenko, Alexander Bork, Milan Ceska, Sebastian Junges, Joost-Pieter Katoen, and Filip Macák. “Search and Explore: Symbiotic Policy Synthesis in POMDPs”. In: *CAV (3)*. Vol. 13966. Lecture Notes in Computer Science. Springer, 2023, pp. 113–135 (cit. on pp. 3, 48, 53).
- [Ang87] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Inf. Comput.* 75.2 (1987), pp. 87–106 (cit. on p. 49).
- [Ans+20] Sten Anslan, Mina Azizi Rad, Johannes Buckel, Paula Echeverria Galindo, Jinlei Kai, Wengang Kang, Laura Keys, Philipp Maurischat, Felix Nieberding, Eike Reinosch, Handuo Tang, Tuong Vi Tran, Yuyang Wang, and Antje Schwalb. “Reviews and syntheses: How do abiotic and biotic processes respond to climatic variations in the Nam Co catchment (Tibetan Plateau)?” In: *Biogeosciences* 17 (2020), pp. 1261–1279 (cit. on p. 39).

- [Anw+18] Syed Muhammad Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi R. Alnowami, and Muhammad Khurram Khan. “Medical Image Analysis using Convolutional Neural Networks: A Review”. In: *J. Medical Syst.* 42.11 (2018), p. 226 (cit. on pp. 1, 2).
- [Ark23] Jeremy Arkes. *Regression Analysis : A Practical Introduction*. Taylor & Francis Group, 2023 (cit. on p. 41).
- [Ash+20] Pranav Ashok, Vahid Hashemi, Jan Kretínský, and Stefanie Mohr. “Deep-Abstract: Neural Network Abstraction for Accelerating Verification”. In: *ATVA*. Vol. 12302. Lecture Notes in Computer Science. Springer, 2020, pp. 92–107 (cit. on pp. 11, 16, 17, 22, 23).
- [Aut24] The LCZero Authors. *LeelaChessZero*. Version 0.30.0. May 24, 2024. URL: <https://lczero.org/> (cit. on p. 1).
- [Bay] Bayerisches Landesamt für Umwelt, [www.lfu.bayern.de](http://www.lfu.bayern.de). *GKD Bayern*. URL: <https://www.gkd.bayern.de/de/> (visited on 06/22/2021) (cit. on p. 41).
- [BB24] Christopher M. Bishop and Hugh Bishop. *Deep Learning - Foundations and Concepts*. Springer, 2024 (cit. on p. 11).
- [BCN06] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2006, pp. 535–541 (cit. on p. 16).
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008 (cit. on p. 12).
- [BKQ22] Alexander Bork, Joost-Pieter Katoen, and Tim Quatmann. “Under-Approximating Expected Total Rewards in POMDPs”. In: *TACAS (2)*. Vol. 13244. Lecture Notes in Computer Science. Springer, 2022, pp. 22–40 (cit. on pp. 3, 48).
- [Bon02] Blai Bonet. “An epsilon-Optimal Grid-Based Algorithm for Partially Observable Markov Decision Processes”. In: *ICML*. Morgan Kaufmann, 2002, pp. 51–58 (cit. on p. 4).
- [Bor+20] Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. “Verification of Indefinite-Horizon POMDPs”. In: *ATVA*. Vol. 12302. Lecture Notes in Computer Science. Springer, 2020, pp. 288–304 (cit. on pp. 3, 48, 52).
- [Bou+23a] Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, and Mohamed Ghazel. “A review of abstraction methods towards verifying neural networks”. In: *ACM Trans. Embed. Comput. Syst.* (2023) (cit. on p. 23).
- [Bou+23b] Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, and Mohamed Ghazel. “INNAbstract: An INN-Based Abstraction Method for Large-Scale Neural Network Verification”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2023) (cit. on p. 23).

- [Bri+23] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. “First three years of the international verification of neural networks competition (VNN-COMP)”. In: *Int. J. Softw. Tools Technol. Transf.* 25.3 (2023), pp. 329–339 (cit. on p. 2).
- [Bue+19] Antoine Buetti-Dinh, Vanni Galli, Sören Bellenberg, Olga Ilie, Malte Herold, Stephan Christel, Mariia Boretska, Igor V. Pivkin, Paul Wilmes, Wolfgang Sand, Mario Vera, and Mark Dopson. “Deep neural networks outperform human expert’s capacity in characterizing bioleaching bacterial biofilm composition”. In: *Biotechnology Reports* 22 (2019) (cit. on p. 2).
- [CA21] SueYeon Chung and L. F. Abbott. “Neural population geometry: An approach for understanding biological and artificial neural networks”. In: *CoRR* abs/2104.07059 (2021) (cit. on p. 39).
- [Cai+21] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. “Physics-Informed Neural Networks for Heat Transfer Problems”. In: *Journal of Heat Transfer* 143.6 (2021), p. 060801 (cit. on p. 39).
- [Cai06] Daniel Caissie. “The thermal regime of rivers: a review”. In: *Freshwater biology* 51.8 (2006), pp. 1389–1406 (cit. on p. 39).
- [Car+19] Steven Carr, Nils Jansen, Ralf Wimmer, Alexandru Constantin Serban, Bernd Becker, and Ufuk Topcu. “Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks”. In: *IJCAI*. ijcai.org, 2019, pp. 5532–5539 (cit. on p. 48).
- [Car+23] Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. “Safe Reinforcement Learning via Shielding under Partial Observability”. In: *AAAI*. AAAI Press, 2023, pp. 14748–14756 (cit. on p. 48).
- [CCT16] Krishnendu Chatterjee, Martin Chmelik, and Mathieu Tracol. “What is decidable about partially observable Markov decision processes with  $\omega$ -regular objectives”. In: *J. Comput. Syst. Sci.* 82.5 (2016), pp. 878–911 (cit. on p. 3).
- [CES98] Daniel Caissie, Nassir El-Jabi, and André St-Hilaire. “Stochastic modelling of water temperatures in a small stream using air to water relations”. In: *Canadian Journal of Civil Engineering* 25.2 (1998), pp. 250–260 (cit. on p. 39).
- [CGL92] Edmund M. Clarke, Orna Grumberg, and David E. Long. “Model Checking and Abstraction”. In: *POPL*. ACM Press, 1992, pp. 342–354 (cit. on p. 15).
- [Cha+22] Arnav Chakravarthy, Nina Narodytska, Asmitha Rathis, Marius Vilcu, Mahmood Sharif, and G Singh. “Property-driven evaluation of rl-controllers in self-driving datacenters”. In: *Workshop on Challenges in Deploying and Monitoring Machine Learning Systems (DMML)*. Available at <https://plus-tau.github.io/files/dmml22-rl-controllers.pdf>. 2022 (cit. on p. 46).

- [Che+17] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. “Multi-view 3D Object Detection Network for Autonomous Driving”. In: *CVPR*. IEEE Computer Society, 2017, pp. 6526–6534 (cit. on pp. 1, 2).
- [CJT21] Steven Carr, Nils Jansen, and Ufuk Topcu. “Task-Aware Verifiable RNN-Based Policies for Partially Observable Markov Decision Processes”. In: *J. Artif. Intell. Res.* 72 (2021), pp. 819–847 (cit. on pp. 3, 48).
- [Cla+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. “Counterexample-Guided Abstraction Refinement”. In: *CAV*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 154–169 (cit. on p. 15).
- [CNR17] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. “Maximum Resilience of Artificial Neural Networks”. In: *ATVA*. Vol. 10482. Lecture Notes in Computer Science. Springer, 2017, pp. 251–268 (cit. on p. 2).
- [CNY19] Chih-Hong Cheng, Georg Nührenberg, and Hirotoshi Yasuoka. “Runtime Monitoring Neuron Activation Patterns”. In: *DATE*. IEEE, 2019, pp. 300–303 (cit. on pp. 3, 11, 28).
- [Con+21] Marcello Congro, Vitor Moreira de Alencar Monteiro, Amanda L.T. Brandão, Brunno F. dos Santos, Deane Roehl, and Flávio de Andrade Silva. “Prediction of the residual flexural strength of fiber reinforced concrete using artificial neural networks”. In: *Construction and Building Materials* 303 (2021), p. 124502 (cit. on p. 39).
- [Cub+21] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Ahmadreza Marandi, Marnix Suilen, and Ufuk Topcu. “Robust Finite-State Controllers for Uncertain POMDPs”. In: *AAAI*. AAAI Press, 2021, pp. 11792–11800 (cit. on p. 48).
- [Deu] Deutscher Wetterdienst. *DWD Climate Data Center (CDC): Historical daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for Germany, version v21.3, 2021*. URL: [https://www.dwd.de/DE/klimaumwelt/cdc/cdc%5C\\_node.html](https://www.dwd.de/DE/klimaumwelt/cdc/cdc%5C_node.html) (visited on 06/21/2021) (cit. on p. 41).
- [Dju+23] Andrija Djuricic, Nebojsa Bozanic, Arjun Ashok, and Rosanne Liu. “Extremely Simple Activation Shaping for Out-of-Distribution Detection”. In: *ICLR*. OpenReview.net, 2023 (cit. on p. 27).
- [DKT08] Le Tien Dung, Takashi Komeda, and Motoki Takagi. “Reinforcement Learning for POMDP Using State Classification”. In: *Appl. Artif. Intell.* 22.7&8 (2008), pp. 761–779 (cit. on pp. 3, 48).
- [EGK20] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. “An Abstraction-Based Framework for Neural Network Verification”. In: *CAV (1)*. Vol. 12224. Lecture Notes in Computer Science. Springer, 2020, pp. 43–65 (cit. on pp. 16, 23).



- [Ehl17] Rüdiger Ehlers. “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks”. In: *ATVA*. Vol. 10482. Lecture Notes in Computer Science. Springer, 2017, pp. 269–286 (cit. on p. 2).
- [Geh+18] Timon Gehr, Matthew Mirman, Dana Drachler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. “AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 3–18 (cit. on pp. 16, 40).
- [GKS03] Sally A. Goldman, Stephen Kwek, and Stephen D. Scott. “Learning from examples with unspecified attribute values”. In: *Inf. Comput.* 180.2 (2003), pp. 82–100 (cit. on p. 53).
- [Gra95] My Grandma. *Chocolate Cake*. **Ingredients:** 250g butter, 300g sugar, 4 eggs, 375g flour, 200g sour cream, 50g unsweet cocoa, 15g baking powder (or use self-rising flour). **Instructions:** Mix all ingredients together and pour on a baking tray. Bake at 165°C 25-35min. Add chocolate glaze, if you want. 1995 (cit. on p. 21).
- [GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *ICLR (Poster)*. 2015 (cit. on p. 2).
- [GSY99] Hoshin Vijai Gupta, Soroosh Sorooshian, and Patrice Ogou Yapo. “Status of Automatic Calibration for Hydrologic Models: Comparison with Multi-level Expert Calibration”. In: *Journal of Hydrologic Engineering* 4.2 (1999), pp. 135–143 (cit. on p. 44).
- [Gu+20] Weibin Gu, Kimon P. Valavanis, Matthew J. Rutherford, and Alessandro Rizzo. “UAV Model-based Flight Control with Artificial Neural Networks: A Survey”. In: *J. Intell. Robotic Syst.* 100.3 (2020), pp. 1469–1491 (cit. on p. 1).
- [Guo+17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. “On Calibration of Modern Neural Networks”. In: *ICML*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1321–1330 (cit. on p. 27).
- [Han+15] Song Han, Jeff Pool, John Tran, and William J. Dally. “Learning both Weights and Connections for Efficient Neural Network”. In: *NIPS*. 2015, pp. 1135–1143 (cit. on p. 16).
- [Han98] Eric A. Hansen. “Solving POMDPs by Searching in Policy Space”. In: *UAI*. Morgan Kaufmann, 1998, pp. 211–219 (cit. on p. 48).
- [Has+23] Vahid Hashemi, Jan Kretínský, Sabine Rieder, and Jessica Schmidt. “Runtime Monitoring for Out-of-Distribution Detection in Object Detection Neural Networks”. In: *FM*. Vol. 14000. Lecture Notes in Computer Science. Springer, 2023, pp. 622–634 (cit. on pp. 3, 38).

- [Hau00] Milos Hauskrecht. “Value-Function Approximations for Partially Observable Markov Decision Processes”. In: *J. Artif. Intell. Res.* 13 (2000), pp. 33–94 (cit. on pp. 3, 48).
- [HBH02] Lonnie Hamm, B. Wade Brorsen, and Martin T. Hagan. “Global optimization of neural network weights”. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No.02CH37290)*. Vol. 2. 2002, pp. 1228–1233 (cit. on p. 43).
- [Hec+22] Linus Heck, Jip Spel, Sebastian Junges, Joshua Moerman, and Joost-Pieter Katoen. “Gradient-Descent for Randomized Controllers Under Partial Observability”. In: *VMCAI*. Vol. 13182. Lecture Notes in Computer Science. Springer, 2022, pp. 127–150 (cit. on pp. 3, 48).
- [Hen+22a] Dan Hendrycks, Steven Basart, Mantas Mazeika, Andy Zou, Joseph Kwon, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. “Scaling Out-of-Distribution Detection for Real-World Settings”. In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 8759–8773 (cit. on p. 27).
- [Hen+22b] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. “The probabilistic model checker Storm”. In: *Int. J. Softw. Tools Technol. Transf.* 24.4 (2022), pp. 589–610 (cit. on pp. 48, 53).
- [Hen24] Jon Henley. “Tesla Autopilot feature was involved in 13 fatal crashes, US regulator says”. In: *The Guardian* (Apr. 2024). URL: <https://www.theguardian.com/technology/2024/apr/26/tesla-autopilot-fatal-crash> (visited on 06/04/2024) (cit. on p. 1).
- [HG17] Dan Hendrycks and Kevin Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In: *ICLR (Poster)*. OpenReview.net, 2017 (cit. on pp. 2, 27).
- [HGL21] Rui Huang, Andrew Geng, and Yixuan Li. “On the Importance of Gradients for Detecting Distributional Shifts in the Wild”. In: *NeurIPS*. 2021, pp. 677–689 (cit. on p. 27).
- [HKW11] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. “Transforming Auto-Encoders”. In: *ICANN (1)*. Vol. 6791. Lecture Notes in Computer Science. Springer, 2011, pp. 44–51 (cit. on p. 2).
- [HLS20] Thomas A. Henzinger, Anna Lukina, and Christian Schilling. “Outside the Box: Abstraction-Based Monitoring of Neural Networks”. In: *ECAI*. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 2433–2440 (cit. on pp. 27, 28, 31, 32, 36, 37, 139).
- [HS15] Matthew J. Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *AAAI Fall Symposia*. AAAI Press, 2015, pp. 29–37 (cit. on p. 48).

- [HS22] Yuning He and Johann Schumann. *Runtime Monitoring for Unmanned Aerospace Systems with Neural Network Components*. Presentation at the Meeting of Engineering Computational Technology 2022, <https://ntrs.nasa.gov/citations/20220012988>. 2022 (cit. on p. 3).
- [Hua+19] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. “ReachNN: Reachability Analysis of Neural-Network Controlled Systems”. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019), 106:1–106:22 (cit. on p. 2).
- [HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In: *CoRR* abs/1503.02531 (2015) (cit. on p. 16).
- [JL24] Taylor T. Johnson and Changliu Liu. *VNN-cOMP2020 Report*. May 24, 2024. URL: <https://www.overleaf.com/project/5f0c85e8d15dc10001749fa9> (cit. on p. 2).
- [Jun+18] Sebastian Junges, Nils Jansen, Ralf Wimmer, Tim Quatmann, Leonore Winterer, Joost-Pieter Katoen, and Bernd Becker. “Finite-State Controllers of POMDPs using Parameter Synthesis”. In: *UAI*. AUAI Press, 2018, pp. 519–529 (cit. on p. 48).
- [Kat+17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In: *CAV (1)*. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 97–117 (cit. on pp. 2, 40, 42).
- [Kat+19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. “The Marabou Framework for Verification and Analysis of Deep Neural Networks”. In: *CAV (1)*. Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 443–452 (cit. on pp. 2, 40).
- [Kat+22] Julian Katz-Samuels, Julia B. Nakhleh, Robert D. Nowak, and Yixuan Li. “Training OOD Detectors in their Natural Habitats”. In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 10848–10865 (cit. on p. 28).
- [KFO22] Konstantin Kirchheim, Marco Filax, and Frank Ortmeier. “PyTorch-OOD: A Library for Out-of-Distribution Detection based on PyTorch”. In: *CVPR Workshops*. IEEE, 2022, pp. 4350–4359 (cit. on p. 28).
- [KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>. Accessed: 28.05.24. 2009 (cit. on pp. 31, 37).
- [KHL08] Hanna Kurniawati, David Hsu, and Wee Sun Lee. “SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces”. In: *Robotics: Science and Systems*. The MIT Press, 2008 (cit. on p. 48).

- [KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. In: *Artif. Intell.* 101.1-2 (1998), pp. 99–134 (cit. on pp. 3, 4, 48).
- [Kri+13] Lori A Krider, Joseph A Magner, Jim Perry, Bruce Vondracek, and Leonard C Ferrington Jr. “Air-water temperature relationships in the trout streams of southeastern Minnesota’s carbonate-sandstone landscape”. In: *JAWRA Journal of the American Water Resources Association* 49.4 (2013), pp. 896–907 (cit. on p. 39).
- [LA23] Tobias Ladner and Matthias Althoff. “Automatic Abstraction Refinement in Neural Network Verification using Sensitivity Analysis”. In: *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*. Association for Computing Machinery, 2023 (cit. on p. 23).
- [LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. *MNIST handwritten digit database*. 2010. URL: <http://yann.lecun.com/exdb/mnist> (visited on 05/22/2024) (cit. on pp. 31, 32, 37).
- [LDS89] Yann LeCun, John S. Denker, and Sara A. Solla. “Optimal Brain Damage”. In: *NIPS*. Morgan Kaufmann, 1989, pp. 598–605 (cit. on p. 16).
- [Lee+18] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. “A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks”. In: *NeurIPS*. 2018, pp. 7167–7177 (cit. on p. 27).
- [Liu+20a] Weitang Liu, Xiaoyun Wang, John D. Owens, and Yixuan Li. “Energy-based Out-of-distribution Detection”. In: (2020) (cit. on pp. 3, 27, 36, 37).
- [Liu+20b] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. “Certified Monotonic Neural Networks”. In: *NeurIPS*. 2020 (cit. on p. 2).
- [LLS18] Shiyu Liang, Yixuan Li, and R. Srikant. “Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks”. In: *ICLR (Poster)*. OpenReview.net, 2018 (cit. on pp. 27, 36, 37).
- [Lov91] William S. Lovejoy. “Computationally Feasible Bounds for Partially Observed Markov Decision Processes”. In: *Oper. Res.* 39.1 (1991), pp. 162–175 (cit. on p. 3).
- [Mac+21] David Macêdo, Tsang Ing Ren, Cleber Zanchettin, Adriano L. I. Oliveira, and Teresa Bernarda Ludermir. “Entropic Out-of-Distribution Detection”. In: *IJCNN*. IEEE, 2021, pp. 1–8 (cit. on p. 27).
- [MC21] Sergei Manzhos and Tucker Jr. Carrington. “Neural Network Potential Energy Surfaces for Small Molecules and Reactions”. In: *Chemical Reviews* 121.16 (2021), pp. 10187–10217 (cit. on p. 39).
- [Meu+99a] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. “Solving POMDPs by Searching the Space of Finite Policies”. In: *UAI*. Morgan Kaufmann, 1999, pp. 417–426 (cit. on pp. 4, 48).

- [Meu+99b] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. “Learning Finite-State Controllers for Partially Observable Environments”. In: *UAI*. Morgan Kaufmann, 1999, pp. 427–436 (cit. on pp. 4, 48).
- [MHC03a] Omid Madani, Steve Hanks, and Anne Condon. “On the undecidability of probabilistic planning and related stochastic optimization problems”. In: *Artif. Intell.* 147.1-2 (2003), pp. 5–34 (cit. on p. 3).
- [MHC03b] Omid Madani, Steve Hanks, and Anne Condon. “On the undecidability of probabilistic planning and related stochastic optimization problems”. In: *Artif. Intell.* 147.1-2 (2003), pp. 5–34 (cit. on p. 3).
- [MMR55] J McCarthy, ML Minsky, and N Rochester. *A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE*. 1955. URL: <http://www.cs.toronto.edu/~bor/196f21/docs/dartmouth-1955-AI-summer-project.pdf> (visited on 06/24/2024) (cit. on p. 8).
- [MP43] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133 (cit. on p. 8).
- [MS99] Omid Mohseni and Heinz G. Stefan. “Stream temperature/air temperature relationship: a physical interpretation”. In: *Journal of Hydrology* 218 (1999), pp. 128–141 (cit. on p. 39).
- [NPZ17] Gethin Norman, David Parker, and Xueyi Zou. “Verification and control of partially observable probabilistic systems”. In: *Real Time Syst.* 53.3 (2017), pp. 354–402 (cit. on p. 48).
- [Ope24] OpenAI. *ChatGPT*. Version GPT-4. May 24, 2024. URL: <https://chat.openai.com> (cit. on pp. 1, 2).
- [PA19] Pavithra Prabhakar and Zahra Rahimi Afzal. “Abstraction based Output Range Analysis for Neural Networks”. In: *NeurIPS*. 2019, pp. 15762–15772 (cit. on pp. 16, 23).
- [Pan+24] Joachim Pander, Johannes Kuhn, Roser Casas-Mulet, Luis Habersetzer, and Juergen Geist. “Diurnal patterns of spatial stream temperature variations reveal the need for integrating thermal heterogeneity in riverscape habitat restoration”. In: *Science of The Total Environment* 918 (2024), p. 170786 (cit. on p. 46).
- [PFS98] John M Pilgrim, Xing Fang, and Heinz G Stefan. “Stream temperature correlations with air temperatures in Minnesota: Implications for climate warming 1”. In: *JAWRA Journal of the American Water Resources Association* 34.5 (1998), pp. 1109–1121 (cit. on p. 39).
- [PGT03] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. “Point-based value iteration: An anytime algorithm for POMDPs”. In: *IJCAI*. Morgan Kaufmann, 2003, pp. 1025–1032 (cit. on p. 48).

- [PM21] Pablo N. Pizarro and Leonardo M. Massone. “Structural design of reinforced concrete buildings based on deep neural networks”. In: *Engineering Structures* 241 (2021), p. 112377 (cit. on p. 39).
- [Pör+22] Hans-Otto Pörtner, Debra C. Roberst, Melinda M. B. Tignor, Elvira Poloczanska, Katja Mintenbeck, Andrés Alegría, Marlies Craig, Stefanie Langsdorf, Sina Löschke, Vincent Möller, Andrew Okem, and Bardhyl Rama. “Climate change 2022: Impacts, adaptation and vulnerability. Contribution of working group II to the sixth assessment report of the intergovernmental panel on Climate change”. In: *Cambridge University Press* (2022) (cit. on p. 39).
- [Pra22] Pavithra Prabhakar. “Bisimulations for Neural Network Reduction”. In: *VMCAI*. Vol. 13182. Lecture Notes in Computer Science. Springer, 2022, pp. 285–300 (cit. on pp. 16, 17, 22, 23).
- [PT10] Luca Pulina and Armando Tacchella. “An Abstraction-Refinement Approach to Verification of Artificial Neural Networks”. In: *CAV*. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 243–257 (cit. on p. 16).
- [Ren+20] Jiahao Ren, Xirong Lin, Jinyun Liu, Tianli Han, Zhilong Wang, Haikuo Zhang, and Jinjin Li. “Engineering early prediction of supercapacitors’ cycle life using neural networks”. In: *Materials Today Energy* 18 (2020), p. 100537 (cit. on p. 39).
- [Ren+21] Jie Ren, Stanislav Fort, Jeremiah Z. Liu, Abhijit Guha Roy, Shreyas Padhy, and Balaji Lakshminarayanan. “A Simple Fix to Mahalanobis Distance for Improving Near-OOD Detection”. In: *CoRR* abs/2106.09022 (2021) (cit. on p. 27).
- [RHŠ15] Anamarija Rabi, Marijana Hadzima-Nyarko, and Marija Šperac. “Modelling river temperature from air temperature: case of the River Drava (Croatia)”. In: *Hydrological sciences journal* 60.9 (2015), pp. 1490–1507 (cit. on pp. 39, 40).
- [RN20] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020 (cit. on pp. 8, 43).
- [Rom+24] Tord Romstad, Marco Costalba, Joona Kiiski, and Gary Linscott. *Stockfish*. Version 16.1. May 24, 2024. URL: <https://stockfishchess.org/> (cit. on p. 1).
- [Rou+23] Naghmeh Shafiee Roudbari, Charalambos Poullis, Zachary Patterson, and Ursula Eicker. “TransGlow: Attention-augmented Transduction model based on Graph Neural Networks for Water Flow Forecasting”. In: *ICMLA*. IEEE, 2023, pp. 626–632 (cit. on p. 46).

- [Sal+22] Mohammadreza Salehi, Hossein Mirzaei, Dan Hendrycks, Yixuan Li, Mohammad Hossein Rohban, and Mohammad Sabokrou. “A Unified Survey on Anomaly, Novelty, Open-Set, and Out of-Distribution Detection: Solutions and Future Challenges”. In: *Trans. Mach. Learn. Res.* 2022 (2022) (cit. on p. 27).
- [Sam+21] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, and Klaus-Robert Müller. “Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications”. In: *Proc. IEEE* 109.3 (2021), pp. 247–278 (cit. on p. 41).
- [SB15] Suraj Srinivas and R. Venkatesh Babu. “Data-free Parameter Pruning for Deep Neural Networks”. In: *BMVC*. BMVA Press, 2015, pp. 31.1–31.12 (cit. on p. 16).
- [SBV21] Jonathan Shlomi, Peter W. Battaglia, and Jean-Roch Vlimant. “Graph neural networks in particle physics”. In: *Mach. Learn. Sci. Technol.* 2.2 (2021), p. 21001 (cit. on p. 39).
- [Sch+21] Philippe Schwaller, Daniel Probst, Alain C. Vaucher, Vishnu H. Nair, David Kreutter, Teodoro Laino, and Jean-Louis Reymond. “Mapping the space of chemical reactions using attention-based neural networks”. In: *Nat. Mach. Intell.* 3.2 (2021), pp. 144–152 (cit. on p. 39).
- [SG09] Muzammil Shahbaz and Roland Groz. “Inferring Mealy Machines”. In: *FM*. Vol. 5850. Lecture Notes in Computer Science. Springer, 2009, pp. 207–222 (cit. on p. 49).
- [SGL21] Yiyu Sun, Chuan Guo, and Yixuan Li. “ReAct: Out-of-distribution Detection With Rectified Activations”. In: *NeurIPS*. 2021, pp. 144–157 (cit. on p. 27).
- [Sin+19a] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. “An abstract domain for certifying neural networks”. In: *Proc. ACM Program. Lang.* 3.POPL (2019), 41:1–41:30 (cit. on pp. 16, 40, 42).
- [Sin+19b] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. “Boosting Robustness Certification of Neural Networks”. In: *ICLR (Poster)*. OpenReview.net, 2019 (cit. on pp. 16, 40).
- [SL22] Yiyu Sun and Yixuan Li. “DICE: Leveraging Sparsification for Out-of-Distribution Detection”. In: *ECCV (24)*. Vol. 13684. Lecture Notes in Computer Science. Springer, 2022, pp. 691–708 (cit. on pp. 3, 27).
- [SMI81] K. SMITH. “The prediction of river water temperatures / Prédiction des températures des eaux de rivière”. In: *Hydrological Sciences Bulletin* 26.1 (1981), pp. 19–32 (cit. on p. 39).
- [SMR22] Kady Sako, Berthine Nyunga Mpinda, and Paulo Canas Rodrigues. “Neural Networks for Financial Time Series Forecasting”. In: *Entropy* 24.5 (2022), p. 657 (cit. on p. 1).

- [SPK13] Guy Shani, Joelle Pineau, and Robert Kaplow. “A survey of point-based POMDP solvers”. In: *Auton. Agents Multi Agent Syst.* 27.1 (2013), pp. 1–51 (cit. on p. 48).
- [SS73] Richard D. Smallwood and Edward J. Sondik. “The Optimal Control of Partially Observable Markov Processes over a Finite Horizon”. In: *Oper. Res.* 21.5 (1973), pp. 1071–1088 (cit. on p. 48).
- [SSJ23] Thiago D. Simão, Marnix Suilen, and Nils Jansen. “Safe Policy Improvement for POMDPs via Finite-State Controllers”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.12 (2023), pp. 15109–15117 (cit. on p. 4).
- [ST20] Matthew Sotoudeh and Aditya V. Thakur. “Abstract Neural Networks”. In: *SAS*. Vol. 12389. Lecture Notes in Computer Science. Springer, 2020, pp. 65–88 (cit. on p. 16).
- [ST94] Robert H. Sloan and György Turán. “Learning with Queries but Incomplete Information (Extended Abstract)”. In: *COLT*. ACM, 1994, pp. 237–245 (cit. on p. 53).
- [Sta+11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. “The German Traffic Sign Recognition Benchmark: A multi-class classification competition”. In: *IJCNN*. IEEE, 2011, pp. 1453–1460 (cit. on p. 31).
- [Sun+22] Yiyu Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. “Out-of-Distribution Detection with Deep Nearest Neighbors”. In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 20827–20840 (cit. on p. 27).
- [SV05] Matthijs T. J. Spaan and Nikos Vlassis. “Perseus: Randomized Point-based Value Iteration for POMDPs”. In: *J. Artif. Intell. Res.* 24 (2005), pp. 195–220 (cit. on p. 48).
- [Sze+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. “Intriguing properties of neural networks”. In: *ICLR (Poster)*. 2014 (cit. on p. 2).
- [TES24] TEST. “Three Decades of Activations: A Comprehensive Survey of 400 Activation Functions for Neural Networks”. In: *CoRR* abs/2402.09092 (2024) (cit. on p. 9).
- [Wan+22] Haoqi Wang, Zhizhong Li, Litong Feng, and Wayne Zhang. “ViM: Out-Of-Distribution with Virtual-logit Matching”. In: *CVPR*. IEEE, 2022, pp. 4911–4920 (cit. on p. 27).
- [Wei+23] Dorina Weichert, Alexander Kister, Peter Volbach, Sebastian Houben, Marcus Trost, and Stefan Wrobel. “Explainable production planning under partial observability in high-precision manufacturing”. In: *Journal of Manufacturing Systems* 70 (2023), pp. 514–524 (cit. on p. 4).



- [Win+21] Leonore Winterer, Sebastian Junges, Ralf Wimmer, Nils Jansen, Ufuk Topcu, Joost-Pieter Katoen, and Bernd Becker. “Strategy Synthesis for POMDPs in Robot Planning via Game-Based Abstractions”. In: *IEEE Trans. Autom. Control*. 66.3 (2021), pp. 1040–1054 (cit. on p. 3).
- [WJ21] Guohua Wei and Yi Jin. “Human resource management model based on three-layer BP neural network and machine learning”. In: *J. Intell. Fuzzy Syst.* 40.2 (2021), pp. 2289–2300 (cit. on p. 39).
- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *CoRR* abs/1708.07747 (2017) (cit. on p. 31).
- [Yan+21] Jingkan Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. “Generalized Out-of-Distribution Detection: A Survey”. In: *CoRR* abs/2110.11334 (2021) (cit. on p. 27).
- [Zha+18] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. “Efficient Neural Network Robustness Certification with General Activation Functions”. In: *NeurIPS*. 2018, pp. 4944–4953 (cit. on pp. 2, 40, 41).
- [Zha+23a] Jingyang Zhang, Jingkan Yang, Pengyun Wang, Haoqi Wang, Yueqian Lin, Haoran Zhang, Yiyu Sun, Xuefeng Du, Kaiyang Zhou, Wayne Zhang, Yixuan Li, Ziwei Liu, Yiran Chen, and Hai Li. “OpenOOD v1.5: Enhanced Benchmark for Out-of-Distribution Detection”. In: *CoRR* abs/2306.09301 (2023) (cit. on p. 27).
- [Zha+23b] Jinsong Zhang, Qiang Fu, Xu Chen, Lun Du, Zelin Li, Gang Wang, Xiaoguang Liu, Shi Han, and Dongmei Zhang. “Out-of-Distribution Detection based on In-Distribution Data Patterns Memorization with Modern Hopfield Energy”. In: *ICLR*. OpenReview.net, 2023 (cit. on p. 27).
- [Zhu+19] Senlin Zhu, Emmanuel Karlo Nyarko, Marijana Hadzima-Nyarko, Salim Heddami, and Shiqiang Wu. “Assessing the performance of a suite of machine learning models for daily river water temperature prediction”. In: *PeerJ* 7 (2019), e7065 (cit. on p. 40).
- [Zhu+24] Senlin Zhu, Fabio Di Nunno, Jiang Sun, Mariusz Sojka, Mariusz Ptak, and Francesco Granata. “An optimized NARX-based model for predicting thermal dynamics and heatwaves in rivers”. In: *Science of The Total Environment* 926 (2024), p. 171954 (cit. on p. 46).
- [ZMC94] Jacek M. Zurada, Aleksander Malinowski, and Ian Cloete. “Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network”. In: *ISCAS*. IEEE, 1994, pp. 447–450 (cit. on p. 43).
- [ZX23] Zihan Zhang and Xiang Xiang. “Decoupling MaxLogit for Out-of-Distribution Detection”. In: *CVPR*. IEEE, 2023, pp. 3388–3397 (cit. on p. 27).

## *Bibliography*

- [YZZ18] Guoqiang Zhong, Hui Yao, and Huiyu Zhou. “Merging Neurons for Structure Compression of Deep Networks”. In: *ICPR*. IEEE Computer Society, 2018, pp. 1462–1467 (cit. on p. 16).

# **Publications**

## **A Syntactic vs Semantic Linear Abstraction and Refinement of Neural Networks**

*Reprinted by permission from Springer Nature (License Number 5663520480625): Lecture Notes in Computer Science book series (LNCS, volume 14215) Syntactic vs Semantic Linear Abstraction and Refinement of Neural Networks, Calvin Chau, Jan Křetínský, Stefanie Mohr © 2023 Springer Nature Switzerland AG (2023)*

This paper has been published as a **peer-reviewed conference paper**.

Calvin Chau, Jan Křetínský, Stefanie Mohr (2023). Syntactic vs Semantic Linear Abstraction and Refinement of Neural Networks. In: André, É., Sun, J. (eds) Automated Technology for Verification and Analysis. ATVA 2023. Lecture Notes in Computer Science, vol 14215, pp. 401-421. Springer, Cham.

DOI: [https://doi.org/10.1007/978-3-031-45329-8\\_19](https://doi.org/10.1007/978-3-031-45329-8_19)

### **Summary**

In this work, we introduce a novel abstraction technique for Neural Networks (NNs) using linear combinations of neurons as representatives. Previous approaches for abstraction replace a group of neurons with a single representative that is defined as similar. Instead, we can provide a more flexible and powerful abstraction framework by introducing linear combinations. Furthermore, we provide a formal error estimate to guarantee the difference between the original and the abstraction. We also provide refinement methods for our abstraction to balance accuracy and reduction. Additionally, we empirically compare syntactic and semantic abstraction, where the first only uses of the weights of the NN, and the latter additionally uses sample inputs. Our approach uses semantic information of the NN, and we compare it to  $\delta$ -bisimulations of NNs, which uses the syntax. We find that semantic abstractions allow for a much more significant reduction than syntactic abstractions.

### **Contributions of the author**

Composition, discussion and revision of the entire manuscript. Sole contribution of all proofs and most of the results presented paper. Discussion and development of the ideas, experimentation and evaluation. Co-lead role in the design and implementation of the presented tool.



# Syntactic vs Semantic Linear Abstraction and Refinement of Neural Networks

Calvin Chau<sup>1</sup> , Jan Křetínský<sup>2,3</sup> , and Stefanie Mohr<sup>2</sup>

<sup>1</sup> Technische Universität Dresden, Dresden, Germany  
`calvin.chau@tu-dresden.de`

<sup>2</sup> Technical University of Munich, Munich, Germany  
`{kretinsky,mohr}@in.tum.de`

<sup>3</sup> Masaryk University, Brno, Czech Republic

**Abstract.** Abstraction is a key verification technique to improve scalability. However, its use for neural networks is so far extremely limited. Previous approaches for abstracting classification networks replace several neurons with one of them that is similar enough. We can classify the similarity as defined either syntactically (using quantities on the connections between neurons) or semantically (on the activation values of neurons for various inputs). Unfortunately, the previous approaches only achieve moderate reductions, when implemented at all. In this work, we provide a more flexible framework, where a neuron can be replaced with a *linear combination* of other neurons, improving the reduction. We apply this approach both on syntactic and semantic abstractions, and implement and evaluate them experimentally. Further, we introduce a refinement method for our abstractions, allowing for finding a better balance between reduction and precision.

**Keywords:** Neural network · Abstraction · Machine learning

## 1 Introduction

**Neural Network Abstractions.** Abstraction is a key instrument for understanding complex systems and analyzing complex problems across all disciplines, including computer science. Abstraction of complex systems, such as neural networks (NN), results in smaller systems, which are not only producing equivalent outputs (such as in distillation [13]), but additionally can be mapped to the original system, providing a strong link between the individual parts of the two systems. Consequently, abstraction find various applications. For instance, the smaller (abstract) networks are more understandable and the strong link between the behaviours of the abstract and the original network allows for better explainability of the original behaviour, too; smaller networks are more efficient

---

This research was funded in part by the German Research Foundation (DFG) project 427755713 *GoPro*, the German Federal Ministry of Education and Research (BMBF) within the project *SEMECO Q1* (03ZU1210AG), and the DFG research training group *ConVeY* (GRK 2428).

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023  
É. André and J. Sun (Eds.): ATVA 2023, LNCS 14215, pp. 401–421, 2023.  
[https://doi.org/10.1007/978-3-031-45329-8\\_19](https://doi.org/10.1007/978-3-031-45329-8_19)

in resource usage during runtime; smaller networks are easier to verify. Again, with no formal link between the original network and, say, a distilled or pruned one, verifying the smaller one is of no use to verifying the original one. In contrast, for abstractions, the verification guarantee can be in principle transferred to the original network, be it via lifting a counterexample or a proof of correctness.

Altogether, abstractions of neural networks are a key concept worth investigating *eo ipso*, subsequently offering various applications. However, currently it is still very under-developed. For defining an abstraction, we need a transformation linking the original neurons to those in the abstraction. Equivalently, we need a notion of the *similarity of neurons*, to identify a good representative of a group of neurons. The difficulty in contrast to, e.g., predicate abstraction of programs is that neurons have no inner structure such as values of variables stored in a state. On the one hand, approaches based on bisimilarity [22] offer a solution focusing on the “*syntax*” of neurons: the weights of the incoming connections. The quantities give rise to an equivalence akin to probabilistic bisimulation. On the other hand, in search of a stronger tool, approaches such as [2] try to identify “*semantics*” of the neurons. For instance, given a vector of inputs to the network, the *I/O semantics* of a neuron [2] is the vector of activation values of this neuron obtained on these inputs. This represents a finite-dimensional approximation of the actual semantics of a neuron as a computational device. Either way, replacing several neurons with one that is very similar yields only moderate savings on size if the abstract network is supposed to be similar, i.e., yield mostly the same predictions and ensure a tight connection between the similar neurons.

**Our Contribution.** We focus on studying abstraction irrespective of the use case (verification, smaller networks, explainability), to establish a better principal understanding of this crucial, yet in this context underdeveloped technique. First, we explore a richer abstraction scheme, where a group of neurons can be represented not only by a chosen neuron but also by a *linear combination of neurons*. Thus instead of keeping exactly one representative per group, we can “reuse” the chosen representatives in many linear combinations; in other words, the representatives can attain many roles, partially representing many groups, which reduces their required count. We provide several algorithms to do so, ranging from resource-intensive algorithms aiming to show the limits of the approach to efficient heuristics approximating the former ones quite closely. We apply these algorithms to the semantic approach of [2] as well as to the syntactic, bisimulation-like approach similar to [22] not implemented previously. Experimental results confirm the *greater power of this linear-combination* approach; further, they provide insight into the *advantages of semantic similarity over the syntactic* one, pointing out the more advantageous future research directions.

Further, we provide a formal link between the concrete and abstract neurons by proving an error bound induced by the abstraction, showing the abstraction is valid and (approximately) simulates the original network. We show the bound is better than the one based on bisimulation. While still not very practical, the experiments show that even on unseen data, the error is always closely bounded by the error on the data used for generating the abstraction, and mostly even

a lot smaller. This empirical version of the concept of error could thus enable the transfer of reasoning about the abstraction to the original network in a yet much tighter way.

In addition, we suggest *abstraction-refinement* procedures to better fine-tune the trade-off between the precision and the size of the abstraction. The experiments reveal that a more aggressive abstraction followed by a refinement provides better results than a direct, moderate abstraction. Hence involving our refinement in the abstraction process improves the resulting quality, opening new lines of attack on efficient neural network abstractions.

**Summary.** Our contribution can be summarized as follows:

- We define abstractions of neural networks with (approximate) equivalences being linear equations over semantics of neurons. We provide a theoretical bound on the induced error, see Theorem 1. We reflect this idea also on the syntactic, bisimulation-based abstraction.
- We implement both approaches and compare them mutually as well as to their previous, special cases with equivalences being (approximate) identities. We perform the experiments on a number of standard benchmarks, such as MNIST, CIFAR, or FashionMNIST, concluding advantages of semantic over syntactic approaches and of linear over identity-based ones.
- We introduce an abstraction-refinement procedure and also evaluate its benefits experimentally.

**Related Work.** There are various approaches for verification of NN, however, we are **not** presenting another verifier. Instead, we introduce an approach that is **orthogonal to verification** and could be integrated with an existing verifier. Therefore, we do not compare our approach to any verification tool and refer the interested reader to the Verification of Neural Networks Competition [4] for an overview of existing approaches [16, 26, 31, 33].

Network compression techniques share many similarities with abstraction [7] and either focus on reducing the memory footprint [14, 15] or computation time of the model [12], but in contrast, do not provide any formal relation to the original network, rendering them inappropriate for understanding redundancies or verification. Knowledge distillation is a prominent technique, which can reduce networks by a significant amount, but completely loses any connection to the original network [13], and can thus not be used in verification. There is some progress in using abstract domains for scalable verification, like [26, 27, 29], but they do not produce an abstracted NN for verification. Instead, they apply abstraction only tightly entangled together with the verification algorithm. These approaches also try to generate a more scalable verification, however, the key difference is that they do not return an actual abstracted network that could be reused or manually inspected. Katz et al. [8] introduce an abstraction scheme for NN, in which they decompose neurons into several parts, before merging them again to obtain an over-approximation of the original network. However, their approach is limited to networks with one output neuron. For networks with more output neurons, the property to be verified needs to be baked into the

network, making the approach significantly less flexible. Additionally, this tight entanglement of specification and neural network does not allow for retrieving the abstraction later and reusing it for anything else than to verify that specific property. This strongly contrasts our generic and usage-agnostic abstraction and their property-restricted abstractions.

Some other works use abstraction after representing a neural network as an interval neural network [23], or more generally, by using more complex abstract domains [28]. While theoretically interesting, the practicality of these works has not been investigated. There are two approaches that we consider to be the closest to our work: a bisimulation-based approach [22], and *DeepAbstract* [2], which we will more closely introduce in the preliminaries, and compare to in the experiments.

## 2 Preliminaries

In this work, we focus on classification feedforward neural networks. Such a neural network  $N$  consists of several layers  $1, 2, \dots, L$ , with 1 being the *input layer*,  $L$  being the *output layer* and  $2, \dots, L - 1$  being the *hidden layers*. Each layer  $\ell$  contains  $n_\ell$  *neurons*. Neurons of one layer are connected to neurons of the previous and next layers by means of weighted connections. Associated with every layer  $\ell$  that is not an output layer is a *weight matrix*  $W^{(\ell)} = (w^{(\ell)}(i, j)) \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$  where  $w^{(\ell)}(i, j)$  gives the weights of the connections to the  $i^{\text{th}}$  neuron in layer  $\ell + 1$  from the  $j^{\text{th}}$  neuron in layer  $\ell$ . We use the notation  $W_{i,*}^{(\ell)} = [w^{(\ell)}(i, 1), \dots, w^{(\ell)}(i, n_\ell)]$  to denote the incoming weights of neuron  $i$  in layer  $\ell + 1$  and  $W_{*,j}^{(\ell)} = [w^{(\ell)}(1, j), \dots, w^{(\ell)}(n_{\ell+1}, j)]^\top$  to denote the outgoing weights of neuron  $j$  in layer  $\ell$ . Note that  $W_{i,*}^{(\ell)}$  and  $W_{*,j}^{(\ell)}$  correspond to the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of  $W^{(\ell)}$  respectively. A vector  $\mathbf{b}^{(\ell)} = [b_1^{(\ell)}, \dots, b_{n_\ell}^{(\ell)}] \in \mathbb{R}^{n_\ell}$  called *bias* is also associated with each hidden layer  $\ell$ . The input and output of a neuron  $i$  in layer  $\ell$  is denoted by  $h_i^{(\ell)}$  and  $z_i^{(\ell)}$  respectively. We call  $\mathbf{h}^\ell = [h_1^{(\ell)}, \dots, h_{n_\ell}^{(\ell)}]^\top$  the vector of *pre-activations* and  $\mathbf{z}^\ell = [z_1^{(\ell)}, \dots, z_{n_\ell}^{(\ell)}]^\top$  the vector of *activations* of layer  $\ell$ . The neuron takes the input  $\mathbf{h}^\ell$ , and applies an *activation function*  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  element-wise on it. The output is then calculated as  $\mathbf{z}^\ell = \phi(\mathbf{h}^\ell)$ , where standard activation functions include tanh, sigmoid, or ReLU [21]. We assume that the activation function is Lipschitz continuous, which in particular holds for the aforementioned functions [30]. In a feedforward neural network, information flows strictly in one direction: from layer  $\ell_m$  to layer  $\ell_n$  where  $\ell_m < \ell_n$ . For an  $n_1$ -dimensional input  $\mathbf{x} \in \mathcal{X}$  from some input space  $\mathcal{X} \subseteq \mathbb{R}^{n_1}$ , the output  $\mathbf{y} \in \mathbb{R}^{n_L}$  of the neural network  $N$ , also written as  $\mathbf{y} = N(\mathbf{x})$  is iteratively computed as:

$$\begin{aligned} \mathbf{h}^{(0)} &= \mathbf{z}^{(0)} = \mathbf{x} \\ \mathbf{h}^{(\ell+1)} &= W^{(\ell)} \mathbf{z}^{(\ell)} + \mathbf{b}^{(\ell+1)} \end{aligned} \tag{1}$$

$$\mathbf{z}^{(\ell+1)} = \phi(\mathbf{h}^{(\ell+1)}) \tag{2}$$

$$\mathbf{y} = \mathbf{z}^{(L)}$$



where  $\phi(\mathbf{x})$  is the column vector obtained by applying  $\phi$  component-wise to  $\mathbf{x}$ . We abuse the notation and write  $\mathbf{z}^{(\ell)}(\mathbf{x})$ , when we want to specify that the output of layer  $\ell$  is computed by starting with  $\mathbf{x}$  as input to the network.

## 2.1 Syntactic and Semantic Abstractions

We are interested in a general abstraction scheme that is not only useful for verification, but also for revealing redundancies, while keeping a formal link to the original network. We distinguish between two types of abstraction: semantic and syntactic. Syntactic abstraction makes use of the weights of the network, the syntactic information, and allows for overapproximation guarantees that are not restricted to specific inputs. However, as we shall see in the experiments, the semantic abstraction can capture the behavior of the original network on typical input data much more accurately than its syntactic counterpart. This comes at the cost of a more challenging error analysis.

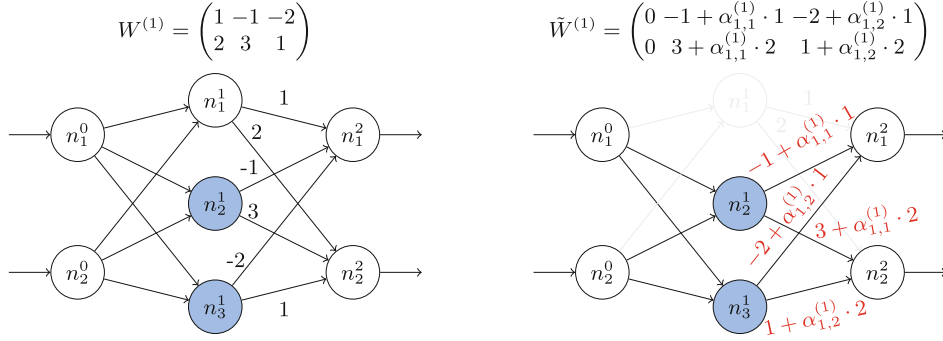
**Semantic Information.** In line with *DeepAbstract* [2], we will create the semantic information based on a set of inputs, the *I/O set*,  $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathcal{X}$ , which is typically a subset of the training dataset. We use the inputs  $\mathbf{x}_j \in X$ , feed them to the network and store the output values  $\{\mathbf{z}^{(\ell)}(\mathbf{x}_j)\}_{\mathbf{x}_j \in X}$  of a layer  $\ell$  in a matrix  $\mathbf{Z}^{(\ell)} = (z_i^{(\ell)}(\mathbf{x}_j))_{i,j}$ . Note that the columns are the  $\mathbf{z}^{(\ell)}(\mathbf{x}_j)$  and the rows, denoted as  $\mathbf{Z}_{i,*}^{(\ell)}$ , correspond to the values one neuron  $i$  produces for all inputs  $\mathbf{x}_j$ . We refer to the vector  $\mathbf{Z}_{j,*}^{(\ell)}$  as the *semantics* of neuron  $j$ . This collection of matrices  $\mathbf{Z}^{(\ell)}$  for all layers contains the semantic information of the network.

**DeepAbstract.** Since we will compare our approach to *DeepAbstract* [2], we will give a concise description of the idea of their work. First, it generates the semantic information  $\mathbf{Z}$ . For one layer  $\ell$ , it clusters the rows of the matrix by using standard clustering techniques, e.g. k-means clustering [3]. Each cluster is considered to be a group of neurons that have similar semantics and similar behavior. Thus, only one group representative is chosen to remain and the rest is replaced by the representatives.

**Bisimulation.** The idea of [22] is to apply the notion of bisimulation to NN. A bisimulation declares two neurons as equivalent if they agree on their incoming weights, biases, and activation functions. Additionally, the paper introduces a  $\delta$ -bisimulation that allows neurons to be equivalent only up to  $\delta$ , i.e. two neurons  $i, j$  of layer  $\ell$  with the same activation function are considered to be  $\delta$ -bisimilar, if for all  $k$  :  $|w^{(\ell-1)}(i, k) - w^{(\ell-1)}(j, k)| \leq \delta$  and  $|b_i^{(\ell)} - b_j^{(\ell)}| \leq \delta$ .

## 3 Linear Abstraction

Our abstraction of a NN is based on the idea that huge NN in their practical application are usually trained with more neurons than necessary. Since there



**Fig. 1.** Linear Abstraction - On the left, the original network with the basis  $B$  in blue. On the right, the abstracted network with the removed neuron  $n_1^1$  and the changed output weights of the basis neurons  $n_2^1, n_3^1$ , where we assume that  $n_1^1$  can be simulated by  $\alpha_{1,1}^{(1)} \cdot n_2^1 + \alpha_{1,2}^{(1)} \cdot n_3^1$ . (Color figure online)

are techniques to avoid “overfitting”, users of machine learning tend to use NN that are bigger than necessary for their task [19]. Intuitively, such networks thus contain redundancies. We want to remove these redundancies to decrease the size of the network and make it more scalable for verification.

Existing approaches group together similar neurons, and then choose a representative. Instead, we propose to replace a neuron with a linear combination of other neurons. More specifically, we want to replace a neuron  $i$  of layer  $\ell$ , not by one single neuron  $j$ , but rather by a clever combination of several neurons, called the *basis*,  $B^{(\ell)} \subset \{1, \dots, n_\ell\} \setminus \{i\}$ , which is a subset of all neurons of this layer and in this case given as their indices. We assume that the behavior of a neuron can be simulated by a linear combination of the behavior of the basis neurons, i.e. by  $\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} \cdot \mathbf{Z}_{j,*}^{(\ell)}$  for some  $\alpha_{i,j}^{(\ell)} \in \mathbb{R}$ .

**Example.** Consider the neural network in Fig. 1. It has an input layer with two neurons  $n_1^0, n_2^0$ , one hidden layer with three neurons  $n_1^1, n_2^1, n_3^1$ , and an output layer with two neurons  $n_1^2, n_2^2$ . We assume that we are given the basis  $B^{(1)} = \{n_2^1, n_3^1\}$ , marked with blue color in the figure, and the linear coefficients  $\alpha_{1,1}^{(1)}, \alpha_{1,2}^{(1)}$ . That is, we assume that  $n_1^1$  can be simulated by the linear combination  $\alpha_{1,1}^{(1)} \cdot n_2^1 + \alpha_{1,2}^{(1)} \cdot n_3^1$ . We can remove neuron  $n_1^1$  and its outgoing weights  $[1, 2]^\top$ , and add the outgoing weights scaled by the linear coefficients to the basis neurons instead. We add  $\alpha_{1,1}^{(1)} \cdot [1, 2]^\top$  to the outgoing weights of neuron  $n_2^1$ , so we get  $[-1, 3]^\top + \alpha_{1,1}^{(1)} \cdot [1, 2]^\top = [-1 + \alpha_{1,1}^{(1)} \cdot 1, 3 + \alpha_{1,1}^{(1)} \cdot 2]^\top$ , and respectively, we get  $[-2 + \alpha_{1,2}^{(1)} \cdot 1, 1 + \alpha_{1,2}^{(1)} \cdot 2]^\top$  as the outgoing weights of neuron  $n_3^1$ .

The computational overhead to compute a linear combination compared to finding a representative is negligible, as we will see in our experiments (see Sect. 5.2). On the other hand, they provide more expressive power, subsuming the aforementioned clustering-based approach [2]. In particular, we can detect scaled weights that previous approaches failed to identify.

Please note that although it is possible to replace a neuron with a linear combination of any other neurons in the network, we will only use neurons

from the same layer due to more efficient support by modern neural network frameworks.

In the following sections, we will answer three questions: How can one find a set of neurons that serves as a basis (Sect. 3.1)? How to find the coefficients for the linear combination (Sect. 3.2)? How to replace a neuron, once its representation as a linear combination is given (Sect. 3.3)?

### 3.1 Finding the Basis

Our approach is meant to find a sufficient smaller subset of neurons in one layer, which is enough to represent the behavior of the whole layer. We will make use of the semantic information of a layer  $\ell$ , given as  $\mathbf{Z}^{(\ell)} = (z_i^{(\ell)}(\mathbf{x}_j))_{i,j}$  (see Sect. 2.1). Based on this, we try to find a basis of neurons, i.e. a set of indices for neurons in this layer  $\{j_1, \dots, j_{k_\ell}\} = B^{(\ell)} \subset \{1, \dots, n_\ell\}$ , which can represent the whole space as well as possible. To this end we want to find a subset of size  $k = |B^{(\ell)}|$  such that  $\|\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} \cdot \mathbf{Z}_{j,*}^{(\ell)} - \mathbf{Z}_{i,*}^{(\ell)}\|$  is minimized. We denote with

$$A_B = \begin{bmatrix} | & & | \\ \mathbf{Z}_{j_1,*}^{(\ell)} & \dots & \mathbf{Z}_{j_{k_\ell},*}^{(\ell)} \\ | & & | \end{bmatrix} \quad (3)$$

the matrix containing the activations  $\mathbf{Z}_{j,*}^{(\ell)}$  of the neurons in the basis as columns.

**Greedy Algorithm.** The problem of finding an optimal basis of size  $k$  w.r.t.  $L_2$  distance can be seen as a variation of the *column subset selection problem* which is known to be NP-complete [25]. As a consequence, we use a variant of a greedy algorithm [1]. While it does not always yield the optimal solution, it has been observed to work reasonably well in practice [9, 10].

It has already been observed that layers closer to the output usually contain more condensed information and more redundancies, and can, thus, be compressed more aggressively [2]. We present a greedy algorithm that chooses which layer contains more information and needs a larger basis instead of decreasing the basis sizes equally fast in each layer.

In Algorithm 1, we see that the procedure iteratively removes neurons from the basis. To this end, it iterates over all layers  $l \in \{1, \dots, L\}$  in the network. It tries to remove one neuron at a time from the basis. Then it computes the projection error of the smaller basis, which is defined as  $\|\mathbf{Z}^{(\ell)\top} - \Pi_{A_B} \mathbf{Z}^{(\ell)\top}\|$ , where  $\Pi_{A_B}$  is the matrix that projects the columns of  $\mathbf{Z}^{(\ell)\top}$  onto the column space of  $A_B$ . The columns of  $A_B$  are the rows of  $\mathbf{Z}^{(\ell)}$  whose neurons belong to  $B$ . It greedily evaluates all neurons in all layers and selects the best neuron of the best layer to be removed. After checking every layer, the algorithm decides on the best layer and neuron to be removed, i.e. the one with the smallest error.

Since the approach thoroughly evaluates all possibilities, its runtime depends on both the number of layers and neurons. A natural alternative would be a heuristic that guides us similarly well through the search space. We provide our choice of heuristic below.

**Algorithm 1.** Greedy algorithm over all layers

---

```

1: Given:  $k$  neurons to be removed
2:  $\forall l \in \{1, \dots, L\} : B^{(l)} \leftarrow \{1, \dots, n_l\}$ 
3:  $error_{min} \leftarrow \infty, l_{best} \leftarrow -1, n_{best} \leftarrow -1$ 
4: for  $i \in 1, \dots, k$  do
5:   for  $l \in 1, \dots, L$  do
6:     for  $j \in 0, \dots, n_l$  do
7:       Compute the projection error  $error_j$  of  $A_{B^{(l)} \setminus \{j\}}$ 
8:       if  $error_j < error_{min}$  then
9:          $l_{best} \leftarrow l$ 
10:         $n_{best} \leftarrow j$ 
11:         $error_{min} \leftarrow error_j$ 
12:    $B^{l_{best}} \leftarrow B^{l_{best}} \setminus \{n_{best}\}$ 
13: return  $B^1, \dots, B^L$ 

```

---

**Variance-Based Heuristic.** Instead of a step-wise decision that takes a lot of computation time, we propose to use a variance-based heuristic. We define the variance of a vector  $\mathbf{v} \in \mathbb{R}^n$  in the usual way by  $\text{Var}(\mathbf{v}) = \sum_{i=1}^n (v_i - \text{Mean}(\mathbf{v}))^2$  where  $\text{Mean}(\mathbf{v})$  is the mean of the vector values. W.l.o.g. let the neurons be numbered in such a way that  $\text{Var}(\mathbf{z}_1^{(\ell)}) \geq \dots \geq \text{Var}(\mathbf{z}_{n_\ell}^{(\ell)})$ . We then choose the basis to contain the neurons with the  $k_\ell$  largest variances, i.e.  $B = \{1, \dots, k\}$ . We assume that neurons with a higher variance in their output values carry more information, and are, therefore, more relevant. Indeed, we can see in our experiments, i.e. Fig. 2, that the heuristic-based approach can achieve similar results, but in far less time.

### 3.2 Finding the Coefficients

Given a basis  $B^{(\ell)}$  for some layer  $\ell$ , computed with the before-mentioned approach, we want to find the coefficients that can be used to replace the remaining neurons which are not part of the basis. We fix a neuron  $i$  in layer  $\ell$  that we want to replace and whose values are stored in  $\mathbf{Z}_{i,*}^{(\ell)}$ , and we want to minimize  $\|\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} \cdot \mathbf{Z}_{j,*}^{(\ell)} - \mathbf{Z}_{i,*}^{(\ell)}\|$  for  $\alpha_{i,j}^{(\ell)}$ .

Since we want to find a linear combination of vectors, a natural choice is **linear programming**. The linear program is straightforward and can be found in [6, Appendix C]. Note that with the linear program, we are minimizing the  $L_1$ -distance between the neuron's values and its replacement, i.e.  $\|\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} \cdot \mathbf{Z}_{j,*}^{(\ell)} - \mathbf{Z}_{i,*}^{(\ell)}\|_1$ .

In a different way, we can also consider the vectors  $\mathbf{Z}_{j,*}^{(\ell)}$  for  $j \in B^{(\ell)}$  to span a vector space. If we are given a subset  $\{\mathbf{Z}_{j,*}^{(\ell)} | j \in B^{(\ell)} \subset \{1, \dots, n_\ell\}\}$  that forms a basis for this space, i.e.  $\text{span}((\mathbf{Z}_{j,*}^{(\ell)})_{j \in B^{(\ell)}}) = \text{span}((\mathbf{Z}_{j,*}^{(\ell)})_{j \in \{1, \dots, n_\ell\}})$ , we can represent any other vector  $\mathbf{z}_i^{(\ell)}$  in terms of this basis. However, we usually cannot represent one neuron perfectly by a linear combination of other neurons. **Orthogo-**

**nal projection** gives us the closest point in the subspace  $\text{span}((\mathbf{Z}_{j,*}^{(\ell)})_{j \in B^{(\ell)}})$  for any vector, in terms of  $L_2$ -distance. Then,  $\boldsymbol{\alpha} = [\alpha_{i,j_1}^{(\ell)}, \dots, \alpha_{i,j_{k_\ell}}^{(\ell)}]^\top := (A_B^\top A_B)^{-1} A_B^\top \mathbf{Z}_{i,*}^{(\ell)}$  gives us the coefficients for the orthogonal projection of  $\mathbf{Z}_{i,*}^{(\ell)}$  on the linear space spanned by the columns of  $A_B$ . For a more detailed description of orthogonal projection see e.g. [17, Chapter 6.8]. Note that we assume that the columns of  $A_B$  are linearly independent. If not we can simply replace the respective neurons directly.

### 3.3 Replacement

Assuming, we have a basis  $B^{(\ell)}$  of this layer and we already know the coefficients  $\alpha_{i,j}^{(\ell)} \in \mathbb{R}$  for  $j \in B^{(\ell)}$  that we need to simulate the behavior of neuron  $i$ . This means, we have a linear combination  $\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} \cdot \mathbf{Z}_{j,*}^{(\ell)}$ , which we want to use instead of neuron  $i$  itself. We will replace the outgoing weights  $W^{(\ell)}$  of this layer, such that for all  $j \in B^{(\ell)}$

$$\tilde{W}_{*,j}^{(\ell)} = [w^{(\ell)}(1,j) + \alpha_{i,j}^{(\ell)} w^{(\ell)}(1,i), \dots, w^{(\ell)}(n_{\ell+1},j) + \alpha_{i,j}^{(\ell)} w^{(\ell)}(n_{\ell+1},i)]^\top \quad (4)$$

$$= W_{*,j}^{(\ell)} + \alpha_{i,j}^{(\ell)} W_{*,i}^{(\ell)} \quad (5)$$

Furthermore, we set  $\tilde{W}_{*,i}^{(\ell)} = [0, \dots, 0]^\top$ , and  $\tilde{W}_{i,*}^{(\ell)} = [0, \dots, 0]^\top$ . This means that we will not use the output of neuron  $i$  anymore, but rather a weighted sum of the outputs of neurons in  $B^{(\ell)}$ , and that we will not even compute the value of  $i$ . Additionally, we keep track of the changes we apply to the different neurons with a matrix  $D^{(\ell)} = (d_{j,i}^{(\ell)}) \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ . Initially,  $D^{(\ell)}$  is 0 and after each replacement, we add  $\alpha_{i,j}^{(\ell)} \cdot w^{(\ell)}(i, i')$  to  $d_{j,i'}^{(\ell)}$ , for  $j \in B^{(\ell)}$  and  $i' \in \{1, \dots, n_{\ell+1}\}$ . This is necessary for restoring neurons at a later point.

In the optimal case, the replacement will not change the overall behavior of the neural network. We can derive a the same semantic equivalence from [22] incorporated into our setting:

**Proposition 1 (Semantic Equivalence).** *Let  $N$  be a neural network with  $L$  layers,  $\ell$  a layer of  $N$ ,  $i$  a neuron of this layer, and  $B^{(\ell)} \subset \{1, \dots, n_\ell\} \setminus \{i\}$  a chosen basis. Let  $\tilde{N}$  be the NN after replacing neuron  $i$  by a linear combination of basis vectors with coefficients  $\alpha_{i,j}^{(\ell)}$ , with the procedure as described above.*

*If for all inputs  $\mathbf{x} \in X \subset \mathcal{X}$ ,  $z_i^{(\ell)}(\mathbf{x}) = \sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} z_j^{(\ell)}(\mathbf{x})$ , then  $N$  and  $\tilde{N}$  are semantically equal, i.e. for all inputs  $\mathbf{x} \in X$ ,  $\tilde{N}(\mathbf{x}) = N(\mathbf{x})$ .*

It is easy to see that this proposition is true, for a full proof see [6, Appendix A]. However, the proposition assumes equality of  $z_i^{(\ell)}(\mathbf{x})$  and  $\sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} z_j^{(\ell)}(\mathbf{x})$  for  $\mathbf{x} \in X$ , which virtually never holds for real-world neural networks. Therefore, we want to minimize the difference  $|z_i^{(\ell)}(\mathbf{x}) - \sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} z_j^{(\ell)}(\mathbf{x})|$ , which will not yield a semantically equivalent abstraction, but an abstraction with very similar behavior. We can then **quantify the difference** between the output of the

original network and the abstraction, i.e. the *induced error* with the following Theorem.

**Theorem 1 (Over-approximation Guarantee).** *Let  $N$  be an NN with  $L$  layers. For each layer  $\ell$ , we have a basis of neurons  $B^{(\ell)}$ , and a set of replaced neurons  $I^{(\ell)}$ . Then, let  $\tilde{N}$  be the network after replacing neurons in  $I^{(\ell)}$  as described above.*

*We can over-approximate the error between the output of the original network  $N^L$  and the output of the abstraction  $\tilde{N}^L$  for  $\mathbf{x} \in X \subset \mathcal{X}$  by*

$$\|\tilde{N}^L(\mathbf{x}) - N^L(\mathbf{x})\| \leq b(1 - a^{L-1})/(1 - a)$$

*with  $a = \lambda(\|W\| + \eta)$ ,  $b = \lambda\|W\|\epsilon$ , with  $\lambda^{(\ell)}$  being the Lipschitz-constant of the activation function in layer  $\ell$ ,  $\lambda = \max_{\ell} \lambda^{(\ell)}$ ,  $\|W\| = \max_{\ell} \|W^{(\ell)}\|_1$ ,  $\eta = \max_{\ell} \eta^{(\ell)}$ , and  $\epsilon = \max_{\ell} \epsilon^{(\ell)}$ , assuming that for all layers  $\ell \in \{1, \dots, L\}$  and for all inputs  $\mathbf{x} \in X$ , we have*

- *for  $i \in I^{(\ell)}$  :  $|z_i^{(\ell)}(\mathbf{x}) - \sum_{j \in B^{(\ell)}} \alpha_{i,j}^{(\ell)} z_j^{(\ell)}(\mathbf{x})| \leq \epsilon^{(\ell)}$*
- *$|\sum_{i \in I^{(\ell)}} W_{*,i}^{(\ell)} \sum_{t \in B^{(\ell)}} \alpha_{i,t}^{(\ell)}| \leq \eta^{(\ell)}$*

In other words, we can over-approximate the difference in the output of the original and the abstracted network by a value that depends on the weight matrices, the activation function and the tightness of the abstracted neurons to their replacements. The proof can be found in [6, Appendix B]. This Theorem provides us with the **theoretical guarantee** that, given our abstraction, we can provide a valid over-approximation of the output of the original network.

**Comparison to the  $\delta$ -Bisimulation.** Let us recap the error definition from [22]. The difference of the bisimulation and the original network is bounded by  $[(2a)^k - 1]b/(2a - 1)$ , where  $a = \lambda|S|\|W\|$  and  $b = \lambda(|P|L(\mathcal{N})\|x\| + 1)\delta^1$ . In this notation,  $|S|$  is the maximum number of neurons per layer in the whole network,  $|P|$  the maximum number of neurons in the bisimulation (can be understood as the number of neurons in an abstraction),  $L(\mathcal{N})$  is the maximum Lipschitz-constant of all layers, and  $\delta$  is the maximum absolute difference of the bias and sum of the incoming weights.

The drawbacks of that approach are twofold: (i) the error is based on one specific input, and (ii) it makes use of the Lipschitz-constant of the whole network. Calculating the Lipschitz constant of an NN is still part of ongoing research [11] and not a trivial problem. In contrast, we improve on both. Our error calculation generalizes over a set of inputs. Additionally, we use local information, stored in the weight-matrices, to circumvent using the Lipschitz-constant of the NN.

## 4 Refinement

For certain inputs the abstraction might not reflect the behavior of the original network. For these inputs, so-called *counterexamples*, we may want to *refine* the

<sup>1</sup> Please note that this statement is slightly different from the paper  $((2a)^k)$  instead of  $(2/a)^k$ , which we believe to be a typo in the paper.

abstraction, as opposed to starting the abstraction from the original network again. We consider an input to be a counterexample whenever the abstraction assigns it a different label than the original network. However, a counterexample can be any input that does not align with the specifications.

We propose to refine the abstraction by restoring some of the replaced neurons. To do this, we need to know which neurons should be replaced and how. We first briefly mention three heuristics to choose a neuron for restoration. Afterward, we explain how to restore a neuron. Note that the refinement offers more than a “roll-back” of the most recent step of the abstraction since it picks the step-to-be-rolled-back in retrospect reflecting all other steps, leading to a more informed choice. This could in principle be done directly in the abstraction phase, but at an infeasible cost of a huge look-ahead.

**Refinement Heuristics.** We propose three different heuristics: difference-guided, gradient-guided, and look-ahead.

- The *difference-guided* refinement looks at the difference of a neuron in the original and its representation as a linear combination in the abstraction. It replaces the neuron with the largest difference.
- The *gradient-guided* refinement additionally takes the gradient of the NN into account, that is computed as in the training phase of the NN. This takes into account how the whole network would need to change to fix the counterexample.
- The *look-ahead* is the most greedy method and would try out every replaced neuron. It would check how much the network would improve if the neuron was replaced and then chooses the neuron with the highest improvement.

More details on the approaches can be found in [6, Appendix D].

**Restoration of a Neuron.** The restoration principle can be seen as the counterpart of the replacement. Let  $\tilde{N}$  be the network obtained by replacing several neurons in the original network  $N$ , where we want to restore a deleted neuron  $i$  of layer  $\ell$ . To do this, we need not only to get the original neuron back, including its incoming and outgoing weights but also to remove the additional outgoing weights from the basis neurons. Intuitively, the restoration removes the linear combination, ensures that the original outgoing weights for the neuron are used, and adjusts the incoming weights of the neuron. We may have changed layer  $\ell - 1$ , and thus we cannot restore the original incoming weights of neuron  $i$ , but we have to adapt it to changes in the basis  $B^{(\ell-1)}$ . This can be done with the following changes:

- $\forall j \in B^{(\ell)}: \tilde{W}_{*,j}^{(\ell)} = \tilde{W}_{*,j}^{(\ell)} - \alpha_j W_{*,j}^{(\ell)}$
- $\tilde{W}_{*,i}^{(\ell)} = W_{*,i}^{(\ell)}$
- $\forall j \in B^{(\ell-1)}: \tilde{w}^{(\ell-1)}(i, j) = w^{(\ell-1)}(i, j) + d_{j,i}^{(\ell-1)}$

Afterward, we subtract  $\alpha_j \cdot w^{(\ell)}(i, i')$  from  $d_{j,i'}^{(\ell)}$  for  $i' \in \{1, \dots, n_{\ell+1}\}$  and  $j \in B^{(\ell)}$ .



## 5 Experimental Results

Our experimental section is divided into several parts: The first one covers how the different methods for finding a basis and the coefficients compare, as described in Sect. 3.2 and Sect. 3.1. The second part shows experiments on our approach in comparison to existing works, namely *DeepAbstract* [2] and our implementation of bisimulation [22] (which was not implemented before). The third part contains the comparison between the abstraction based on syntactic and semantic information. The fourth part describes our experiments on abstraction refinement. Finally, the last part contains experiments on the error induced by our abstraction. Note that supplemental experiments can be found in the Appendix.

Lastly, the work of Katz et al. [8] tightly couples the abstraction with the subsequent particular verification, by integrating the specification as layers into the network. It is, thus, not clear how an abstraction from [8] could be extracted from the tool and reused for another purpose. Additionally, our abstraction would have to be connected with some verification algorithm (DeepPoly, as done by DeepAbstract, or some other) to compare. *Any comparison of the two works would then mostly compare the different verification tools, not really the abstractions.* Although a comparison of different verifiers linked to our LiNNA is an interesting next step into one of the possible applications, it is out of the scope of this paper, which examines the abstraction itself (see Introduction).

**Implementation.** We implemented the approach in our tool *LiNNA* (Linear Neural Network Abstraction)<sup>2</sup>. We used networks that were trained on MNIST [20], CIFAR-10 [18], and FashionMNIST [32] for our experiments. In the following, we refer to the corresponding trained networks with “ $L \times n$ ”, where  $L$  denotes the number of hidden layers and  $n$  is the number of neurons in these hidden layers. All experiments were conducted on a computer with Ubuntu 22.04 LTS with 2.6 GHz Intel® Core™ i7 processors, and 32 GB of RAM.

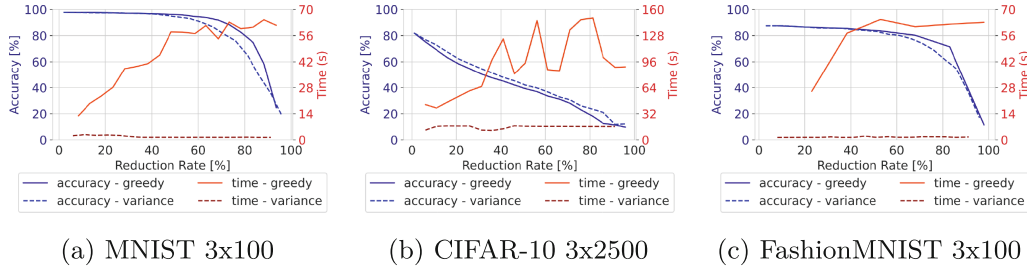
**Performance Measures.** We will compare the approaches mostly on (i) the reduction rate and (ii) the accuracy on a test set. Intuitively, the *reduction rate* describes how much the NN was reduced by abstraction. If an NN  $N$  has in total  $n$  neurons, but after reduction, there are  $m$  neurons left, then the reduction rate is then defined as  $RR(N) = 1 - \frac{m}{n}$ . The *accuracy* of a NN on a test set is defined as the ratio of how many inputs are predicted with the correct label. This is the key performance indicator in machine learning and shows how well a network generalizes to unseen data. In evaluating our abstraction, we follow the same principle since we want to know how well the NN generalizes after abstraction. Note that this test set was not used for training or computing the abstraction.

### 5.1 Abstraction

**Finding the Basis.** We have given two different methods in Sect. 3.1 to find a good basis  $B$ . While the orthogonal projection yields an equally good abstraction

<sup>2</sup> <https://github.com/cxlvinchau/LiNNA>.





**Fig. 2.** *Finding the basis for replacement* - Evaluation on different datasets. The plots contain a comparison of LiNNA while using the greedy variant (solid) and the variance-based heuristic (dashed) for finding a basis with orthogonal projection. Comparison of accuracy (blue) in percent and computation time (red) in seconds. (Color figure online)

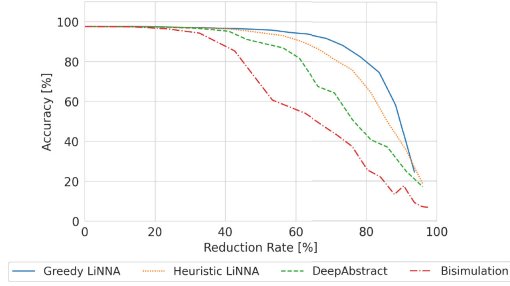
compared to linear programming, it outperforms the latter in terms of runtime by magnitudes. Hence, we conducted the rest of the experiments with orthogonal projection. The full comparison between orthogonal projection and linear programming can be found in [6, Fig. 14, Appendix E].

When we compare the greedy and the heuristic-based approach, shown in Fig. 2, we see that the former outperforms the latter in terms of accuracy on MNIST and FashionMNIST. On CIFAR-10, the variance-based approach is slightly better. However, the variance-based approach is always faster than the greedy approach and scales better, as can be seen for all datasets. Unsurprisingly, the greedy approach takes more time for higher reduction rates, because it needs to evaluate many candidates. The variance-based approach just takes the best neurons according to their variance, which has to be calculated only once. Therefore, the calculation is constant in terms of removed neurons.

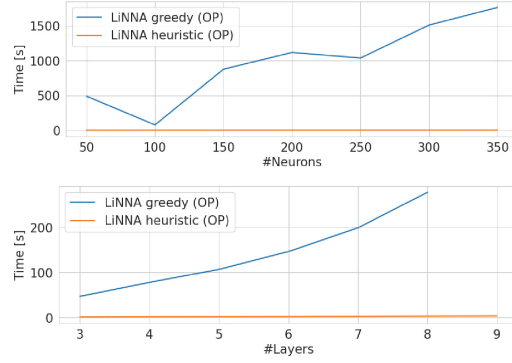
The plots show one more difference in the behavior: On MNIST and FashionMNIST, we see a quite stable accuracy until a reduction rate of 60%. We cannot see the same behavior on CIFAR-10. We believe this is due to the accuracy and size of the networks. Whereas it is fairly easy to train a feedforward network for MNIST and FashionMNIST on a regular computer, this is more challenging for CIFAR-10. We plan to include more extensive experiments including more involved NN architectures in future work. Finally, our abstraction relies on the assumption that NNs contain a lot of redundant information.

We want to emphasize, that in machine learning, it is common to train a huge network that contains many more neurons than necessary to solve the task [34]. After the introduction of regularization techniques (e.g. [24]), the problem of over-fitting (e.g. [5]) has become often negligible. Therefore, the automatic response to a bad neural network is often to increase its size, either in depth or in width. Our approach can detect these cases and abstract away the redundant information.

**Finding the Coefficients.** We have in total four different approaches to finding the coefficients: greedy or heuristic-based linear programming, and greedy or heuristic-based orthogonal projection. All four have similar accuracies for the



**Fig. 3.** *Comparison of LiNNA to related work* - LiNNA (greedy and heuristic-based variant), *DeepAbstract* [2], and our implementation of the bisimulation [22] is evaluated in terms of accuracy on the test set for a certain reduction rate. The experiment was conducted on an MNIST  $3 \times 100$  network.

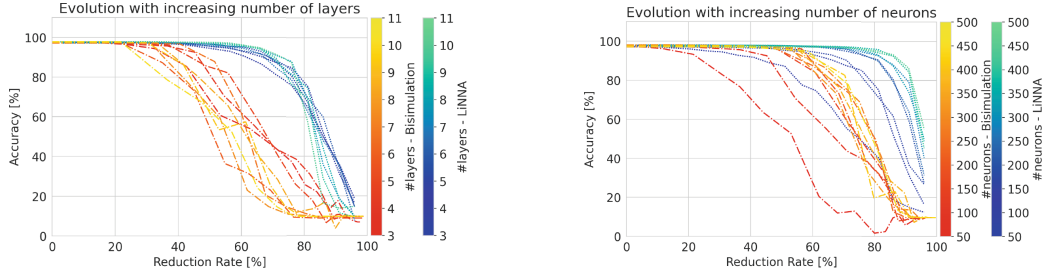


**Fig. 4.** *Scalability of LiNNA* - Average runtime for 20 different reduction rates on one network. The plot at the top depicts the runtime for MNIST networks with 4 layers, w.r.t. number of neurons. The plot at the bottom shows the runtime for MNIST networks with 100 neurons per layer, w.r.t. number of layers.

same reduction rate, whereas the heuristic ones are mostly just slightly worse than the greedy ones. For a more detailed evaluation, please refer to [6, Appendix G]. The runtimes of the four approaches, however, differ a lot. Take for example an MNIST  $3 \times 100$  network. We assume that the abstraction is performed by starting with the full network and reducing up to a certain reduction rate. Thus, we have runtimes for each of the approaches for each reduction rate. We take the average over all the reductions and get 47 s for the greedy orthogonal projection, 5130 s for the greedy linear programming, 1 s for the heuristic orthogonal projection, and 2 s for the heuristic linear programming. Linear programming takes a lot more time than orthogonal projection, and, as already seen before, the heuristic approaches are much faster than the greedy ones. Please refer to [6, Appendix J] for more experiments on the runtime. Therefore, we propose to use the heuristic approach and the orthogonal projection.

**Scalability.** We evaluate how our approach scales to networks of different sizes. We evaluate (1) how our approach scales with an increasing number of layers, and (2) how it scales with a fixed number of layers but an increasing number of neurons. We show our experiments in Fig. 4. The runtime is the average runtime over 20 different reduction rates on the same network. One can imagine this as averaging the runtimes shown in Fig. 2. We can see that the variance-based approach has almost constant runtime, whereas the runtime of the greedy approach is increasing for both a higher number of layers and neurons.

**Final Assessment.** We have four possibilities on how to abstract an NN: greedy orthogonal projection, greedy linear programming, heuristic-based orthogonal



**Fig. 5.** Evolution of the accuracy on the test set for different reduction rates, for an increasing number of layers, or neurons. We show LiNNA (blue-green) for semantic abstraction, and for syntactic abstraction, bisimulation (red-yellow). The networks were trained on MNIST and have a fixed number of neurons (100) on the left, and a fixed number of layers (4) on the right.

projection, and heuristic-based linear programming. Given that the orthogonal projection outperforms linear programming in terms of accuracy and computation time, we propose to use orthogonal projection. We believe that it is sufficient to use the heuristic-based approach, thereby gaining faster runtimes and only barely sacrificing any accuracy. Whenever we refer to *LiNNA* from now on without any additions, it will be the heuristic-based orthogonal projection.

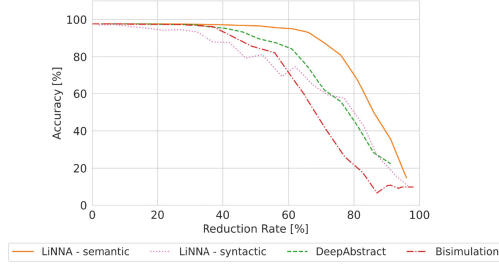
## 5.2 Comparison to Existing Work

We want to show how our approach compares to existing works, i.e. *DeepAbstract* and the *bisimulation*. Since there is no implementation available for the latter, we implemented it ourselves. Please refer to [6, Appendix F] for the details. The results of the comparison are shown in Fig. 3. It is evident that *DeepAbstract* achieves higher accuracies than the *bisimulation*, but *LiNNA* outperforms *DeepAbstract* and the *bisimulation* in terms of accuracy for all reduction rates.

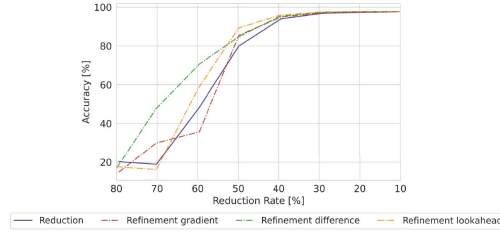
Concerning the runtime, we measure the runtime of each approach for a certain reduction rate, starting from the full network. We find that (in the median) *LiNNA* (greedy) needs 55 s up to 199 s, *LiNNA* (heuristic) 2 s up to 3 s, *DeepAbstract* 187 s up to 2420 s, and the *bisimulation* 1 s up to 2 s, on MNIST networks of different sizes (starting from  $4 \times 50$  up to  $11 \times 100$ ). The details can be found in [6, Appendix J]. The *bisimulation* performs best, however just slightly ahead of the heuristic-based *LiNNA*. The greedy *LiNNA*, as well as *DeepAbstract* both have a much higher computation time.

However, in terms of accuracy, greedy *LiNNA* seems to be the best-performing approach, given sufficient time. Due to efficiency, we suggest using heuristic-based *LiNNA*, as it is as fast as the *bisimulation*, but its accuracy is a lot better and even close to greedy *LiNNA*.

Since we are interested in the general behavior of the abstraction, we want to see how the methods work for varying sizes of networks, but not only in terms of scalability. In Fig. 5, we show the trend for *bisimulation* and *LiNNA* for an increasing number of layers resp. neurons per layer. On the left, we fix the



**Fig. 6.** *Syntactic VS. Semantic* - This plot shows the difference between using semantic resp. syntactic information for the abstraction on an MNIST  $5 \times 100$  network. Semantic: LiNNA (semantic) and DeepAbstract. Syntactic: LiNNA (syntactic) and the bisimulation.



**Fig. 7.** *Refinement* - This plot shows the accuracy of an MNIST  $5 \times 100$  network that was abstracted and refined to a certain reduction rate  $R$ . There is also a plot for an abstraction to the same reduction rate as after the refinement but without refining.

number of neurons per layer to 100 and incrementally increase the number of layers. On the right, we fix the number of layers to four and increase the number of neurons.

We can see that the performance of the networks from the bisimulation varies a lot and gets slightly worse when there are more layers, whereas LiNNA has a very small variation and the performance of the abstractions increases slightly for more layers. Both approaches compute abstractions that perform better the more neurons are in a layer, but LiNNA converges to a much steeper curve at high reduction rates.

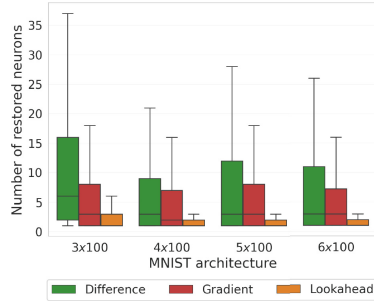
For NNs with 400 or more neurons, LiNNA can reduce 80% of the neurons without a significant loss in accuracy, whereas the bisimulation can do the same only for up to a reduction rate of 55%.

### 5.3 Semantic vs Syntactic

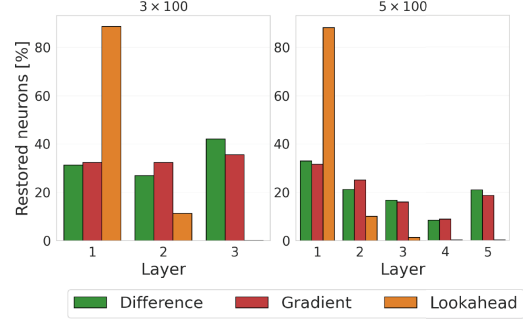
In the following, we want to show the differences between semantic and syntactic abstractions. Recall that syntactic abstraction makes use of the weights of the network, the syntactic information, with no consideration of the actual behavior of the NN on the inputs. Semantic abstraction, on the other hand, focuses on the values of the neurons on an input dataset, which also incorporates information about the weights. DeepAbstract and LiNNA, both use semantic information, whereas bisimulation uses syntactic information. We additionally evaluate the performance of LiNNA on syntactic information.

*Which type of information is better for abstraction: semantic or syntactic?* Note that both DeepAbstract and the bisimulation represent a group of neurons by one single representative, whereas LiNNA makes use of a linear combination.

We summarize our results in Fig. 6. For smaller reduction rates, the bisimulation performs better than LiNNA on syntactic information; for higher reduction rates it is reversed. In general, the approaches based on semantics (DeepAbstract



**Fig. 8.** Comparison of refinement techniques on different architectures for MNIST. The respective networks were abstracted with a reduction rate of 50%. The lines show the variance, the box represents 50% of the data, the line in the box shows the median.



**Fig. 9.** Refinement on different layers - We considered abstractions that were obtained with a 50% reduction rate and fixed 1000 counterexamples. The plots depict the percentage of restored neurons in the layers of the different MNIST networks.

and LiNNA - semantic) outperform the other two approaches w.r.t. accuracy. While abstraction based on syntactic information can provide global guarantees for any input, abstraction based on semantic information relies on the fact that its inputs during abstraction are similar to the ones it will be evaluated on later. However, we see that still the semantic information is more appropriate for preserving accuracy because it combines the knowledge about possible inputs with the knowledge about the weights.

#### 5.4 Refining the Network

We propose refinement of the abstraction in cases where it does not capture all the behavior anymore, instead of restarting the abstraction process. We consider networks that are abstracted up to certain reduction rates, i.e. 20%, 30%, ..., 90%, and use the refinement to regain 10% of the neurons. For example, we reduce the network by 90% and then use refinement to get back to a reduction rate of 80%. We evaluate this refined network on the test dataset and plot its accuracy. Additionally, we show the accuracy of the same NN which is directly reduced to an 80% reduction rate, without refinement. This plot is shown in Fig. 7 for a  $5 \times 100$  network, trained on MNIST.

The gradient and look-ahead refinement have a similar performance. However, the difference-based approach even outperforms the direct reduction itself. This behavior can be explained by the fact that the refinement and the abstraction look at different metrics for removing/restoring neurons. The refinement can focus directly on optimizing for the inputs at hand, whereas the abstraction was generated on the training set. In conclusion, the refinement can even improve the abstraction and it is beneficial to abstract slightly more than required, and refine for the relevant inputs, rather than having a finer abstraction directly.

**Comparison of the Different Approaches.** We collect images that are labeled differently by the abstraction and observe the number of neurons that are restored in order to fix the classification of each image. We ran the experiment on different networks that were abstracted with a 50% reduction rate and considered 1000 counterexamples for each network. The results are summarized in Fig. 8, where we have boxplots for each refinement method on four different network architectures. The look-ahead approach is the most effective technique since it requires the smallest number of restored neurons. In the median, it only requires 1 to 2 operations. The gradient-based approach performs noticeably worse but outperforms the difference-based approach on all networks. The computation time, however, gives a different perspective: Repairing one counterexample takes on average <1s for the difference-based approach, 1s for the gradient-based, but the look-ahead approach takes on average 4s. Interestingly, the look-ahead approach restores fewer neurons but performs worse in accuracy. The difference-based performs better in terms of accuracy while restoring more neurons.

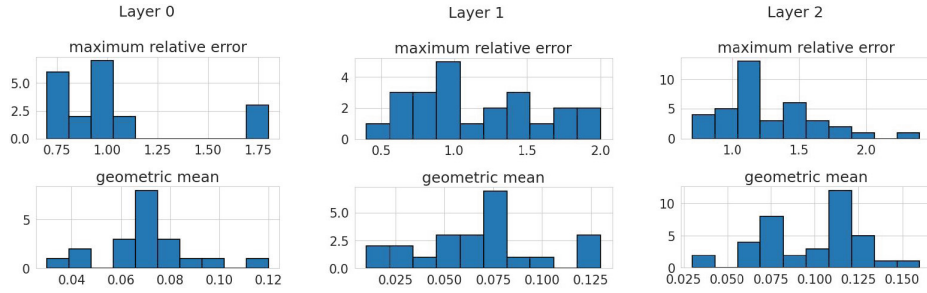
**Insight on the Relevance of Layers.** We also investigated in which layers the different refinement techniques tend to restore the neurons. The plots in Fig. 9 illustrate the percentage of restored neurons in each layer. Notably, the look-ahead approach restores most neurons in the first layer, and very few or none in the later layers, whereas the other approaches have a more uniform behavior. However, the more layers the network has, the more the gradient- and difference-based approaches tend to restore more neurons in the first layer. As reported already by [2], the first layers seem to have a larger influence on the network’s output and hence should be focused on during refinement. It is even more interesting that the difference-based approach does not focus on the first layers as much as the look-ahead approach, but it is better in terms of accuracy.

## 5.5 Error Calculation

We want to show how the abstraction simulates the original network on unseen data not only w.r.t. the output but on every single neuron. In other words, is the discrepancy between the concrete and abstract network higher on the *test* data than on the *training* data that generate the abstraction, or does the link between the neuron and its linear abstraction generalize well?

In Fig. 10, we look at this ratio (“relative error of the abstraction”), i.e. the absolute difference of (activation values of) a simulated abstract neuron to the original neuron, once on the test dataset divided by the maximum value on the training dataset. We can see that there are cases where the error can be greater than one (meaning “larger than on the training set”), see the first row of the plot. However, the geometric mean, defined as  $(\prod_{i=1}^N a_i)^{\frac{1}{N}}$ , calculated over all images is very small. Note that more experiments can be found in [6, Appendix L]. In conclusion, we can say that our abstraction is close to the original also on the test dataset, although the theoretical error calculation does not guarantee so tight a simulation. Future work should reveal how to further utilize the empirical proximity in transferring the reasoning from the abstraction to the original.





**Fig. 10.** Histograms of the relative error of the values of the neurons in an MNIST  $3 \times 100$  network and its abstraction (reduced by 30%). The first row shows the maximum relative error of each neuron in the NN, that occurred for some input from the test set. The second row shows the geometric mean of the relative error of each neuron over 100 images of the test set.

## 6 Conclusions

The focus of this work was to examine abstraction not as a part of a verification procedure, but rather as a stand-alone transformation, which can later be used in different ways: as a preprocessing step for verification, as means of obtaining an equivalent smaller network, or to gain insights about the network and its training, such as identifying where redundancies arise in trained neural networks. (This is analogous to the situation of bisimulation, which has been largely investigated on its own not necessarily as a part of a verification procedure, and its use in verification is only one of the applications.)

We have introduced *LiNNA*, which abstracts a network by replacing neurons with *linear combinations* of other neurons and also equip it with a *refinement* method. We bound the error and thus the difference between the abstraction and the original network in Theorem 1. The theorem yields a lower and an upper bound on the network's output, thereby providing its over-approximation.

We showed that the linear extension provides better performance than existing work on abstraction for classification networks, both DeepAbstract, and the bisimulation-based approach. We focused our experimental evaluation on *accuracy*, since the aim of the abstraction is to faithfully mimic the *whole classification process* in the smaller, abstract network, not just one concrete property to be verified, which describes only a very specific aspect of the network. Interestingly, the practical error is dramatically smaller than the worst-case bounds. We hope this first, experimental step will stimulate interest in research that could utilize this actual advantage, which is currently not supported by any respective theory.

Furthermore, we show that the use of *semantic information should be preferred* over syntactic information because it allows for higher reductions while preserving similar behavior and being cheap since the I/O sets can be quite small. Bringing back semantics could take us closer to the efficiency of classical software abstraction, where the semantics of states is the very key, going way beyond bisimulation.

## References

1. Altschuler, J., et al.: Greedy column subset selection: new bounds and distributed algorithms. In: Balcan, M., Weinberger, K.Q. (eds.) *Proceedings of the 33rd International Conference on Machine Learning, ICML, New York City, NY, USA*, vol. 48, pp. 2539–2548. JMLR Workshop and Conference Proceedings. JMLR.org (2016)
2. Ashok, P., Hashemi, V., Křetínský, J., Mohr, S.: DeepAbstract: neural network abstraction for accelerating verification. In: Hung, D.V., Sokolsky, O. (eds.) *ATVA 2020*. LNCS, vol. 12302, pp. 92–107. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-59152-6\\_5](https://doi.org/10.1007/978-3-030-59152-6_5)
3. Bishop, C.M.: *Pattern Recognition and Machine Learning*. Information Science and Statistics, 5th edn. Springer, Cham (2007)
4. Brix, C., et al.: First three years of the international verification of neural networks competition (VNN-COMP). *Int. J. Softw. Tools Technol. Transfer* 1–11 (2023). <https://doi.org/10.1007/s10009-023-00703-4>
5. Caruana, R., Lawrence, S., Giles, C.: Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: Leen, T., Dietterich, T., Tresp, V. (eds.) *Advances in Neural Information Processing Systems*, vol. 13. MIT Press (2000)
6. Chau, C., Křetínský, J., Mohr, S.: Syntactic vs semantic linear abstraction and refinement of neural networks (2023)
7. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. Preprint [arXiv:1710.09282](https://arxiv.org/abs/1710.09282) (2017)
8. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) *CAV 2020*. LNCS, vol. 12224, pp. 43–65. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-53288-8\\_3](https://doi.org/10.1007/978-3-030-53288-8_3)
9. Farahat, A.K., Ghodsi, A., Kamel, M.S.: A fast greedy algorithm for generalized column subset selection. Preprint [arXiv:1312.6820](https://arxiv.org/abs/1312.6820) (2013)
10. Farahat, A.K., Ghodsi, A., Kamel, M.S.: An efficient greedy method for unsupervised feature selection. In: *11th International Conference on Data Mining, Vancouver, BC, Canada*, pp. 161–170. IEEE (2011)
11. Fazlyab, M., et al.: Efficient and accurate estimation of Lipschitz constants for deep neural networks. In: Wallach, H., et al. (eds.) *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates Inc. (2019)
12. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. Preprint [arXiv:1412.6115](https://arxiv.org/abs/1412.6115) (2014)
13. Hinton, G., Vinyals, O., Dean, J., et al.: Distilling the knowledge in a neural network. In: *NeurIPS Deep Learning Workshop* (2014)
14. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708 (2017)
15. Jian, X., Jinyu, L., Yifan, G.: Restructuring of deep neural network acoustic models with singular value decomposition. In: *Interspeech*, pp. 2365–2369 (2013). <https://doi.org/10.21437/interspeech.2013-552>
16. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) *CAV 2019*. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-25540-4\\_26](https://doi.org/10.1007/978-3-030-25540-4_26)
17. Kirkwood, J.R., Kirkwood, B.H.: *Elementary Linear Algebra*. Chapman and Hall/CRC (2017)



18. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
19. Lawrence, S., Giles, C., Tsoi, A.: Lessons in neural network training: overfitting may be harder than expected. In: Anon (ed.) Proceedings of the National Conference on Artificial Intelligence, pp. 540–545. AAAI (1997)
20. LeCun, Y.: The MNIST database of handwritten digits (1998). <http://yann.lecun.com/exdb/mnist/>
21. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. [http://robotics.stanford.edu/~amaas/papers/relu\\_hybrid\\_icml2013\\_final.pdf](http://robotics.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf)
22. Prabhakar, P.: Bisimulations for neural network reduction. In: Finkbeiner, B., Wies, T. (eds.) VMCAI 2022. LNCS, vol. 13182, pp. 285–300. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-94583-1\\_14](https://doi.org/10.1007/978-3-030-94583-1_14)
23. Prabhakar, P., Rahimi Afzal, Z.: Abstraction based output range analysis for neural networks. In: Wallach, H., et al. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates Inc. (2019)
24. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). <https://doi.org/10.1016/j.neunet.2014.09.003>
25. Shitov, Y.: Column subset selection is NP-complete. Linear Algebra Appl. **610**, 52–58 (2021). <https://doi.org/10.1016/j.laa.2020.09.015>
26. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL) (2019). <https://doi.org/10.1145/3290354>
27. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: 7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA. OpenReview.net (2019)
28. Sotoudeh, M., Thakur, A.V.: Abstract neural networks. In: Pichardie, D., Sighireanu, M. (eds.) SAS 2020. LNCS, vol. 12389, pp. 65–88. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-65474-0\\_4](https://doi.org/10.1007/978-3-030-65474-0_4)
29. Tran, H.-D., et al.: Robustness verification of semantic segmentation neural networks using relaxed reachability. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 263–286. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-81685-8\\_12](https://doi.org/10.1007/978-3-030-81685-8_12)
30. Virmaux, A., Scaman, K.: Lipschitz regularity of deep neural networks: analysis and efficient estimation. In: Bengio, S., et al. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS, Montréal, Canada, pp. 3839–3848 (2018)
31. Wang, S., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Ranzato, M., et al. (eds.) Advances in Neural Information Processing Systems, vol. 34, pp. 29909–29921. Curran Associates Inc. (2021)
32. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. Preprint [arXiv:1708.07747](https://arxiv.org/abs/1708.07747) (2017)
33. Xu, K., et al.: Fast and complete: enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021)
34. Zhang, C., et al.: Understanding deep learning requires rethinking generalization. CoRR, abs/1611.03530 (2016). <http://arxiv.org/abs/1611.03530>

## **B** Assessment of Neural Networks for Stream-Water-Temperature Prediction

©2021 IEEE. Reprinted, with permission, from Stefanie Mohr, Konstantina Drainas and Jürgen Geist. “Assessment of Neural Networks for Stream-Water-Temperature Prediction”. IEEE, Conference Proceedings: 20th IEEE International Conference on Machine Learning and Applications (ICMLA), 2021.

This paper has been published as a **peer-reviewed conference paper**.

Stefanie Mohr, Konstantina Drainas and Jürgen Geist. Assessment of Neural Networks for Stream-Water-Temperature Prediction. In: 20th IEEE International Conference on Machine Learning and Applications (ICMLA), Pasadena, CA, USA, 2021, pp. 891-896.

DOI: <https://doi.org/10.1109/ICMLA52953.2021.00147>

### **Summary**

In this work, we explore the application of NNs for predicting the temperature of streams, an area of growing importance due to climate change. We highlight the inadequacies of traditional metrics like Root Mean Squared Error (RMSE) for evaluating the performance of NNs, and introduce several new methods for a comprehensive analysis: robustness analysis, which evaluates how small changes affect the output; min-max analysis, which determines the potential output range of the NN; and impact analysis, which evaluates the influence of the input features on the model predictions. These methods aim to improve the reliability and understandability of non-expert users of NNs. We validate these methods on six German streams, demonstrating that NNs can provide accurate predictions of water temperature.

### **Contributions of the author**

Composition and revision of the manuscript with significant role in writing Sections 3 and 4. Discussion and revision of the entire manuscript and the results. Discussion and development of the ideas, experimentation and evaluation with the following notable individual contributions: Development of the analysis methods, implementation of the analysis, and evaluation.

# Assessment of Neural Networks for Stream-Water-Temperature Prediction

Stefanie Mohr

*Chair of Theoretical Computer Science  
Technical University of Munich  
Munich, Germany  
0000-0002-8630-3218*

Konstantina Drainas

*Chair of Aquatic Systems Biology  
Technical University of Munich  
Munich, Germany  
0000-0001-6771-2123*

Prof. Dr. Jürgen Geist

*Chair of Aquatic Systems Biology  
Technical University of Munich  
Munich, Germany  
0000-0001-7698-3443*

**Abstract**—Climate change results in altered air and water temperatures. Increases affect physicochemical properties, such as oxygen concentration, and can shift species distribution and survival, with consequences for ecosystem functioning and services. These ecosystem services have integral value for humankind and are forecasted to alter under climate warming. A mechanistic understanding of the drivers and magnitude of expected changes is essential in identifying system resilience and mitigation measures. In this work, we present a selection of state-of-the-art Neural Networks (NN) for the prediction of water temperatures in six streams in Germany. We show that the use of methods that compare observed and predicted values, exemplified with the Root Mean Square Error (RMSE), is not sufficient for their assessment. Hence we introduce additional analysis methods for our models to complement the state-of-the-art metrics. These analyses evaluate the NN's robustness, possible maximal and minimal values, and the impact of single input parameters on the output. We thus contribute to understanding the processes within the NN and help applicants choose architectures and input parameters for reliable water temperature prediction models.

**Index Terms**—neural network, prediction, water temperature, climate change, verification, evaluation, robustness

## I. INTRODUCTION

Recent years have already shown the impact of climate change on various organisms, among them keystone species from aquatic ecosystems, such as macroinvertebrates. These animals, which by convention can be retained by a 500  $\mu\text{m}$  mesh net and lack a backbone, are often overlooked in the public discourse, even though they are abundant in nearly all types of streams and rivers all over the world [1] and have great relevance for the functioning of stream ecosystems, since they play a very important role in energy flow and hence also for higher levels of the food chain [2]. However, certain macroinvertebrate species and many fish species have stringent cold-water temperature requirements, which makes them particularly vulnerable to warmer water conditions. Air temperature has a strong influence on the temperature of rivers, particularly in broad and flat headwater streams that show tight coupling to atmospheric processes due to the high ratio of stream surface to water depth [3]. Consequently, increases in air temperature will also affect water temperature, leading to reduced cold water patches as possible habitats [4]. In Germany, an increase by 1.5°C in the annual mean

temperature between 1881 and 2018 has already been detected [5]. Similarly, an increase in mountain-lake water temperature between 0.1°C and 1.1°C per decade has been observed [6]. For ecological stakeholders, the knowledge of whether and how much temperature will change is highly important for the introduction of preventive policies, e.g. investing in shady riparian vegetation to reduce stream water temperature [7].

To be able to plan and execute such preventive policies, good and reliable models for water temperature prediction are essential. Their creation has two main advantages: Firstly, it will help to predict future changes in temperature and thus helps in choosing reasonable prevention and mitigation methods. Secondly, based on a few years of measurements, water temperature in streams can be predicted instead of measured. Even though measurements are more precise, predictions would not only reduce costs and effort but also establish possibilities for researchers as well as for ecological stakeholders all over the world to work with data that up to now have been hard to obtain.

Different approaches for water temperature prediction exist and have already been used for various aquatic systems. Linear regression, for example, is often used to describe the relationship between air and water temperature [8], [9], [10], [11], [12]. However, when plotting air against water temperature, physical effects concerning high and low temperatures lead to a non-linear, s-shaped relationship between the two parameters [13]. Additionally, atmospheric conditions, topography, stream discharge, bedform, and riparian vegetation play a relevant role as an influence on water temperature [8], [14]. Hence, linear regression based on air temperature can not be considered optimal for water temperature prediction. Addressing the non-linear relationship, Neural Networks (NN) seem very promising for reliable water temperature predictions.

We use the knowledge gained in hitherto studies to create NN for water temperature prediction and introduce analysis methods besides the commonly used metrics. Since these metrics exclusively compare observed and predicted values, we do not rely on them but complement them by our additional analysis methods, to assess and hence choose proper NN for water temperature prediction.

As much as NN are known to be effective in learning, they are also known to be vulnerable to perturbations [15], [16].

In our setting, a small change in the output is not necessarily a problem. However, users of river temperature models, such as hydrologists or employees in water management offices, may not be aware that there is a difference between a NN-model and a classical model, e.g. linear regression. That is why we need a more thorough analysis of the NN to evaluate its behavior more critically. For this application, we introduce three methods: *Robustness*-, *Min/Max*- and *Impact-Analysis*.

The *Robustness-Analysis* mimics the approach that is already known for classification [17], [18]. It evaluates the impact of a small perturbation on the output value. In a linear regression model, a perturbation of the input would be linearly transformed to the output. For NN, this is not as simple because their behavior is not linear. That is why the calculation of a robustness-value is a problem that has gained a lot of interest in the last few years [15], [16], [17], [18]. We adopt an approach from [19] that evaluates a NN in terms of its robustness.

Furthermore, the *Min/Max-Analysis* is meant to determine minimal and maximal values that the NN can take. For classical models, this value is almost obvious or at least can be evaluated by simple calculus. NN make this more difficult due to their complicated structure. However, we adopt the idea of [20] that generates specific input vectors to the NN that fulfill any pre-defined property. Even though the application was different, we can use a slightly adapted approach here.

The purpose of the *Impact-Analysis* is to help future developers of prediction models. The target groups who will want to use our approach may not be computer scientists themselves, but hydrologists or biologists. Thus, we need a method that visualizes clearly which input values were most important for the decision-making of the NN. On the one hand, this can help in adapting the model, on the other hand, it may also result in insights into the correlation between stream temperatures and other features. Therefore, we use a similar approach to [21] to determine which input features contribute most to calculating the output value.

We aim to introduce assessment methods for non-experts in the field of NN. Therefore, we not only implement methods with understandable output but also use easily accessible data, to not limit our approach to specific measurement requirements. Our data were collected at different streams in Germany, which were selected as described in Section III-A. Currently, we have an individual model for each stream trained on data from one measurement site each. This means that the input data was not gathered along the whole length of the streams but at one certain measurement site per stream.

We summarize our contributions as follows:

- We create NN for six different streams in Germany
- We show that metrics like the RMSE are not sufficient to assess NN for water temperature prediction properly
- We introduce thorough analysis methods for regression problems in NN
  - to compare it to classical approaches
  - to be able to choose robust architectures and input combinations for reliable predictions

- to make it more understandable to the user

## II. STATE OF THE ART

### A. Water temperature prediction

As changes in water temperature have been identified as a key component of aquatic ecosystem health, their accurate prediction becomes increasingly important. Hence, several approaches for water temperature prediction already exist. The simplest approach is linear regression, presuming a linear relationship between air and water temperature [8], [9], [10], [11], [12]. Another approach is stochastic modeling, e.g., multiple regression analysis [8], [22], [23], second-order Markov processes [22], Box and Jenkins time-series models [22], [23], and second-order autoregressive models [23]. Machine learning approaches, including Gaussian process regression, decision trees, and NN, have been tested, too [24]. In comparison, NN have shown either well [24] or even best [11], [25] performance and are becoming increasingly popular [11], [24], [25], [26], [27].

The studies using NN for water temperature prediction assessed their models based on bias, Coefficient of Determination [25], Coefficient of Correlation, Coefficient of Efficiency, Adjusted Coefficient of Efficiency [11], Mean Absolute Error, Willmott Index of Agreement [24], Mean Square Error [26], and the Root Mean Square Error (RMSE) [11], [24], [25]. Even though there is a wide variety in these metrics, they all mainly compare observed and predicted values, not considering the underlying processes and the reliability of the NN's predictions.

Also, these metrics do not consider input parameters and their impact on the output, besides changes in prediction accuracy. Nonetheless, several approaches have tried to improve water temperature prediction by additional input parameters that are supposed to complement air temperature from current and previous days as input parameters: runoff/relative change in flow [23], [24], which is the part of the effective precipitation that flows into the stream [28], declination of sun [26], soil temperature [29], riparian vegetation [29] and day of the year [24]. The most promising of those parameters is the day of the year because it improves the performance and does not require additional effort in data collection. Another parameter, the runoff, was found to usually play a relatively small role in water temperature prediction but shows to be increasingly important for high-altitude catchments [24].

To the best of our knowledge, the current best RMSE values vary between 0.46°C [24] and 1.58°C [27] in NN approaches over different streams, NN architectures, and input parameters.

### B. Explainable AI

While all the metrics for evaluating classical regression models, like mentioned above, determine the difference between prediction and observation, NN need more thorough analysis. It is an open problem to understand and interpret their exact behavior, which is why they go by the name of *black-box* models. Among others, some techniques visualize patterns in input images that trigger the NN [30], evaluate their

sensitivity to certain inputs [21], calculate Integrated Gradients [31], or perform layer-wise relevance propagation [32] which is specifically interesting for inputs that are images. A broad overview of many possible techniques can be found in [33]. Almost all of the recent works focus on classification networks whose interpretability is hard to determine. However, for regression networks, this problem is different. In our case, there is just one continuous output value that has to be observed and not categorical labels. That is why we use the gradients of the input values, similar to [31], as a method to visualize the behavior of the NN.

### C. Verification

NN are naturally very susceptible to adversarial attacks, as many works have demonstrated in recent years [15], [20]. Consequently, various verification techniques for NN are being developed these days. Most of these focus on proving the robustness of the NN [18], [19]. Local robustness is usually defined for classification networks: On a small area around each input to the NN, it should still predict the same label as for the original. On regression networks, this property cannot be defined in the same way. We cannot expect the NN to predict the same value for a slightly perturbed input. We still want to bind this prediction error in a small region around the inputs. Unfortunately, many tools for verification only support classification networks. That is why we decided to use a variant of *DeepPoly* [19]. This tool uses an abstraction of all possible input values, namely a particular neighborhood, and propagates this through the network. In the end, it results in an interval of possible output values of the NN.

## III. OUR APPROACH

### A. Data

We used the following values as possible input features for the NN:

- *air temperature* [ $^{\circ}\text{C}$ ]: daily mean values, measured in the distance of 3.25 to 47.03 km from water temperature measurement sites
- *runoff* [ $\text{m}^3/\text{s}$ ]: the part of the effective precipitation that flows into the streams and rivers, measured as the amount of water per unit time at each of the selected measurement sites
- *day of the year*: the date represented by a value in the interval of 0 to 365

The daily mean air temperature [ $^{\circ}\text{C}$ ] is provided by the German Meteorological Center (Deutscher Wetterdienst, abbr. DWD) [34]. The daily mean water temperature [ $^{\circ}\text{C}$ ] and runoff [ $\text{m}^3/\text{s}$ ] are provided by the "Gewässerkundlicher Dienst Bayern" (abbr. GkD) [35], a department of the Bavarian Environmental Agency. The data are free of cost and accessible via download [34], [35]. For our experiments, we use the data from six German streams selected by the mean annual runoff  $\overline{MQ} \leq 1 \text{ m}^3/\text{s}$  and a minimum of 1500 data points. For each stream, there is one measurement site for water temperature and runoff by the GkD and one to four surrounding measurement sites for air temperature by the DWD. In

our case, runoff measurements are conducted by the GkD. Hence we can access the data easily. Still, especially thinking about future predictions, it might not be recommendable to include runoff as it is another value that has to be measured or predicted. However, the importance of runoff in water temperature prediction models has already been shown. It seems to increase the performance for high-altitude catchments [24], which one should keep in mind while considering it as an input parameter. Since not all measurement devices were set up at the same time, measurement periods vary between six and 24 years, depending on the measurement site.

### B. Models

The data structure is quite simple, which is why using complex NN is unnecessary. In the course of this work, we use fully connected NN, which have at most three hidden layers and 90 hidden neurons in total. Several previous approaches showed satisfactory performance, even though they used far simpler models than NN. That is why we can use smaller models with a modest number of layers and neurons. For the training of the NN, we used a particular subset of input features, as described in Section III-A.

### C. Model Analysis

Regression models, not only NN, are usually evaluated by metrics that compare measured values with predicted values (see Section II-A). In this work, we use the RMSE (as defined in [11]) as a representative for these metrics because it is a usual and intuitive measure.

For simple models, e.g., linear regression or simple stochastic models, the behavior is well-defined and well-known and therefore does not require thorough analysis. The behavior of NN, on the other hand, is not well-known yet. To address this issue, we introduce three additional analyzing methods. These are supposed to help to understand the behavior of the NN.

1) *Robustness Analysis*: Local robustness, as defined in [36], is usually concerned with classification networks and the prediction of labels. Since our approach is based on a regression problem, we do not check whether labels are identical but the following: Given an input and its neighborhood, how much does the value of the prediction of the NN change at most in this area? More formally: If we have a NN  $f : X \rightarrow \mathbb{R}$  that is defined on its input set  $X$  and predicts values in  $\mathbb{R}$ , and an  $\epsilon$  defining the size of the neighborhood, we want to get a  $\delta$  such that for a subset of inputs of interest  $D$ :

$$\forall x \in D \subset X \quad \forall y \in N_{\epsilon}(x) : |f(y) - f(x)| < \delta \quad (1)$$

where  $N_{\epsilon}$  defines the  $\epsilon$ -neighborhood.

If the robustness is low, even a small perturbation to the input values can result in a big difference in the output values. The tool that we are using to evaluate the robustness of our NN is called *DeepPoly* [19]. It was originally designed to check the local robustness of classification networks but we adapted it to fit our application.

2) *MinMaxAnalysis*: On a simple model, such as a fitted sine function, its minimum and maximum values are obvious and can easily be derived by calculus. However, this is not as simple for NN. For them, it is neither obvious nor an easy calculation. We adapt the idea from [20] and use gradient descent. The principle is similar to backpropagation for learning a NN. Instead of learning optimal weights to minimize the loss function, we optimize the input vector to minimize resp. maximize the output value, while keeping the values in a reasonable range. That is,  $-45^{\circ}\text{C}$  to  $60^{\circ}\text{C}$  for temperature, and  $-1$  to  $40\text{ m}^3/\text{s}$  for the runoff. This yields input vectors with either very high or very low output values. Since the starting position of gradient descent can heavily influence its result, we start the optimization process from a set of randomly chosen input vectors, and we look at the minimum resp. maximum observed output value.

3) *Impact Analysis*: What we call *impact analysis* is similar to *sensitivity analysis*, which is used to determine which input the NN is sensitive on [21]. The key idea is to measure how much an input feature contributes to the calculation of the output value. This can be done by calculating the gradient of the input features based on the loss function. This value indicates how much a change in the input feature will affect the computation of the output value. On the one hand, it gives an idea of which features are necessary, and on the other hand, it helps in inspecting the differences between the streams. If the influence of the parameters on the streams is different, this could indicate a relationship between the streams that is unknown so far.

#### IV. EXPERIMENTS

With the help of a vast grid search, we found that the use of the default hyper-parameters is suitable for accurate water temperature prediction in all cases. To be able to use *DeepPoly*, we chose ReLU as an activation function. For our fully connected NN, we determined three hidden layers to return consistently good results based on the grid search. Based on this selection, we compare combinations of several input parameters with different numbers of nodes per layer, using normalized input data. The dataset is split into test-, train- and validation data; resp. 25%, 7.5% and 67.5% of the whole data. The optimizer for the training is L-BFGS that is run on a total of 10000 epochs. Details on the dataset and training can be found in [37].

In Section IV-A, we fix the architecture to 9-7-13 nodes, since this performs well according to robustness in Section IV-B. Also, we fix input parameters in Section IV-B to those showing the lowest RMSE values in Section IV-A.

##### A. Input feature selection

In Table I we show a subset of our results, with the combinations of our input values as follows: only air temperature (A), air temperature and runoff (A+R), air temperature and day of the year (A+D), and the combination of all three of them (A+R+D). Additionally, we present a comparison between only having one weather station for the prediction, and as

TABLE I  
RMSE VALUES FOR NN WITH DIFFERENT INPUT COMBINATIONS.  
A: AIR TEMPERATURE. R: RUNOFF. D: DAY OF THE YEAR.

Stream	A	A+R	A+D	A+R+D
<b>One airstation</b>				
Aubach	0.84	0.84	0.49	<b>0.48</b>
Abens	0.88	0.69	0.72	<b>0.55</b>
Bernauer Ache	1.02	0.91	0.83	<b>0.70</b>
Grosse Ohe	1.10	0.96	0.68	<b>0.59</b>
Otterbach	1.74	1.72	1.57	<b>1.54</b>
Sulzbach	1.09	1.06	0.92	<b>0.91</b>
<b>All airstations</b>				
Aubach	0.79	0.79	0.48	<b>0.47</b>
Abens	0.87	0.68	0.67	<b>0.52</b>
Bernauer Ache	1.00	0.92	0.83	<b>0.67</b>
Grosse Ohe	0.97	0.51	0.80	<b>0.48</b>
Otterbach	1.70	1.68	1.57	<b>1.57</b>
Sulzbach	1.07	1.02	0.91	<b>0.89</b>

many as there are in the direct neighborhood of the stream. We can observe that the obtained RMSE values are already similar to those determined in [24], if we use only air temperature as input. For most streams, it decreases if we add runoff as an input parameter. The RMSE decreases even more if we combine air temperature and day of the year as an input. Moreover, the best RMSE values are obtained if the combination of all three input parameters A+R+D is used. On the other hand, using all weather stations decreases the RMSE in only three out of six streams for the input combinations A+D and A+R+D. For the combination A+R it decreases the RMSE in already five out of six streams and for only air temperature as input, even all six streams show decreased RMSE values.

In summary, we can observe that the best RMSE values are obtained if all available input parameters are used. However, we obtain comparable results with the A+D input combination for several streams, confirming that the runoff does not necessarily have a big impact. Moreover, even among the best RMSE values obtained with the A+R+D input combination, we find a high variety in prediction accuracy between the different streams, ranging between  $0.47^{\circ}\text{C}$  and  $1.56^{\circ}\text{C}$ . We conjecture that there are stream-dependent parameters that we do not use in this work and that should be investigated further by hydrologists.

##### B. Robustness Analysis

While the robustness analysis is meant to be an evaluation method, we already use it to choose a good architecture by training three NN with different architectures for each stream, using the best input parameter combinations as identified in Section IV-A. We evaluate the different NN on the RMSE and the NN's robustness. To test the robustness, we add a perturbation of 0.01 to the air temperature and discharge which corresponds to approximately  $1^{\circ}\text{C}$  perturbation on the temperature and  $0.4\text{ m}^3/\text{s}$  on the discharge. The perturbation analysis generates the minimum and maximum possible outcome for the perturbed input, as displayed in Table II: On the one hand, bigger NN usually generate higher possible perturbations. On

TABLE II  
DIFFERENT NN ARCHITECTURES (SHAPE) FOR EACH STREAM WITH  
CORRESPONDING RMSE, AVERAGE DIFFERENCE AFTER PERTURBATION  
(MEANPERTURB) AND MINIMUM AND MAXIMUM VALUES THAT THE  
BEST NN CAN TAKE FOR EACH STREAM.

Stream	Shape	RMSE	MeanPerturb	Max	Min
Aubach	4-5-6	0.66	0.99		
Aubach	9-7-13	0.47	<b>0.7</b>	49.52	-1.96
Aubach	30-30-30	<b>0.44</b>	0.82		
Abens	4-5-6	0.63	<b>0.78</b>		
Abens	9-7-13	0.52	1.1	43.78	-19.70
Abens	30-30-30	<b>0.50</b>	1.7		
Bernauer Ache	4-5-6	0.83	<b>1.04</b>		
Bernauer Ache	9-7-13	0.67	1.19	42.38	-10.71
Bernauer Ache	30-30-30	<b>0.63</b>	1.62		
Grosse Ohe	4-5-6	0.53	1.18		
Grosse Ohe	9-7-13	0.48	<b>1.07</b>	50.14	-0.39
Grosse Ohe	30-30-30	<b>0.47</b>	1.18		
Otterbach	4-5-6	1.70	1.45		
Otterbach	9-7-13	<b>1.54</b>	<b>1.36</b>	48.82	-5.63
Otterbach	30-30-30	1.55	1.44		
Sulzbach	4-5-6	1.05	1.66		
Sulzbach	9-7-13	0.89	<b>1.29</b>	61.16	-0.36
Sulzbach	30-30-30	<b>0.81</b>	3.12		

the other hand, for the biggest NN (30-30-30), the RMSE is best in five out of six streams. This confirms that choosing NN architecture should not be based on the RMSE alone, since it is inconclusive in terms of NN robustness. As we aim to find a good balance between prediction accuracy and perturbation behavior, we use the medium-sized architecture (9-7-13) for further analysis.

#### C. MinMax-Analysis

When using unrealistic input values, e.g. temperatures of  $-45^{\circ}\text{C}$  and  $60^{\circ}\text{C}$  for each of the two weather stations for the last three days combined with very low discharge on January 1<sup>st</sup>, we can obtain very high and low values. For instance, for Sulzbach, this combination leads to a maximum value of  $61.16^{\circ}\text{C}$  (see Table II). These input value combinations are artificial and unrealistic. Still, we observe that all minimum and maximum values displayed in Table II can only be obtained with input values at the boundary of the allowed input values, in our case  $-45^{\circ}\text{C}$  and  $60^{\circ}\text{C}$  for air temperature, and  $-1$  to  $40 \text{ m}^3/\text{s}$  for runoff. To apply this approach in the hydrological context for guaranteeing that the NN does not predict unreasonable values, we have to define reasonable values for each relevant waterbody first. With this information, the MinMax-Analysis is a powerful tool to test the NN's reliability, especially in the context of prediction of future water temperatures.

#### D. Impact Analysis

For the impact analysis, we have chosen the model of Aubach as an example, since this model performed best in all categories. It is depicted in Figure 1. On the x-axis, the respective value's importance for the prediction is represented. On the y-axis, all input features of the NN are displayed. The

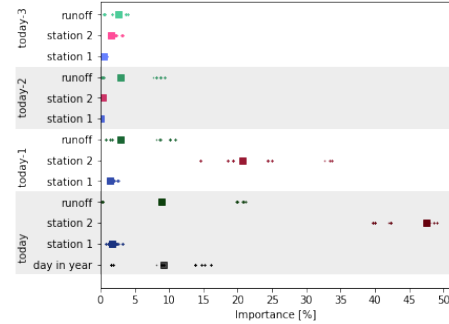


Fig. 1. Impact analysis for Aubach. The x-axis shows the importance for the prediction of each value on the y-axis. The dots represent all values that have occurred in the analysis, the squares mark the median of those. Stations 1 and 2 represent air temperatures, measured at the respective weather stations.

dots represent all of the important values that have occurred, the square indicates their median. Comparing with the feature selection, we can confirm that besides air temperature the day of the year has an impact on the prediction, as well as the runoff. Interestingly, the air temperature values of weather station two have a much higher impact on the result than the values of weather station one, confirming the only low decrease of RMSE between the use of one vs. the use of all stations as seen in Table I.

#### V. CONCLUSION AND FUTURE WORK

We show that precise water temperature predictions for German streams are possible with NN-based models. However, the precision of the resulting NN varies widely, which is an open point for further examination. Furthermore, the best achieved RMSE of 0.44 is still not as small as the usual measurement inaccuracy, which is at  $0.3^{\circ}\text{C}$ . A future goal is thus to improve the precision to a point where the model is just as good as a direct measurement or at least close enough. We confirm that the day of the year is an important factor, and so is the runoff. Additionally, we apply methods already known for classification problems in an adapted version to our regression problem. This improves the understanding of the behavior of NN for users that are not familiar with it, especially their limitations and their sensitivity to changes. With this, we contribute to more reliable water temperature predictions that will be applied to different climate change scenarios.

Besides the improvement of the training methods, we suggest finding additional easily accessible data on streams that might improve water temperature prediction from a hydrological perspective. One could make use of satellite images to include vegetation patterns as an additional input parameter. We also think it would be beneficial to use our methods of assessment, especially the impact analysis, and further examine whether and why certain input features are more important in some models, and less in others. Additionally, the heterogeneity between the different streams should be examined further, for

which our methods will contribute to an interdisciplinary step forward. Concerning the analysis of the NN, the possibilities are also not yet exhausted. There are various other verification tools, that can be used to check certain properties of the NN, e.g. [38]. Some tools try to evaluate NN-based systems which could be used, e.g. [39], to have a more precise evaluation of the minimal and maximal value of the NN.

#### ACKNOWLEDGMENT

This work was financially supported by the DFG Research Training Group on Continuous Verification of Cyber-Physical Systems (GRK 2428) and the AquaKlif project of the Bayklif network funded by the Bavarian State Ministry of Science and Arts (Bayerisches Staatsministerium für Wissenschaft und Kunst). We also want to thank Jan Křetínský, Romy Wild, and Lisa Kaule for their continuous support.

#### REFERENCES

- [1] F. R. Hauer and V. H. Resh, "Chapter 15 - Macroinvertebrates," in *Methods in stream ecology*, F. R. Hauer and G. A. Lamberti, Eds. London and San Diego and Cambridge, MA and Oxford: AP Academic Press an imprint of Elsevier, 2017, pp. 297–319.
- [2] J. B. Wallace and J. R. Webster, "The role of macroinvertebrates in stream ecosystem function," *Annual review of entomology*, vol. 41, pp. 115–139, 1996.
- [3] T. Gomi, R. C. Sidle, and J. S. Richardson, "Understanding processes and downstream linkages of headwater systems: headwaters differ from downstream reaches by their close coupling to hillslope processes, more temporal and spatial variation, and their need for different means of protection from land use," *BioScience*, vol. 52, no. 10, pp. 905–916, 2002.
- [4] J. Kuhn, R. Casas-Mulet, J. Pander, and J. Geist, "Assessing stream thermal heterogeneity and cold-water patches from uav-based imagery: A matter of classification methods and metrics," *Remote Sensing*, vol. 13, no. 7, p. 1379, 2021.
- [5] Federal Environment Agency, "Monitoringbericht 2019 zur Deutschen Anpassungsstrategie an den Klimawandel," 2019.
- [6] W. Kuefner, A. M. Hofmann, J. Geist, and U. Raeder, "Evaluating climate change impacts on mountain lakes by applying the new silicification value to paleolimnological samples," *Science of The Total Environment*, vol. 715, p. 136913, 2020.
- [7] H. Trimmel, P. Weihs, D. Leidinger, H. Formayer, G. Kalny, and A. Melcher, "Can riparian vegetation shade mitigate the expected rise in stream temperatures due to climate change during heat waves in a human-impacted pre-alpine river?" *Hydrology and Earth System Sciences*, vol. 22, no. 1, pp. 437–461, 2018.
- [8] D. Caissie, "The thermal regime of rivers: a review," *Freshwater Biology*, vol. 51, no. 8, pp. 1389–1406, 2006.
- [9] L. A. Krider, J. A. Magner, J. Perry, B. Vondracek, and L. C. Ferrington, "Air-Water Temperature Relationships in the Trout Streams of South-eastern Minnesota's Carbonate-Sandstone Landscape," *JAWRA Journal of the American Water Resources Association*, vol. 49, no. 4, pp. 896–907, 2013.
- [10] J. M. Pilgrim, X. Fang, and H. G. Stefan, "Stream Temperature Correlations with Air Temperatures in Minnesota: Implications for Climate Warming," *JAWRA Journal of the American Water Resources Association*, vol. 34, no. 5, pp. 1109–1121, 1998.
- [11] A. Rabi, M. Hadzima-Nyarko, and M. Šperac, "Modelling river temperature from air temperature: case of the River Drava (Croatia)," *Hydrological Sciences Journal*, vol. 60, no. 9, pp. 1490–1507, 2015.
- [12] K. Smith, "The prediction of river water temperatures / Prédiction des températures des eaux de rivière," *Hydrological Sciences Bulletin*, vol. 26, no. 1, pp. 19–32, 1981.
- [13] O. Mohseni and H. G. Stefan, "Stream temperature/air temperature relationship: a physical interpretation," *Journal of Hydrology*, vol. 218, no. 3–4, pp. 128–141, 1999.
- [14] R. L. Beschta, "Riparian shade and stream temperature: an alternative perspective," *Rangelands*, vol. 19, no. 2, pp. 25–28, 1997.
- [15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *EuroS&P*. IEEE, 2016.
- [16] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.
- [17] C. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *ATVA*, 2017.
- [18] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *ATVA*, 2017.
- [19] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *POPL*, vol. 3, 2019.
- [20] C. Szegedy, W. Zaremba, I. Sutskever, J. B. Estrach, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLR*, 2014.
- [21] J. M. Zurada, A. Malinowski, and I. Cloete, "Sensitivity analysis for minimization of input data dimension for feedforward neural network," in *Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94*, vol. 6. IEEE, 1994, pp. 447–450.
- [22] D. Caissie, N. El-Jabi, and A. St-Hilaire, "Stochastic modelling of water temperatures in a small stream using air to water relations," *Canadian Journal of Civil Engineering*, vol. 25, no. 2, pp. 250–260, 1998.
- [23] B. Ahmadi-Nedushan, A. St-Hilaire, T. B. M. J. Ouarda, L. Bilodeau, É. Robichaud, N. Thiémond, and B. Bobée, "Predicting river water temperatures using stochastic models: case study of the Moisie River (Québec, Canada)," *Hydrological Processes*, vol. 21, no. 1, pp. 21–34, 2007.
- [24] S. Zhu, E. K. Nyarko, M. Hadzima-Nyarko, S. Heddum, and S. Wu, "Assessing the performance of a suite of machine learning models for daily river water temperature prediction," *PeerJ*, vol. 7, p. e7065, 2019.
- [25] J.-F. Chenard and D. Caissie, "Stream temperature modelling using artificial neural networks: application on Catamaran Brook, New Brunswick, Canada," *Hydrological Processes*, vol. 22, no. 17, pp. 3361–3372, 2008.
- [26] A. P. Piotrowski, M. J. Napiorkowski, J. J. Napiorkowski, and M. Osuch, "Comparing various artificial neural network types for water temperature prediction in rivers," *Journal of Hydrology*, vol. 529, pp. 302–315, 2015.
- [27] M. Hadzima-Nyarko, A. Rabi, and M. Šperac, "Implementation of Artificial Neural Networks in Modeling the Water-Air Temperature Relationship of the River Drava," *Water Resources Management*, vol. 28, no. 5, pp. 1379–1394, 2014.
- [28] "Abfluss Bayern," 22.06.2021. [Online]. Available: <https://www.gkd.bayern.de/de/fluesse/abfluss>
- [29] A. St-Hilaire, G. Morin, N. El-Jabi, and D. Caissie, "Water temperature modelling in a small forested stream: implication of forest canopy and soil temperature," *Canadian Journal of Civil Engineering*, vol. 27, no. 6, pp. 1095–1108, 2000.
- [30] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *ICLR*, 2014.
- [31] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, 2017, pp. 3319–3328.
- [32] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS one*, vol. 10, no. 7, p. e0130140, 2015.
- [33] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.
- [34] Deutscher Wetterdienst, "Dwd climate data center (cdc): Historical daily station observations (temperature, pressure, precipitation, sunshine duration, etc.) for germany, (version v21.3, 2021)." [Online]. Available: [https://www.dwd.de/DE/klimaumwelt/cdc/cdc\\_node.html](https://www.dwd.de/DE/klimaumwelt/cdc/cdc_node.html)
- [35] Bayerisches Landesamt für Umwelt, [www.lfu.bayern.de](http://www.lfu.bayern.de), "GKD Bayern," 22.06.2021. [Online]. Available: <https://www.gkd.bayern.de/de/>
- [36] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *CAV (1)*, 2017.
- [37] S. Mohr, K. Drinias, and J. Geist, "Assessment of neural networks for stream-water-temperature prediction." [Online]. Available: <https://arxiv.org/abs/2110.04254>
- [38] G. Katz *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *CAV*, 2019, pp. 443–452.
- [39] M. Sotoudeh and A. V. Thakur, "Syrenn: A tool for analyzing deep neural networks," in *TACAS*, 2021.



## **C** Learning Explainable and Better Performing Representations of POMDP Strategies

*Licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.*

This paper has been published as a **peer-reviewed conference paper**.

Alexander Bork, Debraj Chakraborty, Kush Grover, Jan Křetínský, Stefanie Mohr (2024). Learning Explainable and Better Performing Representations of POMDP Strategies. In: Finkbeiner, B., Kovács, L. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2024. Lecture Notes in Computer Science, vol 14571, pp. 299-319. Springer, Cham.

DOI: [https://doi.org/10.1007/978-3-031-57249-4\\_15](https://doi.org/10.1007/978-3-031-57249-4_15)

### **Summary**

In this work, we develop a novel method for learning Finite State Controllers (FSCs) to represent strategies of Partially Observable Markov Decision Processes (POMDPs). Since strategies for POMDPs typically require memory, FSCs are a standard model to represent them. So far, however, there has been no focus on generating such a representation for any given strategy. While there is some work on directly generating FSCs [And+22], this approach does not scale well. Our approach is very efficient and can be used in addition to any other solution method for POMDPs.

We use a modification of the L\*-algorithm for automata learning. This is (a) efficient and highly scalable and (b) generates small and, therefore, explainable FSC representations. Furthermore, we present heuristics to improve the given strategy, which yields even better-performing strategies.

### **Contributions of the author**

Composition and revision of the manuscript with significant role in writing Section 3. Discussion and development of the ideas, implementation and evaluation with the following notable individual contributions: design and implementation of the learning process and integration into STORM and development of the heuristics.



# Learning Explainable and Better Performing Representations of POMDP Strategies <sup>★</sup>

Alexander Bork<sup>1</sup>, Debraj Chakraborty<sup>2</sup>, Kush Grover<sup>3</sup>, Jan Křetínský<sup>2,3</sup>, and Stefanie Mohr<sup>3</sup> (✉)

<sup>1</sup> RWTH Aachen University, Aachen, Germany

`alexander.bork@cs.rwth-aachen.de`

<sup>2</sup> Masaryk University, Brno, Czechia

`{chakraborty,kretinsky,jan.kretinsky}@fi.muni.cz`

<sup>3</sup> Technical University of Munich, Munich, Germany

`{kush.grover,stefanie.mohr}@tum.de`

**Abstract.** Strategies for partially observable Markov decision processes (POMDP) typically require memory. One way to represent this memory is via automata. We present a method to learn an automaton representation of a strategy using a modification of the  $L^*$ -algorithm. Compared to the tabular representation of a strategy, the resulting automaton is dramatically smaller and thus also more explainable. Moreover, in the learning process, our heuristics may even improve the strategy’s performance. We compare our approach to an existing approach that synthesizes an automaton directly from the POMDP, thereby solving it. Our experiments show that our approach can lead to significant improvements in the size and quality of the resulting strategy representations.

## 1 Introduction

**Partially Observable Markov Decision Processes (POMDPs)** combine non-determinism, probability and partial observability. Consequently, they have gained popularity in various applications as a model of planning in an unsafe and only partially observable environment. Coming from the machine learning community [30], they also gained interest in the formal methods community [25,11,14,22]. They are a very powerful model, able to faithfully capture real-life scenarios where we cannot assume perfect knowledge, which is often the case. Unfortunately, the great power comes with the hardness of analysis. Typical objectives of interest such as reachability or total reward already result in undecidable problems [25]. Namely, the resolution of the non-determinism (a.k.a. synthesis of a *strategy*, policy, scheduler, or controller) cannot be done algorithmically while guaranteeing optimality w.r.t. the objective. Consequently, *heuristics*

---

<sup>★</sup> This research was funded in part by the German Research Foundation (DFG) project 427755713 GPro, the MUNI Award in Science and Humanities (MUNI/I/1757/2021) of the Grant Agency of Masaryk University, the DFG GRK 2428 (ConVeY) and the DFG RTG 2236/2 (UnRAVeL).

to synthesize practically well-performing strategies became of significant interest. Let us name several aspects playing a key role in applicability of such synthesis procedures:

- ① *quality* of the synthesized strategies,
- ② *size* and *explainability* of the representation of the synthesized strategies,
- ③ *scalability* of the computation method.

**Strategy Representation.** While ① and ③ are of obvious importance, it is important to note the aspect ②. A strategy is a function mapping the current history (sequence of observations so far) to an action available in the current state. When written as a list of history-action pairs, it results in a large and incomprehensible table. In contrast, when equivalently written as a Mealy machine transducing the stream of observation to a stream of actions, its size may be dramatically lower (making it easier to implement and more efficient to execute) and its representation more explainable (making it easier to certify). Besides, better understandability allows for easier maintenance and modification. To put it in a contrast, explicit (table-like) or, e.g., neural-network representations of the function can hardly be hoped to be understandable by any human (even domain expert). Compact and understandable representations of strategies have recently gained attention, e.g., [12,27], also for POMDP [21,3], and even tool support [7] and [4], respectively. See [6] for detailed aspects of motivation for compact representations.

**Current Approaches** For POMDP, the state of the art is torn into two streams.

On the one hand, tools such as STORM [11] feature a classic *belief-based* analysis, which essentially blows up the state space, making it easier to analyze. Consequently, it is still reasonably scalable ③, but the size of the resulting strategy is even larger than that of the state space of the POMDP and is simply given as a table, i.e., not doing well w.r.t. the representation ②. Moreover, to achieve the scalability (and in fact even termination), the analysis has to be stopped at some places (“cut-offs”), resulting in poorer performance ①. On the other hand, the exhaustive bounded synthesis as in PAYNT [4] tries to synthesize a small Mealy machine representing a good strategy (while thus solving the POMDP) and if it fails, it tries again with an increased allowed size of the automaton. While this approach typically achieves better quality ① and, by principle, better size and explainability ②, it is extremely expensive and does not scale at all if the strategy requires a larger automaton ③. While symbiotic approaches are emerging [2], the best of both worlds has not been achieved yet.

**Our Contribution** We design a highly scalable postprocessing step, which improves the quality and the representation of the strategy. It is compatible with any framework producing any strategy representation, requiring only that we can query the strategy function (which action corresponds to a given observation sequence). In particular, STORM, which itself is scalable, can thus profit from

improving the quality and the representation of the produced strategies. Our procedure learns a compact representation of the given strategy as a Mealy machine using *automata-learning* techniques, in two different ways. First, through learning the complete strategy, we get its automaton representation, which is *fully equivalent* and thus achieving also the same value. Second, we provide *heuristics* learning small modifications of the strategy. Indeed, for some inputs (observation sequences), we ignore what the strategy suggests, in particular when the strategy is not defined, but also when it explicitly states that it is unsure about its choice (such as at the cut-off points, where the sequences become too long and the strategy was not optimised well at these later points). Whenever we ignore the strategy, we try to devise with a possibly better solution. For instance, we can adopt the decision that the currently learnt automaton suggests, or we can reflect other decisions in similar situations. This way we produce a *simpler strategy* (thus also comparatively smaller), which can, in principle, *fix the suboptimal decisions* of the strategy stemming from the limitations of the original analysis (such as bounds on the exploration) or any other irregularities. Of course, this only works well if the true optimal strategy is “sensible”, i.e., has inner structure allowing for a simple automaton representation. For practical, hence sensible, problems, this is typically the case.

*Summary of our contribution:*

- We provide a method to take any POMDP strategy and transform it into an equivalent or similar (upon choice) automaton, yielding **small** size and potential for **explainability**.
- Thereby we often improve the **quality** of the strategy.
- The experiments confirm the improvements and frequent proximity to best known values (typically of PAYNT) on the simpler benchmarks.
- The experiments indicate great **scalability** even on harder benchmarks where the comparison tool times out. The auspicious comparison on simpler benchmarks warrants the trust in good absolute quality and size on the harder ones.

**Related Work** Methods to solve planning problems on POMDPs have been studied extensively in the literature [34,18,32]. Many state-of-the-art solvers use point-based methods like *PBVI* [29], *Perseus* [35] and *SARSOP* [23] to treat bounded and unbounded discounted properties. For these methods, strategies are typically represented using so called  $\alpha$ -vectors. Apart from a significant overhead in the analysis, they completely lack of explainability. Notably, while the *SARSOP* implementation provides an export of its computed strategies in an automaton format, we have not been able to find an explanation of how it is generated.

Methods based on the (partial) exploration and solving of the belief MDP underlying the POMDP [28,10,11] have been implemented in the probabilistic model checkers *STORM* [20] and *PRISM* [24]. The focus of these methods is optimizing infinite-horizon objectives *without* discounting. Recent work [2] describes

how strategies are extracted from the results of these belief exploration methods. The resulting strategy representation, however, is rather large and contains potentially redundant information.

Orthogonal to the methods above, there are approaches that *directly* synthesize strategies from a space of candidates [16,26]. The synthesized strategy is then applied to the POMDP to yield a Markov chain. Analyzing this Markov chain yields the objective value achieved by the strategy. Methods used for searching policies include using inductive synthesis [3], gradient decent [19] or convex optimization [1,21,15]. [2] describes an integration of a belief exploration approach [11] with inductive synthesis [3].

Our approach is orthogonal to the solution methods in that it uses an *existing* strategy representation and learns a new, potentially more concise finite-state controller representation. Furthermore, our modifications of learned strategy representations shares similarities with approaches for strategy improvement [36,13,33].

## 2 Preliminaries

For a countable set  $S$ , we denote its power set by  $2^S$ . A (*discrete*) *probability distribution* on a countable set  $S$  is a function  $d : S \rightarrow [0, 1]$  such that  $\sum_{s \in S} d(s) = 1$ . We denote the set of all probability distributions on the set  $S$  as  $\text{Dist}(S)$ . For  $d \in \text{Dist}(S)$ , the *support* of  $d$  is  $\text{supp}(d) = \{s \in S \mid d(s) > 0\}$ . We use the Iverson bracket notation where  $[x] = 1$  if the expression  $x$  is true and 0 otherwise. For two sets  $S, T$ , we define the set of concatenations of  $S$  with  $T$  as  $S \cdot T = \{s \cdot t \mid s \in S, t \in T\}$ . We analogously define the set of  $n$ -times concatenation of  $S$  with itself as  $S^n$  for  $n \geq 1$  and  $S^0 = \{\epsilon\}$  is the set containing the empty string. We denote by  $S^* = \bigcup_{i=0}^{\infty} S^i$  the set of all *finite strings* over  $S$  and by  $S^+ = \bigcup_{i=1}^{\infty} S^i$  the set of all *non-empty* finite strings over  $S$ . For a finite string  $w = w_1 w_2 \dots w_n$ , the string  $w[0, i]$  with  $w[0, 0] = \epsilon$  and  $w[0, i] = w_1 \dots w_i$  for  $0 < i \leq n$  is a *prefix* of  $w$ . The string  $w[i, n] = w_i \dots w_n$  with  $0 < i \leq n$  is a *suffix* of  $w$ . A set  $W \subseteq S^*$  is *prefix-closed* if for all  $w \in S^*$ ,  $w = w_1 \dots w_n \in W$  implies  $w[0, i] \in W$  for all  $0 \leq i \leq n$ . A set  $W' \subseteq S^*$  is *suffix-closed* if  $\epsilon \notin W'$  and for all  $w \in S^*$ ,  $w = w_1 \dots w_n \in W'$  implies  $w[i, n] \in W'$  for all  $0 < i \leq n$ .

**Definition 1 (MDP).** A Markov decision process (MDP) is a tuple  $\mathcal{M} = (S, A, P, s_0)$  where  $S$  is a countable set of states,  $A$  is a finite set of actions,  $P : S \times A \rightarrow \text{Dist}(S)$  is a partial transition function, and  $s_0 \in S$  is the initial state.

For an MDP  $\mathcal{M} = (S, A, P, s_0)$ ,  $s \in S$  and  $a \in A$ , let  $\text{Post}^{\mathcal{M}}(s, a) = \{s' \mid P(s, a, s') > 0\}$  be the set of successor states of  $s$  in  $\mathcal{M}$  that can be reached by taking the action  $a$ . We also define the set of *enabled actions* in  $s \in S$  by  $A(s) = \{a \in A \mid P(s, a) \neq \perp\}$ . A *Markov chain* (MC) is an MDP with  $|A(s)| = 1$  for all  $s \in S$ . For an MDP  $\mathcal{M}$ , a *finite path*  $\rho = s_0 a_0 s_1 \dots s_i$  of length  $i \geq 0$  is a sequence of states and actions such that for all  $t \in [0, i - 1]$ ,  $a_t \in A(s_t)$  and  $s_{t+1} \in \text{Post}^{\mathcal{M}}(s_t, a_t)$ . Similarly, an infinite path is an infinite sequence  $\rho =$

$s_0 a_0 s_1 a_1 s_2 \dots$  such that for all  $t \in \mathbb{N}$ ,  $a_t \in A(s_t)$  and  $s_{t+1} \in \text{Post}^{\mathcal{M}}(s_t, a_t)$ . For an MDP  $\mathcal{M}$ , we denote the set of all finite paths by  $\text{FPaths}_{\mathcal{M}}$ , and of all infinite paths by  $\text{IPaths}_{\mathcal{M}}$ .

**Definition 2 (POMDP).** A partially observable MDP (POMDP) is a tuple  $\mathcal{P} = (\mathcal{M}, Z, \mathcal{O})$  where  $\mathcal{M} = (S, A, P, s_0)$  is the underlying MDP with finite number of states,  $Z$  is a finite set of observations, and  $\mathcal{O} : S \rightarrow Z$  is an observation function that maps each state to an observation.

For POMDPs, we require that states with the same observation have the same set of enabled actions, i.e.,  $\mathcal{O}(s) = \mathcal{O}(s')$  implies  $A(s) = A(s')$  for all  $s, s' \in S$ . This way, we can lift the notion of enabled actions to an observation  $z \in Z$  by setting  $A(z) = A(s)$  for some state  $s \in S$  with  $\mathcal{O}(s) = z$ . The notion of observation  $\mathcal{O}$  for states can be lifted to paths: for a path  $\rho = s_0 a_0 s_1 a_1 \dots$ , we define  $\mathcal{O}(\rho) = \mathcal{O}(s_0) a_0 \mathcal{O}(s_1) a_1 \dots$ . Two paths  $\rho_1$  and  $\rho_2$  are called *observation-equivalent* if  $\mathcal{O}(\rho_1) = \mathcal{O}(\rho_2)$ . We call an element  $\bar{o} \in Z^*$  an *observation sequence* and denote the observation sequence of a path  $\rho = s_0 a_0 s_1 \dots$  by  $\bar{\mathcal{O}}(\rho) = \mathcal{O}(s_0) \mathcal{O}(s_1) \dots$ .

b 0	y 1
b 2	g 3

Fig. 1: Running example: POMDP

*Example 1.* Consider the POMDP graphically depicted in Fig. 1, modeling a basic robot planning task. A robot is dropped uniformly at random in one of four grid cells. Its goal is to reach cell 3. The robot's sensors cannot distinguish cells 0 and 2, while cells 1 and 3 provide unique information. For the POMDP model, we use states 0, 1, 2, and 3 to indicate the robot's position. We mimic the random initialization by introducing a unique initial state  $s_0$  with a unique observation  $\mathbf{i}$  (init).  $s_0$  has a single action

that reaches any of the other four states with equal probability 0.25. Thus, the state space of the POMDP is  $S = \{s_0, 0, 1, 2, 3\}$ . To represent the observations of the robot, we use three observations  $\mathbf{b}$ ,  $\mathbf{y}$  and  $\mathbf{g}$ , so  $Z = \{\mathbf{i}, \mathbf{b}, \mathbf{y}, \mathbf{g}\}$ . States 0 and 2 have the same observation, while states 1 and 3 are uniquely identifiable, formally  $\mathcal{O} = \{(s_0 \rightarrow \mathbf{i}), (0 \rightarrow \mathbf{b}), (1 \rightarrow \mathbf{y}), (2 \rightarrow \mathbf{b}), (3 \rightarrow \mathbf{g})\}$ . The goal is for the robot to reach state 3. In each state, it can choose to move up, down, left, or right,  $A = \{s, u, d, l, r\}$ . In each step, executing the chosen action may fail with a probability of  $p = 0.5$ , causing the robot to remain in its current cell without changing states.

**Definition 3 (Strategy).** A strategy for an MDP  $\mathcal{M}$  is a function  $\pi : \text{FPaths}_{\mathcal{M}} \rightarrow \text{Dist}(A)$  such that for all paths  $\rho \in \text{FPaths}_{\mathcal{M}}$ ,  $\text{supp}(\pi(\rho)) \subseteq A(\text{last}(\rho))$ .

A strategy  $\pi$  is *deterministic* if  $|\text{supp}(\pi(\rho))| = 1$  for all paths  $\rho \in \text{FPaths}_{\mathcal{M}}$ . Otherwise, it is *randomized*. A strategy  $\pi$  is called *memoryless* if it depends only on  $\text{last}(\rho)$  i.e. for any two paths  $\rho_1, \rho_2 \in \text{FPaths}_{\mathcal{M}}$ , if  $\text{last}(\rho_1) = \text{last}(\rho_2)$  then  $\pi(\rho_1) = \pi(\rho_2)$ . As general strategies have access to *full* state information, they are unsuitable for partially observable domains. Therefore, POMDPs require a notion of strategies based only on observations. For a POMDP  $\mathcal{P}$ , we call a

strategy *observation-based* if for any  $\rho_1, \rho_2 \in \text{FPaths}_{\mathcal{M}}$ ,  $\mathcal{O}(\rho_1) = \mathcal{O}(\rho_2)$  implies  $\pi(\rho_1) = \pi(\rho_2)$ , i.e. the strategy has same output on observation-equivalent paths.

We are interested in representing observation-based strategies approximating optimal objective values for infinite horizon objectives without discounting, also called *indefinite-horizon objectives*, i.e., maximum/minimum reachability probabilities and expected total reward objectives. We emphasize that our general learning framework also generalizes straightforwardly to strategies for different objectives. In contrast to fully observable MDPs, deciding if a given strategy is optimal for an indefinite-horizon objective on a POMDP is generally undecidable [25]. In fact, optimal behavior requires access to the *full* history of observations, necessitating an arbitrary amount of memory. As such, our goal is to learn a small representation of a strategy using only a finite amount of memory that approximates optimal values as well as possible.

We represent these strategies as *finite-state controllers (FSCs)* – automata that compactly encode strategies with access to memory and randomization in a POMDP.

**Definition 4 (Finite-State Controller).** A finite-state controller (FSC) is a tuple  $\mathcal{F} = (N, \gamma, \delta, n_0)$  where  $N$  is a finite set of nodes,  $\gamma : N \times Z \rightarrow \text{Dist}(A)$  is an action mapping,  $\delta : N \times Z \rightarrow N$  is the transition function, and  $n_0$  is the initial node.

We denote by  $\pi_{\mathcal{F}}$  the strategy represented by the FSC  $\mathcal{F}$  and use  $\mathfrak{F}$  for the set of all FSCs for a POMDP  $\mathcal{P}$ . Given an FSC  $\mathcal{F} = (N, \gamma, \delta, n_0)$  that is currently in node  $n$ , and a POMDP  $\mathcal{P}$  with underlying MDP  $\mathcal{M} = (S, A, P, s_0)$ , in state  $s$ , the action to play by an agent following  $\mathcal{F}$  is chosen randomly from the distribution  $\gamma(n, \mathcal{O}(s))$ .  $\mathcal{F}$  then updates its current node to  $n' = \delta(n, z)$ . The state of the POMDP is updated according to  $P$ . As such, an FSC induces a Markov chain  $\mathcal{M}_{\mathcal{F}} = (S \times N, \{\alpha\}, P^{\mathcal{F}}, (s_0, n_0))$  where  $P^{\mathcal{F}}((s, n), \alpha, (s', n'))$  is  $[\delta(n, \mathcal{O}(s)) = n'] \cdot \sum_{a \in A(s)} \gamma(n, \mathcal{O}(s))(a) \cdot P(s, a, s')$ .

An FSC can be interpreted as a Mealy machine: nodes correspond directly to states of the Mealy machine, which takes observations as input. The set of output symbols is the set of all distributions over actions occurring in the FSC.

### 3 Learning a Finite-State Controller

We present a framework for learning a concise finite-state controller representation from a given strategy for a POMDP. Our approach mimics an extension of the  $L^*$  automaton learning approach [5] for learning Mealy machines [31]. The main difference in our approach is that we have a sparse learning space: not all observations of a POMDP are possible to reach from all states. Thus, there are many observation sequences that can never occur in the POMDP. To mark situations where this occurs, i.e. where a learned FSC has complete freedom to decide what to do, we introduce a “don’t-care” symbol  $\dagger$ .

Furthermore, for some policy computation methods, the strategy we receive as input may be incomplete. Although some observation sequence can appear

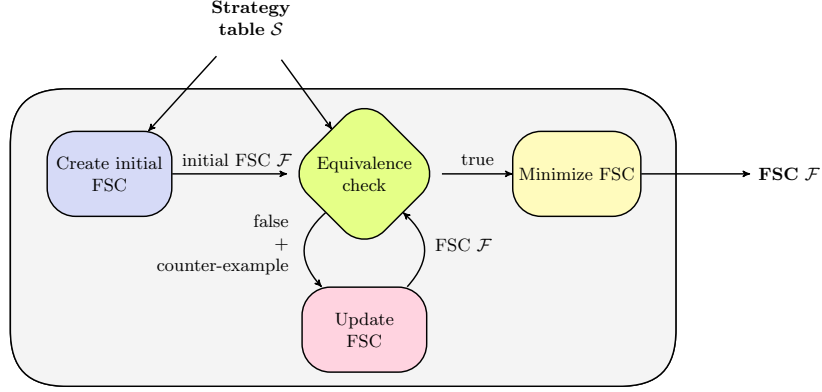


Fig. 2: Depiction of the FSC learning framework

in the POMDP, the strategy does not specify what to do when it occurs. This can for example be caused by reaching the depth limit in an exploration based approach. We use a “don’t-know” symbol  $\chi$  to mark such cases. While the non-occurring sequences do not directly influence the learning process, they cannot be ignored completely. These  $\chi$  need to be replaced by actual actions using some heuristics for the final FSC to yield a complete strategy (see Section 3.4).

An overview of the learning process is depicted in Fig. 2. We expect as input a (partially defined) strategy in the form of a table that maps observation sequences in the POMDP to a distribution over actions.

**Definition 5 (Strategy Table).** A strategy table  $\mathcal{S}$  for a POMDP  $\mathcal{P}$  is a relation  $\mathcal{S} \subseteq Z^* \times (\text{Dist}(A) \cup \{\chi\})$ . A row of  $\mathcal{S}$  is an element  $(\bar{o}, d) \in \mathcal{S}$ .

For  $(\bar{o}, d) \in \mathcal{S}$ , if  $\text{supp}(d)$  contains only a single action  $a$ , we write it as  $(\bar{o}, a)$ . We say a strategy table  $\mathcal{S}$  is *consistent* if and only if for  $\bar{o} \in Z^*$ ,  $(\bar{o}, d_1) \in \mathcal{S}$  and  $(\bar{o}, d_2) \in \mathcal{S}$  implies  $d_1 = d_2$ , i.e. each observation sequence has at most one unique output. A consistent strategy table  $\mathcal{S}$  (partially) defines an observation-based strategy  $\pi_{\mathcal{S}}$  with  $\pi_{\mathcal{S}}(\rho) = d$  if and only if  $(\bar{\mathcal{O}}(\rho), d) \in \mathcal{S}$  and  $d \neq \chi$ . For consistent strategy tables, the FSC resulting from our approach correctly represents the partially defined strategy.

*Example 2.* Table 1 depicts a strategy table for the POMDP described in Example 1. The table does not specify what to do in state 3 as at that point, the robot has already achieved its target. The action chosen at that point is irrelevant. Intuitively, the strategy table describes that the robot should go right as long as it sees **b**, and goes down once it sees **y**. The FSC in Figure 3 fully captures the behaviour described by the strategy table and thus accurately represents it.

In our framework, the input strategy table is used to build an initial FSC which is then compared to the input. If the initial FSC is already equivalent to the given strategy table, we are done and we output the FSC. Otherwise, we get a counterexample and use it to update the FSC. This process of checking for



Observation sequence	Action
i	s
i y	d
i b	r

Table 1: Example strategy table for the POMDP in Example 1. It only contains observation sequences of length at most 2.

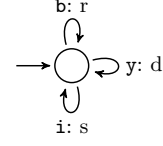


Fig. 3: FSC representing the strategy table of Table 1.

equivalence and updating the FSC is repeated until the FSC is equivalent to the table.

In the sequel, we first explain how our learning approach works on general input of the form described above. Then we show how the learning approach is integrated with an existing POMDP solution method by means of the belief exploration framework from [11]. Lastly, we introduce heuristics for improvement of the learned policies when the information in the table is incomplete.

### 3.1 Automaton Learning

The regular  $L^*$  approach is used to learn a DFA for a regular language. It is intuitively described as: a *teacher* has information about an automaton and a *student* wants to learn that automaton. The student asks the teacher whether specific words are part of the language (*membership query*). At some point, the student proposes a solution candidate (in case of  $L^*$ , a DFA) and asks the teacher whether it is correct, i.e. whether the proposed automaton accepts the language (*equivalence query*). Instead of the membership query of standard  $L^*$ , the extension to Mealy machines [31] uses an *output query*, since we are not interested in the membership of a word in a language but rather the output of the Mealy machine corresponding to a specific word. As such, our learning approach needs access to an output query, specifying the output of the strategy table for a given observation sequence, and an equivalence query, checking whether an FSC accurately represents the strategy table. We formally define the two types of queries.

**Definition 6 (Output Query (OQ)).** *The output query for a strategy table  $\mathcal{S}$  is the function  $OQ_{\mathcal{S}} : Z^* \rightarrow \text{Dist}(A) \cup \{\chi, \dagger\}$  with  $OQ_{\mathcal{S}}(\bar{o}) = d$  if  $(\bar{o}, d) \in \mathcal{S}$  and  $OQ_{\mathcal{S}}(\bar{o}) = \dagger$  otherwise.*

**Definition 7 (Equivalence Query (EQ)).** *The equivalence query for a strategy table  $\mathcal{S}$  is a function  $EQ_{\mathcal{S}} : \mathfrak{F} \rightarrow Z^*$  defined as follows:  $EQ_{\mathcal{S}}(\mathcal{F}) = \epsilon$  if for all  $(\bar{o}, d) \in \mathcal{S}$  and for all  $\rho$  with  $\bar{\mathcal{O}}(\rho) = \bar{o}$ ,  $\pi_{\mathcal{F}}(\rho) = d$ . Otherwise,  $EQ_{\mathcal{S}}(\mathcal{F}) = c$  where  $c \in \{\bar{o} \mid (\bar{o}, d) \in \mathcal{S}, \exists \rho \in \text{FPaths}_{\mathcal{M}}(\mathcal{P}) : \bar{\mathcal{O}}(\rho) = \bar{o} \wedge \pi_{\mathcal{F}}(\rho) \neq d\}$  is a counterexample where  $\mathcal{S}$  and  $\mathcal{F}$  have different output.*

The output query (OQ) takes an observation sequence  $\bar{o}$ , and outputs the distribution (or the  $\chi$  symbol) suggested by the strategy table. If the given observation sequence is not present in the strategy table, it returns the  $\dagger$  symbol,

i.e., a "don't care"-symbol. The equivalence query (EQ) takes a hypothesis FSC  $\mathcal{F}_{hyp}$  and asks whether it accurately represents  $\mathcal{S}$ . In case it does not, an observation sequence where  $\mathcal{F}_{hyp}$  and  $\mathcal{S}$  differ is generated as a counterexample.

Using the definitions of these two queries, we formalise our problem statement as follows:

**Problem Statement:** Given a POMDP  $\mathcal{P}$ , a strategy table  $\mathcal{S}$ , an output query  $OQ_{\mathcal{S}}$  and an equivalence query  $EQ_{\mathcal{S}}$ , compute a small FSC  $\mathcal{F}$  such that  $EQ_{\mathcal{S}}(\mathcal{F}) = \epsilon$ .

**Learning Table** We aim at solving the problem using a learning framework similar to  $L^*$ . We learn an FSC by creating a *learning table* which keeps track of the observation sequences and the outputs the learner assumes they should yield in the strategy. Formally, it is defined as follows:

**Definition 8 (Learning Table).** A learning table for POMDP  $\mathcal{P}$  is a tuple  $\mathcal{T} = (\mathbf{R}, \mathbf{C}, \mathcal{E})$  where  $\mathbf{R} \subset Z^*$  is a prefix-closed finite set of finite strings over the observations representing the upper row indices, the set  $\mathbf{R} \cdot Z$  are the lower rows indices and  $\mathbf{C} \subset Z^+$  is a suffix-closed finite set of non-empty finite strings over  $Z$  – the columns.  $\mathcal{E} : (R \cup R \cdot Z) \times C \rightarrow \text{Dist}(A) \cup \{\chi, \dagger\}$  is a mapping that represents the entries of the table.

	i	b	y
$\epsilon$	s	†	†
i	†	r	d
b	†	†	†
y	†	†	†

Table 2:

Running example -  
initial table

Intuitively speaking, the table is divided into *upper* and *lower* rows. Initially, the *columns* of the learning table are the observations in the POMDP. Additional columns may be added in the learning process to further refine the behavior of the learned FSC. Upper rows effectively result in nodes of the learned FSC, while lower rows specify destinations of the transitions. For a row in the upper rows, each entry represents the output of the FSC corresponding to their respective observation (column). For an upper row, if a column is labelled only with an observation, the corresponding *entry* represents the output of the FSC on that observation. As an ex-

ample, Table 2 contains the initial learning table for our running example. We do not include observation  $\mathbf{g}$  for the target state as we are not interested in the behavior of the strategy after the target has been reached.

We say that two rows  $r_1, r_2 \in R \cup R \cdot Z$  are *equivalent* ( $r_1 \equiv r_2$ ) if they fully agree on their entries, i.e.,  $r_1 \equiv r_2$  if and only if  $\mathcal{E}(r_1, c) = \mathcal{E}(r_2, c)$  for all  $c \in C$ . The equivalence class of a row  $r \in R \cup R \cdot Z$  is  $[r] = \{r' \mid r \equiv r'\}$ .

**From Learning Table to FSC** To transform a learning table into an FSC, the table needs to be of a specific form. In particular, it needs to be *closed* and *consistent*. A learning table is *closed* if for each lower row  $l \in R \cdot Z$ , there is an upper row  $u \in R$  such that  $l \equiv u$ .

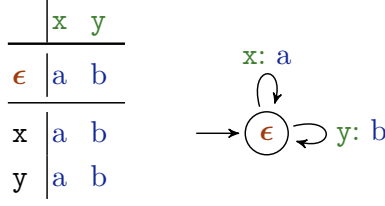


Fig. 4: Transformation of a learning table to an FSC.

A learning table is *consistent* if for each  $r_1, r_2 \in R$  such that  $r_1 \equiv r_2$ , we have  $r_1 \cdot e \equiv r_2 \cdot e$  for all  $e \in Z$ . Closure of a learning table guarantees that each transition – defined in the FSC by a lower row – leads to a valid node, i.e. the node corresponding to the equivalent upper row. Consistency, on the other hand, guarantees that the table unambiguously defines the output of a node in the FSC given an observation.

Using the notions of closure and consistency, we can define the transformation of a learning table into the learned FSC :

**Definition 9 (Learned FSC).** *Given a closed and consistent learning table  $\mathcal{T} = (R, C, \mathcal{E})$ , we obtain a learned FSC  $\mathcal{F}_{\mathcal{T}} = (N_{\mathcal{T}}, \gamma_{\mathcal{T}}, \delta_{\mathcal{T}}, n_{0, \mathcal{T}})$  where:*

*$N_{\mathcal{T}} = \{[r] \mid r \in R\}$ , i.e., the nodes are the upper rows of the table;  $\gamma_{\mathcal{T}}([r], o) = \mathcal{E}(r, o)$  for all  $o \in Z$ , i.e. the output of a transition is defined by its entry in the table;  $\delta_{\mathcal{T}}([r], o) = [r \cdot o]$  for all  $r \in R, o \in Z$ , i.e., the destination of a transition from node  $[r]$  with observation  $o$  is the node corresponding to the upper row equivalent to the lower row  $r \cdot o$ ;  $n_{0, \mathcal{T}} = [\epsilon]$ , i.e., the initial state is  $[\epsilon]$ .*

*Example 3.* We demonstrate how to transform a table to an FSC in Fig. 4. The upper rows become states, the lower rows show the transitions. In this example, on both the observations  $x, y$ , we stay in the state and play action  $a$  and  $b$ , respectively.

### 3.2 Algorithm

We present our algorithm for learning an FSC from a strategy table. We have already seen the abstract view of the approach in Fig. 2. Algorithm 1 contains the pseudo-code for our learning algorithm. It consists of four main parts, also pictured in Fig. 2: **initialization**, **equivalence check**, **update of the FSC**, **minimization**.

First, we initialise the learning table. The columns are initially filled with all available observations  $Z$ , i.e. we set  $C \leftarrow Z$ . We start with a single upper row  $\epsilon$ , representing the empty observation sequence. In the lower rows, we add the observation sequences of length 1. The entries of the table are then filled using output queries. For example, consider the strategy table in Table 1. The learning table after initialisation is shown in Table 2. The strategy table only contains observation sequences starting with  $i$ . Thus, for any sequence starting with  $b$  or  $y$ , all entries are  $\dagger$ .

After initialising the table, we check whether it is closed. If the table is not closed, all rows in the lower part of the table that do not occur in the upper part are moved to the upper part. Formally, we set  $R \leftarrow R \cup \{l\}$  for all  $l \in R \cdot Z$  with  $l \not\equiv u$  for all  $u \in R$ . In our example, this means that we move the rows  $(i \mid \dagger r d)$  and  $(b \mid \dagger \dagger \dagger)$  to the upper part of the table.

**Algorithm 1** Learning an FSC**Input:** POMDP  $\mathcal{P}$ , strategy table  $\mathcal{S}$ 


---

```

1:  $R \leftarrow \{\epsilon\}, C \leftarrow Z$ 
2: for all  $r \in R \cup R \cdot Z, e \in C$  do
3:    $\mathcal{E}(r, e) \leftarrow \text{OUTPUTQUERY}(r \cdot e)$ 
4: end for
5:  $\text{MAKECLOSEDANDCONSISTENT}(R, C, \mathcal{E})$ 
6:  $c \leftarrow \text{EQUIVALENCEQUERY}(\mathcal{S}, \mathcal{F}_{(R, C, \mathcal{E})})$ 
7: while  $c \neq \epsilon$  do
8:    $C \leftarrow C \cup \text{set of all prefixes of } c$ 
9:   for all  $r \in R \cup R \cdot Z, e \in C$  do
10:     $\mathcal{E}(r, e) \leftarrow \text{OUTPUTQUERY}(r \cdot e)$ 
11:   end for
12:    $\text{MAKECLOSEDANDCONSISTENT}(R, C, \mathcal{E})$ 
13:    $c \leftarrow \text{EQUIVALENCEQUERY}(\mathcal{S}, \mathcal{F}_{(R, C, \mathcal{E})})$ 
14: end while
15:  $\mathcal{T} \leftarrow (R, C, \mathcal{E}), \mathcal{T} \leftarrow \text{MINIMIZE}(\mathcal{T})$ 

```

---

**Output:** FSC  $\mathcal{F}_{\mathcal{T}}$  generated from  $\mathcal{S}$

---

Once the table is closed (and naturally consistent), we check for each row in the given strategy table  $\mathcal{S}$  whether it coincides with the action provided for this observation sequence by our hypothesis FSC  $\mathcal{F}_{hyp}$ . This is done formally using the equivalence query, i.e. we check if  $EQ_{\mathcal{S}}(\mathcal{F}_{hyp}) = \epsilon$ . If our hypothesis is not correct, we get a counterexample  $c \in Z^+$  where the output of  $\mathcal{S}$  and  $\mathcal{F}_{hyp}$  differ. We add all non-empty prefixes of  $c$  to  $C$  and fill the table. We repeat this until  $\mathcal{F}_{hyp}$  is equivalent to the strategy table  $\mathcal{S}$ .

After the equivalence has been established, we use the “don’t-care” entries  $\dagger$  to further minimise the FSC. These entries only appear for observation sequences that do not occur in the strategy table. Thus, changing them to any action does not change the FSC’s behaviour with respect to the strategy table. We use this fact to merge nodes of the FSC to obtain a smaller one that still captures the behaviour of the strategy table. It is not trivial to already exploit “don’t care” entries during the learning phase. Two upper rows that are compatible in terms of the outputs they suggest, i.e. they either agree or have a  $\dagger$  where the other suggests an output, might be split when a new counterexample is added. As such, we postpone minimisation of the FSC until the learning is finished.

### 3.3 Proof of Concept: Belief Exploration

For integrating our learning approach with an existing POMDP solution framework, we need to consider how the strategy table is constructed. Assume that the solution method outputs *some* representation of a strategy. For strategies that are equivalent to some FSC, one possibility is to pre-compute the strategy table. However, it is not clear how to determine the length of observation sequences that need to be considered. A more reasonable view is considering the strategy

representation as a *symbolic* representation of the strategy table as long as it permits computable output and equivalence queries.

We demonstrate how this works by considering the belief exploration framework of [11]. The idea of belief exploration is to explore (a fragment of) the *belief MDP* corresponding to the POMDP. Then, model checking techniques are used on this finite MDP to optimise objectives and find a strategy. States of the belief MDP are *beliefs* – distributions over states of the POMDP that describe the likelihood of being in a state given the observation history. The strategy output of the belief exploration is a memoryless deterministic strategy  $\pi_{bel}$  that maps each belief to the optimal action. It is well-known that there is a direct correspondence between strategies on the belief MDP and its POMDP [34]. A decision in a belief corresponds to a decision in the POMDP for all observation sequences that lead to the belief in the belief MDP. Thus,  $\pi_{bel}$  can also be interpreted as a strategy for the POMDP that we want to learn using our approach.

First, assume that the belief MDP is finite. Defining the computation of the output query is conceptually straightforward. During each output query, we search for the belief  $b$  that corresponds to the observation sequence in the belief MDP. If we find it, the output is  $\pi_{bel}(b)$ , otherwise the query outputs “don’t care” ( $\dagger$ ). For the equivalence query, we consider one representative observation sequence for each belief  $b$ . We compare whether  $\pi_{bel}(b)$  coincides with the output of the hypothesis FSC on the corresponding observation sequence. If not, this sequence is a counterexample. To deal with infinite belief MDPs, [11] employs a partial exploration of the reachable belief space of the POMDP. At the points where the exploration has been stopped (*cut-off states*), they use approximations based on pre-computed, small strategies on the POMDP to yield a finite abstraction of the belief MDP. The strategy  $\pi_{bel}$  computed on this abstraction, however, does not output valid actions for the POMDP in the cut-off states. We modify the output query described above and introduce a set of  $\chi$  symbols, i.e.,  $\chi_0, \dots, \chi_n$ . On observation sequences of cut-off states, the output query returns “don’t-know” corresponding to that cutoff, i.e.,  $\chi_i$  for “cut-off” strategy  $i$ . This allows us to later integrate the strategies used for approximation in our learned FSC or even substitute these strategies by different ones.

### 3.4 Improving Learned FSCs for Incomplete Information

FSCs learned using the learning approach described in Section 3.2 may still contain transitions with output “don’t-know” ( $\chi$ ). To make the FSC applicable to a POMDP, these outputs need to be replaced by distributions over actions of the POMDP. For this purpose, we suggest two heuristics. They are designed to be general, i.e. they do not consider any information that the underlying POMDP solution method provides. Furthermore, they use the idea that already learned behavior might offer a basis for generalization. As a result, the information already present in the FSC is used to replace the “don’t-know” outputs. We note that additional heuristics can take for example the structure of the POMDP or information available in the POMDP solution method used to generate the strategy table into account. For illustrating the heuristics, we assume that all

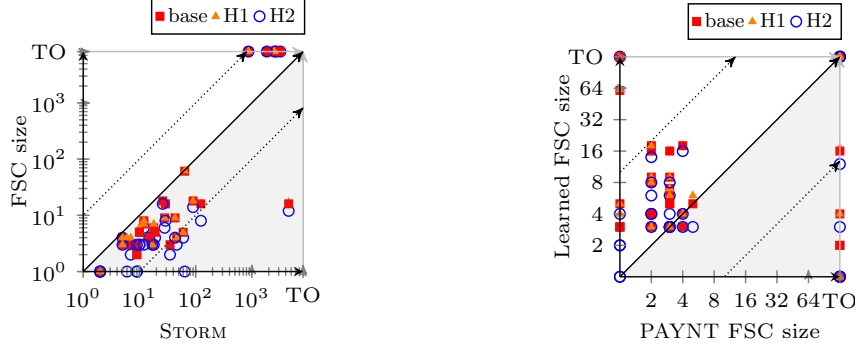
output distributions are Dirac. We denote the number of transitions in the FSC with observation  $o$  with output not equal to  $\dagger_i$  or  $\chi_i$  for some  $i$  by  $\#(o)$  and the number of transitions with output action  $a$  for  $o$  by  $\#(o, a)$ .

- *Heuristic 1 – Distribution:* Intuitively, this heuristic replaces “don’t know” by a distribution over all actions that the FSC already chooses for an observation. The resulting FSC therefore represents a randomized strategy, i.e. the strategy may probabilistically choose between actions. This happens only in nodes of the FSC where “don’t know” occurs. Furthermore, this does not mean that the FSC itself is randomized; its structure remains deterministic. Only some outputs represent randomization over actions. In this method, we replace the  $i$ th “don’t know”  $\chi_i$  by an action distribution where the probability of action  $a$  under observation  $o$  is given by  $\frac{\#(o, a)}{\#(o)}$ . If  $\#(o) = 0$ , we keep  $\chi_i$  instead which, in the belief exploration approach of STORM, represents a precomputed cutoff strategy. In approaches where the strategy does not provide any information at all it can be replaced by  $\dagger$ . Intuitively, we try to copy the behavior of the FSC for an observation and since the optimal action is unknown, we use a distribution over all possible actions.
- *Heuristic 2 – Minimizing Using  $\dagger$ -transitions:* As for ease of implementation and explainability, smaller FSCs are preferable, this heuristic aims at replacing  $\chi_i$  outputs such that we can minimise the FSC as much as possible. For this purpose, we simply replace all occurrences of  $\chi_i$  by  $\dagger$ , i.e. we replace “don’t-know” by “don’t-care” outputs. This allows the FSC to behave arbitrarily on these transitions. By then applying an additional minimisation step, we can potentially reduce the size of our FSC. Intuitively, this allows for a smaller FSC that might be able to generalize better than specifying all actions directly. Note that this heuristic will transform any deterministic FSC into a smaller representation that is still deterministic, and will not induce any randomization.

## 4 Experimental Evaluation

We implemented a prototype of the policy automaton learning framework on top of version 1.8.1 of the probabilistic model checker STORM [20]. As input, our implementation takes the belief MC induced by the optimal policy on the belief MDP abstraction computed by STORM’s belief exploration for POMDPs [11]. This Markov chain, labeled with observations and actions chosen by the computed strategy, encodes all information necessary for our approach as described in Section 3.3. We apply our learning techniques to obtain a finite-state controller representation of a policy. This FSC can be exported into a human-readable format or analyzed by building the Markov chain induced by the learned policy *directly* on the POMDP. As a baseline comparison for the learned FSC, we use the tool PAYNT [4]. Recall that PAYNT uses a technique called *inductive synthesis* to directly synthesize FSCs with respect to a given objective.

Recent research has shown that PAYNT’s performance greatly improves when working in tandem with belief exploration [2]. As such, the comparison



(a) Size of input MC (from STORM) vs. size of learned FSC (number of nodes) (b) Size of learned FSC in comparison to PAYNT (in number of nodes)

Fig. 5: Comparison of the resulting FSC size

made here does not show the full capabilities of PAYNT. The tandem approach is likely to outperform our approach in many cases. We want to, however, show a comparison of our approach with a more basic, and thus more comparable, method. We emphasize furthermore that integrating our approach in the framework of [2] is a promising prospect for future work.

**Setup.** The experiments are run on two cores of an Intel<sup>®</sup> Xeon<sup>®</sup> Platinum 8160 CPU using 64GB RAM and a time limit of 1 hour. We run STORM’s POMDP model checking framework using default parameters. In particular, we use the heuristically determined exploration depth for the belief MDP approximation and apply cut-offs where we choose not to explore further. We refer to [11] for more information. For PAYNT, we use abstraction-refinement with multi-core support [3]. We run experiments for the two heuristics described in Section 3.4. Additionally, we provide another result described as the “base” approach. This is specific to the input given by STORM and encodes the strategy obtained from STORM exactly by keeping the cut-off strategies, represented as  $\chi_i$  (see the extended version of this paper [8] for more technical details).

**Benchmarks.** As benchmarks for our evaluation, we consider the models from [2]. The benchmark set contains models taken from the literature [3,10,11,17] meant to illustrate the strengths and weaknesses of the belief exploration and inductive synthesis approaches. As such, they also showcase how our learning approach transforms the output of the belief exploration concerning the size and quality of the computed FSC. An overview of the used benchmarks is available as part of the extended version of this paper [8].

## 4.1 Results

Our approach is general and meant to be used on top of other algorithms to transform possibly big and hardly explainable strategies into small FSCs. However, we want to explore whether our results are comparable to state-of-the-art work for directly learning FSCs. Therefore, we compare our FSCs to PAYNT.

First, we talk about the size of the FSC generated by our method compared to the MC generated by STORM and the FSCs generated by PAYNT. Secondly, we show the scalability of our approach by comparing the runtime with PAYNT. Lastly, we discuss the quality of the synthesized FSCs compared to PAYNT and also discuss the trade-off between runtime and quality of the FSC.

**Small and Explainable FSCs.** Given a strategy table, our approach results in the *smallest possible* FSC for the represented strategy. As an overview, in Fig. 5a, we show a comparison of the sizes of the belief MC from STORM to the size of our FSC. The dashed line corresponds to a 10-fold reduction in size, showing our approach’s usefulness. We generate FSCs of sizes 1 to 64; however, more than 80% of the FSCs are smaller than ten nodes, and only two are bigger than 60. More than half of the generated FSCs have less than four nodes. In one case, we reduce 4517 states in the MC to an FSC of size 12.

We claim that these concise representations can generally be considered explainable, in particular when compared to huge original strategy representations. When the given strategy is deterministic, our learning approach would construct a deterministic FSC which is easy to explain. While improving the FSC by replacing the “don’t know” actions (Section 3.4), heuristic 2 still keeps the FSC deterministic as it only replaces the  $\chi$  actions with  $\dagger$  actions before minimisation. Heuristic 1 often introduces some randomization when it replaces the  $\chi$  actions with a distribution. But even in that case, they are only in selected sink states which does not impede explainability.

In Fig. 5b, we provide the size comparison of PAYNT’s FSCs and ours. Our FSCs are slightly bigger than PAYNT’s in general, but our approach also returns smaller FSCs in some cases. This is to be expected since the approach of PAYNT is iteratively searching through the space of FSCs, starting with only one memory node and adding memory only once it is necessary. Therefore, it is meant to find the smallest possible FSC. However, PAYNT times out much more often because of its exhaustive search on small FSC. Additionally, our FSC are bound to be as big as necessary to represent the given strategy. Let us consider the benchmark `grid-avoid-4-0`. In this model, a robot moves in a grid of size four by four with no knowledge about its position. It starts randomly at any place in the grid and has to move towards a goal without falling into a “hole”. PAYNT produces a strategy of size 3, which moves right once and then iterates between moving right and down. The nature of STORM’s exploration leads to a strategy that moves right three times and then down forever. This can be represented in an FSC of size at least 5.

**Scalability.** Regarding scalability, Figure 6 shows that our approach outperforms PAYNT on almost all cases. The dotted lines show differences by a factor of 10. There are only two benchmarks, for which our approach times out and

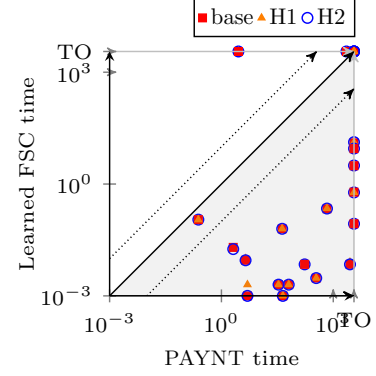


Fig. 6: Runtime comparison: our approach vs. PAYNT



Table 3: Comparison to PAYNT on value, size, and time (in that order) on selected benchmarks. The reported time for our approach includes the time of STORM for producing the strategy table **and** the time for learning the FSC.

Category	Model	STORM	Learning heuristics			PAYNT
			base	H <sub>1</sub>	H <sub>2</sub>	
<b>A</b>	problem-paynt-storm-combined	8.07 18	8.07 6	<b>7.67</b> 7	<b>7.67</b> 3	<b>7.67</b> 3
	Rmin	<1s		<1s		349s
	problem-storm-extended	3009.0 64	3009.0 61	<b>98.0</b> 62	<b>98.0</b> 1	<b>98.0</b> 1
	Rmin	<1s		<1s		<1 s
	refuel-20	0.14 46	0.14 4	<b>0.23</b> 4	<b>0.23</b> 3	TO
	Pmax	73s	s 75s	<b>74s</b>	<b>74s</b>	
<b>B</b>	grid-avoid-4-01	0.75 10	0.75 5	0.9 6	0.67 3	<b>0.93</b> 5
	Pmax	<1s		<1s		726s
	posterior-awareness	12.0 5	12.0 4	12.0 4	12.0 4	<b>11.99</b> 4
	Rmin	<1s		<1s		<1 s
<b>C</b>	4x5x2-95	1.29 26	1.29 18	1.28 18	1.26 16	<b>2.02</b> 4
	Rmax	<1s		<1s		2807s
	query-s2	395.66 43	395.66 9	391.9 9	343.94 4	<b>486.69</b> 2
	Rmax	<1s		<1s		5s
<b>C</b>	drone-4-1	0.75 3217	TO	TO	TO	<b>0.87</b> 1
	Pmax	1s				<b>2250s</b>

PAYNT does not. In one of these cases, PAYNT also takes more than 2000s to produce a result.

**Runtime and Quality of FSCs.** Comparing the quality of results, we need to put into consideration that our approach often runs within a fraction of the available time. We run STORM with its default values to get a strategy. As demonstrated in [3], running STORM using non-default parameters, specifically larger exploration thresholds, results in better strategies at the cost of longer runtimes. Our approach directly profits from such better input strategies.

Since the learning is done in far less than a second for most of the benchmarks, we suggest using a portfolio of the heuristics. This allows us to output the optimal solution among all our heuristics with negligible computational overhead. To simplify the presentation of our results, we categorize the benchmarks into three groups: A, B, and C, based on the overall performance of our method. Due to space constraints, we provide detailed results for only a selection of benchmarks for each category and do not discuss benchmarks for which both approaches experienced timeouts. The complete set of results is given in [8].

*Category A.* This category represents benchmarks where our approach is arguably favored, assuming the portfolio approach. There are a total of 19 benchmarks in this category, and we observe that we can improve all variants of properties using heuristics. Only one time, PAYNT produces a slightly better probability value (0.93 vs 0.9), but it takes significantly more time (726s vs < 1s).

There are 7 cases where we can generate FSCs while PAYNT times out and on 6 out of these 7 cases, we get the **smallest** FSCs reported in state-of-the-art [2]. In this category, we also include benchmarks on which the heuristics improved on STORM’s strategy to achieve the same value as PAYNT while being more efficient, e.g. **problem-paynt-storm-combined**. Also, for the benchmark **problem-storm-extended**, designed to be difficult for STORM, we reduce the approximate total reward from 3009 to 98, resulting in an FSC of size **1** in  $< 1s$ .

*Category B.* This category contains benchmarks on which there is no clear front-runner. There are a total of 7 benchmarks in this category. Three of these benchmarks are similar to **posterior-awareness**, where the results produced and the time taken are quite similar for both approaches. The other 4 benchmarks (similar to **4x5x2-95**) show that the value generated by our approach is significantly worse; however, it takes significantly less time. Depending on the situation, this trade-off between quality and runtime may favor either approach.

*Category C.* This category shows the weakness of our method compared to PAYNT. In this category, there are a total of 3 benchmarks, out of which our approach times out 2 times. It is notable that the **drone**-benchmarks seem to be generally hard: PAYNT needs 2250s for **drone-4-1**, and both approaches time out for the bigger instances. There is only one benchmark, **query-s2**, where we produce a worse value without any significant time advantage over PAYNT.

## 5 Conclusion

In this paper, we present an approach to learn an FSC for representing POMDP strategies. Our FSCs are (i) always *smaller* than the given representation, and (ii) the FSC structure is simple, which together increases the strategy’s *explainability*. The structure of the FSC is always deterministic. Additionally, one of our heuristics only generates deterministic output actions (without randomization). The other heuristic typically represents a randomized strategy. However, only output actions are randomized, *not* the FSC structure. Besides, this randomization happens in only a very restricted form. Further, our heuristics achieved notable improvements in the *performance* of many strategies produced by STORM and provably perform equal or better than the baseline, while retaining negligible resource consumption. Altogether, our comparison against PAYNT underscores the competitiveness of our method, frequently yielding FSCs of comparable quality with significant improvements in terms of runtime and size.

This attests to the scalability and efficiency of our approach and also highlights its applicability in scenarios challenging for other tools.

Concerning future work, several directions open up. Further heuristics can be designed to solve some of the patterns occurring in the cases where our approach could not match the size achieved by PAYNT. Furthermore, we would like to integrate our approach into other approaches in order to improve them, in particular the tandem synthesis approach from [2] is a suitable candidate.

*Data Availability.* The artifact accompanying this paper [9] contains source code, benchmark files, and replication scripts for our experiments.

## References

1. Amato, C., Bernstein, D.S., Zilberstein, S.: Optimizing fixed-size stochastic controllers for pomdps and decentralized pomdps. *Auton. Agents Multi Agent Syst.* **21**(3), 293–320 (2010), <https://doi.org/10.1007/s10458-009-9103-z>
2. Andriushchenko, R., Bork, A., Ceska, M., Junges, S., Katoen, J., Macák, F.: Search and explore: Symbiotic policy synthesis in pomdps. In: *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III. Lecture Notes in Computer Science*, vol. 13966, pp. 113–135. Springer (2023), [https://doi.org/10.1007/978-3-031-37709-9\\_6](https://doi.org/10.1007/978-3-031-37709-9_6)
3. Andriushchenko, R., Ceska, M., Junges, S., Katoen, J.: Inductive synthesis of finite-state controllers for pomdps. In: *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands. Proceedings of Machine Learning Research*, vol. 180, pp. 85–95. PMLR (2022), <https://proceedings.mlr.press/v180/andriushchenko22a.html>
4. Andriushchenko, R., Ceska, M., Junges, S., Katoen, J., Stupinský, S.: PAYNT: A tool for inductive synthesis of probabilistic programs. In: *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I. Lecture Notes in Computer Science*, vol. 12759, pp. 856–869. Springer (2021). [https://doi.org/10.1007/978-3-030-81685-8\\_40](https://doi.org/10.1007/978-3-030-81685-8_40), [https://doi.org/10.1007/978-3-030-81685-8\\_40](https://doi.org/10.1007/978-3-030-81685-8_40)
5. Angluin, D.: Learning regular sets from queries and counterexamples. *Information and computation* **75**(2), 87–106 (1987), [https://doi.org/10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6)
6. Ashok, P., Jackermeier, M., Jagtap, P., Kretínský, J., Weininger, M., Zamani, M.: dtcontrol: decision tree learning algorithms for controller representation. In: *HSCC*. pp. 17:1–17:7. ACM (2020), <https://dl.acm.org/doi/abs/10.1145/3365365.3383468>
7. Ashok, P., Jackermeier, M., Křetínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtcontrol 2.0: Explainable strategy representation via decision tree learning steered by experts. In: *TACAS (2). Lecture Notes in Computer Science*, vol. 12652, pp. 326–345. Springer (2021), [https://doi.org/10.1007/978-3-030-72013-1\\_17](https://doi.org/10.1007/978-3-030-72013-1_17)
8. Bork, A., Chakraborty, D., Grover, K., Kretinsky, J., Mohr, S.: Learning Explainable and Better Performing Representations of POMDP Strategies. *arXiv preprint arXiv:2401.07656* (2024), <https://doi.org/10.48550/arXiv.2401.07656>
9. Bork, A., Chakraborty, D., Grover, K., Mohr, S., Kretinsky, J.: Artifact for Paper: Learning Explainable and Better Performing Representations of POMDP Strategies, <https://doi.org/10.5281/zenodo.10437018>
10. Bork, A., Junges, S., Katoen, J., Quatmann, T.: Verification of indefinite-horizon pomdps. In: *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science*, vol. 12302, pp. 288–304. Springer (2020), [https://doi.org/10.1007/978-3-030-59152-6\\_16](https://doi.org/10.1007/978-3-030-59152-6_16)
11. Bork, A., Katoen, J.P., Quatmann, T.: Under-approximating expected total rewards in pomdps. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. pp. 22–40. Springer (2022), [https://doi.org/10.1007/978-3-030-99527-0\\_2](https://doi.org/10.1007/978-3-030-99527-0_2)
12. Brázdil, T., Chatterjee, K., Chmelik, M., Fellner, A., Křetínský, J.: Counterexample explanation by learning small strategies in markov decision processes. In: *CAV*

- (1). Lecture Notes in Computer Science, vol. 9206, pp. 158–177. Springer (2015), [https://doi.org/10.1007/978-3-319-21690-4\\_10](https://doi.org/10.1007/978-3-319-21690-4_10)
13. Carr, S., Jansen, N., Wimmer, R., Serban, A.C., Becker, B., Topcu, U.: Counterexample-guided strategy improvement for pomdps using recurrent neural networks. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 5532–5539. ijcai.org (2019), <https://doi.org/10.24963/ijcai.2019/768>
14. Chatterjee, K., Chmelik, M., Tracol, M.: What is decidable about partially observable markov decision processes with  $\omega$ -regular objectives. Journal of Computer and System Sciences **82**(5), 878–911 (2016), <https://doi.org/10.1016/j.jcss.2016.02.009>
15. Cubuktepe, M., Jansen, N., Junges, S., Marandi, A., Suilen, M., Topcu, U.: Robust finite-state controllers for uncertain pomdps. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 11792–11800. AAAI Press (2021), <https://doi.org/10.1609/aaai.v35i13.17401>
16. Hansen, E.A.: Solving pomdps by searching in policy space. In: Cooper, G.F., Moral, S. (eds.) UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998. pp. 211–219. Morgan Kaufmann (1998), <https://dl.acm.org/doi/abs/10.5555/2074094.2074119>
17. Hauskrecht, M.: Incremental methods for computing bounds in partially observable markov decision processes. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA. pp. 734–739. AAAI Press / The MIT Press (1997), <https://dl.acm.org/doi/10.5555/1867406.1867520>
18. Hauskrecht, M.: Value-function approximations for partially observable markov decision processes. J. Artif. Intell. Res. **13**, 33–94 (2000), <https://doi.org/10.1613/jair.678>
19. Heck, L., Spel, J., Junges, S., Moerman, J., Katoen, J.: Gradient-descent for randomized controllers under partial observability. In: Verification, Model Checking, and Abstract Interpretation - 23rd International Conference, VMCAI 2022, Philadelphia, PA, USA, January 16-18, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13182, pp. 127–150. Springer (2022), [https://doi.org/10.1007/978-3-030-94583-1\\_7](https://doi.org/10.1007/978-3-030-94583-1_7)
20. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022), <https://doi.org/10.1007/s10009-021-00633-z>
21. Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J., Becker, B.: Finite-state controllers of pomdps using parameter synthesis. In: Globerson, A., Silva, R. (eds.) Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018. pp. 519–529. AUAI Press (2018)
22. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**(1), 99–134 (1998), [https://doi.org/10.1016/S0004-3702\(98\)00023-X](https://doi.org/10.1016/S0004-3702(98)00023-X)

23. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Brock, O., Trinkle, J., Ramos, F. (eds.) *Robotics: Science and Systems IV*, Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, June 25-28, 2008. The MIT Press (2008), <https://doi.org/10.15607/RSS.2008.IV.009>
24. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science*, vol. 6806, pp. 585–591. Springer (2011), [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
25. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence* **147**(1-2), 5–34 (2003), [https://doi.org/10.1016/S0004-3702\(02\)00378-8](https://doi.org/10.1016/S0004-3702(02)00378-8)
26. Meuleau, N., Kim, K., Kaelbling, L.P., Cassandra, A.R.: Solving pomdps by searching the space of finite policies. In: Laskey, K.B., Prade, H. (eds.) *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 30 - August 1, 1999. pp. 417–426. Morgan Kaufmann (1999), <https://dl.acm.org/doi/10.5555/2073796.2073844>
27. Neider, D., Topcu, U.: An automaton learning approach to solving safety games over infinite graphs. In: *TACAS. Lecture Notes in Computer Science*, vol. 9636, pp. 204–221. Springer (2016), [https://doi.org/10.1007/978-3-662-49674-9\\_12](https://doi.org/10.1007/978-3-662-49674-9_12)
28. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic systems. *Real Time Syst.* **53**(3), 354–402 (2017), <https://doi.org/10.1007/s11241-017-9269-4>
29. Pineau, J., Gordon, G.J., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps. In: Gottlob, G., Walsh, T. (eds.) *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 9-15, 2003. pp. 1025–1032. Morgan Kaufmann (2003)
30. Russell, S.J.: *Artificial intelligence a modern approach*. Pearson Education, Inc. (2010), <https://dl.acm.org/doi/book/10.5555/1671238>
31. Shahbaz, M., Groz, R.: Inferring mealy machines. In: Cavalcanti, A., Dams, D. (eds.) *FM 2009: Formal Methods, Second World Congress*, Eindhoven, The Netherlands, November 2-6, 2009. *Proceedings. Lecture Notes in Computer Science*, vol. 5850, pp. 207–222. Springer (2009), [https://doi.org/10.1007/978-3-642-05089-3\\_14](https://doi.org/10.1007/978-3-642-05089-3_14)
32. Shani, G., Pineau, J., Kaplow, R.: A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems* **27**, 1–51 (2013), <https://doi.org/10.1007/s10458-012-9200-2>
33. Simão, T.D., Suilen, M., Jansen, N.: Safe policy improvement for pomdps via finite-state controllers. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence. AAAI'23/IAAI'23/EAAI'23*, AAAI Press (2023), <https://doi.org/10.1609/aaai.v37i12.26763>
34. Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable markov processes over a finite horizon. *Oper. Res.* **21**(5), 1071–1088 (1973), <https://doi.org/10.1287/opre.21.5.1071>
35. Spaan, M.T.J., Vlassis, N.: Perseus: Randomized point-based value iteration for pomdps. *J. Artif. Intell. Res.* **24**, 195–220 (2005), <https://doi.org/10.1613/jair.1659>

36. Thomas, P., Theocharous, G., Ghavamzadeh, M.: High-confidence off-policy evaluation. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 29 (2015), <https://dl.acm.org/doi/10.5555/2888116.2888134>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



## **D** Monitizer: Automating Design and Evaluation of Neural Network Monitors

*Licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.*

This paper is to be published as a **peer-reviewed conference paper**.

Muqsit Azeem, Marta Grobelna, Sudeep Kanav, Jan Křetínský, Stefanie Mohr, and Sabine Rieder: Monitizer: Automating Design and Evaluation of Neural Network Monitors. In: Gurfinkel, A., Ganesh, V. (eds) Computer Aided Verification. CAV 2024. Lecture Notes in Computer Science, vol 14682. Springer, Cham.

DOI: [https://doi.org/10.1007/978-3-031-65630-9\\_14](https://doi.org/10.1007/978-3-031-65630-9_14)

### Summary

With Neural Networks (NNs) being employed increasingly in safety-critical systems, their unpredictable behavior on Out-of-Distribution (OOD) inputs can create great risk. Therefore, we need *monitors* to check for such inputs. To address this, we introduce MONITIZER to automate the design, optimization, and evaluation of NN monitors.

MONITIZER solves several challenges in monitoring: First, it provides a modular framework that allows users to apply monitors from the literature or develop their own; second, it offers an optimization mechanism to tune the hyperparameters of the monitors for better performance; last, MONITIZER contains a transparent evaluation based on a hierarchy of OOD classes. It also offers a one-click solution for automatic selection and optimization of the best monitor, reducing the need for extensive user expertise. Additionally, it includes a library of 19 monitors, 9 datasets, and 15 NNs, making it usable straight out of the box.

For developers and researchers, MONITIZER provides a framework to integrate and develop novel monitoring techniques with minimal effort.

### Contributions of the author

Composition and revision of the manuscript with significant role in writing sections 3 and 4. Discussion and development of the ideas, implementation and evaluation with the following notable individual contributions: Leading role in the design and implementation of the presented tool, and creation of the software artifact for the conference submission.



# Monitizer: Automating Design and Evaluation of Neural Network Monitors

Muqsit Azeem<sup>1</sup>, Marta Grobelna<sup>1</sup>, Sudeep Kanav<sup>2</sup>,  
Jan Křetínský<sup>1,2</sup>, Stefanie Mohr<sup>1</sup>, and Sabine Rieder<sup>1,2,3</sup>

<sup>1</sup> Technical University of Munich, Munich, Germany

jan.kretinsky@tum.de

<sup>2</sup> Masaryk University, Brno, Czech Republic

<sup>3</sup> Audi AG, Ingolstadt, Germany



**Abstract.** The behavior of neural networks (NNs) on previously unseen types of data (out-of-distribution or OOD) is typically unpredictable. This can be dangerous if the network’s output is used for decision making in a safety-critical system. Hence, detecting that an input is OOD is crucial for the safe application of the NN. Verification approaches do not scale to practical NNs, making runtime monitoring more appealing for practical use. While various monitors have been suggested recently, their optimization for a given problem, as well as comparison with each other and reproduction of results, remain challenging.

We present a tool for users and developers of NN monitors. It allows for (i) application of various types of monitors from the literature to a given input NN, (ii) optimization of the monitor’s hyperparameters, and (iii) experimental evaluation and comparison to other approaches. Besides, it facilitates the development of new monitoring approaches. We demonstrate the tool’s usability on several use cases of different types of users as well as on a case study comparing different approaches from recent literature.

## 1 Introduction

*Neural networks (NNs)* are increasingly used in safety-critical applications due to their good performance even on complex problems. However, their notorious unreliability makes their safety assurance even more important. In particular, even if the NN is well trained on the data that it is given and works well on similar data (so-called *in-distribution (ID) data*), it is unclear what it does if presented with a significantly different input (so-called *out-of-distribution (OOD) data*). For instance, what if an NN for traffic signs recognition trained on pictures taken

---

This research was funded in part by the German Research Foundation (DFG) project 427755713 GOPro and the MUNI Award in Science and Humanities MUNI/I/1757/2021 of the Grant Agency of Masaryk University.

© The Author(s) 2024

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14682, pp. 265–279, 2024.

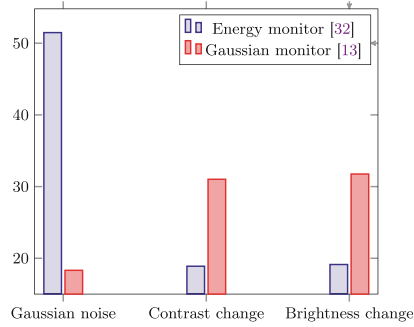
[https://doi.org/10.1007/978-3-031-65630-9\\_14](https://doi.org/10.1007/978-3-031-65630-9_14)



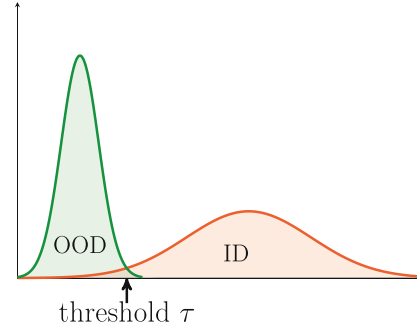
in Nevada is now presented with a traffic sign in rainy weather, a European one, or a billboard with an elephant?

To ensure safety in all situations, we must at least recognize that the input is OOD; thus, the network’s answer is unreliable, no matter its confidence. Verification, a classic approach for proving safety, is extremely costly and essentially infeasible for practical NNs [34]. Moreover, it is mainly done for ID or related data [6, 34]. For instance, robustness is typically proven for neighborhoods of essential points, which may ensure correct behavior in the presence of noise or rain, but not elephants [18, 24, 25, 35]. In contrast, runtime verification and particularly runtime *monitoring* provide a cheap alternative. Moreover, the industry also finds it appealing as it is currently the only formal-methods approach applicable to industrial-sized NNs.

*OOD runtime monitoring methods* have recently started flourishing [7, 14, 20, 22, 32, 42]. Such a runtime monitor tries to detect if the current input to the NN is OOD. To this end, it typically monitors the behavior of the network (e.g., the output probabilities or the activation values of the neurons) and evaluates whether the obtained values resemble the ones observed on known ID data. If not, the monitor raises an alarm to convey suspicion about OOD data.



(a) Accuracy of two monitoring techniques on different OOD data



(b) Threshold value  $\tau$  to optimally separate OOD and ID

**Fig. 1.** Illustration of challenges for OOD detection

*Challenges:* While this approach has demonstrated potential, several practical issues arise:

- How can we *compare* two monitors and determine which one is better? Considering the example of autonomous driving, an OOD input could arise from the fact that some noise was introduced by sensors or the brightness of the environment was perturbed. A monitor might perform well on one kind of OOD input but may not on another [44], as better performance in one class of OOD data does not imply the same in another class (see Fig. 1a).

- Applying a particular monitoring technology to a concrete NN involves significant tweaking and *hyperparameter tuning*, with no push-button technology available. OOD monitors typically compute a value from the input and the behavior of the NN. The input is considered OOD if this value is smaller than a configurable *threshold*  $\tau$  (see Fig. 1)b. The value of this threshold has a significant influence on the performance of the monitors. More inputs would be classified as OOD if the threshold value is high, and vice versa. Moreover, OOD monitors generally have *multiple parameters* that require tuning, thereby aggravating the complexity of manual configuration.
- As OOD monitoring can currently be described as a search for a good heuristic, many more heuristics will appear, implying the need for streamlining their handling and fair comparison.

In this paper, we provide the infrastructure for users and developers of NN monitors aiming at detecting OOD inputs (onwards just “monitors”).

*Our contributions* can be summarized as follows:

- We provide a modular tool called MONITIZER for automatic learning/constructing, optimizing, and evaluating monitors.
- MONITIZER supports (i) *easy practical use*, providing various recent monitors from the literature, which can directly be optimized and applied to user-given networks and datasets with no further inputs required; the push-button solution offers automatic choice of the best available monitor without requiring any knowledge on the side of the user; (ii) *advanced development use*, with the possibility of easily integrating a new monitor or new evaluation techniques. The framework also foresees and allows for the integration of monitoring other properties than OOD.
- We provide a library of 19 well-known monitors from the scientific literature to be used off-the-shelf, accompanied by 9 datasets and 15 NNs, which can be used for easy but rich automatic evaluation and comparison of monitors on various OOD categories.
- We demonstrate the functionality for principled use cases accompanied by examples and a case study comparing a few recent monitoring approaches.

Altogether, we are giving users the infrastructure for automatic creation of monitors, development of new methods, and their comparison to similar approaches.

## 2 Related Work

**NN Monitoring Frameworks.** OPENOOD [47,48] contains task-specific benchmarks for OOD detection that consist of an ID and multiple OOD datasets for specific tasks (e.g., Open Set Recognition and Anomaly Detection). Both OPENOOD and MONITIZER contain several different monitors and benchmarks. MONITIZER provides functionality to tune the monitors for the given objective, supports a comprehensive evaluation of monitors on a specific ID dataset by

automatically providing generated OOD inputs by, e.g., the addition of noise, and can easily be extended with more datasets. OPENOOD, in contrast to MONITIZER, does not support hyperparameter tuning and generation of OOD inputs.

Samuels et al. propose a framework to optimize an OOD monitor during runtime on newly experienced OOD inputs [26]. While this contains optimization, the framework is specific to one monitor and is based on active learning. MONITIZER is meant to work in an offline setting and optimize a monitor before it is deployed. Additionally, MONITIZER is built for extensibility and reusability, which the other tool is not, e.g., it lacks an executable.

PYTORCH-OOD [27] is a library for OOD detection, yet despite its name, it is *not* part of the official PyTorch-library. It includes several monitors, datasets, and supports the evaluation of the integrated monitors. Both MONITIZER and PYTORCH-OOD provide a library of monitors and datasets. However, there are significant differences. MONITIZER supports optimization of monitors, allowing us to return monitors optimal for a chosen objective, provides a more structured view of the dataset, and provides a transparent and detailed evaluation showing how a monitor performs on different OOD classes. Besides, we provide a one-click solution to easily evaluate the whole set of monitors and automatically return the best available option, fine-tuned to the case. Consequently, MONITIZER is a tool that is much easier to use and extend. Last but not least, it is an alternative implementation that allows cross-checking outcomes, thereby making monitoring more trustworthy.

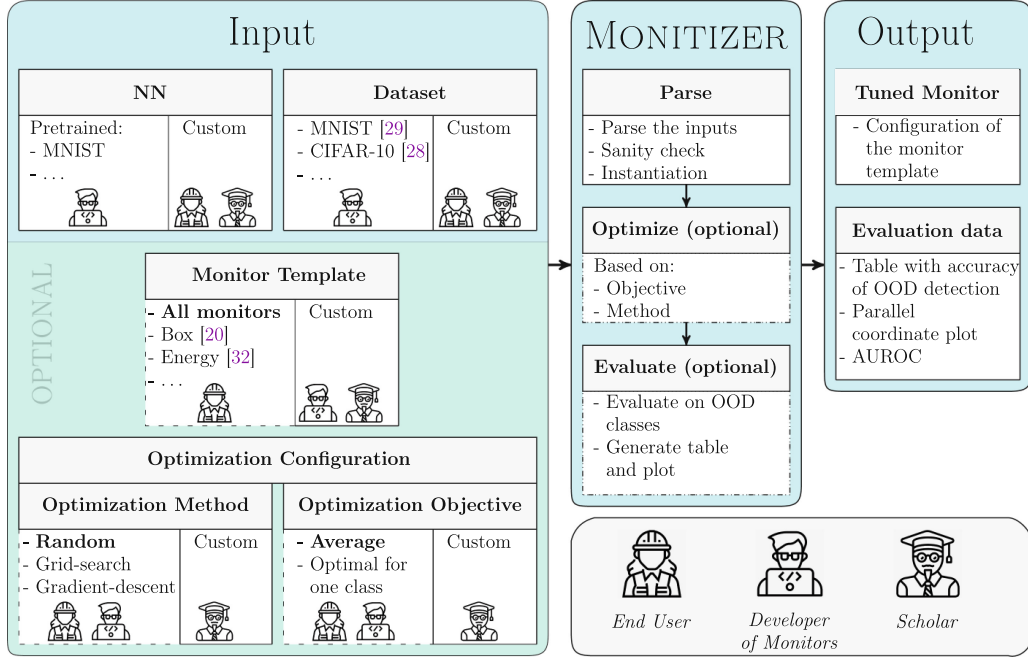
**OOD Benchmarking.** Various datasets have been published for OOD benchmarking [15, 16, 19, 37, 38], Breitenstein et al. present a classification for different types of OOD data in automated driving [5], and Ferreira et al. propose a benchmark set for OOD with several different categories [11].

### 3 Monitizer

MONITIZER aims to assist the developers and users of NN monitors and developers of new monitoring techniques by supporting optimization and transparent evaluation of their monitors. It structures OOD data in a hierarchy of classes, and a monitor can be tuned for any (combination) of these classes. It also provides a one-click solution to evaluate a set of monitors and return the best available option optimized for the given requirement.

#### 3.1 Overview

MONITIZER offers two main building blocks, as demonstrated in Fig. 2: optimization and evaluation of NN monitors. NN monitors are typically parameterized and usually depend on the NN and dataset. Before one can evaluate them, they need to be configured and possibly tuned. We refer to monitors that are not yet configured as *monitor templates*. MONITIZER optimizes the monitor templates and evaluates them afterward on several different OOD classes, i.e., types of OOD data.



**Fig. 2.** Architecture of MONITIZER: The required inputs are an NN and the dataset (both can be chosen from existing options). The dashed area indicates optional inputs, and the bold-faced option indicates the default value. The icons(see footnote 1) indicate which types of users are expected to use each of the options.

MONITIZER needs at least two inputs (see Fig. 2): an NN, and an ID-dataset. The user can also provide a monitor template and an optimization configuration (consisting of an optimization objective and optimization method). If these are not provided, MONITIZER reverts to the default values (i.e., evaluating all monitors using the AUROC-score without optimization). For both inputs, the user can choose from the options we offer or provide a custom implementation.

MONITIZER optimizes the provided monitor based on the optimization objectives and method on the given ID dataset. An example of optimization would be:<sup>1</sup> maximize the detection accuracy on blurry images, but keep the accuracy on ID images at least 70%. Optimization is necessary to obtain a monitor that is ready to use. However, it is possible to evaluate a monitor template on its default values for the parameters using the *AUROC*-score (Area Under the Receiver Operating Characteristic Curve)<sup>2</sup>.

On successful execution, MONITIZER provides the user with a configuration of the monitor template and the evaluation result. This can be either a table with the accuracy of OOD detection for each OOD dataset along with a parallel coordinate plot for the same (in case of optimization) or the AUROC score.

<sup>1</sup> Thanks to Flaticon.com for the Icons.

<sup>2</sup> The ROC (Receiver Operating Characteristic) curve shows the performance of a binary classifier with different decision thresholds. The AUROC computes the area under this curve. The best possible value is 1, indicating perfect prediction.

### 3.2 Use Cases

We envision three different types of users for MONITIZER:

#### 1. The End User

*Context:* The end user of a monitor, e.g., an engineer in the aviation industry, is interested in the end product, not in the intricacies of the underlying monitoring technique. She intends to evaluate one or all monitors provided by MONITIZER for her custom NN and dataset, and wants to come to a conclusion on which one to use. She has an NN that needs to be monitored. Additionally, she has her own proprietary ID dataset, e.g., the one on which the NN was trained. She wants a monitor fulfilling some requirement, e.g., one that is optimal on average for all classes or one that can detect a specific type of OOD that her NN is not able to handle properly.

*Usage:* Such a user can obtain a monitor tuned to her needs using MONITIZER without much effort. MONITIZER supports this feature out of the box. It provides various monitors (19 at present) that can be optimized for a given network. In case she wants to use a custom NN or a dataset, she has to provide the NN as PyTorch-dump or in onnx-format [4] and add some lines of code to implement the interface for loading her data.

*Required Effort:* After providing the interface for her custom dataset, the user only has to trigger the execution. The execution time depends on the hardware quality, the NN's size, the chosen monitor's complexity, and the dataset's size.

#### 2. The Developer of Monitors

*Context:* The developer of monitoring techniques, e.g., a researcher working in runtime verification of NNs, aims to create novel techniques and assess their performance in comparison to established methods.

*Usage:* Such a user can plug their novel monitor into MONITIZER and evaluate it. MONITIZER directly provides the most commonly used NNs and datasets for academic evaluation.

*Required Effort:* The code for the monitor needs to be in Python and should implement the functions specified in the interface for monitors in MONITIZER. Afterward, she can trigger the evaluation of her monitoring technique.

#### 3. The Scholar

*Context:* An expert in monitoring, e.g., an experienced researcher in NN runtime verification, intends to explore beyond the current boundaries. She might want to adapt an NN monitor to properties other than OOD, or to experiment with custom NNs or datasets.

*Usage:* MONITIZER provides interfaces, and instructions on how to integrate new NNs, datasets, monitors, custom optimization methods and objectives.

*Required Effort:* The required integration effort depends on the complexity of the concrete use case. For example, adding an NN would take much less time than developing a new monitor.

More detailed examples are available in [1].

### 3.3 Phases of MONITIZER

An execution of MONITIZER is typically a sequence of three phases: parse, optimize, and evaluate. As mentioned, the user can decide to skip the optimization or the evaluation.

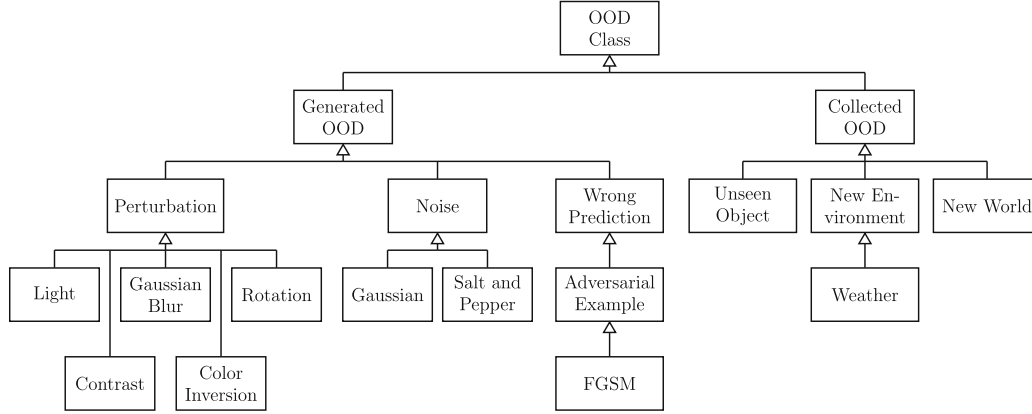
**Parse.** This phase parses the input, loads the NN and dataset, and instantiates the monitor. It also performs sanity checks on the inputs, e.g., the datasets are available in the file system, the provided monitor is implemented correctly, etc.

**Optimize.** This phase tunes the parameters of a given monitor template to maximize an objective. It depends on two inputs, the optimization method and the optimization objective, that the user has to give.

An illustrative depiction of this process can be found in [1]. The optimization method defines the search space and generates a new candidate monitor by setting its parameters. MONITIZER then uses the optimization objective to evaluate this candidate. If the objective is to optimize at least one OOD class, MONITIZER evaluates the monitor on a validation set of this class, which is distinct from the test set used in the evaluation later. The optimization method obtains this result and decides whether to continue optimizing or stop and return the best monitor that it has found.

MONITIZER provides three optimization methods: random, grid-search, and gradient descent. Random search tries out a specified number of random sets of parameters and returns the monitor that worked best among these. Grid-search specifies a search grid by looking at the minimal and maximal values of the parameters. It then defines a grid on the search space. The monitor is infused with these parameters for each grid vertex and evaluated on the objective. Gradient-descent follows the gradient of the objective function towards the optimum.

MONITIZER supports multi-objective optimization of monitors. A user can specify a set of OOD classes to optimize for and the minimum required accuracy for ID detection. Single objective optimization is a special case when only one OOD class is specified for optimization. Based on a configuration value, MONITIZER would generate a set of different weight combinations for the objectives and create and evaluate a monitor for each of these combinations. If there are two objectives, MONITIZER generates a Pareto frontier plot; in the case of more than two objectives, the tool generates a table. The user obtains the performance of the optimized monitor for each weight-combination of objectives.



**Fig. 3.** Class diagram depicting the different types of OOD data.

**Evaluate.** The evaluation of NN monitors in MONITIZER is structured according to the OOD classification (detailed in the next section). We introduce this classification of OOD data to enable a clearer evaluation and gain knowledge about which monitor performs well on which particular class of OOD. Typically, no monitor performs well on every class of OOD [44]. We highlight this in our evaluation to ensure a fair and meaningful comparison between monitors rather than restricting to a non-transparent and possibly biased average score.

After evaluation, MONITIZER reports the detection accuracy for each OOD class and can also produce a parallel-coordinates-plot displaying the reported accuracy. MONITIZER can also provide confidence intervals for the evaluation quality, which is explained in [1].

### 3.4 Classification of Out-of-Distribution Data

We now introduce our classification of OOD data. At the top level, an OOD input can either be *generated*, i.e., obtained by distorting ID data [3, 14, 17, 31, 41], or it can be *collected* using data from some other available dataset.



(a) DTD [8]



(b) CIFAR-10 [28]



(c) MNIST [29]



(d) KMNIST [9]

The notion of generated OOD is straightforward. These classes are created by slightly distorting ID data, for example, by increasing the contrast or adding noise. An important factor is the amount of distortion, e.g., the amount of noise, as it influences the NN's performance and needs to be high enough to transform an ID into an OOD input.

We explain the idea of collected OOD with the help of an example shown in Fig. 4. Consider an ID

**Fig. 4.** Examples for OOD



dataset that consists of textures (Fig. 4a). Images containing objects (Fig. 4b) differ from images showing just a texture. But, when we consider a dataset of numbers as ID (Fig. 4c), it seems much more similar to a dataset of letters (Fig. 4d) than textures are to objects. In the first case, the datasets have no common meaning or concept, as if they were belonging to a *new world*. In the second case, the environment and the underlying concept are similar, but an *unseen object* is placed in it.

Figure 3 shows our classification of the OOD data. It is based on the kind of OOD data we found in the literature (discussed in Sect. 2). [1] contains a detailed description of each class and an illustrative figure.

**OOD Benchmarks Implementation.** Note that the generated OOD will be automatically created by MONITIZER for any given ID dataset. The collected OOD data has to be manually selected. We provide a few preselected datasets (for example, KMNIST [9] as unseen objects for MNIST [29]) in the tool. A user can easily add more when needed. However, for a user like the developer of monitors, MNIST and CIFAR-10 are often sufficient to test new monitoring methodologies, as related work has shown [13, 20].

### 3.5 Library of Monitors, NNs, and Datasets

MONITIZER currently includes 19 monitors, accompanied by 9 datasets and 15 NNs. In the following, we give an overview of the available options.

*Monitors.* MONITIZER provides different highly cited monitors, which are also included in other tools such as OPENOOD/PYTORCH-OOD. We extended this list by adding monitors from the formal methods community (e.g., BOX monitor, GAUSSIAN monitor). The following monitors are available in MONITIZER: ASH-B, ASH-P, ASH-S [10], BOX-MONITOR [20], DICE [42], ENERGY [32], ENTROPY [33], GAUSSIAN [13], GRADNORM [23], KL MATCHING [15], KNN [43], MAXLOGIT [50], MDS [30], SOFTMAX [17], ODIN [31], REACT [41], MAHALANOBIS [39], SHE [49], TEMPERATURE [12] VIM [45].

*Datasets.* The following datasets are available in MONITIZER: CIFAR-10, CIFAR-100 [28], DTD [8], FashionMNIST [46], GTSRB [21], ImageNet [40], K-MNIST [9], MNIST [29], SVHN [36].

*Neural Networks* MONITIZER provides at least one pretrained NN for each available dataset. The library contains more NNs trained on commonly used datasets in academia, such as MNIST and CIFAR-10, allowing users to evaluate monitors on different architectures. [1] contains a detailed description of the pretrained NNs.

## 4 Summary of Evaluation by Case Study

We demonstrate the necessity of having a clear evaluation in Table 1. The full table containing all available OOD datasets can be found in [1]. We evaluate the



**Table 1.** Comparison of the AUROC-score of all implemented monitors on different OOD datasets multiplied by 100 (and rounded to the nearest integer). All monitors were evaluated on a fully connected network trained on MNIST. The cells are colored according to the relative performance of a monitor (column) in a specific OOD class (row). The monitors are divided in three ranks and the darker color represents better performance. If several monitors have the same score, they all belong to the better group.

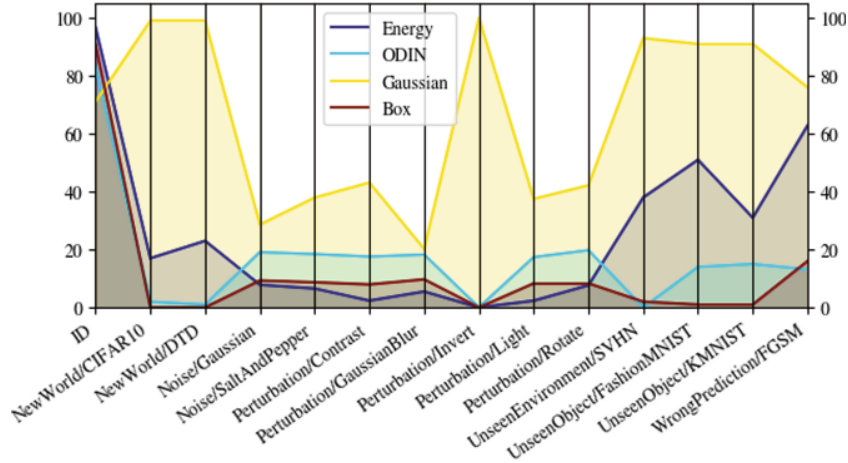
Perturbations	ASH-B [10]	ASH-P [10]	ASH-S [10]	DICE [42]	Energy [32]	Entropy [33]	Gauss [13]	GradNorm [23]	KL Matching [15]	KNN [43]	MDS [30]	Mahalanobis [39]	MaxLogit [50]	ODIN [31]	ReAct [41]	SHE [49]	Softmax [17]	Temperature [12]	VIM [45]
Gaussian	64	65	65	65	65	37	48	89	35	48	62	66	35	50	56	38	61	65	46
Contrast	45	41	41	41	41	56	44	20	56	42	64	49	59	50	51	57	46	41	50
Invert	28	21	21	21	21	47	0	0	39	0	100	100	79	43	92	88	56	21	0
Rotate	60	62	62	61	61	38	43	79	39	41	69	67	39	50	59	41	62	61	41
KMNIST	64	82	81	81	82	18	16	84	18	10	98	97	18	54	84	30	82	82	14

available monitors on a network trained on the MNIST dataset on a GPU and depict the AUROC score. The values of MDS and Mahalanobis can differ when switching between CPU and GPU; refer to [1] for details. The BOX monitor [20] is not included as it does not have a single threshold and, therefore, no AUROC score can be computed. The table shows the ranking of the monitors for the detection of Gaussian noise, increased contrast, color inversion, rotation, and a new, albeit similar dataset (KMNIIST). A darker color indicates a better ranking. One can see that there is barely any common behavior among the monitors. For example, while GRADNORM performs best on Gaussian noise, it performs worst on inverted images.

This also shows that it is important for the user to define her goal for the monitor. Not every monitor will be great at detecting a particular type of OOD, and she must carefully choose the right monitor for her setting. MONITIZER eases this task. In addition, it highlights the need for a clear evaluation of new monitoring methods in scientific publications.

We illustrate further features of MONITIZER using the following four monitors: ENERGY [32], ODIN [31], BOX [20], and GAUSSIAN [13]. The first two were proposed by the machine-learning community, and the latter two by the formal methods community.

The output produced by MONITIZER in the form of tables and plots (depicted in Fig. 5) helps the user see the effect of the choice of monitor, chosen objective, and dataset on the monitor’s effectiveness. MONITIZER allows users to experiment with different choices and select the one suitable for their needs. Figure 5 shows the evaluation of the mentioned monitors with the MNIST dataset as ID



**Fig. 5.** The monitor templates were optimized on MNIST as ID and for detecting New-World / CIFAR-10 as OOD while keeping 70% accuracy on ID. All monitors were optimized randomly.

data and an optimization with the goal of detecting pre-selected images of the CIFAR-10 dataset as those are entirely unknown to the network. The optimization was performed randomly. This resulted in the GAUSSIAN monitor only correctly classifying around 70% of ID data, whereas the other monitors have higher accuracy on ID data. Consequently, the other monitors perform worse than the GAUSSIAN monitor in detecting OOD data, as there is a tradeoff between good performance on ID and OOD data. This highlights the necessity of proper optimization for each monitor. See [1] for a detailed evaluation where we report on the experiments with different monitors, optimization objectives, and datasets.

Our experiments show that different monitors have different strengths and limitations. One can tune a monitor for a specific purpose (e.g., detecting a particular OOD class with very high accuracy); however, this affects its performance in other OOD classes.

## 5 Conclusion

MONITIZER is a tool for automating the design and evaluation of NN monitors. It supports developers of new monitoring techniques, potential users of available monitors, and researchers attempting to improve the state of the art. In particular, it optimizes the monitor for the objectives specified by the user and thoroughly evaluates it.

MONITIZER provides a library of 19 monitors, accompanied by 9 datasets and 15 NNs (at least one for each dataset), and three optimization methods (random, grid-search, and gradient descent). Additionally, all these inputs can be easily customized by a few lines of Python code, allowing a user to provide their monitors, datasets, and networks. The framework is extensible so that the user can implement their custom optimization methods and objectives.

MONITIZER is an open-source tool providing a freely available platform for new monitors and easing their evaluation. It is publicly available at <https://gitlab.com/live-lab/software/monitizer>.

**Data Availability Statement.** A reproduction package including all our results is available at Zenodo [2].

## References

1. Azeem, M., Grobelna, M., Kanav, S., Křetínský, J., Mohr, S., Rieder, S.: Monitizer: Automating design and evaluation of neural network monitors. CoRR (2024). <https://arxiv.org/abs/2405.10350>
2. Azeem, M., Grobelna, M., Kanav, S., Křetínský, J., Mohr, S., Rieder, S.: Reproduction package for article ‘monitizer: automating design and evaluation of neural network monitors. In: Proceedings of CAV 2024, Zenodo (2024). <https://doi.org/10.5281/zenodo.10933013>
3. Bai, H., Canal, G., Du, X., Kwon, J., Nowak, R.D., Li, Y.: Feed two birds with one scone: exploiting wild data for both out-of-distribution generalization and detection. In: ICML 2023. PMLR, vol. 202, pp. 1454–1471. PMLR (2023), <https://proceedings.mlr.press/v202/bai23a.html>
4. Bai, J., Lu, F., Zhang, K., et al.: ONNX: Open neural network exchange (2019). <https://github.com/onnx/onnx>
5. Breitenstein, J., Termöhlen, J., Lipinski, D., Fingscheidt, T.: systematization of corner cases for visual perception in automated driving. In: Proceedings of IV, pp. 1257–1264. IEEE (2020). <https://doi.org/10.1109/IV47402.2020.9304789>
6. Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: a principled case study. In: Proceedings of CAV, pp. 219–231. Springer (2022). [https://doi.org/10.1007/978-3-031-13185-1\\_11](https://doi.org/10.1007/978-3-031-13185-1_11)
7. Cheng, C., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. In: Proceedings of DATE, pp. 300–303. IEEE (2019). <https://doi.org/10.23919/DATE.2019.8714971>
8. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , Vedaldi, A.: Describing textures in the wild. In: Proceedings of CVPR (2014). <https://doi.org/10.1109/CVPR.2014.461>
9. Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., Ha, D.: Deep learning for classical japanese literature. CoRR (2018). <https://doi.org/10.48550/arXiv.1812.01718>
10. Djurisic, A., Bozanic, N., Ashok, A., Liu, R.: Extremely simple activation shaping for out-of-distribution detection. In: Proceedings of ICLR. OpenReview.net (2023). <https://openreview.net/forum?id=ndYXTEL6cZz>
11. Ferreira, R.S., Arlat, J., Guiochet, J., Waeselynck, H.: Benchmarking safety monitors for image classifiers with machine learning. In: PRDC 2021, pp. 7–16. IEEE (2021). <https://doi.org/10.1109/PRDC53464.2021.00012>
12. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: Proc. ICML, pp. 1321–1330. PMLR (2017). <https://proceedings.mlr.press/v70/guo17a.html>

13. Hashemi, V., Křetínský, J., Mohr, S., Seferis, E.: Gaussian-based runtime detection of out-of-distribution inputs for neural networks. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 254–264. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-88494-9\\_14](https://doi.org/10.1007/978-3-030-88494-9_14)
14. Hashemi, V., Křetínský, J., Rieder, S., Schmidt, J.: Runtime monitoring for out-of-distribution detection in object detection neural networks. In: Proc. FM. LNCS, vol. 14000, pp. 622–634. Springer (2023). [https://doi.org/10.1007/978-3-031-27481-7\\_36](https://doi.org/10.1007/978-3-031-27481-7_36)
15. Hendrycks, D., et al.: Scaling out-of-distribution detection for real-world settings. In: Proc. ICML. PMLR, vol. 162, pp. 8759–8773. PMLR (2022). <https://proceedings.mlr.press/v162/hendrycks22a.html>
16. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. In: ICLR. OpenReview.net (2019). <https://openreview.net/forum?id=HJz6tiCqYm>
17. Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. In: Proc. ICLR. OpenReview.net (2017). <https://openreview.net/forum?id=Hkg4TI9xl>
18. Henriksen, P., Lomuscio, A.R.: Efficient neural network verification via adaptive refinement and adversarial search. In: Proceedings of ECAI. FAIA, vol. 325, pp. 2513–2520. IOS Press (2020). <https://doi.org/10.3233/FAIA200385>
19. Henriksson, J., et al.: Towards structured evaluation of deep neural network supervisors. In: Proceedings of AITest, pp. 27–34. IEEE (2019). <https://doi.org/10.1109/AITest.2019.00-12>
20. Henzinger, T.A., Lukina, A., Schilling, C.: Outside the box: abstraction-based monitoring of neural networks. In: Proceedings of ECAI, FAIA, vol. 325, pp. 2433–2440. IOS Press (2020). <https://doi.org/10.3233/FAIA200375>
21. Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., Igel, C.: Detection of traffic signs in real-world images: the german traffic sign detection benchmark. In: Proceedings of IJCNN. pp. 1–8. IEEE (2013). <https://doi.org/10.1109/IJCNN.2013.6706807>
22. Hsu, Y., Shen, Y., Jin, H., Kira, Z.: Generalized ODIN: detecting out-of-distribution image without learning from out-of-distribution data. In: Proceedings of CVPR, pp. 10948–10957. IEEE/CVF (2020). <https://doi.org/10.1109/CVPR42600.2020.01096>
23. Huang, R., Geng, A., Li, Y.: On the importance of gradients for detecting distributional shifts in the wild. In: NeurIPS, vol. 34, pp. 677–689 (2021), [https://proceedings.neurips.cc/paper\\_files/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html](https://proceedings.neurips.cc/paper_files/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html)
24. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: a calculus for reasoning about deep neural networks. FMSD **60**(1), 87–116 (2022). <https://doi.org/10.1007/s10703-021-00363-7>
25. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Proceedings of CAV. LNCS, vol. 11561, pp. 443–452. Springer (2019). [https://doi.org/10.1007/978-3-030-25540-4\\_26](https://doi.org/10.1007/978-3-030-25540-4_26)
26. Katz-Samuels, J., Nakhleh, J.B., Nowak, R.D., Li, Y.: Training OOD detectors in their natural habitats. In: Proc. ICML. PMLR, vol. 162, pp. 10848–10865. PMLR (2022). <https://proceedings.mlr.press/v162/katz-samuels22a.html>
27. Kirchheim, K., Filax, M., Ortmeier, F.: PyTorch-OOD: a library for out-of-distribution detection based on PyTorch. In: CVPR Workshops 2022, pp. 4350–4359. IEEE/CVF (2022). <https://doi.org/10.1109/CVPRW56347.2022.00481>

28. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
29. LeCun, Y., Cortes, C., Burges, C.: MNIST handwritten digit database **2**
30. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In: NeurIPS, vol. 31, pp. 7167–7177 (2018). <https://proceedings.neurips.cc/paper/2018/hash/abdeb6f575ac5c6676b747bca8d09cc2-Abstract.html>
31. Liang, S., Li, Y., Srikant, R.: Enhancing the reliability of out-of-distribution image detection in neural networks. In: Proceedings of ICLR. OpenReview.net (2018). <https://openreview.net/forum?id=H1VGkIxRZ>
32. Liu, W., Wang, X., Owens, J., Li, Y.: Energy-based out-of-distribution detection. NeurIPS **33**, 21464–21475 (2020). [https://proceedings.neurips.cc/paper\\_files/paper/2020/hash/f5496252609c43eb8a3d147ab9b9c006-Abstract.html](https://proceedings.neurips.cc/paper_files/paper/2020/hash/f5496252609c43eb8a3d147ab9b9c006-Abstract.html)
33. Macêdo, D., Ren, T.I., Zanchettin, C., Oliveira, A.L., Ludermir, T.: Entropic out-of-distribution detection. In: Proceedings of (IJCNN), pp. 1–8. IEEE (2021). <https://doi.org/10.1109/IJCNN52387.2021.9533899>
34. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The third international verification of neural networks competition (VNN-COMP 2022): Summary and results. CoRR (2022). <https://doi.org/10.48550/arXiv.2212.10376>
35. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.T.: PRIMA: general and precise neural network certification via scalable convex hull approximations. PACMPL **6**(POPL), 1–33 (2022). <https://doi.org/10.1145/3498704>
36. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning
37. Olber, B., Radlak, K., Popowicz, A., Szczepankiewicz, M., Chachula, K.: Detection of out-of-distribution samples using binary neuron activation patterns. In: Proceedings of CVPR, pp. 3378–3387. IEEE/CVF (2023). <https://doi.org/10.1109/CVPR52729.2023.00329>
38. Pinggera, P., Ramos, S., Gehrig, S., Franke, U., Rother, C., Mester, R.: Lost and found: detecting small road hazards for self-driving vehicles. In: Proceedings of IROS, pp. 1099–1106. IEEE (2016). <https://doi.org/10.1109/IROS.2016.7759186>
39. Ren, J., Fort, S., Liu, J., Roy, A.G., Padhy, S., Lakshminarayanan, B.: A simple fix to mahalanobis distance for improving near-ood detection. CoRR (2021). <https://doi.org/10.48550/arXiv.2106.09022>
40. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015). <https://doi.org/10.1007/s11263-015-0816-y>
41. Sun, Y., Guo, C., Li, Y.: ReAct: Out-of-distribution detection with rectified activations. In: NeurIPS. vol. 34, pp. 144–157 (2021). <https://proceedings.neurips.cc/paper/2021/hash/01894d6f048493d2cacde3c579c315a3-Abstract.html>
42. Sun, Y., Li, Y.: DICE: Leveraging sparsification for out-of-distribution detection. In: Proceeding of ECCV. LNCS, vol. 13684, pp. 691–708. Springer (2022). [https://doi.org/10.1007/978-3-031-20053-3\\_40](https://doi.org/10.1007/978-3-031-20053-3_40)
43. Sun, Y., Ming, Y., Zhu, X., Li, Y.: Out-of-distribution detection with deep nearest neighbors. In: Proceedings of ICML, pp. 20827–20840. PMLR (2022). <https://proceedings.mlr.press/v162/sun22d>
44. Tajwar, F., Kumar, A., Xie, S.M., Liang, P.: No true state-of-the-art? OOD detection methods are inconsistent across datasets. CoRR (2021). <https://doi.org/10.48550/arXiv.2109.05554>

45. Wang, H., Li, Z., Feng, L., Zhang, W.: ViM: Out-of-distribution with virtual-logit matching. In: Proceedings of CVPR, pp. 4921–4930. IEEE/CVF (2022). <https://doi.org/10.1109/CVPR52688.2022.00487>
46. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms
47. Yang, J., et al.: OpenOOD: benchmarking generalized out-of-distribution detection. In: NeurIPS (2022). [http://papers.nips.cc/paper\\_files/paper/2022/hash/d201587e3a84fc4761eadc743e9b3f35-Abstract-Datasets\\_and\\_Benchmarks.html](http://papers.nips.cc/paper_files/paper/2022/hash/d201587e3a84fc4761eadc743e9b3f35-Abstract-Datasets_and_Benchmarks.html)
48. Zhang, J., et al.: OpenOOD v1.5: Enhanced benchmark for out-of-distribution detection. CoRR (2023). <https://doi.org/10.48550/arXiv.2306.09301>
49. Zhang, J., et al.: Out-of-distribution detection based on in-distribution data patterns memorization with modern hopfield energy. In: Proceedings of ICLR (2022). <https://openreview.net/forum?id=KkazG4lgKL>
50. Zhang, Z., Xiang, X.: Decoupling maxlogit for out-of-distribution detection. In: Proceedings of CVPR, pp. 3388–3397. IEEE/CVF (2023). <https://doi.org/10.1109/CVPR52729.2023.00330>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



## **E Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neural Networks**

*Reprinted by permission from Springer Nature (License Number 5702360791464): Lecture Notes in Computer Science book series (LNCS, volume 12974) Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neural Networks, Vahid Hashemi, Jan Křetínský, Stefanie Mohr, and Emmanouil Seferis © 2021 Springer Nature Switzerland AG (2021)*

This paper has been published as a **peer-reviewed short conference paper**.

Vahid Hashemi, Jan Křetínský, Stefanie Mohr, and Emmanouil Seferis (2021). Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neural Networks. In: Feng, L., Fisman, D. (eds) Runtime Verification. RV 2021. Lecture Notes in Computer Science(LNCS), vol 12974, pp. 254-264. Springer, Cham.

DOI: [https://doi.org/10.1007/978-3-030-88494-9\\_14](https://doi.org/10.1007/978-3-030-88494-9_14)

### **Summary**

This paper introduces a lightweight and scalable method for Out-of-Distribution (OOD) detection of Neural Networks (NNs). We use the activation values of individual neurons across the network and model them as Gaussian distributions. Whenever a new input falls outside the expected range, the monitor raises an alarm. We have a different approach to previous work [HLS20] by allowing outliers and not using strict interval abstractions.

We validate our methods against traditional approaches by performing experiments on several standard datasets. We use different NN architectures and compare the monitor's performance on both In-Distribution (ID) and OOD inputs.

### **Contributions of the author**

Composition, discussion and revision of the entire manuscript. Discussion of all results presented in the paper. Discussion and development of the ideas, experimentation and evaluation. Significant contributions towards the overall design of the implemented approach.





# Gaussian-Based Runtime Detection of Out-of-distribution Inputs for Neural Networks

Vahid Hashemi<sup>2</sup> , Jan Křetínský<sup>1</sup> , Stefanie Mohr<sup>1</sup> ,  
and Emmanouil Seferis<sup>1,2</sup>

<sup>1</sup> Technical University of Munich, Munich, Germany  
[mohr@in.tum.de](mailto:mohr@in.tum.de)

<sup>2</sup> AUDI AG, Ettingerstr. 60, 85057 Ingolstadt, Germany

**Abstract.** In this short paper, we introduce a simple approach for runtime monitoring of deep neural networks and show how to use it for out-of-distribution detection. The approach is based on inferring Gaussian models of some of the neurons and layers. Despite its simplicity, it performs better than recently introduced approaches based on interval abstractions which are traditionally used in verification.

## 1 Introduction

Learning deep neural networks (DNN) [2] has shown remarkable success in practically solving a large number of hard and previously intractable problems. However, direct applications in safety-critical domains, such as automated driving, are hindered by the lack of practical methods to *guarantee their safety*, e.g. [3, 4]. This poses a serious problem for industrial adoption of DNN-based systems. Companies struggle to comply with safety regulations such as SOTIF [19], both due to lack of techniques to demonstrate safety in the presence of DNN as well as due to the actual lack of safety, e.g. accidents in automated cars due to errors in DNN-based perception system used [5].

One of the key requirements is the ability to detect novel inputs [20], for which the DNN has not been trained and thus the only responsible answer is “don’t know”. Such inputs are also called *out-of-distribution (OOD) examples* [10]. Whenever such inputs occur, an alarm should be raised announcing the unreliability of the current output of the DNN, so that rectifying actions can be taken. Various runtime monitors for this task have already been proposed recently. Cheng et al. [1] monitor which subsets of neurons in a given layer are activated for known inputs; whenever a very different subset is activated, an alarm is raised. Henzinger et al. [16] monitor activation values of neurons and envelop the tuples into hyper-boxes (multidimensional intervals) along the

---

This research was funded in part by the DFG research training group *CONVEY* (GRK 2428), the DFG project 383882557 - Statistical Unbounded Verification (KR 4890/2-1), the project *Audi Verifiable AI*, and the BMWi funded *KARLI* project (grant 19A21031C).

© Springer Nature Switzerland AG 2021

L. Feng and D. Fisman (Eds.): RV 2021, LNCS 12974, pp. 254–264, 2021.

[https://doi.org/10.1007/978-3-030-88494-9\\_14](https://doi.org/10.1007/978-3-030-88494-9_14)



program analysis tradition; whenever a very different tuple is observed (outside of the boxes), an alarm is raised.

In this short paper, we propose a very light-weight and scalable approach. Similarly to [16], we monitor the activation values. However, instead of discrete and exact enveloping, we learn a more continuous and fuzzy representation of the recorded experience, namely a Gaussian model of each monitored neuron. Whenever many neurons have sufficiently improbable activation values on the current input, we raise an alarm. Surprisingly, our simple monitor is equally or even more accurate than the similar state-of-the-art [16] even though we take no correlation of the activation values of different neurons into account and instead we monitor each of the neurons separately, in contrast to the multi-dimensional boxes of [16].

**Our Contribution** can be summarized as follows:

- We present a new and simple method for OOD detection based on Gaussian models of neuron activation values.
- We show that our method performs better than state-of-the-art techniques for out-of-distribution (OOD) detection.

**Related Work.** In our work, we focus on the detection of OOD-inputs, arguably [10] one of the major problems in AI safety.

*State of the Art.* A recent work by Henzinger et al. [16] is very similar to our approach. The authors consider the neuron activations of one layer for all samples of the training data. For each class in the dataset, they collect the activation vectors of the class samples, and cluster them using k-Means [17]. They increase the number of clusters successively, until the relative improvement drops below a given threshold  $\tau$ . For each cluster, they construct a box abstraction that contains all samples of that cluster. In the end, each class in the data corresponds to a set of boxes. Finally, during testing, they check whether the activation vector of a new sample is contained in one of the boxes of its predicted class; if not, they raise an alarm. This approach can be extended to more layers, by taking the element-wise boolean AND of the layer “decisions”. That is, an input is accepted if only if it is contained in the abstractions of all monitored layers. While the idea of looking at the activations of neurons in a layer is similar to our approach, the difference is in the detection of OOD samples. In contrast to using box-abstractions, we use Gaussian models. This reflects better the actual distribution of values of the neurons, as can be seen in Sect. 4.2.

*OOD-Detection.* Previous works have suggested, for example, using the maximum class probability or the entropy of the predicted class distribution as an OOD indicator [11], or training a classifier to distinguish clean and perturbed data, using ensembles of classifiers trained on random shuffles of the training data [12]. Besides, two popular approaches closely resemble the methods of runtime monitoring, namely ODIN [13] and the Mahalanobis-based detector [14]. ODIN first applies temperature scaling on the softmax outputs of a DNN to

reduce the standard DNN overconfidence, and then applies a small adversarial-like perturbation of the input. If after that the maximum class score is below some threshold, the sample is considered to be OOD.

In contrast, the detector of [14] measures the probability density of a test sample by using a distance-based classifier. Another line of work involves generative models for OOD detection, attempting to model the distribution of the data, such as in [15]. By definition, OOD detection runs at test time, and thus many proposed approaches can be viewed under this setting. Other related approaches include using Bayesian learning methods [9], which can output prediction uncertainties, DNN testing [3], which are methods attempting to find problematic inputs, or building DNN architectures that are robust by construction, for example using interval bound propagation, abstract interpretation, or other methods [6–8].

## 2 Preliminaries

### 2.1 Deep Neural Networks

DNNs come in various architectures suitable for different tasks, however, at the core, they are composed of multiple *layers* of computation units called *neurons*. The task of a neuron is to read an input, calculate a weighted sum, apply a function called the *activation function* on it and output the result, called the *activation value*. We number the layers  $1, 2, \dots, L$  where layer 1 is called the input layer, layers  $2, \dots, (L - 1)$  are called the hidden layers and layer  $L$  is called the output layer.

More formally, given an input  $\vec{x}$  to the DNN, we have:

$$\begin{aligned}\vec{h}^1 &= \vec{x} \\ \vec{h}^{l+1} &= \vec{\phi}^{l+1}(\vec{h}^l) \quad l = 2, \dots, L\end{aligned}$$

where  $\phi_i^l(\vec{x})$  defines the element-wise computation of the neurons  $i = 1, \dots, N_l$  in layer  $l$ . The details of the computation are not necessary to understand the following work.

DNN can perform various tasks, the most usual being classification and regression. Whereas the first type labels its input with a category from a finite subset of classes, the second type outputs non discrete but real values. We consider only classification DNNs in this work. Neuron activations are vectors of activation values produced by neurons in some layer of a DNN. It is generally believed that layers closer to the output encode more complex features. This result has been supported by our results, which can be seen in Table 1. We refer to  $h_i^l$   $i = 1, \dots, N_l$  as the activation of neuron  $i$  in layer  $l$ .

## 3 Our Solution Approach

In this section, we discuss our approach for synthesizing an OOD detector based on Gaussian models. In statistics, Gaussian models are used to model the behavior of data samples. We adapt this idea to model the behavior of a neuron by a Gaussian model.

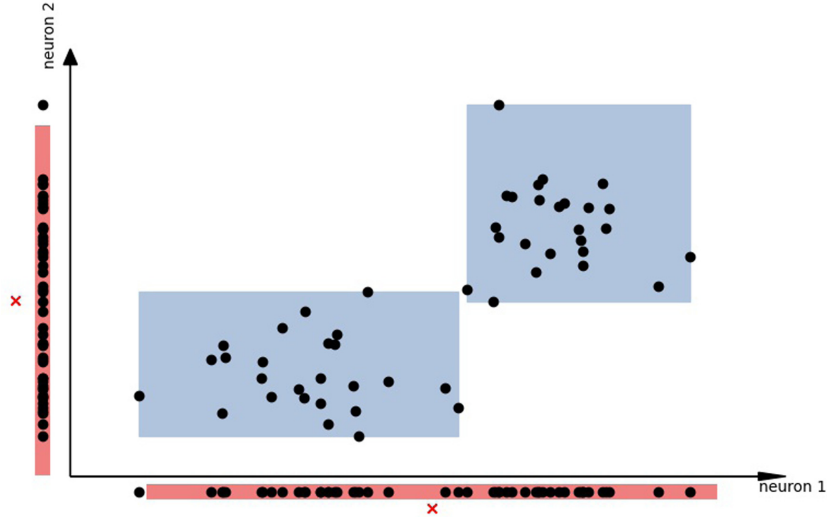
Consider a DNN as a classifier that distinguishes between  $\{c_1, \dots, c_{N_L}\} = C$  classes. One layer  $l$  of this DNN contains  $N_l$  neurons. For each class  $c_o \in C$  in the training data set, we feed samples  $x_j^{c_o}$ ,  $j = 1, \dots, m$ , into the network, and record the activations  $n_i^{c_o}(x_j)$  of each neuron for  $i = 1, \dots, N_l$ .

We collect those vectors  $\tilde{n}_i^{c_o}$ , and calculate the mean and standard deviation,  $\mu_i^{c_o}, \sigma_i^{c_o}$  of these values for each monitored neuron. We assume that the distribution of these values is approximately Gaussian. Thus, we expect the majority of samples to fall within the range  $[\mu_i^{c_o} - k\sigma_i^{c_o}, \mu_i^{c_o} + k\sigma_i^{c_o}]$ , where  $k$  is typically a value close to 2, containing 95% of the samples. During testing, we feed a new sample  $x$  to the DNN. We then do the following: we record the class  $c$  that our DNN predicts on  $x$ , and also retrieve the neuron activation values  $n_i^c(x)$ . We check if the activations of each neuron  $i$  falls within its range for the predicted class.

More formally, we check if

$$\forall i = 1, \dots, N_l : n_i^c(x) \in [\mu_i^c - k\sigma_i^c, \mu_i^c + k\sigma_i^c] \quad (1)$$

For a better understanding, we have the intuition depicted in Fig. 1. The data in the plot is random but shall give an idea of how the approach works. There are two neurons that output different values, which are depicted as black dots. On the one hand, they are shown in a 2D-plane, which is used for the abstraction of Henzinger et al.; on the other hand, they are shown projected onto one dimension next to the axes, for our approach. The approach of Henzinger et al. fits interval boxes to the values that the neurons can take. The interval boxes are drawn in blue. Our approach calculates intervals based on fitted Gaussians. The mean of the Gaussian is depicted as a red cross next to the neuron activations. The red line marks the interval that we consider as good for the neuron.



**Fig. 1.** This is an intuition of the Gaussian models on neuron activations. Black dots mark the values of the neurons. Once, in a 2D-plane together with the blue boxes that represent the abstraction of Henzinger et al., and once projected to one dimension only. The red lines mark the interval  $[\mu_i^c - k\sigma_i^c, \mu_i^c + k\sigma_i^c]$  for the two neurons respectively. Those intervals are the basis for our approach of OOD-detection. (Color figure online)

Each neuron “votes” independently if the new sample is valid or not. Samples within the distribution are expected to obtain a large number of votes, while OOD samples should obtain less. Thus, we collect the votes of all neurons, and then we compare them to a threshold; if they are below it, we consider  $x$  as an OOD sample, otherwise we consider it as correct. In that way, we can detect OOD inputs at runtime.

Note that this approach can also be extended to use multiple layers. For this, we compute the votes for each of the monitored layers. If they are below the threshold in at least one of the layers, we flag the sample as OOD.

An issue here is finding appropriate voting thresholds. For that, we use a suitable validation set. Normally, we should not make assumptions for the OOD data, and assume that we do not have access to them. In this case, we can use a suitable surrogate validation set, containing another unrelated dataset, e.g. adversarial examples or noisy images. In case we monitor more than one layer, the voting thresholds are computed individually for each layer.

## 4 Experiments

In this section, we analyze the experimental results of our approach. We will apply our approach for OOD detection to some example datasets and DNNs. We use the setting of Henzinger et al. [16], and we compare our result with theirs.

### 4.1 Datasets and Training

There are 4 datasets on which we evaluate our approach: MNIST, F-MNIST, CIFAR-10 and GTSRB (German Traffic Sign Recognition Benchmark) [18].

- MNIST is a dataset that contains images of digits. They shall be classified into ten classes, i.e. 0, ..., 9.
- F-MNIST consists of images of clothes, which shall also be classified into ten categories.
- CIFAR-10 is made of images of ten distinct classes from different settings.
- GTSRB contains images of German traffic signs that can be categorized into 43 classes.

All of the four datasets are used for classification. We train two different architectures of DNNs, NN1 and NN2, with the architectures of [16]. Those are:

- **NN1:** Conv(40), Max Pool, Conv(20), Max Pool, FC(320), FC(160), FC(80), FC(40), FC(k)
- **NN2:** BN(Conv(40)), Max Pool, BN(Conv(20)), Max Pool, FC(240), FC(84), FC(k)

Here, *FC* is a fully connected layer, *Conv* is a convolutional layer, *MaxPool* is  $2 \times 2$  max pooling, and *BN* is batch normalization. The activation function is always the RELU. NN1 was trained 10 epochs for MNIST, and 30 for F-MNIST, while NN2 was trained 200 epochs for CIFAR-10 and 10 for GTSRB. A learning

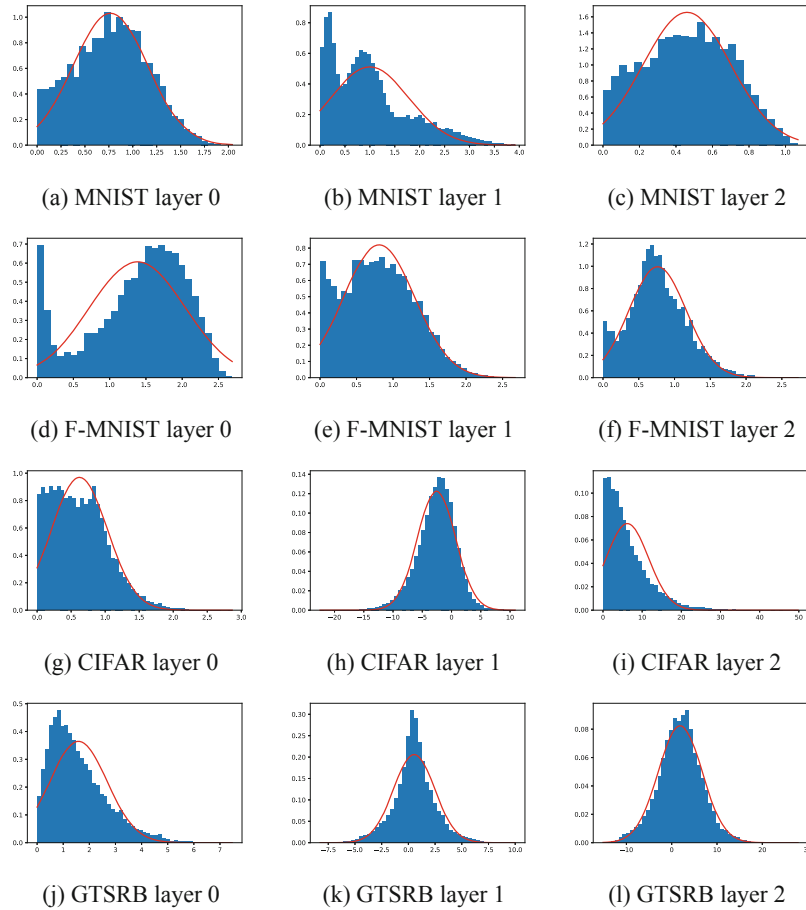
rate of  $10^{-3}$  and batch size 100 were used during training. NN1 is used for MNIST and F-MNIST, while NN2 is used for CIFAR-10 and GTSRB.

The evaluation is performed on two measures: the detection rate (DTERR) and the false alarm rate (FAR). The detection rate counts how many samples were correctly marked as OOD out of all OOD inputs. The false alarm rate (also known as Type-1-error) counts how many samples were marked as OOD but are not OOD, out of all marked inputs.

## 4.2 Gaussian Assumption

In this work, we used Gaussian distributions in order to approximate the output of each neuron. To verify that this is a valid assumption, we show in the following the distribution of values of the neurons.

We pick each dataset and select one random neuron from one of the monitored layers. Then, we plot the histogram of that neuron’s output. We also show the Gaussian distribution we would expect to have, according to the measured  $\mu$  and  $\sigma$  (Fig. 2).



**Fig. 2.** Histogram of neuron outputs, along with the Gaussian distribution with the sample mean and variance.

We see that there are some small differences. For some neurons, the Gaussian assumption is very accurate, e.g. f, h, k, and l. For some other cases the histograms indicate a slightly different behavior, e.g. a, d, e, g, i. However, in general they show that the neuron’s outputs follow more a Gaussian behavior than a uniformly distributed one. It seems especially that the problem is rather that the parameters  $\mu$  and  $\sigma$  do not exactly fit the true underlying Gaussian. One could think of calculating the parameters differently, or even using other models in future. Overall, the assumption that the neuron’s outputs are Gaussian-like seems to be true.

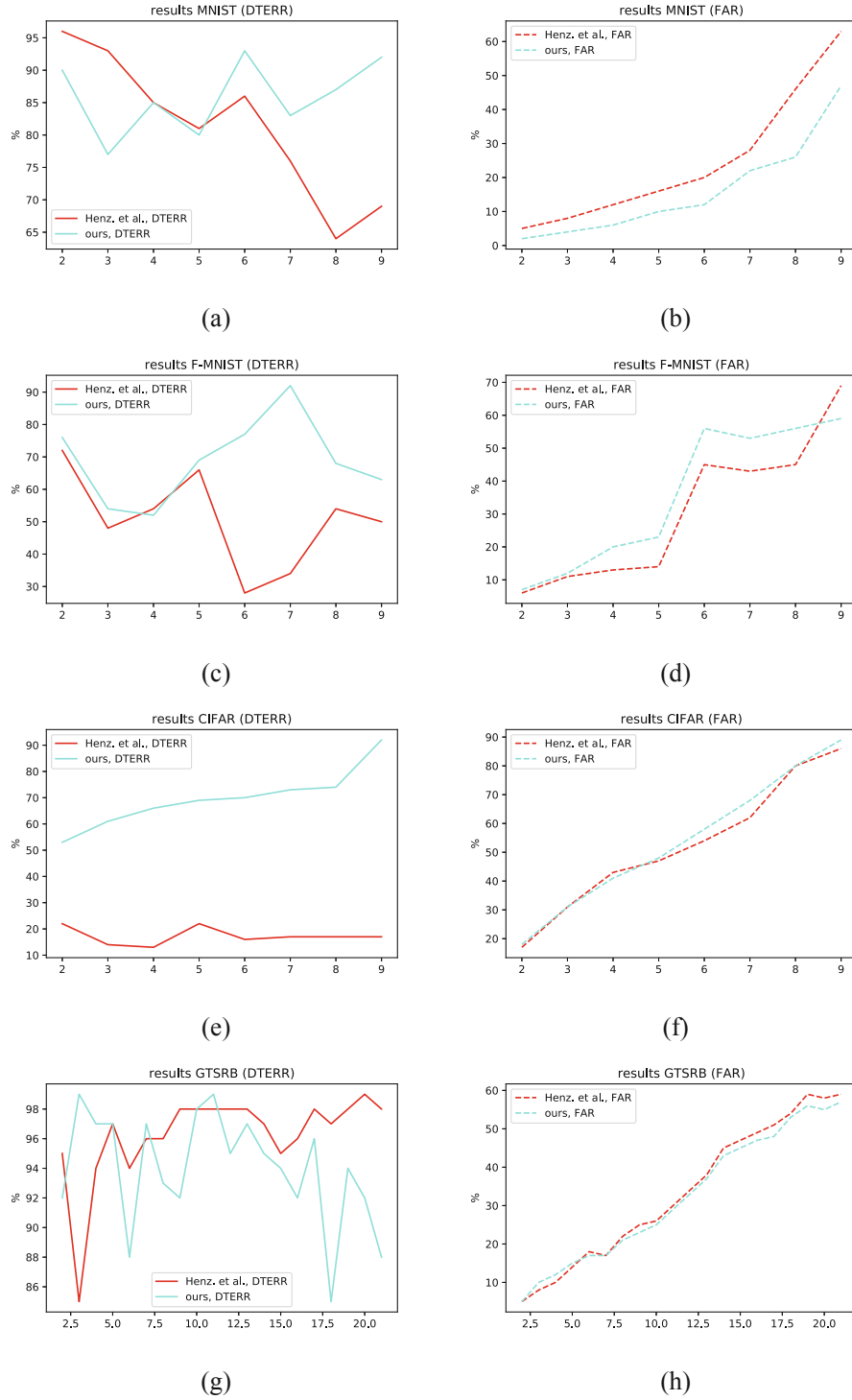
### 4.3 Evaluation Steps

Following the setting of Henzinger et al., we perform the following steps for each dataset: we train the DNN for the first  $k$  classes of the dataset, and consider the rest as OOD. This results, for example, in a DNN that was only trained on the digits from 0 to 5. Digits from 6 to 9 are considered as OOD. Having now constructed the networks and datasets in this way, we can apply our approach, and compare the results with the ones of Henzinger et al.. We monitor all linear hidden layers of the DNNs for both approaches. We use the interval  $[\mu - 2\sigma, \mu + 2\sigma]$  for each neuron and class label, while for Henzinger’s approach, we use the parameters mentioned in their paper. Note that the monitor of Henzinger et al. outputs boolean values (e.g.  $x$  is inside or outside of the boxes), while ours outputs numerical scores (e.g. number of “votes” for an input  $x$ ). In order to be able to compare the two approaches, we have to select a threshold for our approach, in order to convert its output to a boolean value (e.g.  $votes(x) < \tau \Rightarrow OOD$ ).

For this, we set the threshold at a quantile of the in-distribution data, so that the FAR is similar to the one of [16]. For example, for a quantile  $q = 50\%$ , we set the voting threshold in a way that 50% of the known in-distribution data pass through. Having set the FARs on a similar level, we can then compare the detected errors of the approaches.

In the case where we monitor more than one layer, we use the same quantile  $q$  in every one of them, and then combine votes as described before, i.e.  $x$  is accepted if the votes of each layer are above the corresponding threshold. Having a different quantile threshold for every layer improves performance, but might also be prone to overfitting. Note also that the threshold  $q$  is not the same across experiments: in each run, we modify it in order to match the FAR of [16] on that particular experiment. The results are shown in Fig. 3.

Each of the datasets has its own plot, where we have in red the values that the approach of [16] achieves, and in blue the values of our approach. For both of them, we measure the detection rate (DTERR) shown as a solid line in the left plot, and the false alarm rate (FAR) shown as a dashed line in the right plot. We see that the performance of our approach is mostly comparable or better than [16]. Especially, on CIFAR, our approach clearly outperforms the approach of Henzinger et al. in terms of the detection rate.



**Fig. 3.** Comparison between the approaches of Henzinger et al. and ours, for the cases of MNIST, F-MNIST, CIFAR, and GTSRB datasets. Here, *DTERR* and *FAR* are shown in separate diagrams. The number of classes on which the network was trained is depicted on the  $x$  axis.

Overall, our approach seems promising and shows already good results. However, we also have to indicate some problems with both approaches, namely the occasional low detection rate or high false alarm rates. This is problematic for industrial applications, and shows us the difficulties involved, and the need for stronger approaches.

#### 4.4 Parameter Study

In this section, we perform a study on the parameters of our approach. For simplicity, we focused on the MNIST dataset. The DNN in this case was thus NN1 with a total of eight layers. We want to particularly investigate the effects of the number of layers.

**Table 1.** Results on different layers, and different combination of layers. The evaluation is performed on the detection rate and the false alarm rate. Layers closer to the output layer show a higher detection rate than layers earlier in the DNN. The combination of several layers only results in a small improvement compared to the usage of only one layer.

Layers	<i>DTERR</i> (%)	<i>FAR</i> (%)	Layers	<i>DTERR</i> (%)	<i>FAR</i> (%)
5	44.6	8.7	(5,6)	70.3	8.3
6	69.2	5.3	(6,7)	77.0	6.5
7	73.0	5.4	(7,8)	76.4	6.8
8	73.5	6.4	(6,7,8)	79.5	7.5
			(5,6,7,8)	80.0	9.8

At first, we look at different layers in the DNN. The fifth layer seems to contain less important information in comparison to layer six, seven, and eight. When we only monitor layer five, the *DTERR* is almost 20% lower as for the other layers, while the *FAR* does not change significantly. We can thus verify the intuition that the features of the later layers in a DNN are more meaningful. If we combine the voting of several layers, we can see that the detection rate is slightly increased. Especially, the bad *DTERR* of 44% by only using layer five can be drastically improved by adding the knowledge of layer six, namely to 70%; while the *FAR* even decreases slightly. The combination of other layers can still increase the *DTERR* up to 80.0%, however, it comes with a slightly higher false alarm rate. Thus, for a more light-weight approach, it could be recommended to stick with fewer layers. Additionally, there may also be another different voting system for several layers, e.g. incorporate a weighted voting system for the layers and granting later layers more influence on the result.



## 5 Outlook

A natural next step would be to use additional information given by the correlation between the neurons. So far, we only considered the Gaussians of each neuron independently.

Instead, we can consider a subset of neurons  $n_k \in S$  and fit a joint Gaussian distribution  $N(\mu_S, \Sigma_S)$  on them. This subset can be an entire layer, where we fit a Gaussian distribution on the entire vector of layer activations, but it can also be a smaller subset of neurons. This offers the advantage of reduced computations, and an easier estimation of the covariance matrix (which is hard in high dimensions). The approach is flexible, and allows us to consider arbitrary subsets of neurons with varying sizes. Predictions can then be combined again by voting. For multidimensional Gaussian distributions, a simple threshold with  $\mu$  and  $\sigma$  is no longer possible. Instead, one can use the Mahalanobis distance,  $M^2(x) = (x - \mu)^T \Sigma^{-1} (x - \mu)$ , which is a notion of distance from the distribution center. A suitable threshold for  $M(x)$  is then to be calculated.

Besides, for a subset of neurons, a more precise model that can be used is a mixture of Gaussians. This might be more accurate since the Gaussian distributions as above are only imprecise approximations of the true distribution, while in contrast, Gaussian mixture models can approximate any probability distribution to any precision.

## 6 Conclusion

In this work, we considered the problem of runtime monitoring of DNNs, which forms an important step towards applying deep learning to safety-critical systems. Specifically, we focused on the sub-problem of OOD detection, and developed a lightweight detection method based on Gaussian models of neuron activation values. This can be extended in various ways as described before, and gives more accurate results than the recent work of Henzinger et al. [16]. Interestingly, the results suggest that reflecting correlation of the activation values (as in [16]) is less important than handling outliers through voting on learnt models (as here). Actually, the rigid and complete coverage by the boxes does not seem as adequate as the learnt approximations.

While we showed already a good efficiency on OOD inputs, the industrial requirements suggest that further improvements are necessary to reach real-world applicability. Our preliminary results invite further investigation along these directions. In particular, runtime monitoring by more complex probabilistic models, such as Gaussian mixtures, or using DNN-based probability estimation methods such as Normalizing Flows seem very promising.

## References

1. Cheng, C.-H., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. DATE (2019). <https://arxiv.org/abs/1809.06573>

2. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). <https://www.deeplearningbook.org/>
3. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks. CoRR (2018). <https://arxiv.org/abs/1812.08342>
4. Ortega, P., Maini, V.: Building safe artificial intelligence: specification, robustness, and assurance. Deep Mind blog (2018). <https://medium.com/@deepmindsafetyresearch/building-safe-artificial-intelligence-52f5f75058f1>
5. Wikipedia: List of self-driving car fatalities. Wikipedia article (2018). <https://en.wikipedia.org/wiki/List-of-self-driving-car-fatalities>
6. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: ICML (2018). <https://arxiv.org/abs/1711.00851>
7. Gowal, S., et al.: On the effectiveness of interval bound propagation for training verifiably robust models. In: NIPS (2018). <https://arxiv.org/abs/1810.12715>
8. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: ICML (2018). <https://files.sri.inf.ethz.ch/website/papers/icml18-diffai.pdf>
9. McAllister, R., et al.: Concrete problems for autonomous vehicle safety: advantages of bayesian deep learning. In: IJCAI (2017). <https://www.ijcai.org/Proceedings/2017/661>
10. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety. CoRR (2016). <https://arxiv.org/abs/1606.06565>
11. Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. In: ICLR (2017). <https://arxiv.org/abs/1610.02136>
12. Lakshminarayanan, B., Pritzel, A., Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles. In: NIPS (2017). <https://arxiv.org/abs/1612.01474>
13. Liang, S., Li, Y., Srikant, R.: Enhancing the reliability of out-of-distribution image detection in neural networks. In: ICLR (2018). <https://arxiv.org/abs/1706.02690>
14. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In: NIPS (2018). <https://arxiv.org/abs/1807.03888>
15. Ren, J., et al.: Likelihood ratios for out-of-distribution detection. In: NeurIPS (2019). <https://arxiv.org/abs/1906.02845>
16. Henzinger, T.A., Lukina, A., Schilling, C.: Outside the box: abstraction-based monitoring of neural networks. In: ECAI (2020). <https://arxiv.org/abs/1911.09032>
17. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
18. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German traffic sign recognition benchmark: a multi-class classification competition. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 1453–1460 (2011)
19. ISO/PAS 21448. Road vehicles - Safety of the intended functionality. <https://www.iso.org/obp/ui/#iso:std:70939:en>
20. Pimentel, M.A.F., Clifton, D.A., Clifton, L.A., Tarassenko, L.: A review of novelty detection. Signal Process. **99**, 215–249 (2014)

## **F** Predicting stream water temperature with artificial neural networks based on open-access data

*Licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.*

This paper has been published as a **peer-reviewed journal paper**.

Konstantina Drainas, Lisa Kaule, Stefanie Mohr, Bhumika Uniyal, Romy Wild, and Jürgen Geist: Predicting stream water temperature with artificial neural networks based on open-access data. In: Hydrological Processes 2023: Volume 37, Issue 10.

DOI: <https://doi.org/10.1002/hyp.14991>

### **Summary**

This work presents Neural Networks (NNs) for predicting stream water temperatures. We build a model that forecasts water temperature at various time scales, hourly and daily. Our models demonstrate high accuracy and better performance than some existing traditional methods. We show that NNs can handle non-linearity and complex hydrological variables, which significantly influence water temperature, much better.

Our findings suggest that NNs are a good environmental management and monitoring tool, offering a promising approach for water resource professionals to predict stream temperatures under changing climatic conditions. This capability is particularly crucial for preserving aquatic ecosystems sensitive to temperature fluctuations. Additionally, we show that using precise, high-quality input data is essential since this improves predictive accuracy.

Finally, we show that our analysis methods from Publication (B) help inspect NNs and show that it is important to treat them carefully.

### **Contributions of the author**

Composition and revision of the manuscript. Discussion and development of the ideas, and evaluation of the analysis methods.







Received: 22 February 2023 | Revised: 24 August 2023 | Accepted: 28 August 2023

DOI: 10.1002/hyp.14991

## RESEARCH ARTICLE

WILEY

# Predicting stream water temperature with artificial neural networks based on open-access data

Konstantina Drainas<sup>1</sup>  | Lisa Kaule<sup>2</sup>  | Stefanie Mohr<sup>3</sup>  | Bhumika Uniyal<sup>4</sup>  | Romy Wild<sup>1</sup>  | Juergen Geist<sup>1</sup> 

<sup>1</sup>Aquatic Systems Biology, TUM School of Life Sciences, Technical University of Munich, Freising, Germany

<sup>2</sup>Department of Hydrology, Bayreuth Center of Ecology and Environmental Research (BayCEER), University of Bayreuth, Bayreuth, Germany

<sup>3</sup>Foundations of Software Reliability and Theoretical Computer Science, TUM School of Computation, Information and Technology, Garching, Germany

<sup>4</sup>Professorship of Ecological Services, Bayreuth Center of Ecology and Environmental Research (BayCEER), University of Bayreuth, Bayreuth, Germany

## Correspondence

Juergen Geist, Aquatic Systems Biology, TUM School of Life Sciences, Technical University of Munich, Freising, Germany.  
Email: [geist@tum.de](mailto:geist@tum.de)

## Funding information

Bayerisches Staatsministerium für Wissenschaft und Kunst; Deutsche Forschungsgemeinschaft, Grant/Award Number: GRK 2428

## Abstract

Predictions of stream water temperature are an important tool for assessing potential impacts of climate warming on aquatic ecosystems and for prioritizing targeted adaptation and mitigation measures. Since predictions require reliable baseline data, we assessed whether open-access data can serve as a suitable resource for accurate and reliable water temperature prediction using artificial neural networks (ANNs). For this purpose, we trained and tested ANNs in 16 small ( $\leq 1 \frac{m^3}{s}$ ) headwater streams of major types located in Bavaria, Germany. Between four and eight different combinations of input parameters were trained and tested for each stream ANN, based on data availability. These were air temperature (mean, minimum and maximum), day of the year, discharge, water level and sunshine duration per day. We found that the input combination with the highest accuracy (lowest RMSE) was stream-specific, suggesting that the optimal input combination cannot be generalized across streams. Using a reasonable, but random, input combination resulted in an increase in error (RMSE) of up to >100% compared to the stream-specific optimal combination. Hence, we conclude that the accuracy of water temperature prediction strongly depends on the availability of open-access input data. We also found that environmental parameters such as hydrological characteristics and the proportion of land use in the 5 m riparian strip and the entire catchment are important drivers, affecting the accuracy and reliability of ANNs. ANNs' prediction accuracy was strongly negatively related to river length, total catchment area and water level. High proportions of semi-natural and forested land cover correlated with a higher accuracy, while open-canopy land use types such as grassland were negatively associated with ANN accuracy. In conclusion, open-access data were found to be suitable for accurate and reliable predictions of water temperature using ANNs. However, we recommend incorporating stream-specific environmental information and tailor the combination of input parameters to individual streams in order to obtain optimal results.

## KEYWORDS

artificial neural networks, climate change, machine learning, open-access data, prediction, stream habitat quality, thermal stress, water temperature modelling

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *Hydrological Processes* published by John Wiley & Sons Ltd.

## 1 | INTRODUCTION

With rising atmospheric temperatures, climate change affects stream water temperatures due to the well-established relationship between air and water temperature (Crisp & Howson, 1982; Kothandaraman & Evans, 1972; Mohseni & Stefan, 1999; Webb et al., 2003). This is of particular relevance in small headwater streams, where the relatively low mean stream depth is highly influenced by surface energy fluxes (Leach et al., 2023), which are correlated with air temperature (e.g. solar radiation). Moreover, headwater areas govern processes further downstream and their coldwater spots provide important refugia for coldwater-dependent species in light of climatic change (Kuhn et al., 2021). Stream water temperature is naturally regulated by various drivers: meteorological (air temperature and net radiation), hydro(geo)logical (discharge and groundwater inflow), hydromorphological (stream width and depth), and vegetational, the latter determining shading and evapotranspiration. There are a number of anthropogenic activities that influence variables such as discharge and flow variation, the proportion of surfaces rendered impervious by urbanization, changes in ice cover, and thermal pollution, which can additionally affect the water's thermal properties and natural temperature regimes on a large spatial scale (Caldwell et al., 2015; Nelson & Palmer, 2007). As one of the most common forms of anthropogenic disturbance to ecosystems, land use plays a key role in water temperature regulation, for example due to the partitioning of precipitation into infiltration and surface runoff, which affects water regimes on a (sub-)catchment scale.

Since temperature is the most crucial determinant of abiotic and biotic processes, further anticipated changes in stream water temperature due to global warming and other human impacts are expected to have substantial effects on aquatic ecosystems (Smith, 1981). This not only includes a decreased saturation concentration of oxygen triggered by global warming (Piatka et al., 2021), but also changes in viscosity, vapour pressure, density, and surface tension (Caissie, 2006). Additionally, temperature controls a wide range of biological processes, such as the decomposition rate of organic matter, species composition in aquatic communities, biotic interactions, and energy transfer in aquatic food webs (Woodward et al., 2010). The rapid pace of global warming (IPCC, 2022) creates a need for more detailed predictions of future water temperatures in streams. These are urgently required to enable an assessment of the potential impacts of climate warming on the abiotic stream environment and the consequences for biological communities. An understanding of this is also key to targeting and prioritizing mitigation and adaptation measures.

The importance of predicting stream water temperatures is reflected by the variety of approaches that have already been tested. As stated in Rabi et al. (2015), water temperature prediction models can generally be divided into two major categories: deterministic and statistical. Statistical models are in turn differentiated into parametric and non-parametric ones (for definitions, see e.g. Rabi et al. (2015) or Benyahya et al. (2007)). The availability of data for deterministic models, such as SHADE (Chen et al., 1998) or CEQUEAU (St-Hilaire et al., 2000), is problematic, as many variables are required for catchment and thermal representations, along with complete time series for discharge and meteorological parameters. While parametric statistical models have much lower data requirements

and are simple to use, their structure is specified from the start and hence not flexibly adjustable to the data (Benyahya et al., 2007). This limitation can lead to incorrect water temperature predictions when using linear regression, a technique often applied to describe the relationship between air and water temperature (Ahmadi-Nedushan et al., 2007; Crisp & Howson, 1982; Harvey et al., 2011; Krider et al., 2013; Rabi et al., 2015; Smith, 1981; Webb et al., 2003). At elevated and low air temperatures, physical effects lead to non-linearity (Mohseni & Stefan, 1999), which is beyond the limits of linear regression analysis.

In attempting to deal with the above challenges, the non-parametric statistical approach of Artificial Neural Networks (ANNs) is increasingly popular and has displayed equal or even higher accuracy (as evident from RMSE) than the majority of deterministic and parametric statistical models (Chenard & Caissie, 2008; Feigl et al., 2021; Hadzima-Nyarko et al., 2014; Pilgrim et al., 1998; Piotrowski et al., 2015; Rabi et al., 2015; Zhu et al., 2019,b,c). To the best of our knowledge, the smallest and hence best RMSE values reported for water temperature prediction using ANNs ranged between 0.46°C (Zhu, Heddum, Nyarko, et al., 2019) and 1.58°C (Hadzima-Nyarko et al., 2014). In the following, we refer to this “state of the art” range as “sota-range”.

Besides performance, a major benefit of using ANNs compared to deterministic models are the lower data requirements. While deterministic models require large amounts of data for predictions of water temperature, ANNs have already displayed good results with comparably limited information. It is currently unknown which input parameters produce optimal results, and so their selection varies in different studies. While air temperature is a key input parameter, its format varies greatly, as do the additional input parameters, particularly those concerning the temporal resolution of the data. Several studies used only daily mean air temperatures or once-a-day-measurements (Graf et al., 2019; Hadzima-Nyarko et al., 2014; Qiu et al., 2020; Rabi et al., 2015; Zhu et al., 2019,b), while others are based on daily mean, minimum and maximum air temperatures (Chenard & Caissie, 2008; Feigl et al., 2021; Piotrowski et al., 2015). Most studies used discharge or water level (Chenard & Caissie, 2008; Feigl et al., 2021; Qiu et al., 2020; Zhu et al., 2019,b,c) and/or the day of the year as an additional input (Chenard & Caissie, 2008; Feigl et al., 2021; Hadzima-Nyarko et al., 2014; Qiu et al., 2020; Zhu et al., 2019,b,c), while only one study additionally used global radiation (Feigl et al., 2021) and one the declination of the sun (Piotrowski et al., 2015).

Various measures can be obtained to determine the quality of a prediction, the most prominent one being accuracy. Accuracy is an indicator of how exactly an ANN predicts the output in the context of training and testing. However, climate change and natural variability involve data variations that ANNs might not be sufficiently capable of learning, since data obtained for training and testing cannot be used to represent future climatic developments. It is therefore not sufficient to rely solely on accuracy to determine the suitability of an ANN for its task. For classification networks, there are several methods available that provide more insight into the behaviour of ANNs (for examples, see Bach et al., 2015; Baehrens et al., 2010; Erhan et al., 2009; Huang et al., 2020; Simonyan et al., 2013; and Sundararajan et al., 2017). However, the prediction of water temperature is not a classification but a regression problem. In the field of regression problems, Mohr et al. (2021), to

the best of our knowledge, were the first to develop a methodology, able to give insight into the behaviour of regression models and to measure their behaviour not only on the basis of accuracy calculations but also as a means of examining reliability. Consequently, we included these methods in our study to enable a more holistic picture of water temperature ANN performance.

While determining accuracy and reliability is important for understanding how much trust can be placed in a prediction, these measures do not fully explain the disturbances found in the predictions. Variations in environmental conditions and the land use form surrounding streams are highly relevant for explaining the behaviour of ANNs, how they are influenced by environmental factors, and which conditions allow for a reasonable use of ANNs for predicting water temperature.

Hence, in this study, we address whether it is possible to train accurate and reliable ANNs based on open-access data for small ( $\leq 1 \frac{m^2}{s}$ ) headwater streams in Bavaria, Germany. Additionally, we address whether an optimal combination of input parameters exists and whether these combinations are unique for each stream or can be generalized across streams. To confine the range of environmental conditions in which ANNs can be optimally applied, we studied how environmental parameters such as stream length, hydrological characteristics and proportion of land use types affect ANN accuracy. We hypothesized that open-access data suffices to predict water temperature in small headwater streams with an RMSE in the sota-range. We also hypothesized that the accuracy and reliability of ANNs are influenced by both the input combinations and the environmental parameters of the streams, which would make the optimal input combination stream-specific.

## 2 | MATERIALS AND METHODS

### 2.1 | Study sites

For this study, we investigated 16 streams in major eco-regions of different geological origins throughout Bavaria, Germany. Figure 1 shows the locations of the gauging stations by Gewaesserkundlicher Dienst Bayern (abbr. GkD) for each stream. Figures 2 and 3 depict water temperature time series that were available for each of the 16 gauging stations. We selected streams with a mean annual discharge of  $\leq 1 \frac{m^3}{s}$ , based on open-access data from GkD. Using this criterion, we were able to focus on headwater streams, which are of special interest since they also govern processes further downstream.

### 2.2 | Measures of model performance

To assess the ANNs' performance, we used three different accuracy metrics as described in the following and three newly developed reliability metrics from Mohr et al. (2021) as described in Appendix A.3. Our aim was to prioritize the used accuracy metrics according to their expressive power regarding the reliability of ANNs. Therefore, we conducted correlation analysis (see description in Section 2.6.2) to

identify connections between accuracy and reliability metrics. In the following, we describe the three used accuracy metrics and define them in Formulas 1, 2, and 3.

**RMSE:** According to Moriasi et al. (2007), the root mean square error (for RMSE, see Formula 1) is an error index commonly used in the context of model evaluation. A value of 0 indicates a perfect fit.

We chose this metric since it is regularly used in the context of water temperature predictions with ANNs and is intuitively well understandable.

**R:** The Pearson's product-moment correlation coefficient (for R, see Formula 2) describes the degree of collinearity between predicted and observed data (Rabi et al., 2015). It ranges from  $-1$  to  $1$ , where  $0$  indicates no linear relationship and  $-1$  or  $1$  indicate a linear relationship. In this study, we aimed for a positive correlation between the observed and predicted values that is, values close to  $+1$ .

As the RMSE, this metric is regularly used in the context of water temperature prediction with ANNs but in the contrary does not show the mean error, but the degree of collinearity.

**PBIAS:** According to Gupta et al. (1999) as cited in Moriasi et al. (2007), the Percent bias (PBIAS, see Formula 3) shows whether the predictions are, on average, over- or underestimated. A value of  $0$  indicates a perfect fit, while positive values indicate underestimation and negative ones indicate overestimation.

This metric is uncommon in the field of water temperature prediction with ANNs but opens up a new perspective, since the direction of prediction inaccuracies (over- or underestimation) is displayed. On the contrary, the other two metrics concentrate, in general, on the amount of difference between observation and prediction.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (O_i - P_i)^2}, \quad (1)$$

$$R = \frac{\sum_{i=1}^n (O_i - \bar{O}) (P_i - \bar{P})}{\sqrt{\sum_{i=1}^n (O_i - \bar{O})^2 \sum_{i=1}^n (P_i - \bar{P})^2}}, \quad (2)$$

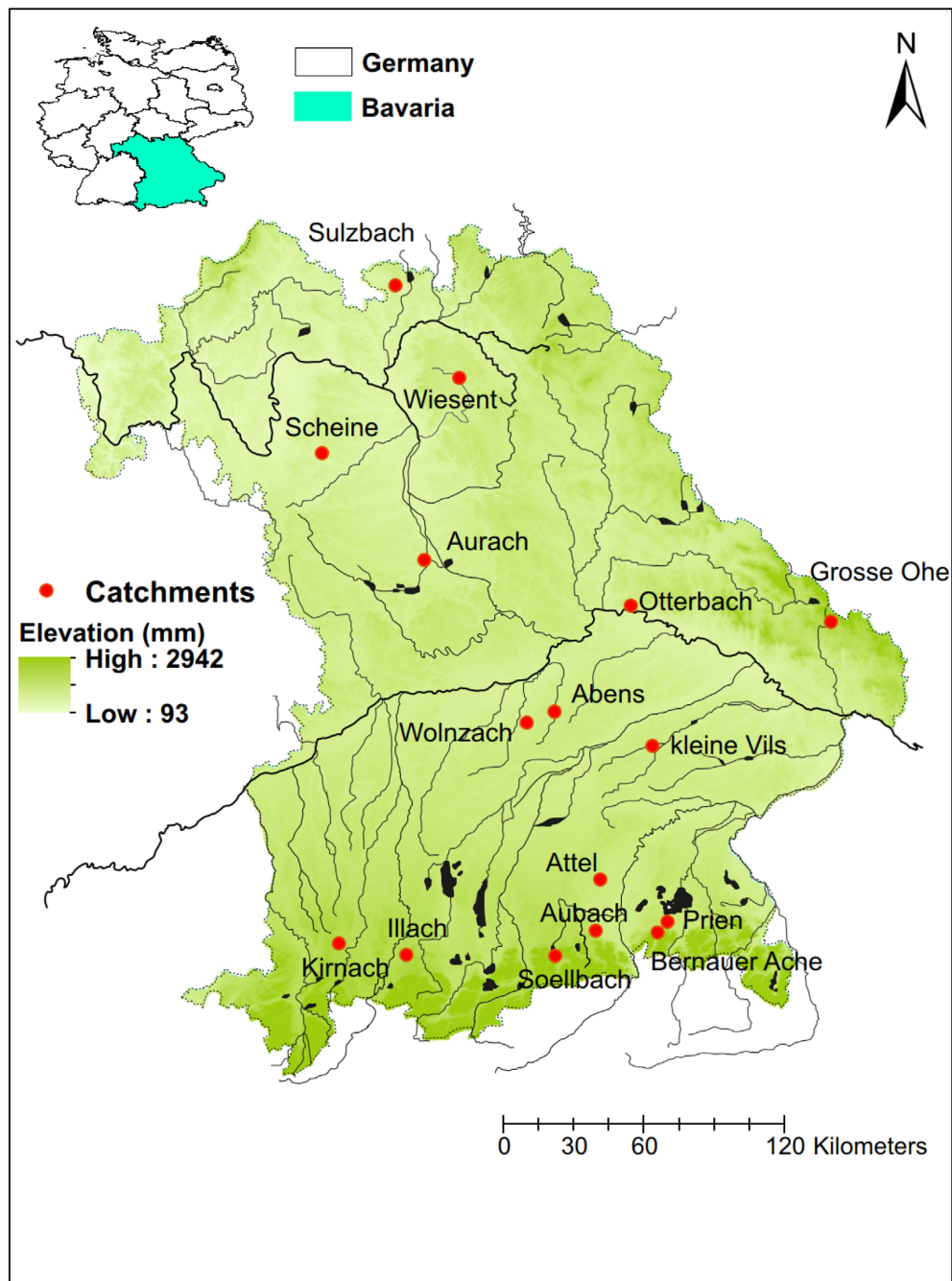
$$PBIAS = \frac{\sum_{i=1}^n (O_i - P_i) \times 100}{\sum_{i=1}^n (O_i)}. \quad (3)$$

To define the evaluation metrics, we followed the notation by Rabi et al. (2015), where  $P_i$  is the  $i$ th predicted water temperature value,  $O_i$  is the  $i$ th observed water temperature value,  $\bar{P}$  is the average of  $P_i$ ,  $\bar{O}$  is the average of  $O_i$  and  $n$  is the size of the dataset.

### 2.3 | Input

The data basis for the ANNs consisted of open-access data supplied by the GkD and Germany's National Meteorological Service (DWD). The data used for each stream consisted of the daily mean water temperatures ( $^{\circ}C$ ), daily discharges ( $Q$ ,  $m^3/s$ ) and daily mean water levels



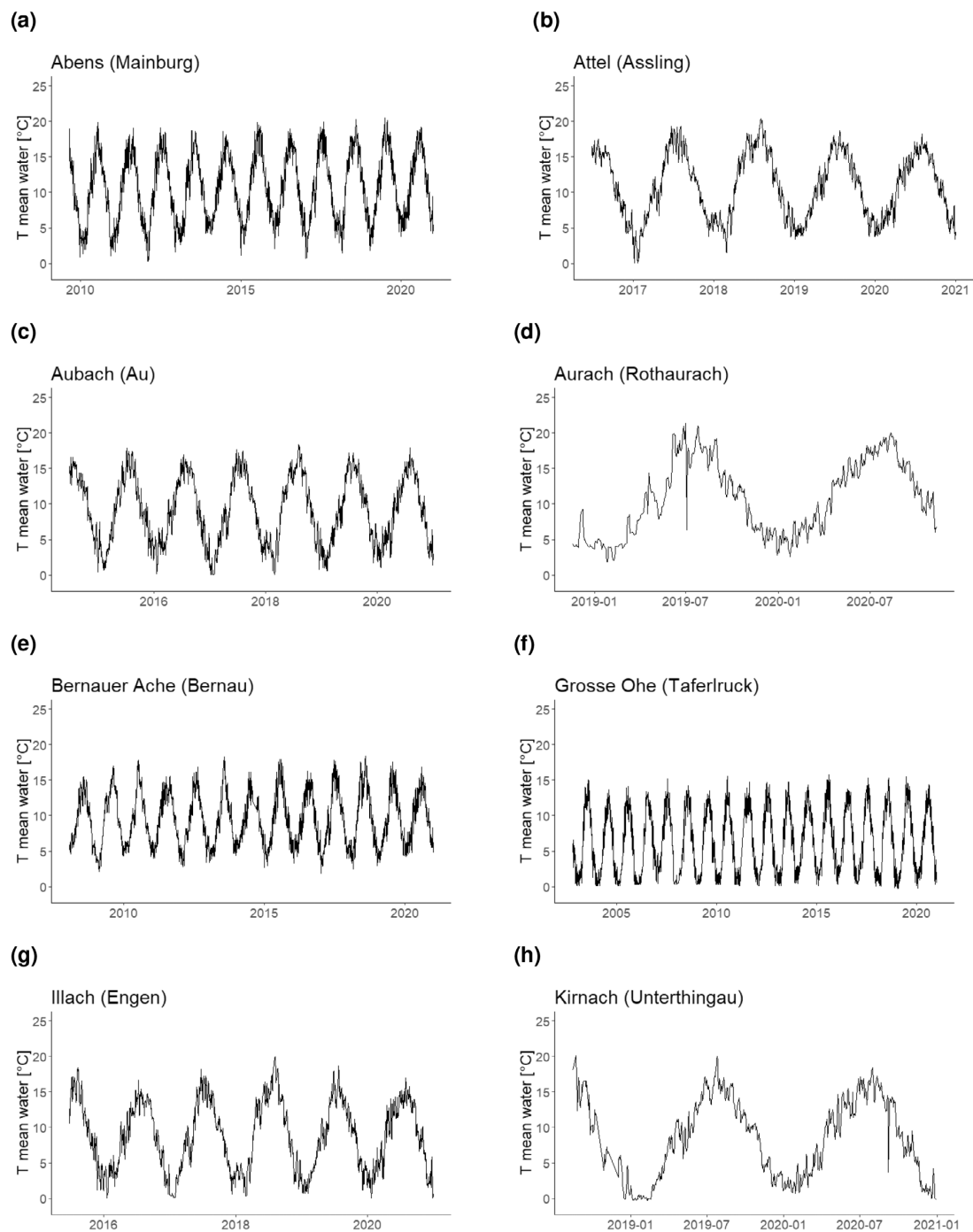


**FIGURE 1** Location of GkD gauging stations throughout Bavaria, Germany.

( $L$ , cm), as obtained from each of the GkD gauging stations. Additionally, the daily minimum, maximum and mean air temperatures ( $T$ , °C) and the daily sunshine durations ( $S$ , defined by DWD as duration of direct solar radiation at a given location) were derived from the two closest DWD gauging stations for each stream. All data sets carried a time stamp, which we recalculated to obtain the day of year ( $D$ ) as

a continuous number. To improve the learning of our ANNs, we employed data normalization, which is a common technique used in machine learning (Han et al., 2011).

We trained and tested all ANNs by inputting data taken from four consecutive days, with the predicted fourth day's daily mean water temperature as the output. This amount of days was found to lead to



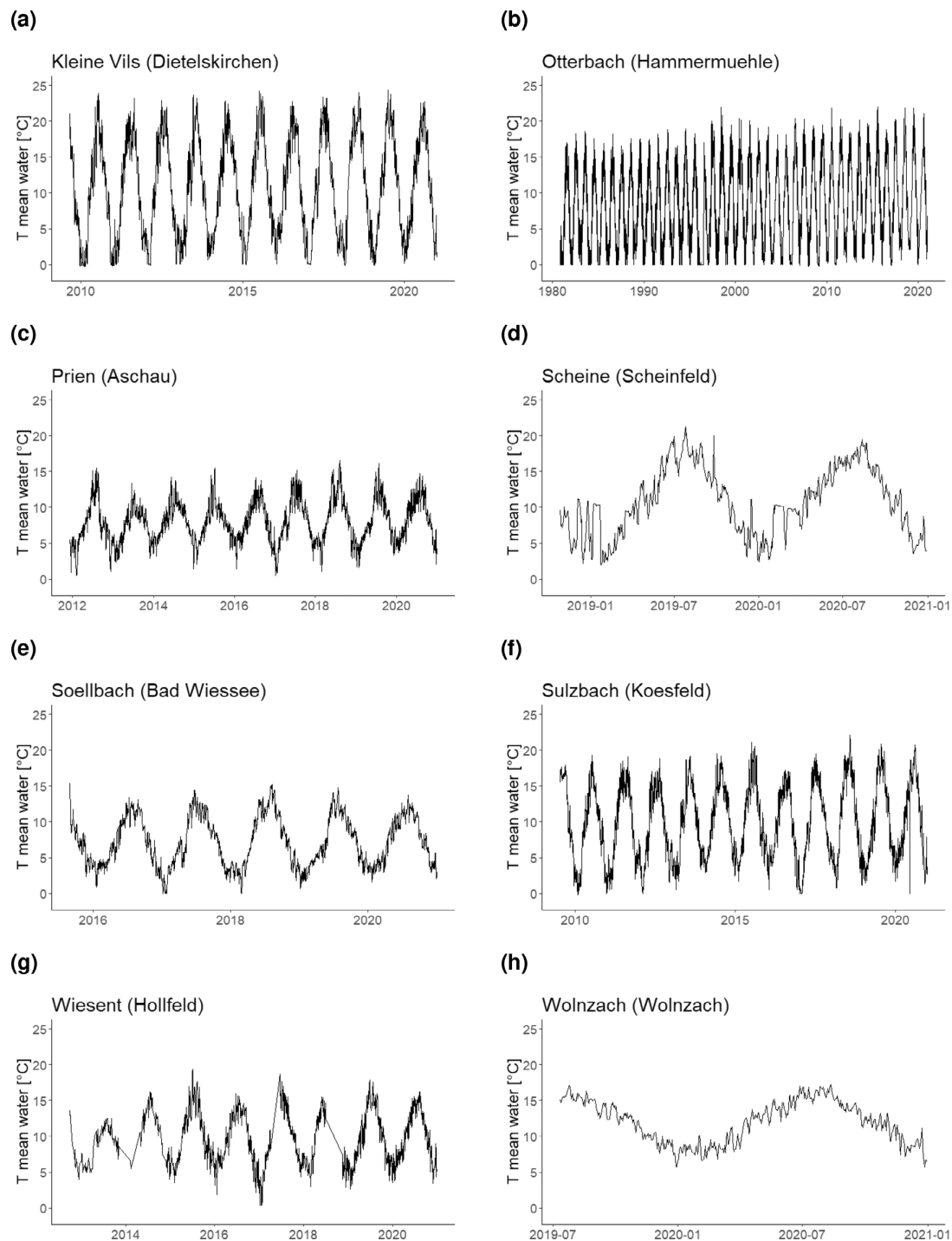
**FIGURE 2** Water temperature time series (l). (a) Abens. (b) Attel. (c) Aubach. (d) Aurach. (e) Bernauer Ache. (f) Grosse Ohe. (g) Illach. (h) Kirnach.

a better accuracy compared to ANNs with the input of 1, 2, 3, or 5 consecutive days at a case study in the Bavarian headwater catchment Mähringsbach (Drainas, 2020).

Since not all data were measured continuously, we chose time periods for each stream during which all the input

parameters were available (except for Scheine and Soellbach, for which no sunshine data was available). The data used for each stream, the DWD stations used for each GkD gauging station, and the distances between them are presented in Appendix A.4 in Table A2.



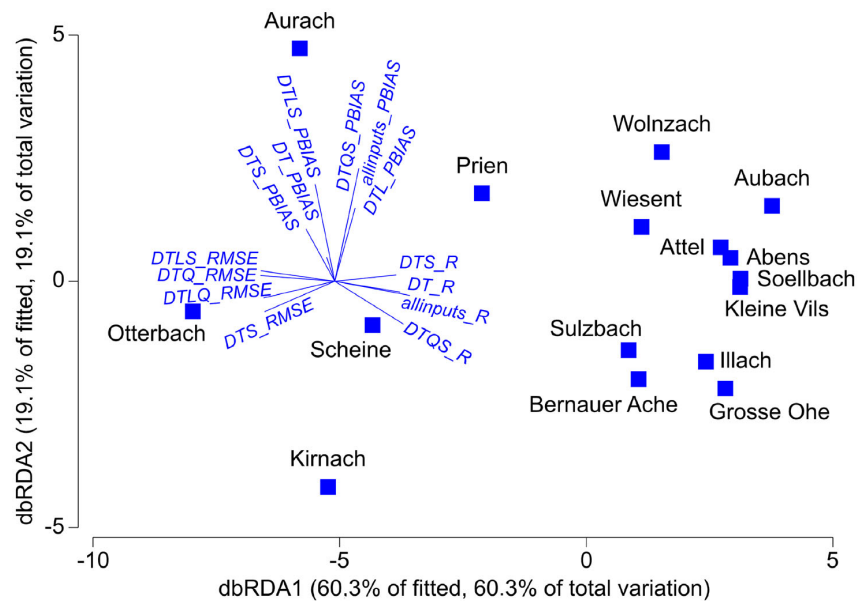


**FIGURE 3** Water temperature time series (II). (a) Kleine Vils, (b) Otterbach. (c) Prien. (d) Scheine. (e) Soellbach. (f) Sulzbach. (g) Wiesent. (h) Wolnzach.

To test the suitability of the different input parameters, we trained and tested ANNs with all possible combinations of input parameters for each stream. However, we rejected any combination with no air temperature or date (Mohseni & Stefan, 1999; Zhu,

Nyarko, Hadzima-Nyarko, et al., 2019). We used the following combinations (names indicate the input parameters as abbreviated above): *DT*, *DTS*, *DTQ*, *DTL*, *DTQS*, *DTLQ*, *DTLS* and *allinputs* (*DTQLS*).

**FIGURE 4** distLM-Eval-Eval-plot. Resemblance: D1 Euclidian distance. Strong correlation of RMSE and R along dbRDA axis 1, discrimination of streams by PBIAS on dbRDA axis 2. Input parameters: D, day of the year; T, air temperature; Q, discharge; L, water level; S, sunshine duration; DTQLS, allinputs; R, PBIAS, evaluation metrics as defined in Formulas 1, 2, 3.



## 2.4 | The modelling approach: Artificial Neural Networks

ANNs are a machine learning approach that is inspired by biological nervous systems (da Silva et al., 2017). They can create output information (in this study water temperature) based on given input information (in this study different input combinations). To do so, they need to learn the relationship between in- and output, for which they need to be trained. Hereby the ANNs have access to the input as well as to the output data and iteratively learn to predict the output based on the input. After the training, the ANNs' ability to predict the output is tested by only presenting the input and comparing the predicted output to the actual output information. In this study, this comparison was conducted by calculating the RMSE.

For the distribution into training and testing phase, the given dataset needs to be divided into a training and a testing dataset. In this study, this was done randomly in a ratio of 90% (training) to 10% (testing) as it is default for an established library (see Appendix A.2 for details).

For this study, we trained and tested nine ANNs for each of the 14 waterbodies, for which all input parameters were available. Additionally, we trained four ANNs for Scheine and four for Soellbach (no sunshine duration available), resulting in a total of 136 ANNs. The ANNs were determined by using search methods from the scikit-learn Python package (Pedregosa et al., 2011) (for a description of the search methods, see Appendix A.1). Optimization was based on the calculated root mean square error (RMSE), which we also employed in this work as an accuracy metric (see Formula 1).

## 2.5 | Examination of environmental parameters

To determine the influence of the environmental parameters on the prediction accuracy of ANNs, we examined the environmental parameters

of two different spatial scales for which an influence can be expected. For that, we used ArcGIS software to create riparian strips upstream of the 16 catchment outlets flanking 5 m left and right (measured from the middle of the stream) of the entire stream length (in the following referred to as "5 m riparian strip"). This width was chosen to capture the direct impact of the landuse on the stream, e.g. in the form of shading. We obtained the stream geometry and catchment spatial maps as input data from the Bavarian Environmental Agency (LfU) and used it to extract various catchment characteristics. Once we had extracted the spatial area of the riparian strips, we intersected the Corine Land Cover (CLC) map (European Union, 2021) with shape files representing the riparian strip and whole catchment area (in the following referred to as "entire catchment resolution") to integrate land use information. The CLC map comprises an inventory of land cover in 44 classes. CLC uses a minimum mapping unit (MMU) of 25 ha for areal phenomena and a minimum width of 100 m for linear phenomena. The MMU for mapping CLC status layers is 25 ha. This means that no objects of less than 25 ha can be present in the database; they are generalized into a neighbouring feature, resulting in >25 ha polygons. Additionally, the minimum mapping width (MMW) of linear elements is 100 m, which means that no objects (e.g., roads, rivers) of less than 100 m width are present in the database.

## 2.6 | Statistical data analysis

### 2.6.1 | Environmental dataset

To determine which environmental and land use variables were linked to ANN accuracy and reliability, we assembled an environmental data set which describes the features of the assessed streams on two spatial scales: the entire catchment and the 5 m riparian zone of the stream banks. The data set included the following environmental

parameters for the entire catchment resolution: total length of all tributaries merging into the final river branch (total river length); the length of the longest river branch of the catchment (longest river length); proportion of land use type in the catchment (agriculture, forest, grassland, semi-natural land use, urban and water surfaces); total catchment size; hydrologic parameters calculated over the entire measurement time span, namely mean water level (MW), highest measured water level (HW), lowest measured water level (NW), mean discharge (MQ) and highest measured discharge (HQ); the number of tributaries (tributaries); the number of days for which data was used (DOD); the number of input data points per output data point (IPO); the distance between the GkD station and the closest DWD station (Dist1) and between the GkD station and the second closest DWD station (Dist2).

Regarding the resolution of the 5 m riparian strip, only the land use variables were provided, along with the total riparian strip area.

In Appendix A.4, the description and results of a principal component analysis (PCA) are presented, showing the distribution of the 16 stream sites along the different environmental gradients captured in our environmental dataset.

## 2.6.2 | Correlation analysis

To determine connections between the prediction accuracy and the environmental parameters, we conducted correlation analysis. For this, we used the calculated accuracy metrics (RMSE, R and PBIAS) for each input combination and each waterbody, once for the entire catchment and once for the 5 m riparian strip resolution. First, we used the Shapiro–Wilk test to examine whether our datasets were normally distributed. We then tested the distribution both for each input combination separately and for each environmental parameter. We then conducted tests to examine the correlation between each input combination and each environmental parameter. If both datasets were normally distributed, we used Pearson product moment correlation. Otherwise, if one or both of the datasets was not normally distributed, we used Spearman rank-order correlation. To visualize the  $\rho$  and  $\text{corr}$  values, we created heatmaps. We conducted all steps of the correlation analysis with RStudio (RStudio Team, 2022; Warnes et al., 2020).

Additionally, we conducted correlation analysis to find connections between reliability and accuracy to assess which accuracy metric is most suitable for displaying the reliability of an ANN's predictions. Details are described in Appendix A.5.

## 2.6.3 | Distance-based linear model

As a multivariate approach, we used distance based Linear Model (DistLM), which is based on a procedure called “distance based redundancy analysis” (dbRDA) (Legendre & Anderson, 1999) and implements a routine that analyses and models the relationship between a multivariate resemblance matrix and a set of given predictor variables. DistLM is applied as a multivariate multiple regression that models the explanatory significance of the environmental predictor variables via

partitioning of variation that facilitates permutation-based significance testing. In our case, we first used the resemblance matrix of the RMSE, R, and PBIAS values of all calculated ANN input combinations and the same data as predictors, to reduce dimensionality. We chose this combination of data, to investigate which of the accuracy metrics and ANN input combinations explained most of the between-stream variability and to identify any redundancy in the three accuracy metrics (Eval-predict). In a subsequent approach, we used the same resemblance matrix but the environmental data set as predictor variables to determine the environmental variables that explain most of the observed variations in the multivariate data set of accuracy metrics of different ANNs (Enviro-predict). For both approaches, we used the DistLM function and redundancy analysis plots (dbRDA-plots) and chose the step-wise method and Adjusted  $R^2$  for model comparison in PRIMER v7 & PERMANOVA+ (Anderson et al., 2008).

## 3 | RESULTS

### 3.1 | Measures of model performance

To prioritize the use of the three different accuracy metrics applied in this study, we evaluated which of them is most suitable to also display the reliability of an ANN. Therefore, we conducted correlation analysis between the accuracy and the reliability metrics (for detailed results see Appendix B.2), which resulted in significant correlations between two of the reliability metrics and the RMSE and one significant correlation between reliability metrics and R and PBIAS each.

### 3.2 | Selection of ANNs and input parameters

The comparison of prediction accuracy of randomly searched ANNs (see RandomizedSearch in Appendix A.1) for all different input combinations (see Tables A3, A4, A5, and A6) showed that the optimal input combination was different for each of the tested streams (see Table 1).

The most frequently used combination of input parameters was DTL (38% of all streams), followed by DTLQ (25% of all streams), all-inputs (21%, if sunshine duration was available), DTQS (14%, if sunshine duration was available) and DTQ (6% of all streams) (Table 2 top). The combinations DT, DTS and DTLS were not selected as input combinations with the greatest predictive power in any of the streams. Consequently, the share of individual input parameters in the composition of ANNs was as follows: day of the year and air temperature were identified as input parameters in 100% of all streams, water level was identified in 81% of all streams, discharge was identified in 63% of all streams, and sunshine duration was identified in 36% of the streams for which sunshine duration was available (Table 2 bottom). Using the most accurate input combination for each stream based on the RMSE, the RMSE values ranged between 0.373°C (Aubach) and 1.667°C (Otterbach), R values ranged between 0.997 (Aubach) and 0.958 (Otterbach), and PBIAS values ranged between −0.767% (Kirnach) and 0.112% (Prien). Comparing the input combinations with the

**TABLE 1** Evaluation of the most suitable ANN for each waterbody, as determined by RandomizedSearch.

Stream	Inputs	RMSE	R	PBIAS
Prien	DTQS	0.733	0.969	0.112
Attel	allinputs	0.453	0.995	0.004
Aubach	DTLQ	0.373	0.997	0.033
Soellbach	DTL	0.419	0.993	−0.090
Bernauer Ache	DTLQ	0.586	0.988	−0.315
Kleine Vils	DTL	0.535	0.997	−0.105
Illach	allinputs	0.503	0.994	−0.136
Otterbach	DTLQ	1.667	0.958	−0.248
Wiesent	DTQS	0.623	0.985	0.158
Sulzbach	DTLQ	0.736	0.990	−0.286
Abens	allinputs	0.468	0.995	−0.084
Aurach	DTL	1.301	0.969	−0.113
Scheine	DTQ	0.920	0.980	−0.676
Grosse Ohe	DTL	0.483	0.994	−0.344
Kirnach	DTL	1.104	0.979	−0.767
Wolnzach	DTL	0.476	0.988	−0.048

Note: Column titles: Stream, name of examined stream; Inputs, input combination used; RMSE, R, PBIAS, evaluation metrics as defined in Formulas 1,2,3. Abbreviations: D, day of the year; DTQLS, allinputs; L, water level; Q, discharge; S, sunshine duration; T, air temperature.

highest accuracy according to the RMSE from each stream, with the combinations of lowest accuracy, the error increased on average by 41% when a random input combination was used, compared to the optimal combination, with a minimum of 5% (Otterbach) and a maximum of 102% (Scheine).

Given the number of ANNs and the accuracy metrics, a DistLM (Eval-predict) was calculated to determine which of the accuracy metrics and ANN input combinations explained the majority of the between-stream variability and to identify redundancy in accuracy metrics. The RMSE and R values were strongly correlated along dbRDA axis 1, implying that the accuracy of calculated ANNs was very similarly reflected by these two metrics (Figure 4). Both allinputs-ANNs of RMSE and R individually explained 58% of the total variability in the dataset according to marginal testing, and the sequential tests furthermore confirmed that the information of R and RMSE of allinputs-ANNs was redundant, as only one of them was included in the best-solution set of variables. However, the PBIAS values calculated on the basis of multiple input combinations were responsible for approximately 35% of the remaining variability in the data set and distinctly discriminated streams on dbRDA axis 2, hence showing that PBIAS provides information that cannot be substituted by the other two used accuracy metrics, and even streams with high accuracy measures, as shown by the small RMSE or high R values, can be subject to under- or overestimation in temperature predictions (Figure 4).

### 3.3 | ANN accuracy metrics

To determine connections between environmental parameters and ANNs' accuracy, we conducted correlation analyses in the spatial scales of entire catchment as well as 5 m riparian strip resolution.

#### 3.3.1 | Entire catchment

With regard to the entire catchment resolution, we observed the most statistically significant associations between accuracy metrics and environmental parameters for RMSE (34), followed by R (16) and PBIAS (6).

For RMSE (see Figure 5a), we detected significantly positive correlations between all ANN input combinations and total river length as well as between all ANN input combinations and the hydrologic parameters MW and HW. Additionally, four ANN input combinations correlated significantly positively with NW (DT, DTL, DTQ and DTLQ), three ANN input combinations correlated significantly positively with the longest river (DT, DTQ and DTLQ), and three ANN input combinations correlated significantly positively with Dist1 (DTQ, DTQS, DTLS). The R of all input combinations was significantly negatively related to total river length and longest river length (Figure 6a). PBIAS correlated significantly positively with total catchment area (DTLS) and Dist1 (DT) and significantly negatively with DOD (DTLS and DTS) and semi-natural land use (DTS) (Figure 7a).

#### 3.3.2 | 5 m riparian strip

Significant associations between accuracy metrics and environmental parameters in the 5 m riparian strip were most numerous for RMSE (26), followed by R (21) and PBIAS (2). The RMSE values of all ANN input combinations, with the exception of DTLS and DTS, were significantly positively correlated with total riparian strip area (Figure 5b). Also, grassland was significantly positively associated with all ANN input combinations except allinputs and DTQS. Semi-natural land use, in contrast was significantly negatively correlated with RMSE values

**TABLE 2** Top: Frequency of input combinations used for the best ANNs as depicted in Table 1. Bottom: Frequency of input parameters used in input combinations in Top.

Input combination	Frequency	Percentage
DTL	6 of 16	38
DTLQ	4 of 16	25
allinputs	3 of 14	21
DTQS	2 of 14	14
DTQ	1 of 16	6
DT	0 of 16	0
DTLS	0 of 14	0
DTS	0 of 14	0
Input Parameter	Frequency	Percentage
Day of the year (D)	16 of 16	100
Air temperature (T)	16 of 16	100
Water level (L)	13 of 16	81
Discharge (Q)	10 of 16	63
Sunshine duration (S)	5 of 14	36

Note: Sunshine duration was only available for 14 streams.

in DTLS, allinputs and DTQS. Also, the proportion of forest and water surface in the riparian-strip area correlated negatively with RMSE values.

Similarly, R values were significantly negatively related to the total riparian strip area (all ANN combinations except DTS and DTLS), negatively related to grassland (Figure 6b) and positively related to semi-natural land use.

For PBIAS values (see Figure 7b), we only observed a significant negative relationship with semi-natural land use (DT and DTS).

### 3.4 | Multivariate analysis of environmental predictors of ANN accuracy metrics

To investigate which of the accuracy metrics and ANN input combinations explained most of the between-stream variability and to identify any redundancy in the three accuracy metrics, we conducted a DistLM. We found that the 14 variables depicted in Figure 8 explained a total variation of  $R^2 = 0.99601$ , Adjusted  $R^2 = 0.94014$ . 60.5% of the 2-D configuration of the 12 streams was explained by dbRDA axis 1 and 19.05% by dbRDA axis 2.

The DistLM's marginal tests indicated a significant relationship between the multivariate configuration of the streams, based on the three accuracy metrics (RMSE, R, PBIAS) with total river length (prop = 0.35,  $p < 0.01$ ), longest river length (prop = 0.28,  $p < 0.01$ ) and Dist1 (prop = 0.20,  $p < 0.05$ ). Total river length correlated with the negative space of dbRDA axis 1, indicating that decreasing accuracy in terms of RMSE and R (see Figure 8) was significantly correlated with the total length of tributary streams. The streams exemplifying this relationship were the Otterbach in the most negative space of dbRDA axis 1, with a total river length of 41.43 km, contrasting the Aubach with a total river length of 7.91 km, in the upper positive

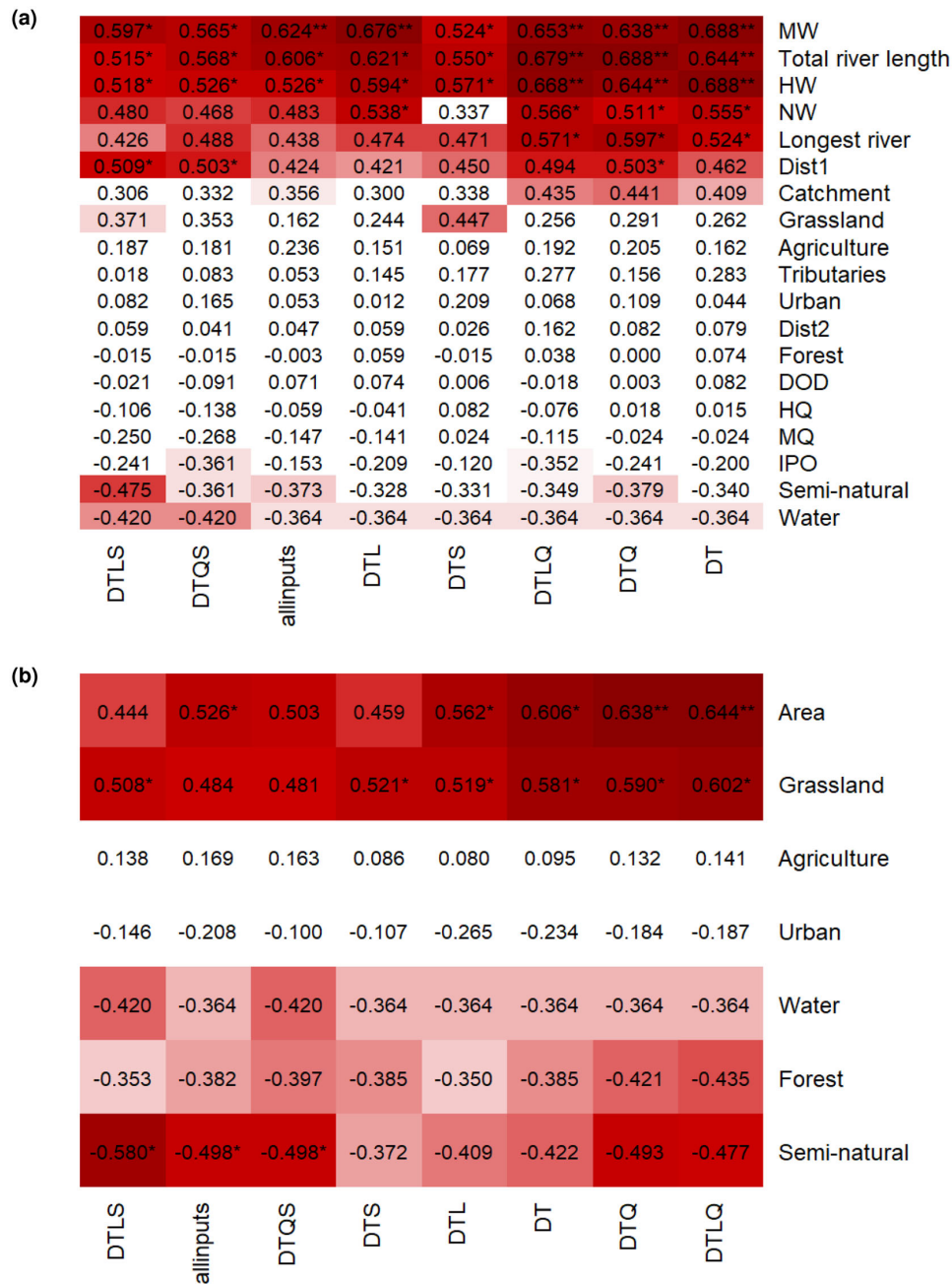
space of dbRDA axis 1. On dbRDA axis 2, streams were mainly separated along a gradient of the parameters: longest river length, Dist1, catchment area as well as proportions of grassland, semi-natural land use and HW. Thus, when relating these findings to the underlying configuration of accuracy metrics, environmental parameters on dbRDA axis 2 were positively associated with overestimation (longest river length, Dist1, catchment area) or underestimation (grassland, semi-natural land use and HW) of water temperature prediction by ANNs.

## 4 | DISCUSSION

In line with our hypothesis, our results suggest that the accuracy and reliability of ANNs' predictions for single streams are highly dependent on input combination and environmental parameters. To understand how environmental parameters affect ANNs' accuracy and reliability, we analysed a broad range of environmental predictors, showing that river length and water levels, the size of the catchment and open-canopy land use types were particularly negatively associated with ANN accuracy in the streams we tested.

### 4.1 | Measures of model performance

To prioritize the use of the accuracy metrics RMSE, R, and PBIAS for the evaluation of ANNs, we examined correlations between these metrics and reliability metrics as established in Mohr et al. (2021). We found, that not all accuracy metrics correlated significantly with all reliability metrics, confirming the finding of Mohr et al. (2021), that the use of accuracy metrics alone is insufficient and should be supplemented by reliability metrics.

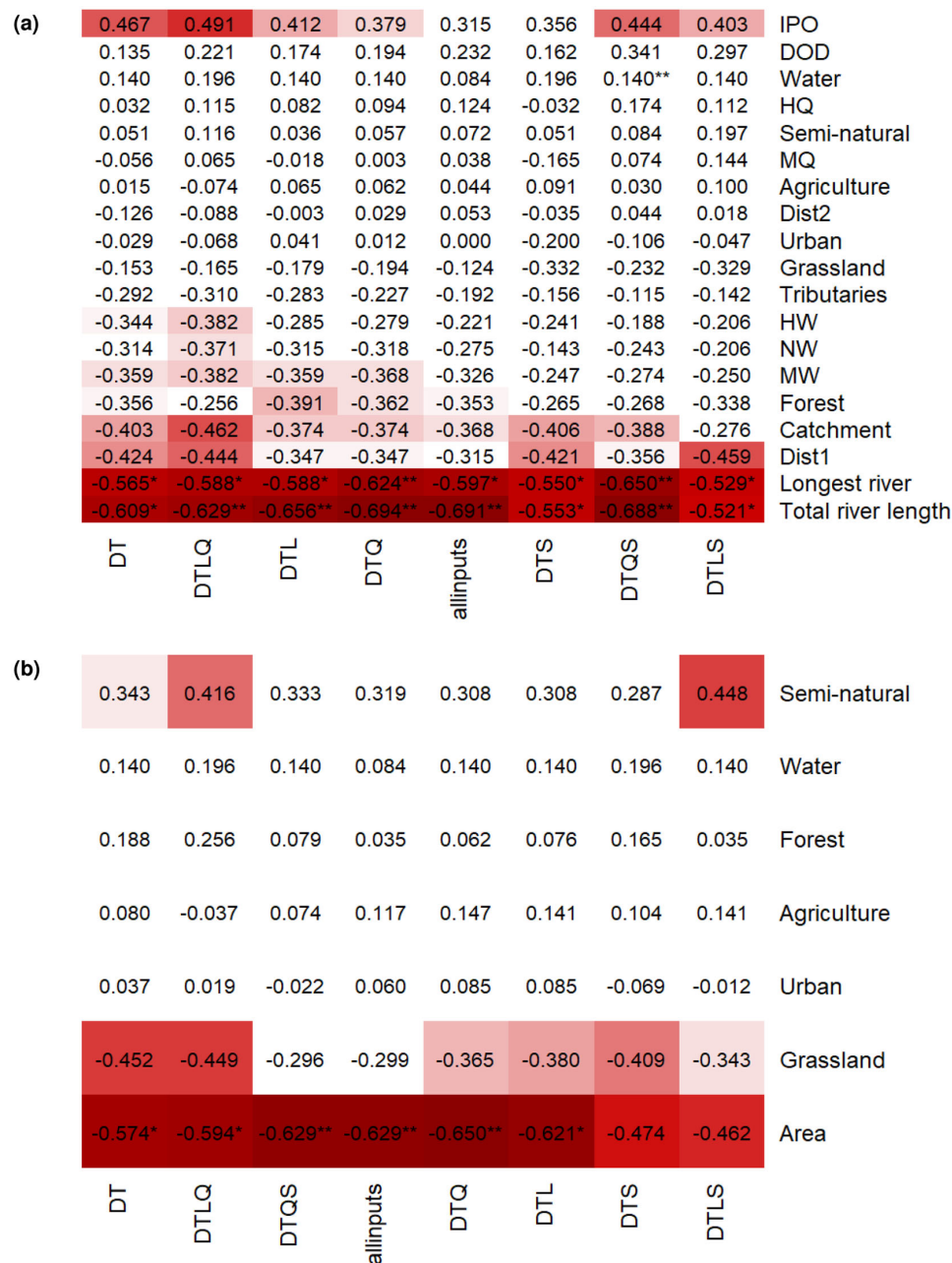


**FIGURE 5** (a) RMSE entire catchment; (b) RMSE 5 m riparian strip. Increasing intensity of red colour indicates increasing correlation (positive as well as negative). Significance is marked with \* $p < 0.05$  and \*\* $p < 0.01$ . Input parameters (x-axis): D, day of the year; T, air temperature; Q, discharge; L, water level; S, sunshine duration; DTQLS, allinputs. Environmental parameters (y-axis): as described in Section 2.6.

Still, we can conclude that as accuracy metric, the RMSE was the most suitable one to reflect the reliability of an ANN, due to two significant correlations with reliability metrics as opposed to one significant correlation for R and PBIAS each. We also observed that the

RMSE had a greater resolution and hence contributed more significant relationships with environmental parameters than R, probably because it had a higher potential to reflect the high-resolution dynamics of hydrologic parameters. This further confirmed the plausibility of its

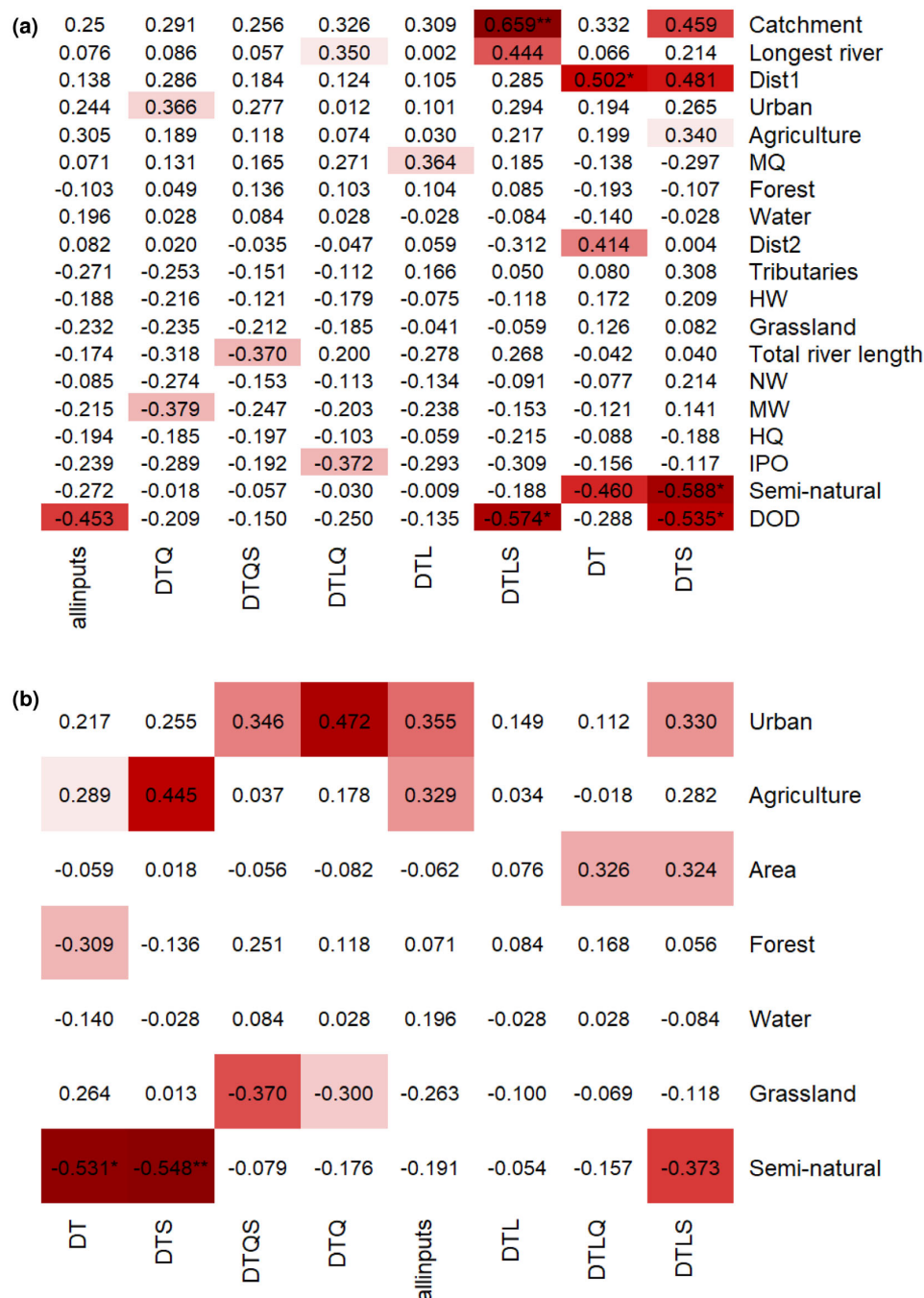




**FIGURE 6** (a) R entire catchment; (b) R 5 m riparian strip. Increasing intensity of red colour indicates increasing correlation (positive as well as negative). Significance is marked with \* $p < 0.05$  and \*\* $p < 0.01$ . Input parameters (x-axis): D, day of the year; DTQLS, allinputs; L, water level; Q, discharge; S, sunshine duration; T, air temperature; Environmental parameters (y-axis): as described in Section 2.6.

frequent use for measuring the accuracy of water temperature prediction with ANNs (Ahmadi-Nedushan et al., 2007; Caissie et al., 1998; Chenard & Caissie, 2008; Cho & Lee, 2012; Feigl et al., 2021; Graf et al., 2019; Hadzima-Nyarko et al., 2014; Qiu et al., 2020; Quan et al., 2020; Rabi et al., 2015; Rahmani et al., 2020; Rehana, 2019; St-Hilaire et al., 2000; Zhu, Nyarko, Hadzima-Nyarko, et al., 2019).

We additionally employed PBIAS as an accuracy metric, which is unusual for water temperature prediction with ANNs. Although we saw advantages of including the PBIAS due to the different aspects of model performance it highlights, in this study we were not able to find any general significant correlations between the assessed environmental parameters and the PBIAS. This might be because the PBIAS

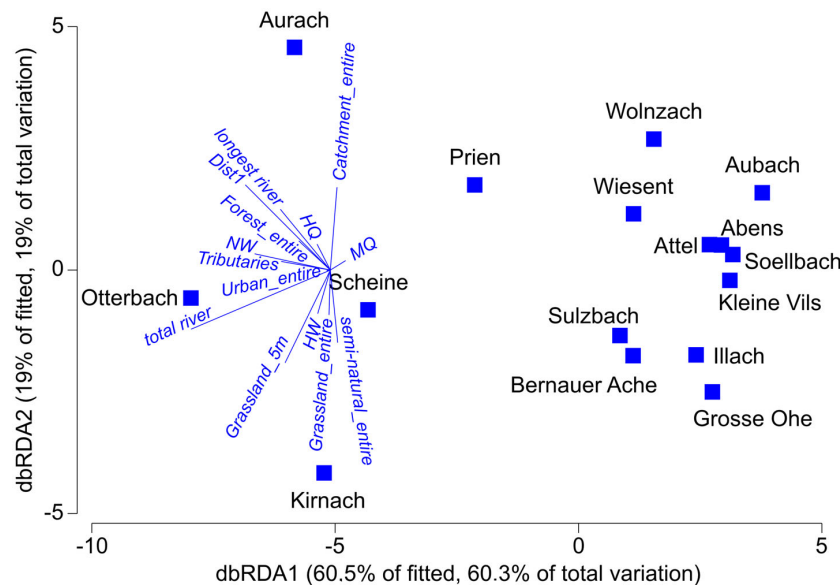


**FIGURE 7** (a) PBIAS entire catchment; (b) PBIAS 5 m riparian strip. Increasing intensity of red colour indicates increasing correlation (positive as well as negative). Significance is marked with \* $p < 0.05$  and \*\* $p < 0.01$ . Input parameters (x-axis): D, day of the year; DTQLS, allinputs; L, water level; Q, discharge; S, sunshine duration; T, air temperature; Environmental parameters (y-axis): As described in Section 2.6.

in general reflects variation in two directions, but in our study the direction of estimation (over- or underestimation) did not necessarily correlate with the examined environmental parameters in only one direction. The overestimation was pronounced for Kirnach, a stream

with a very high proportion of grassland (72.58%) and a very low proportion of semi-natural land cover (0.01%). In contrast, underestimation of water temperature was pronounced for Aurach, a long stream with a large catchment. These findings were also confirmed by the





**FIGURE 8** distLM-Eval-Enviro-plot, Resemblance: D1 Euclidian distance, Correlation between total river length and negative space of dbRDA axis 1, discrimination of streams by longest river length, Dist1, catchment area, proportions of grassland and semi-natural land use, and HW along dbRDA axis 2. Environmental parameters: Total river length: Sum of lengths of all contributing rivers; Longest river length: Length of the longest contributing river; Land use: agriculture, forest, grassland, semi-natural, urban, water; Catchment: Catchment size of all contributing catchments; Area: Total buffer area; MW: Mean water level; HW: Highest measured water level; NW: Lowest measured water level; MQ: Mean discharge; HQ: Highest measured discharge; Tributaries: Number of tributaries; DOD: Number of days for which data was used; IPO: Number of input data points per output data point; Dist1: Distance between GkD station and DWD station 1; Dist2: Distance between GkD station and DWD station 2. Resolution: entire: Entire catchment; 5 m 5 m riparian strip.

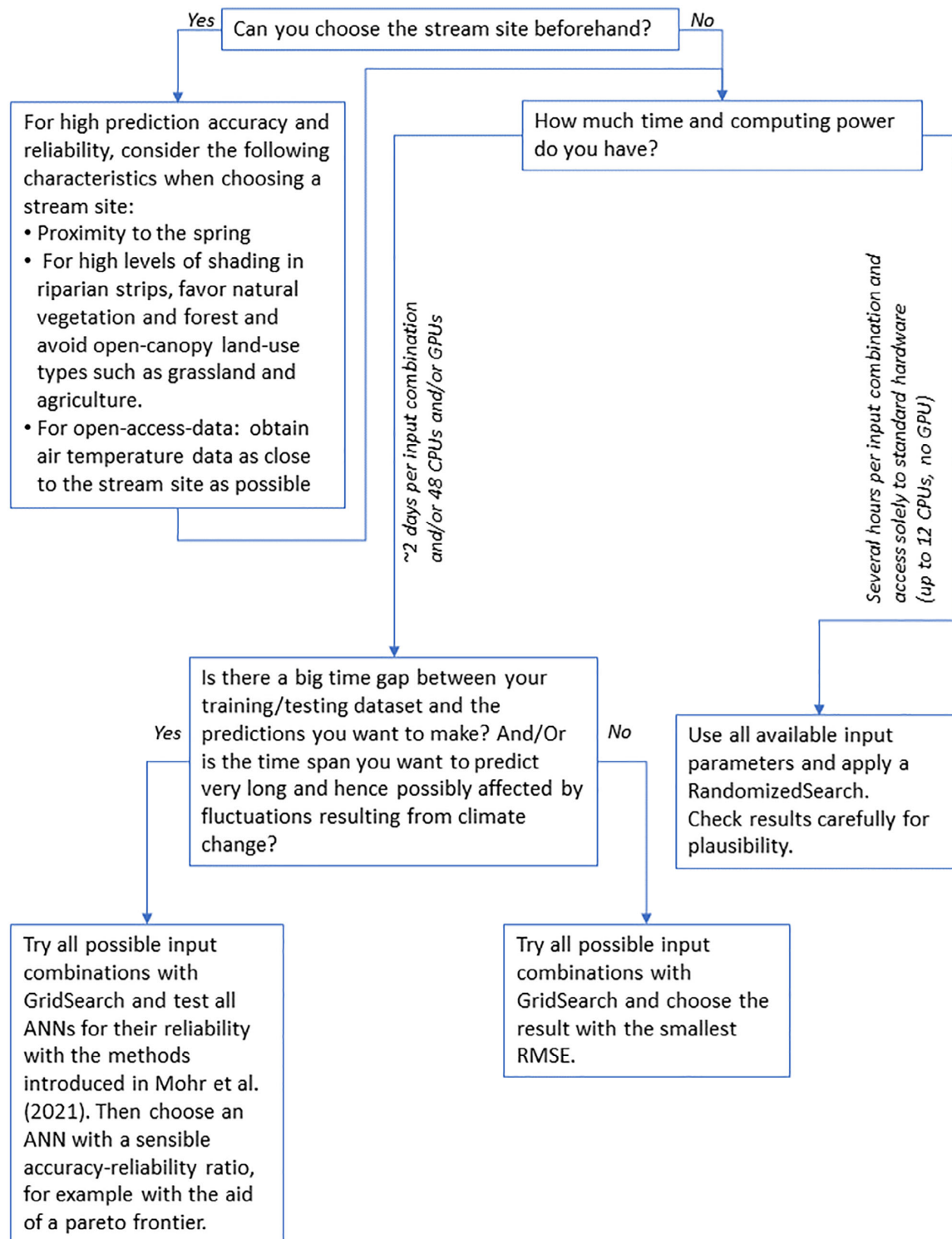
DistLM analysis of environmental predictors of evaluation metrics, in which PBIAS/overestimation was associated with high proportions of grassland, particularly in the 5 m riparian strip. Consequently, it would be advisable to carefully check for both over- and underestimation of water temperature prediction, particularly in catchments with high proportions of open-canopy landscape.

## 4.2 | Input parameters

The most striking finding of this study was that the input combination with the highest accuracy was a stream-specific set of input parameters, suggesting that the optimal input combination cannot be generalized across streams. As important asset, this study used a systematic procedure of training and testing ANNs with different sets of input parameters, which provided us with the opportunity to compare ANN accuracies within single streams. While other studies like Feigl et al. (2021) previously identified that the input combination has an effect on ANN performance in general, our finding, that the optimal input combination is stream-specific, adds an important new insight to this field, which can help to make stream water temperature predictions more accurate in the future. As we were able to show, the error in the prediction (RMSE) could increase to > 100% in a single

stream if a random input combination was used instead of the optimal input combination. Even when using the allinputs combination, the error increased by up to 34%, indicating that allinputs might be more accurate than a random input combination, but still not as accurate as if the combination was determined systematically. This result is in line with the “explosion” of Myth #7 in Maier et al. (2023), where it is stated that an increase in the number of input variables does not necessarily improve model performance, but that these variables need to be selected carefully. Clearly, the search for the optimal input combination is time consuming compared to a fixed procedure using a set of pre-defined input variables. Hence, for supporting the application of ANNs based on our results, we provide a flow chart to facilitate decision-making along the process of water temperature prediction with ANNs (see Figure 9).

Comparing the RMSE values from Table 1 to previous studies that predicted water temperatures with ANNs (sota-range 0.46°C to 1.58°C), only one stream (Otterbach, RMSE = 1.667°C) had an RMSE slightly higher than the sota-range. Further, 12 streams had an RMSE within the sota-range and three streams had RMSE values even lower than the sota-range, namely Attel (RMSE = 0.453°C), Aubach (RMSE = 0.373°C), and Soellbach (RMSE = 0.419°C). To the best of our knowledge, the RMSE values of Attel, Aubach and Soellbach were the smallest ever reported for stream water temperature prediction using ANNs.



**FIGURE 9** Flow chart with recommendations on stream-specific artificial neural network-development.

Based on Zhu, Nyarko, Hadzima-Nyarko, et al. (2019), we expected a minor role of discharge in explaining temperature, since they state that discharge plays a minor role in stream water temperature prediction compared to the day of the year and that discharge's importance increases for high-altitude catchments. Still, in all of the streams of our study, the most accurate ANNs all had water level and/or discharge as inputs. Unfortunately, Zhu, Nyarko, Hadzima-Nyarko, et al. (2019) did not consider water level, which hinders direct comparison with our results. Nevertheless, based on our results, we suggest using at least one hydrologic input parameter for water temperature prediction with ANNs, while we cannot generalize the recommendation to a specific hydrologic parameter, since this is highly stream specific. Still, we conclude that no unique optimal input combination exists for each stream.

### 4.3 | Environmental influences on water temperature prediction

Given the high specificity of input combinations we determined for individual streams, it was a key interest of this study to identify stream environmental conditions that govern the accuracy of ANNs. In light of climate change, such knowledge is also highly relevant for deducing mitigation and management strategies in streams related to securing high oxygen concentrations (Piatka et al., 2021), endangered fish populations (Wild et al., 2023), and temperature refuges (Kuhn et al., 2021; Mejia et al., 2023). In contrast to existing approaches, which mainly consider hyperparameter tuning and dataset length, the associations between environmental parameters and ANN accuracy allow a more mechanistic and realistic assessment of model applicability at individual stream sites, as demonstrated for our datasets. Several significant correlations between environmental parameters and prediction accuracy of ANNs were identified, suggesting key influences of catchment hydrological variables.

Specifically, the accuracy of ANNs (RMSE and R) was strongly related to total river and longest river length, total catchment area, and the hydrological parameters MW, HW, and NW, indicating a decrease in ANN accuracy with increasing river length, catchment size, and water level.

Since stream water temperatures are defined by complex and dynamic physico-chemical, hydrologic and atmospheric processes and not solely based on air temperature (Caissie, 2006; Leach et al., 2023), a possible explanation for the strong negative relationship between ANN accuracy and river length and catchment area could be the increase of air-temperature-unrelated complex influences along the flow path of streams. Beginning at the spring, the water has a specific temperature, depending on its origin and the distance to its spring. As the stream water passes through the landscape, energy exchange is influenced by advective fluxes like evaporation or longitudinal changes in advection and radiation due to changes in vegetation (Coats & Jackson, 2020; Leach et al., 2023). Energy is added by river bank and bed friction, and contact with the atmosphere

increases, as do the radiative fluxes (Dan Moore et al., 2005; Kuhn et al., 2021; Webb et al., 2008). Hence, with increasing river length, the potential number of complex influences increases and thus, the accuracy of the water temperature predictions decreases. This is especially pronounced for models like ANNs, which do not receive additional information on catchment-size related variables but have to learn in the context of local input parameters, measured at the gauging station.

As with river length and catchment size, higher levels of HW, NW and MW were associated with a lower prediction accuracy (RMSE) of ANNs. The relationship between extreme water levels (HW) and ANN accuracy is due to difficulties in predicting the temperatures of water sources entering the stream along its flowpath (e.g. groundwater, hyporheic water, precipitation, anthropogenic water influxes (Nelson & Palmer, 2007; Webb et al., 2008). During spates and high-water events, these water sources contribute different relative quantities to total water volume, and temperature mixing during high water events is then presumably more difficult for ANNs to predict. Additionally, it has been shown that air-water temperature relationships are stronger and more sensitive for flows below median levels (Webb et al., 2003), likely because high water levels lead to a lower water-atmosphere interaction of the surface area compared to total water volume, influencing radiation influx and sensible heat transfer. As a result, depending on surrounding atmospheric temperatures, energy fluxes are often easier to predict for smaller water volumes, which explains the higher prediction accuracy for lower MW and NW values of streams. Hence, the connection between increasing water levels, in particular the HW values and decreasing accuracy in water temperature prediction by ANNs, seems plausible and should be considered when predicting water temperatures in streams during periods of high water.

We found that the land use types semi-natural, forest and water bodies had a positive effect on ANN accuracy. Further, our results showed that high proportions of grassland in the 5 m riparian strip (but not on the entire catchment resolution) correlated with decreasing accuracy (RMSE) in water temperature prediction.

The land use surrounding a stream has a strong influence on its temperature regime and humidity, which controls the water-atmosphere interaction (Webb & Zhang, 1997). It can be assumed that high proportions of grassland facilitated heat-induced evaporation, which can lead to cooling effects especially during high temperature phases (Ouellet & Caissie, 2023), inducing a paradoxical relationship between air temperature (increasing) and water temperature (decreasing). In low temperature phases, this effect is not induced, resulting in an inconsistent relationship between air and water temperature, hence potentially reducing the accuracy of water temperature predictions based on air temperature data.

In general, open-canopy land use such as grassland involves higher levels of radiation and heat fluxes due to a lack of shading and temperature buffering through a micro-climate of complex riparian vegetation. As solar radiation is the most important component of heat transfer in streams (Webb & Zhang, 1997, 1999), accounting for

70% of non-advective heat fluxes in a stream (Webb et al., 2008), open-canopy land use forms are associated with higher air temperatures and lower humidity, which can in turn result in more pronounced temperature extremes and drought conditions in streams. For example, Rutherford et al. (2004) and Ebersole et al. (2003) attributed a maximum temperature decrease of 4°C downstream of shaded areas to the effect of riparian vegetation. In a simulation study, Wondzell et al. (2019) determined that shading through a mature forest can account for a decrease of water temperature of 8°C. Johnson (2004) quantified the net energy transfer in July in a stream in Oregon. Non-shaded, the water temperature gained 580 W/m<sup>2</sup>, but fully shaded, the stream's water lost 149 W/m<sup>2</sup>. Hoess et al. (2022) found that shading by coniferous vegetation could even compensate for a temperature increase caused by pond effluents. Also, without shading, conduction between water and heated alluvial substrates is an often underestimated process influencing stream water temperatures, particularly under forest harvest scenarios (Brown, 1969; Johnson & Jones, 2000). Hence, riparian shading appears to be of paramount importance for controlling and regulating stream water temperatures. Our findings further demonstrated that for the prediction of water temperatures using an air-water-temperature relationship, the land use in the proximate riparian surroundings (in our case the 5 m riparian strip) seemed more important than the catchment's global land use. Also, Kail et al. (2021) found that large trees in the 10 m riparian strip are a better predictor of water temperature than the width of riparian strips (in their case 30 m), due to the presence of large trees that provided direct shade for the streams and hence cooled the stream water highly effectively. As we showed that prediction accuracy (RMSE) was higher in streams with higher proportions of forest and semi-natural land use (5 m riparian strip) and semi-natural land use and water body area (entire catchment), it can be assumed that riparian shade stabilizes water temperatures, hence facilitating more accurate prediction, as water temperatures are likely more linearly and consistently related to atmospheric temperatures. Also the proportion of water bodies is likely related to prediction accuracy, due to their temperature-buffering properties in the catchment. Our results imply that larger proportions of open-canopy land use forms and the associated higher radiation and low levels of shading can lead to high levels of temperature variability, potentially hampering ANN accuracy and reliability. Consequently, we advise greater caution when using ANNs for streams in open-canopy landscapes.

## 5 | CONCLUSIONS

We conclude the following for water temperature prediction in streams with ANNs, based on open-access data:

1. It is possible to use open-access data for water temperature predictions within the sota-range.
  - a. The use of open-access data, however, comes with the problem that there is only a limited number of parameters. Hence, the choice of streams for which the water temperature is to be

predicted is crucial for the accuracy and reliability of the predictions.

- b. For an optimal outcome, all available input parameters should be tested for their suitability (see recommendations in Figure 9).
2. If water temperature is to be predicted for a specific stream, it might not be sufficient to use open-access data, especially if the stream is characterized by specific environmental parameters, which reduce the accuracy and reliability of water temperature prediction.
3. If the ANN is intended to predict water temperature for a future or past time with different climatic conditions compared to present ones, not only the accuracy but also the reliability of the ANN should be considered in the choice of architecture and input parameters (see recommendations in Figure 9). If it is not possible to test reliability, the RMSE is a good (but not in itself sufficient) predictor of ANN reliability and should hence be used.

Our findings highlight that water temperature predictions are more accurate and reliable in headwater streams closer to their source, especially if adjacent land use comprises forests and natural riparian vegetation that lack anthropogenic influences. The finding that ANN prediction accuracy is distinctly compromised by disturbances in the riparian cover, which commonly accumulate along a river's course, leads us to conclude that the lower ANN accuracy reflects the increasing disturbances in the air-water-temperature relationship. We propose that measures of ANN accuracy, as a proxy for an inconsistent air-water-temperature relationship, could even be used to indicate a functional and resilient water temperature regime in headwater streams. Given the importance of small headwater streams and spring ecosystems as refuges and highly specialized environments that feature a broad width of unique and sensitive species requiring special protection (Cantonati et al., 2012; Richardson, 2019), ANN accuracy measures could serve as an indicative tool to identify, evaluate and monitor headwater streams with regard to their temperature integrity and to support decision making regarding where and how to best protect these unique environments. Further, this research highlights that anthropogenic and, specifically, land-use-derived disturbances along stream ecosystems affect stream water temperatures and will consequently exacerbate the climate-change-associated warming of stream water. We have therefore added highly relevant information to the use of ANNs to predict stream water temperatures. In combination with climate change projections, ANNs could prove to be a cost-efficient and invaluable resource for decision makers to use when assessing future developments in stream water temperatures, aiding the evaluation and prioritization of restoration, renaturation and adaptation measures in streams.

## ACKNOWLEDGEMENTS

This work was supported by the Bavarian State Ministry of Science and the Arts in the AquaKlif project within the Bavarian Climate Research Network (bayklif) and by the DFG Research Training Group

on Continuous Verification of Cyber-Physical Systems (GRK 2428). We thank Jan Křetínský and Maximilian Weininger from the Chair for Foundations of Software Reliability and Theoretical Computer Science (Technical University of Munich) for the valuable discussions on model assessment as well as the Proofreading Service of the TUM Graduate School for a final cross-check of the manuscript. Open Access funding enabled and organized by Projekt DEAL.

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## ORCID

Konstantina Drinas  <https://orcid.org/0000-0001-6771-2123>

Lisa Kaule  <https://orcid.org/0000-0002-5881-7561>

Stefanie Mohr  <https://orcid.org/0000-0002-8630-3218>

Bhumika Uniyal  <https://orcid.org/0000-0002-3841-8184>

Romy Wild  <https://orcid.org/0000-0002-4814-6215>

Juergen Geist  <https://orcid.org/0000-0001-7698-3443>

## REFERENCES

- Ahmadi-Nedushan, B., St-Hilaire, A., Ouarda, T. B. M. J., Bilodeau, L., Robichaud, É., Thiémondge, N., & Bobée, B. (2007). Predicting river water temperatures using stochastic models: Case study of the Moisie River (Québec, Canada). *Hydrological Processes*, 21(1), 21–34.
- Anderson, M. J., Gorley, R., & Clarke, K. (2008). *PERMANOVA+ for PRIMER: Guide to software and statistical methods*. PRIMER-E.
- Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS One*, 10(7), e0130140.
- Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., & Müller, K.-R. (2010). How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11, 1803–1831.
- Benyahya, L., Caissie, D., St-Hilaire, A., Ouarda, T. B., & Bobée, B. (2007). A review of statistical water temperature models. *Canadian Water Resources Journal*, 32(3), 179–192.
- Brown, G. W. (1969). Predicting temperatures of small streams. *Water Resources Research*, 5(1), 68–75.
- Caissie, D. (2006). The thermal regime of rivers: A review. *Freshwater Biology*, 51(8), 1389–1406.
- Caissie, D., El-Jabi, N., & St-Hilaire, A. (1998). Stochastic modelling of water temperatures in a small stream using air to water relations. *Canadian Journal of Civil Engineering*, 25(2), 250–260.
- Caldwell, P., Segura, C., Gull Laird, S., Sun, G., McNulty, S. G., Sandercock, M., Boggs, J., & Vose, J. M. (2015). Short-term stream water temperature observations permit rapid assessment of potential climate change impacts. *Hydrological Processes*, 29(9), 2196–2211.
- Cantonati, M., Füreder, L., Gerecke, R., Jüttner, I., & Cox, E. J. (2012). Crenic habitats, hotspots for freshwater biodiversity conservation: Toward an understanding of their ecology. *Freshwater Science*, 31(2), 463–480.
- Chen, Y. D., Carsel, R. F., McCutcheon, S. C., & Nutter, W. L. (1998). Stream temperature simulation of forested riparian areas: I. Watershed-scale model development. *Journal of Environmental Engineering*, 124(4), 304–315.
- Chenard, J.-F., & Caissie, D. (2008). Stream temperature modelling using artificial neural networks: Application on catamaran brook, New Brunswick, Canada. *Hydrological Processes*, 22(17), 3361–3372.
- Cho, H.-Y., & Lee, K.-H. (2012). Development of an air–water temperature relationship model to predict Climate-induced future water temperature in estuaries. *Journal of Environmental Engineering*, 138(5), 570–577.
- Coats, W. A., & Jackson, C. R. (2020). Riparian canopy openings on mountain streams: Landscape controls upon temperature increases within openings and cooling downstream. *Hydrological Processes*, 34(8), 1966–1980.
- Crisp, D. T., & Howson, G. (1982). Effect of air temperature upon mean water temperature in streams in the north Pennines and English Lake District. *Freshwater Biology*, 12(4), 359–367.
- da Silva, I. N., Hernane Spatti, D., Andrade Flauzino, R., Liboni, L. H. B., & dos Reis Alves, S. F. (2017). *Artificial neural networks*. Springer International Publishing.
- Dan Moore, R., Spittlehouse, D., & Story, A. (2005). Riparian microclimate and stream temperature response to forest harvesting: A review 1. *JAWRA Journal of the American Water Resources Association*, 41(4), 813–834.
- Drinas, K. (2020). Prediction of stream water and hyporheic temperature in the context of local climate change: A case study at the bavarian mähringsbach, fichtel mountains. [Unpublished Master's thesis], Technical University of Munich.
- Ebersole, J. L., Liss, W. J., & Frissell, C. A. (2003). Cold water patches in warm streams: Physicochemical characteristics and the influence of shading 1. *JAWRA Journal of the American Water Resources Association*, 39(2), 355–368.
- Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3), 1.
- European Union. (2021). *Copernicus Land Monitoring Service 2021*. European Environment Agency (EEA).
- Feigl, M., Lebedzinski, K., Hermegger, M., & Schulz, K. (2021). Machine-learning methods for stream water temperature prediction. *Hydrology and Earth System Sciences*, 25(5), 2951–2977.
- Graf, R., Zhu, S., & Sivakumar, B. (2019). Forecasting river water temperature time series using a wavelet–neural network hybrid modelling approach. *Journal of Hydrology*, 578, 124115.
- Gupta, H. V., Sorooshian, S., & Yapo, P. O. (1999). Status of automatic calibration for hydrologic models: Comparison with multilevel expert calibration. *Journal of Hydrologic Engineering*, 4(2), 135–143.
- Hadzima-Nyarko, M., Rabi, A., & Šperac, M. (2014). Implementation of artificial neural networks in modeling the water–air temperature relationship of the river Drava. *Water Resources Management*, 28(5), 1379–1394.
- Han, J., Pei, J., & Kamber, M. (2011). *Data mining: Concepts and techniques*. Elsevier.
- Harvey, R., Lye, L., Khan, A., & Paterson, R. (2011). The influence of air temperature on water temperature and the concentration of dissolved oxygen in Newfoundland Rivers. *Canadian Water Resources Journal*, 36(2), 171–192.
- Hoess, R., Generali, K. A., Kuhn, J., & Geist, J. (2022). Impact of fish ponds on stream hydrology and temperature regime in the context of freshwater pearl mussel conservation. *Watermark*, 14(16), 2490.
- Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., & Yi, X. (2020). A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability? *Computer Science Review*, 37, 100270.
- IPCC. (2022). Climate change 2022: Impacts, adaptation and vulnerability. Contribution of working group II to the sixth assessment report of the intergovernmental panel on Climate change. In H.-O. Pörtner, D. C. Roberts, M. Tignor, E. S. Poloczanska, K. Mintenbeck, A. Alegría, M. Craig, S. Langsdorf, S. Löschke, V. Möller, A. Okem, & B. Rama (Eds.). Cambridge University Press.
- Johnson, S. L. (2004). Factors influencing stream temperatures in small streams: Substrate effects and a shading experiment. *Canadian Journal of Fisheries and Aquatic Sciences*, 61(6), 913–923.



- Johnson, S. L., & Jones, J. A. (2000). Stream temperature responses to forest harvest and debris flows in western cascades, Oregon. *Canadian Journal of Fisheries and Aquatic Sciences*, 57(S2), 30–39.
- Kail, J., Palt, M., Lorenz, A., & Hering, D. (2021). Woody buffer effects on water temperature: The role of spatial configuration and daily temperature fluctuations. *Hydrological Processes*, 35(1), e14008.
- Kothandaraman, V. and Evans, R. L. (1972). *Use of air-water relationships for predicting water temperature*. Illinois State Water Survey.
- Krider, L. A., Magner, J. A., Perry, J., Vondracek, B., & Ferrington, L. C. (2013). Air-water temperature relationships in the trout streams of southeastern Minnesota's carbonate-sandstone landscape. *JAWRA Journal of the American Water Resources Association*, 49(4), 896–907.
- Kuhn, J., Casas-Mulet, R., Pander, J., & Geist, J. (2021). Assessing stream thermal heterogeneity and cold-water patches from UAV-based imagery: A matter of classification methods and metrics. *Remote Sensing*, 13(7), 1379.
- Leach, J. A., Kelleher, C., Kurylyk, B. L., Moore, R. D., & Neilson, B. T. (2023). A primer on stream temperature processes. *Wiley Interdisciplinary Reviews Water*, 10, e1643.
- Legendre, P., & Anderson, M. J. (1999). Distance-based redundancy analysis: Testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs*, 69(1), 1–24.
- Maier, H. R., Galelli, S., Razavi, S., Castelletti, A., Rizzoli, A., Athanasiadis, I. N., Sánchez-Marré, M., Acutis, M., Wu, W., & Humphrey, G. B. (2023). Exploding the myths: An introduction to artificial neural networks for prediction and forecasting. *Environmental Modelling & Software*, 167, 105776.
- Mejia, F. H., Ouellet, V., Briggs, M. A., Carlson, S. M., Casas-Mulet, R., Chapman, M., Collins, M. J., Dugdale, S. J., Ebersole, J. L., Frechette, D. M., Fullerton, A. H., Gillis, C. A., Johnson, Z. C., Kelleher, C., Kurylyk, B. L., Lave, R., Letcher, B. H., Myrvold, K. M., Nadeau, T. L., ... Torgersen, C. E. (2023). Closing the gap between science and management of cold-water refuges in rivers and streams. *Global Change Biology*, 29, 5482–5508.
- Mohr, S., Drainas, K., & Geist, J. (2021). Assessment of neural networks for stream-water-temperature prediction. In *20th IEEE international conference on machine learning and applications (ICMLA)*. IEEE.
- Mohseni, O., & Stefan, H. G. (1999). Stream temperature/air temperature relationship: A physical interpretation. *Journal of Hydrology*, 218(3–4), 128–141. PII: S0022169499000347.
- Moriasi, D. N., Arnold, J. G., van Liew, M. W., Bingner, R. L., Harmel, R. D., & Veith, T. L. (2007). Model evaluation guidelines for systematic quantification of accuracy in watershed simulations. *Transactions of the ASABE*, 50(3), 885–900.
- Nelson, K. C., & Palmer, M. A. (2007). Stream temperature surges under urbanization and Climate change: Data, models, and responses. *JAWRA Journal of the American Water Resources Association*, 43(2), 440–452.
- Ouellet, V., & Caissie, D. (2023). Towards a better understanding of the evaporative cooling of rivers: Case study for the little Southwest Miramichi river (New Brunswick, Canada). *Canadian Water Resources Journal/Revue Canadienne Des Ressources Hydriques*, 48, 1–17.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Piatka, D. R., Wild, R., Hartmann, J., Kaule, R., Kaule, L., Gilfedder, B., Peiffer, S., Geist, J., Beierkuhnlein, C., & Barth, J. A. (2021). Transfer and transformations of oxygen in rivers as catchment reflectors of continental landscapes: A review. *Earth-Science Reviews*, 220, 103729.
- Pilgrim, J. M., Fang, X., & Stefan, H. G. (1998). Stream temperature CORRELATIONS with air temperatures IN Minnesota: Implications for CLIMATE warming. *JAWRA Journal of the American Water Resources Association*, 34(5), 1109–1121.
- Piotrowski, A. P., Napiorkowski, M. J., Napiorkowski, J. J., & Osuch, M. (2015). Comparing various artificial neural network types for water temperature prediction in rivers. *Journal of Hydrology*, 529, 302–315.
- Qiu, R., Wang, Y., Wang, D., Qiu, W., Wu, J., & Tao, Y. (2020). Water temperature forecasting based on modified artificial neural network methods: Two cases of the Yangtze River. *The Science of the Total Environment*, 737, 139729.
- Quan, Q., Hao, Z., Xifeng, H., & Jingchun, L. (2020). Research on water temperature prediction based on improved support vector regression. *Neural Computing and Applications*, 34, 1–10.
- Rabi, A., Hadzima-Nyarko, M., & Šperac, M. (2015). Modelling river temperature from air temperature: Case of the river Drava (Croatia). *Hydrological Sciences Journal*, 60(9), 1490–1507.
- Rahmani, F., Lawson, K., Ouyang, W., Appling, A., Oliver, S., & Shen, C. (2020). Exploring the exceptional performance of a deep learning stream temperature model and the value of streamflow data. *Environmental Research Letters*, 16(2), 024025.
- Rehana, S. (2019). River water temperature modelling under Climate change using support vector regression. In S. K. Singh (Ed.), *Hydrology in a changing world, Springer Water Series* (pp. 171–183). Springer.
- Richardson, J. (2019). Biological diversity in headwater streams. *Watermark*, 11(2), 366.
- RStudio Team. (2022). *RStudio: Integrated development environment for R*. RStudio PBC.
- Rutherford, J. C., Marsh, N. A., Davies, P. M., & Bunn, S. E. (2004). Effects of patchy shade on stream water temperature: How quickly do small streams heat and cool? *Marine and Freshwater Research*, 55(8), 737–748.
- Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034.
- Smith, K. (1981). The prediction of river water temperatures / Prédiction des températures des eaux de rivière. *Hydrological Sciences Bulletin*, 26(1), 19–32.
- St-Hilaire, A., Morin, G., El-Jabi, N., & Caissie, D. (2000). Water temperature modelling in a small forested stream: Implication of forest canopy and soil temperature. *Canadian Journal of Civil Engineering*, 27(6), 1095–1108.
- Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning* (pp. 3319–3328). PMLR.
- Warnes, G. R., Bolker, B., Bonebakker, L., Gentleman, R., Huber, W., Liaw, A., Lumley, T., Maechler, M., Magnusson, A., Moeller, S., Schwartz, M., & Venables, B. (2020). gplots: Various R Programming Tools for Plotting Data. R package version 3.1.1.
- Webb, B., & Zhang, Y. (1997). Spatial and seasonal variability in the components of the river heat budget. *Hydrological Processes*, 11(1), 79–101.
- Webb, B., & Zhang, Y. (1999). Water temperatures and heat budgets in dorset chalk water courses. *Hydrological Processes*, 13(3), 309–321.
- Webb, B. W., Clack, P. D., & Walling, D. E. (2003). Water-air temperature relationships in a Devon river system and the role of flow. *Hydrological Processes*, 17(15), 3069–3084.
- Webb, B. W., Hannah, D. M., Moore, R. D., Brown, L. E., & Nobilis, F. (2008). Recent advances in stream and river temperature research. *Hydrological Processes: An International Journal*, 22(7), 902–918.
- Wild, R., Nagel, C., & Geist, J. (2023). *Climate change effects on hatching success and embryonic development of fish: Assessing multiple stressor responses in a large-scale mesocosm study* (164834). Science of The Total Environment.
- Wondzell, S. M., Diabat, M., & Haggerty, R. (2019). What matters most: Are future stream temperatures more sensitive to changing air temperatures, discharge, or riparian vegetation? *JAWRA Journal of the American Water Resources Association*, 55(1), 116–132.
- Woodward, G., Perkins, D. M., & Brown, L. E. (2010). Climate change and freshwater ecosystems: Impacts across multiple levels of organization.

- Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 365(1549), 2093–2106.
- Zhu, S., Heddam, S., Nyarko, E. K., Hadzima-Nyarko, M., Piccolroaz, S., & Wu, S. (2019). Modeling daily water temperature for rivers: Comparison between adaptive neuro-fuzzy inference systems and artificial neural networks models. *Environmental Science and Pollution Research*, 26(1), 402–420.
- Zhu, S., Heddam, S., Wu, S., Dai, J., & Jia, B. (2019). Extreme learning machine-based prediction of daily water temperature for rivers. *Environmental Earth Sciences*, 78(6), 1–17.
- Zhu, S., Nyarko, E. K., Hadzima-Nyarko, M., Heddam, S., & Wu, S. (2019). Assessing the performance of a suite of machine learning models for daily river water temperature prediction. *PeerJ*, 7, e7065.
- Zurada, J. M., Malinowski, A., & Cloete, I. (1994). Sensitivity analysis for minimization of input data dimension for feedforward neural network. In *Proceedings of IEEE international symposium on circuits and systems*, ISCAS...94 (Vol. 6, pp. 447–450). IEEE.

**How to cite this article:** Drainas, K., Kaule, L., Mohr, S., Uniyal, B., Wild, R., & Geist, J. (2023). Predicting stream water temperature with artificial neural networks based on open-access data. *Hydrological Processes*, 37(10), e14991. <https://doi.org/10.1002/hyp.14991>

## APPENDIX

### A | Additional information on Materials and Methods

#### A.1 | Searching algorithms Scikit-learn

Our results were obtained using RandomizedSearch, which, given different options, optimized the architecture and hyperparameter combination of the ANNs for each individual input combination and waterbody.

RandomizedSearch certainly delivers a lower search quality than GridSearch, since it only determines local optima, unlike GridSearch, which delivers global optima. Still, as the ANN accuracy in our study demonstrated no weakness, in contrast to the results in the literature, we can support the use of RandomizedSearch, since it requires considerably less computing power and time. It is important to note, however, that even better results for the RMSE might be obtained with GridSearch and so it may be worth investing more time if fewer streams and less data needs to be processed or the time and capacity investment does not play a relevant role.

**Difference between RandomizedSearch and GridSearch:** While in RandomizedSearch, random sets of hyperparameters are used and tested, GridSearch tests all possible hyperparameter combinations systematically. The process can be accelerated by preselecting hyperparameters to reduce the total number of hyperparameter combinations. Of course, this again reduces the power of the search. In conclusion we suggest not using GridSearch if time and/or computing power are limited (see Figure 9).

#### A.2 | Results of RandomizedSearch

Using scikit-learn's RandomizedSearch, we determined an ANN with the hyperparameter combination leading to the lowest RMSE, for each waterbody and input combination. The RMSE values for all ANNs determined by RandomizedSearch are presented in Figure A1, according to waterbody. Figure A2 shows the same information but sorted by input combination. In Tables A3, A4, A5, and A6, these results are sorted by waterbody. The tables show which input combination for each stream reached what accuracy measures based on which hyperparameter combination. The abbreviations stand for the hyperparameters as indicated in the table below.

These combinations were attained by preselecting values for each hyperparameter based on prior experience. As stated above, preselection can reduce the power of the search, so we recommend including as many values as possible.

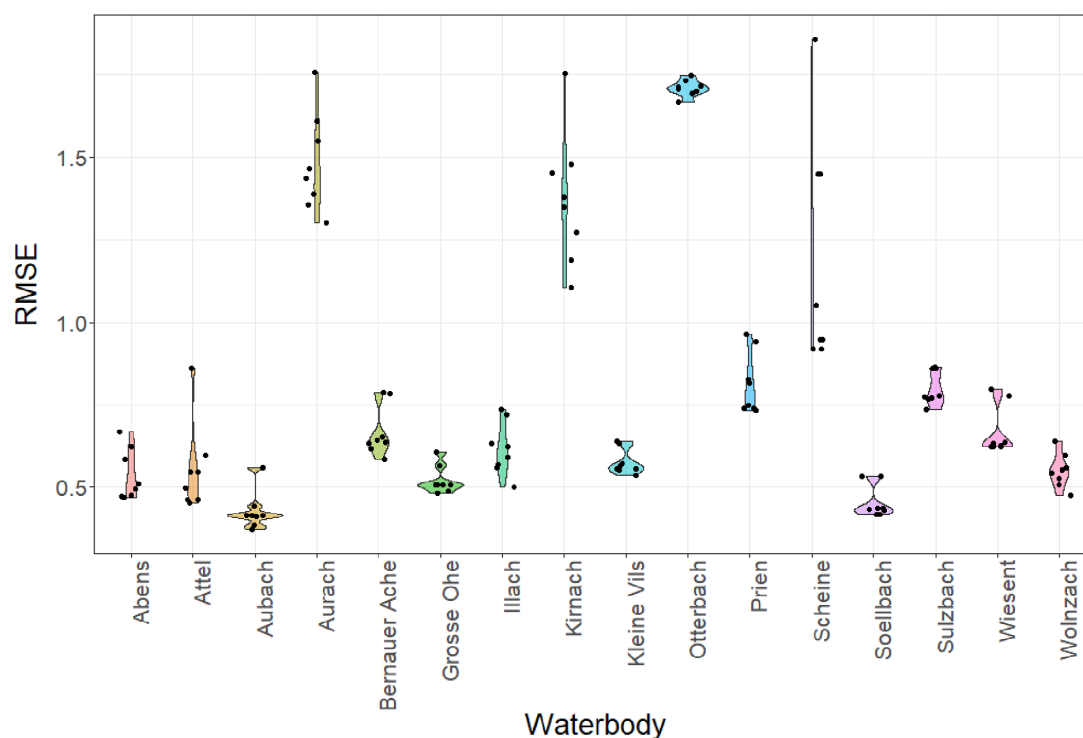
#### A.3 | Reliability of ANNs

Since common accuracy metrics only consider the differences between observed and predicted values, they are not suitable for assessing the reliability of the ANN, especially when it comes to changes in the database as expected for climate change scenarios. Hence, we also applied the reliability methodology as established in Mohr et al. (2021) on ANNs with the *allinputs* input combination, as determined by GridSearch.

##### A.3.1. | Perturbation analysis

Due to climate change, input variables will change in the future, e.g. air temperatures will rise. Since the training and testing datasets are retrieved from the past and cannot display future developments properly, a thorough analysis regarding changes in the input data is

Abbr.	Solver	Maximum iterations	Learning rate	Learning	Layers	Activation function
loc	lbfgs	100 000	0.0001	adaptive	5,20	logistic
rec	lbfgs	100 000	0.0001	constant	80,20,5	relu
tac	lbfgs	100 000	0.001	adaptive	80,10,5	tanh
werec	adam	100 000	0.01	constant	160,80,10	relu
wtac	adam	100 000	0.0001	adaptive	20 160,40	tanh
wtac2	lbfgs	100 000	0.0001	invscaling	80,10	tanh



**FIGURE A1** RMSE values for all input combinations per waterbody.

essential for the assessment of a model's reliability. In this study, we applied perturbation analysis to simulate changes in the input variables. For that, we perturbed every input value except the date by 0.01 (normalized) and evaluated the changes in the mean output. This reliability method is similar to accuracy metrics (comparison of observed vs. predicted data) but differs in that the input is changed.

#### A.3.2. | MinMax analysis

To consider how reasonable an ANN works regarding its predictions, it is useful to know the range of prediction values that the ANN can display. Therefore, we used MinMax analysis, where we chose random input values between 0 and 1 (normalized) to identify the operational range of each ANN. We optimized the initially chosen input and repeated the method 10 times for each ANN for the minimum and 10 times for the maximum value, respectively.

#### A.3.3. | Impact analysis

While the reaction of the ANN to perturbations and its operational range already give a good overview of its reliability, the so-called Impact Analysis, which is a method similar to *sensitivity analysis* (Zurada et al., 1994), can be used to determine which input the ANN

is sensitive to, or, more specifically in our case, can be used to measure the importance of each input feature by determining its contribution to the water temperature calculation. With this information, it can be assessed whether single input parameters are weighted unreasonably high or low and hence predictions of future scenarios might not be reliable.

#### A.4 | Principal component analysis

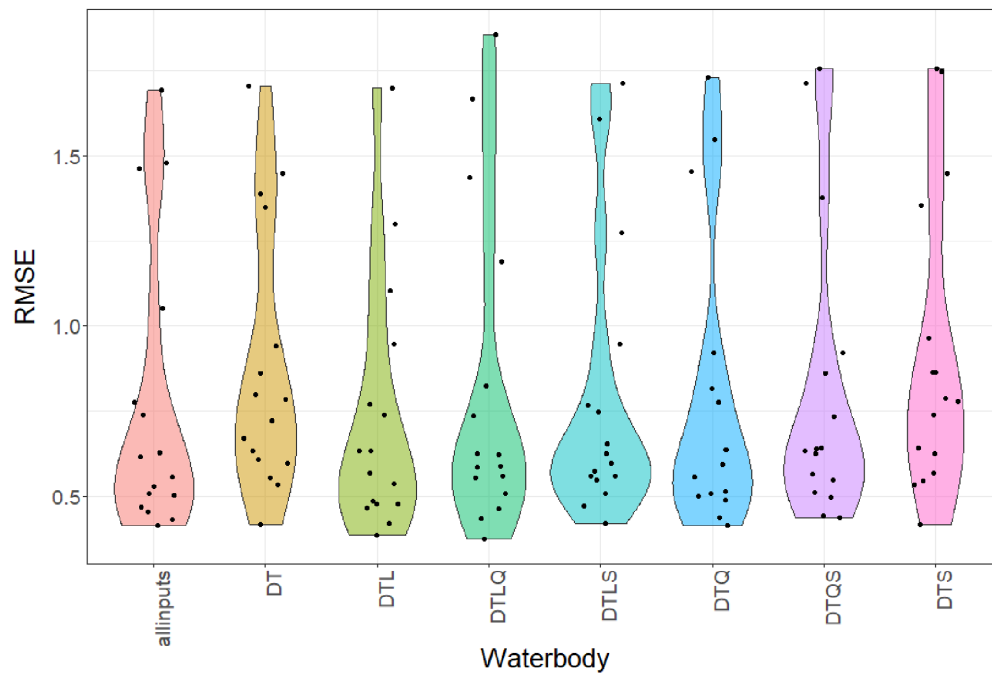
To assess the environmental variables used to distinguish between the 16 assessed streams, we applied a principal component analysis (PCA) based on the normalized environmental variables that we compiled in the environmental dataset. The PCA and all subsequent multivariate analyses were calculated with the statistical software PRIMER v7 & PERMANOVA+ (Anderson et al., 2008).

#### A.5 | Correlation analysis

##### A.5.1. | Reliability metrics

As described for the accuracy metrics above, we also conducted a correlation analysis of the reliability metrics. To do this, we correlated all the environmental parameters of both resolutions (entire catchment





**FIGURE A2** RMSE values for all streams per input combination. Input parameters: D, day of the year; DTQLS, allinputs; L, Water level, Q, discharge; S, sunshine duration; T, air temperature.

Stream	Catchment	Longest river length	MW	HW	NW	MQ	HQ
Prien	52.82	15.15	0.24	220	3	1.70	128.00
Attel	66.14	11.01	0.28	188	10	1.00	24.70
Aubach	13.74	7.91	0.31	197	12	0.40	31.70
Söllbach	24.12	13.15	0.19	173	6	1.08	44.10
Bernauer Ache	36.53	8.80	0.48	215	29	0.80	37.10
Kleine Vils	43.25	9.55	0.51	229	25	0.97	55.90
Illach	25.09	19.99	0.50	225	34	0.79	24.90
Otterbach	91.74	22.79	0.89	235	70	0.83	32.10
Wiesent	135.38	14.46	1.34	195	110	1.05	6.88
Sulzbach	34.69	7.18	1.21	315	103	0.26	12.10
Abens	144.49	19.48	0.33	299	20	0.91	43.60
Aurach	123.59	28.20	1.37	278	110	0.66	20.20
Scheine	63.81	12.22	1.40	316	117	0.41	24.30
Große Ohe	18.70	5.15	0.44	165	19	0.60	23.80
Kirnach	25.31	21.15	0.35	223	10	0.77	49.50
Wolnzach	75.99	13.25	0.21	88	17	0.40	2.90

Note: Catchment, size of catchment (km<sup>2</sup>); HQ, highest water discharge over entire period (m<sup>3</sup>/s); HW, highest water level over entire period (cm); Longest river length, length of longest contributing river (km); MQ, mean water discharge over entire period (m<sup>3</sup>/s); MW, mean water level over total period (m); NW, lowest water level over total period (cm); Stream, name of examined stream.

**TABLE A1** Environmental parameter for distLM I.

**TABLE A2** Environmental parameter for distLM II.

Stream	Gauging station	DOD	IPO	Stations	Distances (km)
Abens	Mainburg	3782	32	02410   <b>05404</b>	19.69   27.67
Attel	Assling	1821	40	01103   <b>04261</b>	12.12   15.73
Aubach	Au	2593	40	<b>04261</b>   03679	14.53   24.45
Aurach	Rothaurach	883	40	04280   <b>03668</b>	4.56   29.13
BernauerAche	Bernau	4815	40	<b>00856</b>   05941	13.90   17.13
GrosseOhe	Taferlruck	4169	40	<b>05800</b>   01832	16.23   28.12
Illach	Engen	1904	32	00125   <b>15 555</b>	12.17   28.56
Kirnach	Unterthingau	1012	40	<b>15 555</b>   02559	12.73   14.15
KleineVils	Dietelskirchen	4355	40	13 710   <b>03366</b>	10.76   26.78
Otterbach	Hammermuehle	8742	40	<b>04104</b>   04559	10.99   30.10
Prien	Aschau	3505	40	05941   <b>04261</b>	15.84   18.01
Scheine	Scheinfeld	996	36	01107   05149	21.55   21.82
Soellbach	BadWiessee	2162	36	02319   03679	20.06   33.67
Sulzbach	Koesfeld	4381	40	<b>00867</b>   07428	3.25   15.31
Wiesent	Hollfeld	2379	40	<b>00320</b>   00282	16.76   27.33
Wolnzach	Wolnzach	817	32	02410   <b>05404</b>	13.56   22.86

Abbreviations: Distances (km), distances between GkD gauging station and DWD station; DOD, number indicating how many days served as data basis for training and testing; Gauging station, name of GkD gauging station from which water temperature, discharge and water level were obtained; IPO, maximum number of input values per output value; Stations, DWD stations from which air temperature data was used, bold indicates that sunshine duration was available (value from closer station preferred if possible); if no station is indicated in bold, no sunshine duration was available; Stream: name of stream investigated.

and 5 m riparian strip) with the mean perturbation values as well as with the results of the MinMax analysis that is, the minimum possible values and the maximum possible values for the ANN with the all-inputs combination for each stream.

#### A.5.2. | Accuracy versus reliability metrics

Finally, we conducted a correlation analysis between the accuracy metrics RMSE, R, and PBIAS and the reliability metrics mean perturbation, minimum of MinMax analysis and maximum of MinMax analysis, to determine whether and to what extent the accuracy and reliability metrics agree with and/or complement each other.

## B | Additional information on Results

### B.1. | Assessment of ANNs

**Impact analysis:** The mean importance, as an indicator of the contribution of individual parameters for water temperature prediction, showed that the impact of individual input parameters strongly varied among streams (Figure B1). Calculating the mean over all streams, we observed that the water level of the current day (L) was the most important, with a value of 14%, followed by the mean air temperature of the closest DWD station of the current day (*mean\_St1*), with a value of 11%. The greatest individual

importance value of 35% was determined for the mean air temperature of the closest DWD station for the current day (*mean\_St1*) at Grosse Ohe, for the water level of the current day (L) at Prien, and for the water level 3 days before the current day (L.3) at Sulzbach. On the other hand, we observed that for all streams, the sunshine duration (S) for the current and the previous days had no impact (0%), contradicting the findings of the accuracy metrics, in which some streams displayed the highest accuracy when S was included as input parameter.

**MinMax analysis:** MinMax analysis was applied to define the specific limits of water temperature prediction for each stream. For this analysis, values between 0 and 1 (normalized) were randomly recombined to identify the ANN's minimum and maximum water temperature predictions for each stream. The results of the MinMax analysis were in line with the above results, showing that the maximum and minimum range of the calculated values varied strongly depending on the stream's specifics.

Observed water temperature minima ( $0.34 \pm 1.10^\circ\text{C}$ ) and maxima ( $19.55 \pm 2.50^\circ\text{C}$ ) in the dataset differed from calculated minima ( $-11.96 \pm 12.06^\circ\text{C}$ ) and maxima ( $74.05 \pm 27.34^\circ\text{C}$ ). The mean delta (observed values minus calculated value) for the minimum values was  $12.30 \pm 11.88^\circ\text{C}$ , with a maximum delta of  $40.64^\circ\text{C}$  (Sulzbach) and a minimum of  $0.09^\circ\text{C}$  (Kleine Vils). The mean delta for the maximum values was  $-54.50 \pm 26.23^\circ\text{C}$  with a maximum of  $-10.67^\circ\text{C}$  (Wiesent) and a minimum of  $-106.89^\circ\text{C}$  (Kirnach) (see Figure B2).

Waterbody	Inputs	Hyperparameter	RMSE	R	PBIAS	NSE
Prien	allinputs	rec	0.738	0.969	−0.111	0.938
Prien	DT	loc	0.940	0.949	−0.042	0.900
Prien	DTQ	loc	0.816	0.962	0.411	0.924
Prien	DTQS	rec	0.733	0.969	0.112	0.939
Prien	DTL	rec	0.737	0.969	0.237	0.938
Prien	DTLQ	wtac2	0.824	0.962	0.290	0.923
Prien	DTLS	rec	0.747	0.968	0.074	0.937
Prien	DTS	werec	0.962	0.946	−0.345	0.895
Mean			0.812	0.962	0.078	0.924
SD			0.087	0.009	0.227	0.017
Var			0.008	0.000	0.051	0.000
Attel	allinputs	rec	0.453	0.995	0.004	0.991
Attel	DT	loc	0.596	0.992	0.071	0.984
Attel	DTQ	tac	0.498	0.994	0.103	0.989
Attel	DTQS	rec	0.547	0.993	0.000	0.986
Attel	DTL	rec	0.464	0.995	0.108	0.990
Attel	DTLQ	tac	0.462	0.995	−0.127	0.990
Attel	DTLS	rec	0.546	0.993	0.158	0.986
Attel	DTS	tac	0.862	0.984	0.110	0.966
mean			0.554	0.993	0.053	0.985
SD			0.126	0.004	0.085	0.008
Var			0.016	0.000	0.007	0.000
Aubach	allinputs	rec	0.412	0.996	0.256	0.992
Aubach	DT	rec	0.415	0.996	0.314	0.992
Aubach	DTQ	tac	0.413	0.996	0.282	0.992
Aubach	DTQS	tac	0.443	0.995	0.272	0.991
Aubach	DTL	tac	0.385	0.997	0.163	0.993
Aubach	DTLQ	tac	0.373	0.997	0.033	0.994
Aubach	DTLS	loc	0.559	0.993	−0.066	0.986
Aubach	DTS	tac	0.416	0.996	0.424	0.992
mean			0.427	0.996	0.210	0.991
SD			0.054	0.001	0.149	0.002
Var			0.003	0.000	0.022	0.000
Soellbach	DT	rec	0.533	0.989	−0.183	0.977
Soellbach	DTQ	tac	0.437	0.992	−0.003	0.985
Soellbach	DTL	wtac2	0.419	0.993	−0.090	0.986
Soellbach	DTLQ	rec	0.433	0.992	−0.093	0.985
Mean			0.455	0.992	−0.079	0.983
SD			0.045	0.002	0.072	0.003
Var			0.002	0.000	0.005	0.000

**TABLE A3** Results of randomized search for Prien, Attel, Aubach and Soellbach, for explanations see Section A.2.

**TABLE A4** Results of randomized search for Bernauer Ache, Kleine Vils, Illach and Otterbach, for explanations see Section A.2.

Waterbody	Inputs	Hyperparameter	RMSE	R	PBIAS	NSE
Bernauer Ache	allinputs	loc	0.616	0.986	−0.350	0.973
Bernauer Ache	DT	tac	0.784	0.978	−0.731	0.956
Bernauer Ache	DTQ	tac	0.637	0.986	−0.378	0.971
Bernauer Ache	DTQS	rec	0.641	0.985	−0.310	0.971
Bernauer Ache	DTL	loc	0.634	0.986	−0.458	0.971
Bernauer Ache	DTLQ	tac	0.586	0.988	−0.315	0.976
Bernauer Ache	DTLS	rec	0.653	0.985	−0.562	0.970
Bernauer Ache	DTS	loc	0.787	0.978	−0.570	0.956
mean			0.667	0.984	−0.459	0.968
SD			0.071	0.004	0.140	0.007
Var			0.005	0.000	0.020	0.000
Kleine Vils	allinputs	loc	0.555	0.997	0.152	0.993
Kleine Vils	DT	tac	0.632	0.996	−0.077	0.991
Kleine Vils	DTQ	rec	0.555	0.997	−0.025	0.993
Kleine Vils	DTQS	rec	0.564	0.997	−0.081	0.993
Kleine Vils	DTL	tac	0.535	0.997	−0.105	0.994
Kleine Vils	DTLQ	tac	0.552	0.997	−0.126	0.993
Kleine Vils	DTLS	tac	0.572	0.997	−0.131	0.993
Kleine Vils	DTS	tac	0.640	0.996	−0.178	0.991
mean			0.576	0.996	−0.071	0.993
SD			0.036	0.000	0.094	0.001
Var			0.001	0.000	0.009	0.000
Illach	allinputs	loc	0.503	0.994	−0.136	0.989
Illach	DT	loc	0.721	0.988	−0.559	0.977
Illach	DTQ	rec	0.592	0.992	−0.659	0.984
Illach	DTQS	rec	0.633	0.991	−0.470	0.982
Illach	DTL	tac	0.567	0.993	−0.026	0.986
Illach	DTLQ	rec	0.622	0.991	−0.091	0.983
Illach	DTLS	rec	0.560	0.993	−0.310	0.986
Illach	DTS	rec	0.737	0.988	−0.967	0.976
mean			0.617	0.991	−0.402	0.983
SD			0.075	0.002	0.302	0.004
Var			0.006	0.000	0.091	0.000
Otterbach	allinputs	rec	1.693	0.957	−0.254	0.915
Otterbach	DT	rec	1.704	0.956	−0.295	0.914
Otterbach	DTQ	werec	1.730	0.955	−1.134	0.912
Otterbach	DTQS	werec	1.713	0.956	−0.889	0.913
Otterbach	DTL	rec	1.700	0.956	−0.160	0.915
Otterbach	DTLQ	rec	1.667	0.958	−0.248	0.918
Otterbach	DTLS	werec	1.714	0.956	0.069	0.913
Otterbach	DTS	werec	1.747	0.954	0.369	0.910
mean			1.708	0.956	−0.318	0.914
SD			0.023	0.001	0.454	0.002
Var			0.001	0.000	0.206	0.000

Waterbody	Inputs	Hyperparameter	RMSE	R	PBIAS	NSE
Wiesent	allinputs	rec	0.626	0.985	−0.060	0.970
Wiesent	DT	rec	0.798	0.976	0.076	0.951
Wiesent	DTQ	tac	0.637	0.985	0.077	0.969
Wiesent	DTQS	rec	0.623	0.985	0.158	0.970
Wiesent	DTL	tac	0.633	0.985	0.297	0.969
Wiesent	DTLQ	tac	0.625	0.985	0.153	0.970
Wiesent	DTLS	rec	0.624	0.985	−0.003	0.970
Wiesent	DTS	rec	0.778	0.977	−0.249	0.954
mean			0.668	0.983	0.056	0.965
SD			0.070	0.004	0.154	0.008
Var			0.005	0.000	0.024	0.000
Sulzbach	allinputs	rec	0.774	0.989	−0.176	0.978
Sulzbach	DT	rec	0.861	0.986	−0.380	0.972
Sulzbach	DTQ	rec	0.776	0.989	−0.309	0.977
Sulzbach	DTQS	tac	0.861	0.986	−0.424	0.972
Sulzbach	DTL	tac	0.770	0.989	−0.320	0.978
Sulzbach	DTLQ	tac	0.736	0.990	−0.286	0.980
Sulzbach	DTLS	rec	0.766	0.989	−0.310	0.978
Sulzbach	DTS	loc	0.863	0.986	−0.380	0.972
mean			0.801	0.988	−0.323	0.976
SD			0.049	0.001	0.071	0.003
Var			0.002	0.000	0.005	0.000
Abens	allinputs	tac	0.468	0.995	−0.084	0.990
Abens	DT	loc	0.670	0.990	0.081	0.980
Abens	DTQ	wtac2	0.512	0.994	0.094	0.988
Abens	DTQS	rec	0.496	0.994	−0.040	0.989
Abens	DTL	wtac2	0.474	0.995	0.135	0.990
Abens	DTLQ	rec	0.585	0.992	−0.045	0.984
Abens	DTLS	tac	0.471	0.995	−0.046	0.990
Abens	DTS	rec	0.623	0.991	0.045	0.982
mean			0.537	0.993	0.017	0.987
SD			0.073	0.002	0.076	0.004
Var			0.005	0.000	0.006	0.000
Aurach	allinputs	rec	1.464	0.961	0.507	0.924
Aurach	DT	rec	1.388	0.965	−0.082	0.931
Aurach	DTQ	rec	1.549	0.957	1.075	0.914
Aurach	DTQS	rec	1.756	0.944	0.877	0.890
Aurach	DTL	rec	1.301	0.969	−0.113	0.940
Aurach	DTLQ	rec	1.436	0.963	0.498	0.926
Aurach	DTLS	rec	1.609	0.954	0.912	0.908
Aurach	DTS	rec	1.355	0.967	0.632	0.934
mean			1.482	0.960	0.538	0.921
SD			0.140	0.008	0.413	0.015
Var			0.019	0.000	0.170	0.000

**TABLE A5** Results of randomized search for Wiesent, Sulzbach, Abens and Aurach, for explanations see Section A.2.

**TABLE A6** Results of randomized search for Scheine, Grosse Ohe, Kirnach and Wolnzach, for explanations see Section A.2.

Waterbody	Inputs	Hyperparameter	RMSE	R	PBIAS	NSE
Scheine	DT	rec	1.449	0.949	0.948	0.900
Scheine	DTQ	rec	0.920	0.980	−0.676	0.960
Scheine	DTL	rec	0.947	0.978	−0.078	0.957
Scheine	DTLQ	rec	1.857	0.921	−1.687	0.836
mean			1.193	0.964	−0.191	0.927
SD			0.328	0.020	0.816	0.042
Var			0.108	0.000	0.666	0.002
Grosse Ohe	allinputs	rec	0.506	0.993	−0.440	0.987
Grosse Ohe	DT	tac	0.607	0.990	−0.603	0.981
Grosse Ohe	DTQ	wtac2	0.488	0.994	−0.517	0.988
Grosse Ohe	DTQS	loc	0.509	0.993	−0.561	0.986
Grosse Ohe	DTL	tac	0.483	0.994	−0.344	0.988
Grosse Ohe	DTLQ	tac	0.508	0.993	−0.297	0.987
Grosse Ohe	DTLS	rec	0.508	0.993	−0.309	0.986
Grosse Ohe	DTS	loc	0.566	0.992	−0.651	0.983
mean			0.522	0.993	−0.465	0.986
SD			0.040	0.001	0.129	0.002
Var			0.002	0.000	0.017	0.000
Kirnach	allinputs	rec	1.479	0.964	−1.794	0.926
Kirnach	DT	rec	1.348	0.969	0.074	0.939
Kirnach	DTQ	rec	1.453	0.964	−0.792	0.929
Kirnach	DTQS	rec	1.378	0.968	−1.317	0.936
Kirnach	DTL	rec	1.104	0.979	−0.767	0.959
Kirnach	DTLQ	loc	1.188	0.976	−1.093	0.952
Kirnach	DTLS	rec	1.273	0.972	−0.184	0.945
Kirnach	DTS	werec	1.755	0.947	−0.482	0.896
mean			1.372	0.967	−0.794	0.935
SD			0.187	0.009	0.569	0.018
Var			0.035	0.000	0.323	0.000
Wolnzach	allinputs	rec	0.528	0.985	0.426	0.970
Wolnzach	DT	rec	0.554	0.983	0.182	0.967
Wolnzach	DTQ	loc	0.508	0.987	0.659	0.972
Wolnzach	DTQS	rec	0.638	0.979	0.066	0.956
Wolnzach	DTL	rec	0.476	0.988	−0.048	0.976
Wolnzach	DTLQ	rec	0.558	0.983	0.005	0.966
Wolnzach	DTLS	rec	0.596	0.981	0.778	0.962
Wolnzach	DTS	rec	0.543	0.984	0.305	0.968
mean			0.550	0.984	0.297	0.967
SD			0.047	0.003	0.285	0.006
Var			0.002	0.000	0.081	0.000

**Perturbation analysis:** We applied perturbation analysis to test the degree to which the ANNs' predictions changed when historical input data varied by 0.01 (normalized). The mean perturbation value over all streams was  $2.620 \pm 2.109^\circ\text{C}$ , with the highest mean perturbation observed in Otterbach ( $9.981^\circ\text{C}$ ) and the lowest in Wolnzach ( $0.985^\circ\text{C}$ ).

## B.2. | Relationship between accuracy and reliability metrics

Correlation analysis (see Table B1) resulted in one highly significant correlation between RMSE and mean perturbation ( $\rho = 0.853$ ,  $p < 0.001$ ) and three significant correlations: between  $R$  and mean perturbation ( $\rho = -0.599$ ,  $p < 0.05$ ), between RMSE and MinMax-max ( $\rho = 0.538$ ,  $p < 0.05$ ) and between PBIAS and MinMax-max ( $\rho = -0.582$ ,  $p < 0.05$ ). There were no significant correlations between any of the accuracy metrics and MinMax-Min.

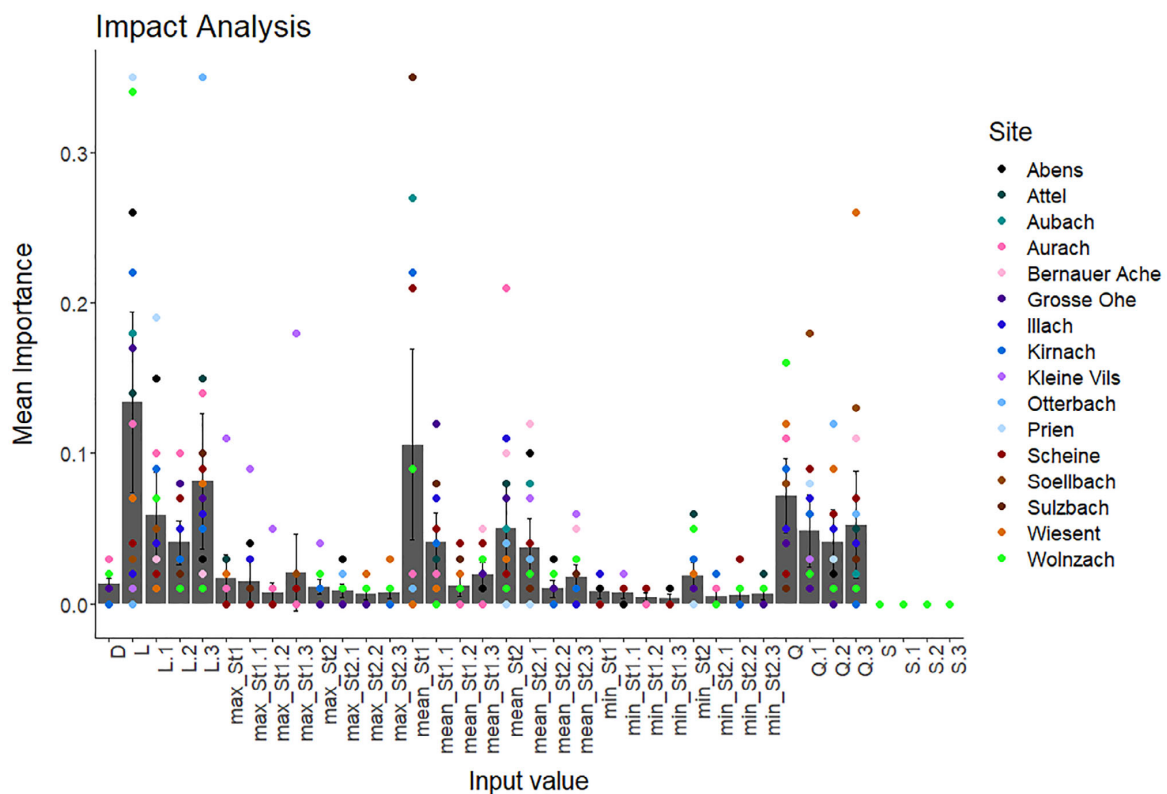
## B.3. | Environmental characteristics of sites

The PCA of environmental conditions across the streams (Figure B3) showed that the 16 sites were broadly distributed along multiple environmental gradients. The first PC axis, covering 26.8% of the observed

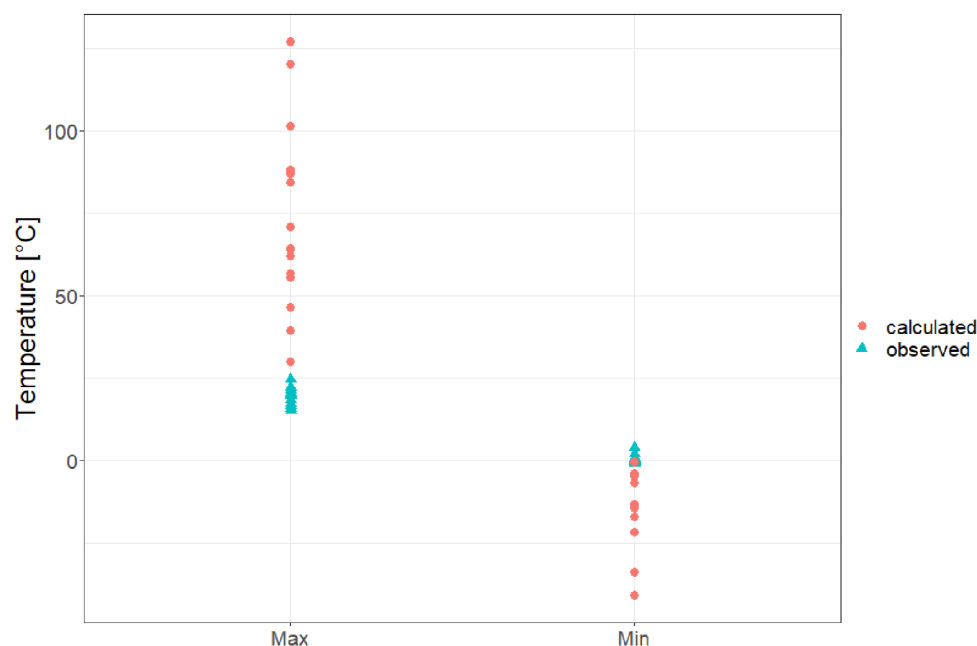
variation (Eigenvalue = 6.44), structured streams primarily according to the proportion of natural and forested vegetation and water bodies in their surroundings. It exemplifies that the streams Grosse Ohe, Soellbach and Bernauer Ache feature a higher share of natural vegetation than such streams as the Scheine, Kleine Vils or Sulzbach. Also hydrological features of the streams investigated, such as NW and HW, were reflected by PC1, with streams in the negative space of PC1 tending to have higher mean and high water levels than those in the positive space. The second PC axis, making up 16.6% of the observed variation in the data set (Eigenvalue = 3.98), grouped streams largely according to the proportion of agricultural and urban land use (with a high share, for example, along Sulzbach and Wolnzach and a low share in Kirmach, Prien and Illach), while the proportion of grassland in the surroundings and the total length of the river upstream from the sampling site grouped streams in the opposite direction. Detailed proportions of land use are depicted in Table C1. Further information on environmental parameters is depicted in Tables A1 and A2.

## B.4. | Environmental predictors of ANN reliability metrics

Regarding the overall catchment resolution (Table B2 top), we determined a significantly positive correlation between mean perturbation



**FIGURE B1** Impact analysis for all input parameters in each stream. Inputs on x-axis indexed as below. Whiskers mark 95% confidence intervals and bars mark mean importance for each input. St indicates the station from which air temperature was received (St1 = DWD station closest to GkD gauging station, St2 = DWD station second-closest to GkD gauging station). The “addendum.No” indicates how many days prior to D the data is from (0.1 = the day before D, 0.2 = 2 days before D, 0.3 = 3 days before D). Input values: D, day of the year; L, water level; max, maximum air temperature; mean, mean air temperature; min, minimum air temperature; Q, discharge; S, sunshine duration.



**FIGURE B2** Comparison of calculated and observed minimum and maximum values for all waterbodies. Calculated values were determined by MinMax analysis, observed values were retrieved from the datasets.

**TABLE B1** Correlations between evaluation and assessment metrics.

	RMSE	R	PBIAS
Mean perturbation	0.853***	−0.600*	−0.185
MinMax_min	−0.053	0.157	−0.091
MinMax_max	0.538*	−0.213	−0.582*

\* $p < 0.05$ ; \*\* $p < 0.01$ ; \*\*\* $p < 0.001$ .

and HW ( $r = 0.67$ ,  $p < 0.01$ ) as well as a significantly negative relationship between the minimum values of the MinMax analysis and DOD ( $r = -0.60$ ,  $p < 0.05$ ) and between the maximum values of the MinMax analysis and semi-natural ( $r = -0.55$ ,  $p < 0.05$ ). For the 5 m riparian strip resolution (see Table B2 bottom), there was a significantly positive correlation between mean perturbation and grassland ( $r = 0.52$ ,  $p < 0.05$ ) as well as between the minimum values of the MinMax analysis and grassland ( $r = 0.50$ ,  $p < 0.05$ ).

## C. | Additional information on the Discussion

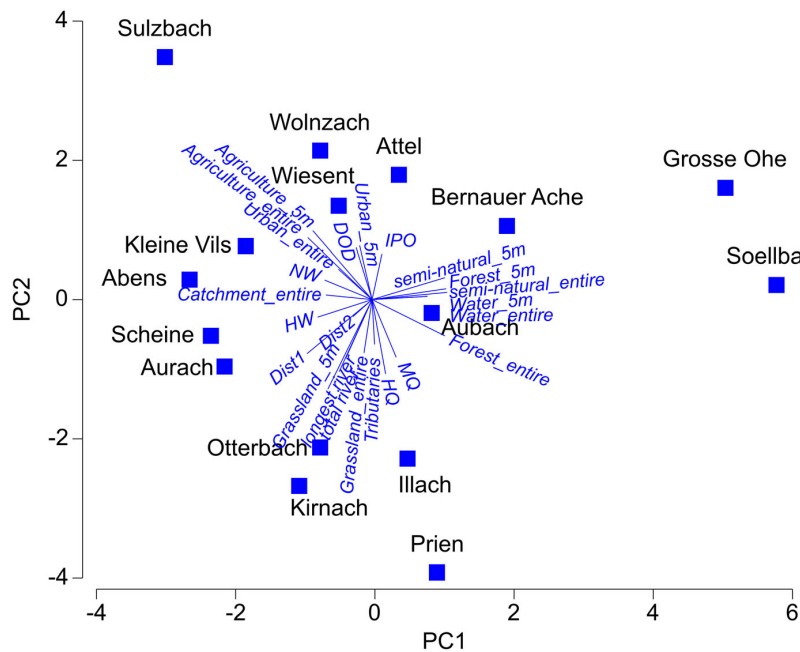
### C.1. | Accuracy and reliability

Not all accuracy vs. reliability metrics correlated significantly. This confirmed the finding of Mohr et al. (2021), that the use of accuracy metrics alone is not sufficient and must be supplemented with reliability metrics. Still, we can conclude that as accuracy metric, the RMSE is the most suitable one of those we used to reflect the reliability of an ANN. We are able to conclude this thanks to the significant correlations both to mean perturbation and to the maximum

values obtained in the MinMax analysis, while only one significant correlation was demonstrated for R and PBIAS, respectively. We also observed that the RMSE had a greater resolution and hence contributed more significant relationships with environmental parameters than R, probably because it had a higher potential to reflect the high-resolution dynamics of hydrologic parameters. This further increased the benefit of the RMSE and confirms the plausibility of its frequent use for measuring the accuracy of water temperature prediction with ANNs (Ahmadi-Nedushan et al., 2007; Caissie et al., 1998; Chenard & Caissie, 2008; Cho & Lee, 2012; Feigl et al., 2021; Graf et al., 2019; Hadzima-Nyarko et al., 2014; Qiu et al., 2020; Quan et al., 2020; Rabi et al., 2015; Rahmani et al., 2020; Rehana, 2019; St-Hilaire et al., 2000; Zhu, Nyarko, Hadzima-Nyarko, et al., 2019).

In this study, we additionally employed PBIAS as an accuracy metric, which is unusual for water temperature prediction with ANNs. Although we see advantages in combining different accuracy metrics and including the PBIAS due to the different aspects of model performance it highlights, in this study we were not able to find any general correlations between the environmental parameters





**FIGURE B3** Principal component analysis plot, all stream sites broadly distributed along multiple environmental gradients. Streams structured by PC axis 1 mainly according to the proportion of natural and forested vegetation and waterbodies in their surroundings, as well as by NW and HW. Structure by PC axis 2 mainly according to proportion of agricultural and urban land use.

**TABLE B2** Correlation analysis of environmental parameters versus robustness measures, entire catchment and 5 m riparian strip.

	Perturbation	Min	Max
Entire catchment			
Total river length	0.406	-0.141	0.191
Longest river length	0.344	0.238	0.413
Agriculture	0.020	0.065	-0.128
Forest	0.003	0.000	-0.178
Grassland	0.456	0.026	0.426
Semi-natural	-0.328	-0.081	-0.546*
Urban	0.068	0.135	-0.074
Water	-0.140	0.084	-0.028
Catchment	0.044	0.221	-0.326
MW	0.421	-0.259	0.459
HW	0.671**	-0.191	0.495
NW	0.250	-0.225	0.268
MQ	-0.041	0.321	-0.289
HQ	0.279	0.115	0.147
DOD	0.038	-0.588*	-0.171
IPO	-0.050	-0.442	-0.070
Dist1	0.397	0.265	0.408
Dist2	0.121	0.024	0.283
Tributaries	0.324	-0.112	0.094
5 m riparian strip			
Total river length	0.406	-0.141	0.191
Longest river length	0.344	0.238	0.203
Agriculture	0.037	0.197	-0.009

TABLE B2 (Continued)

	Perturbation	Min	Max
Forest	−0.421	−0.026	−0.309
Grassland	0.521*	0.001	0.498*
Semi-natural	−0.365	−0.091	−0.220
Urban	−0.162	0.110	−0.415
Water	−0.140	0.084	−0.028
Area	0.341	−0.074	0.097

Abbreviations: Agriculture, forest, grassland, semi-natural, urban, water: land use; Area, total area of riparian strip; Area, total buffer area; Catchment, Total size of all contributing catchments; D, Day of the year; Dist1, distance between GkD station and DWD station 1; Dist2, distance between GkD station and DWD station 2; DOD, number of days for which data was used; DTQLS, allinputs; HQ, highest measured discharge; HW, highest measured water level; IPO, number of input data points per output data point; L, water level; Longest river length, the length of the longest contributing river; Max, maximum determined by MinMax-analysis; Min, minimum determined by MinMax-analysis; MQ, mean discharge; MW, mean water level; NW, lowest measured water level; Perturbation, mean perturbation determined by perturbation analysis; Q, discharge; S, sunshine duration; T, air temperature; Total river length, sum of lengths of all contributing rivers; Tributaries, number of tributaries.

\* $p < 0.05$ ; \*\* $p < 0.01$ ; \*\*\* $p < 0.001$ .

and the PBIAS. This might be because the PBIAS reflects variation in two directions, but the direction of estimation (over- or underestimation) does not necessarily correlate with an environmental parameter in only one direction. Even though the PBIAS showed no consistent trends in the correlation analysis with either environmental parameters or reliability metrics, the significant correlation between the PBIAS and the maximum values of the MinMax analysis showed that with increasing MinMax-max values, the ANNs tended to overestimate water temperature. This overestimation was pronounced for Kirnach, a stream with a very high proportion of grassland (72.58%)

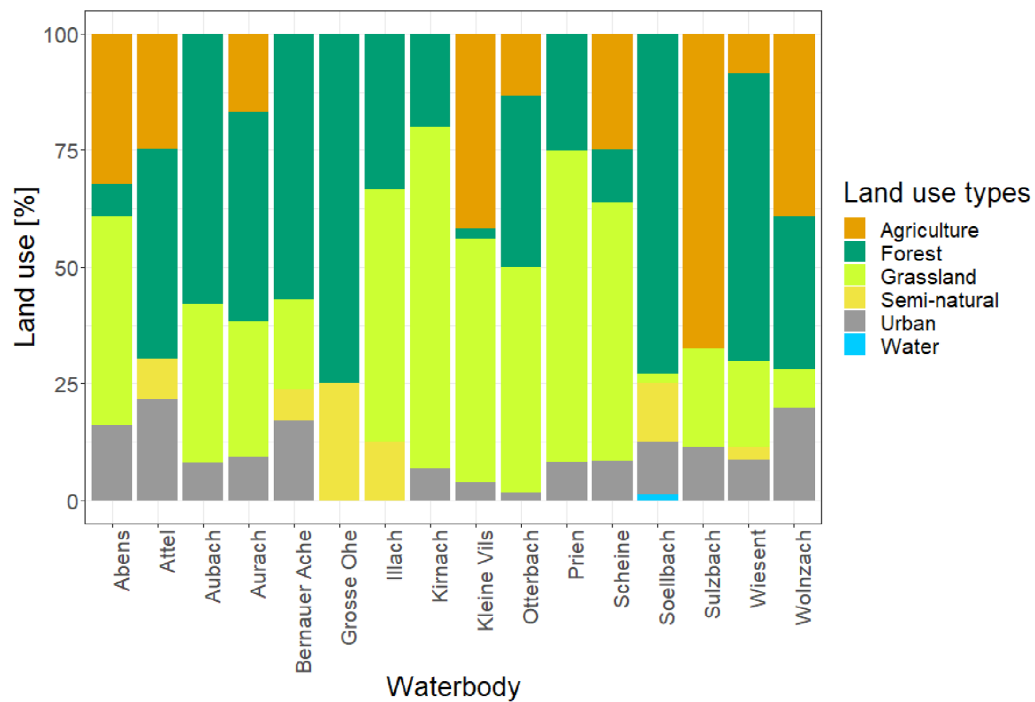
and a very low proportion of semi-natural land cover (0.01%). In contrast, underestimation of water temperature was pronounced for Aurach, a long stream with a large catchment. These findings were also confirmed by the DistLM analysis of environmental predictors of evaluation metrics, in which PBIAS/overestimation was associated with high proportions of grassland, particularly in the 5 m riparian strip. Consequently, it would be advisable to carefully check for both over- and underestimation of the water temperature prediction, particularly in catchments with high proportions of open-canopy landscape (Figures C1–C6).

**TABLE C1** Land use in entire catchment and in 5 m riparian strip.

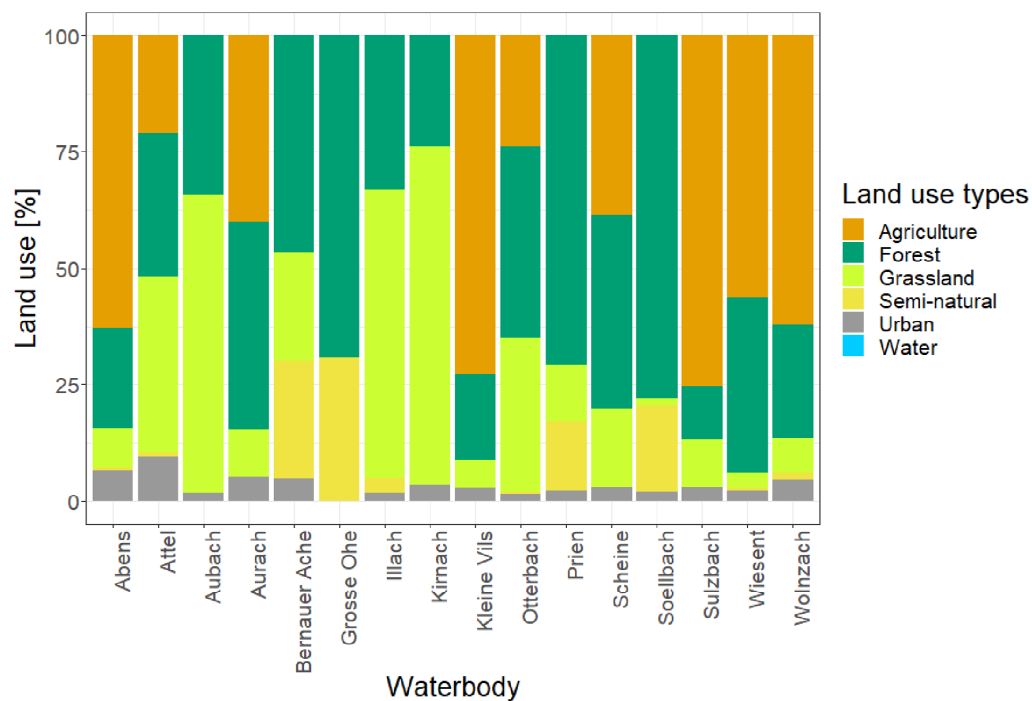
Stream	Total river length	Agriculture	Forest	Grassland	Semi-natural	Urban	Water
Prien	15.15	x	70.91	12.03	14.98	2.09	x
Attel	11.01	21.03	30.81	37.74	1.04	9.38	x
Aubach	7.91	x	34.30	63.92	x	1.79	x
Soellbach	13.15	x	77.92	1.62	18.58	1.83	0.05
Bernauer Ache	8.80	0.08	46.55	23.23	25.37	4.77	x
Kleine Vils	9.55	72.68	18.43	6.01	x	2.88	x
Illach	19.99	x	33.04	61.96	3.32	1.68	x
Otterbach	41.43	23.89	41.12	33.17	0.31	1.52	x
Wiesent	26.13	56.25	37.62	3.27	0.71	2.15	x
Sulzbach	13.89	75.33	11.55	10.12	x	2.99	x
Abens	19.48	62.78	21.59	8.45	0.81	6.38	x
Aurach	28.20	40.06	44.56	10.26	x	5.12	x
Scheine	12.22	38.62	41.44	17.04	x	2.89	x
Große Ohe	11.95	x	69.22	x	30.78	x	x
Kirnach	39.51	x	23.91	72.58	0.01	3.50	x
Wolnzach	13.25	62.06	24.52	7.47	1.43	4.51	x
Stream	Mean river length	Agriculture	Forest	Grassland	Semi-natural	Urban	Water
Prien	15.15	x	25.06	66.77	x	8.18	x
Attel	11.01	24.65	45.09	x	8.75	21.51	x
Aubach	7.91	x	58.12	34.00	x	7.88	x
Soellbach	13.15	x	73.06	1.81	12.62	11.33	1.17
Bernauer Ache	8.80	x	57.02	19.18	6.68	17.12	x
Kleine Vils	9.55	41.76	2.14	52.30	x	3.79	x
Illach	19.99	x	33.26	54.14	12.60	x	
Otterbach	13.81	14.44	44.64	44.72	x	3.05	x
Wiesent	13.07	8.07	62.19	17.77	6.49	8.74	x
Sulzbach	6.95	67.38	x	21.31	x	11.31	x
Abens	19.48	32.37	6.86	44.65	x	16.12	x
Aurach	28.20	16.67	45.02	29.06	x	9.25	x
Scheine	12.22	24.94	11.21	55.46	x	8.39	x
Große Ohe	3.98	x	73.46	x	26.54	x	x
Kirnach	19.75	x	21.32	70.58	x	8.1	x
Wolnzach	13.25	39.14	32.86	8.16	x	19.84	x

Note: Top: land use in entire catchment. Bottom: land use in 5 m riparian strip for whole river.

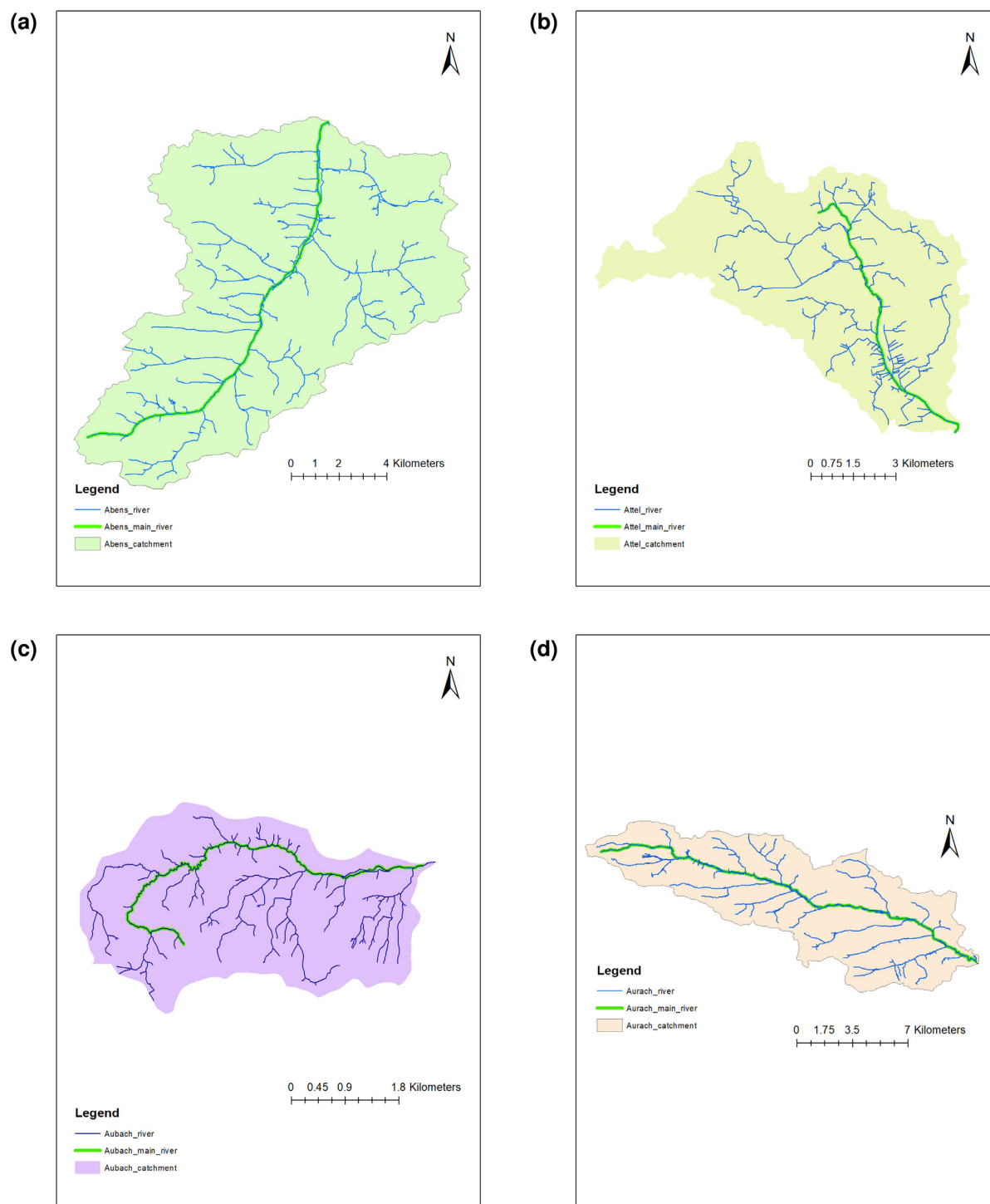
Abbreviations: Agriculture, forest, grassland, semi-natural, urban, water: Proportion of land use in percent, for 5 m riparian strip as mean over all arms; Mean river length: (for 5 m riparian strips) If stream contained more than one arm, this is the mean of the lengths of the arms in km; Stream, name of stream investigated; Total river length, sum of lengths of all contributing rivers in km.



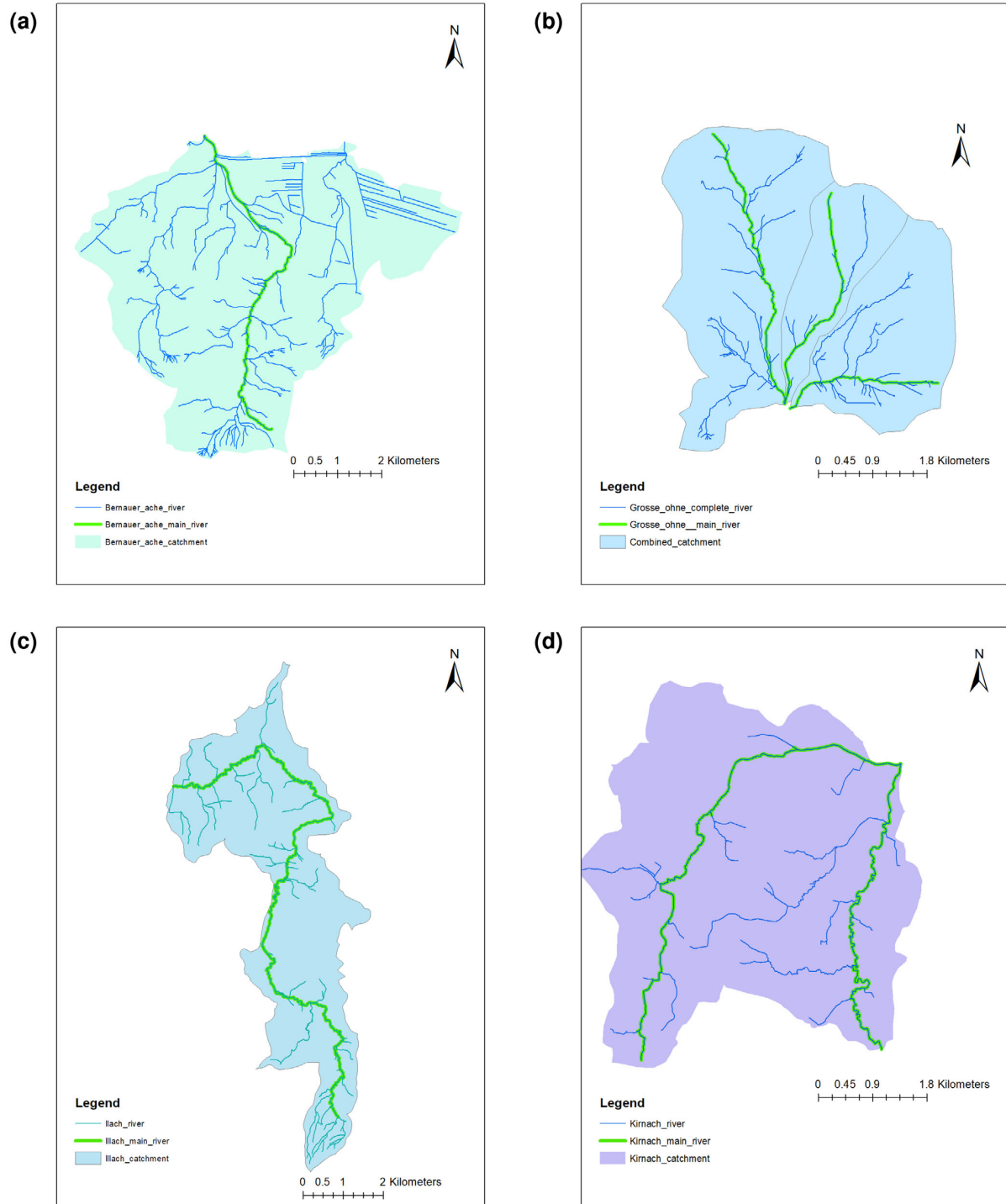
**FIGURE C1** Cumulative barplot illustrating the shares of land use in the 5 m riparian strip.



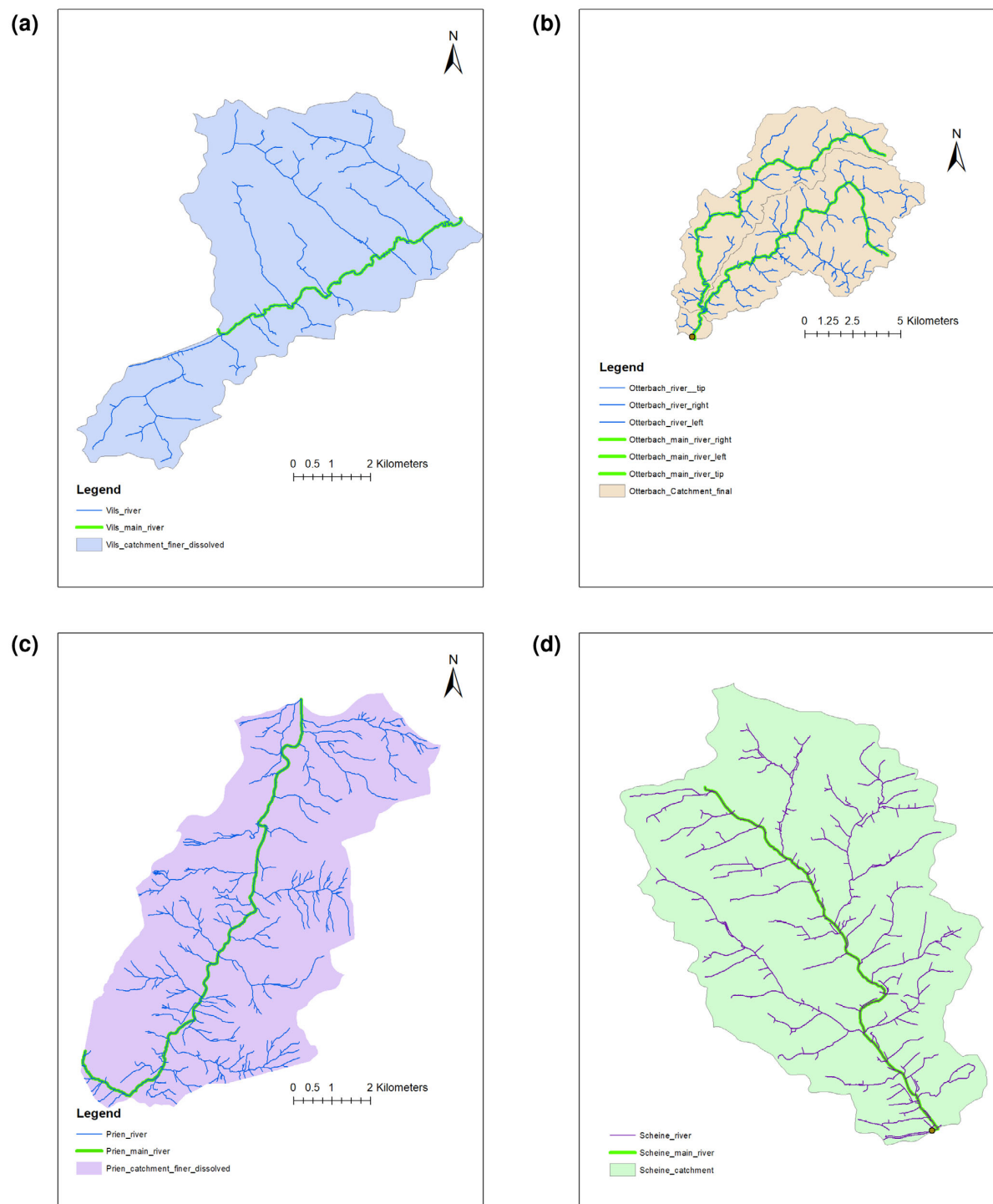
**FIGURE C2** Cumulative barplot illustrating the shares of land use in the entire catchment.



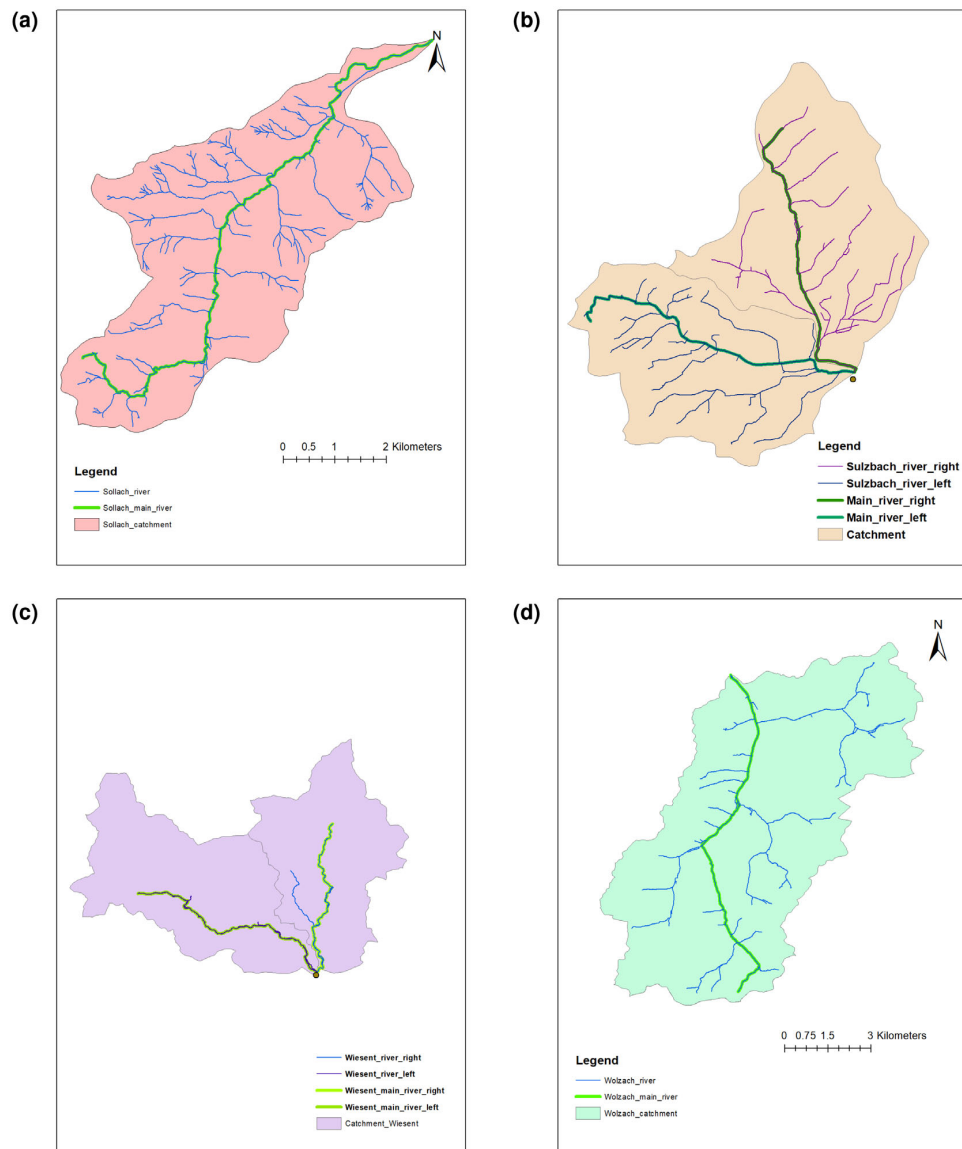
**FIGURE C3** Catchments (a) Abens, (b) Attel, (c) Aubach, and (d) Aurach.



**FIGURE C4** Catchments (a) Bernauer Ache, (b) Grosse Ohe, (c) Illach, and (d) Kirmach.



**FIGURE C5** Catchments (a) Kleine Vils, (b) Otterbach, (c) Prien, and (d) Scheine.



**FIGURE C6** Catchments (a) Soellbach, (b) Sulzbach, (c) Wiesent, and (d) Wolnzach.



