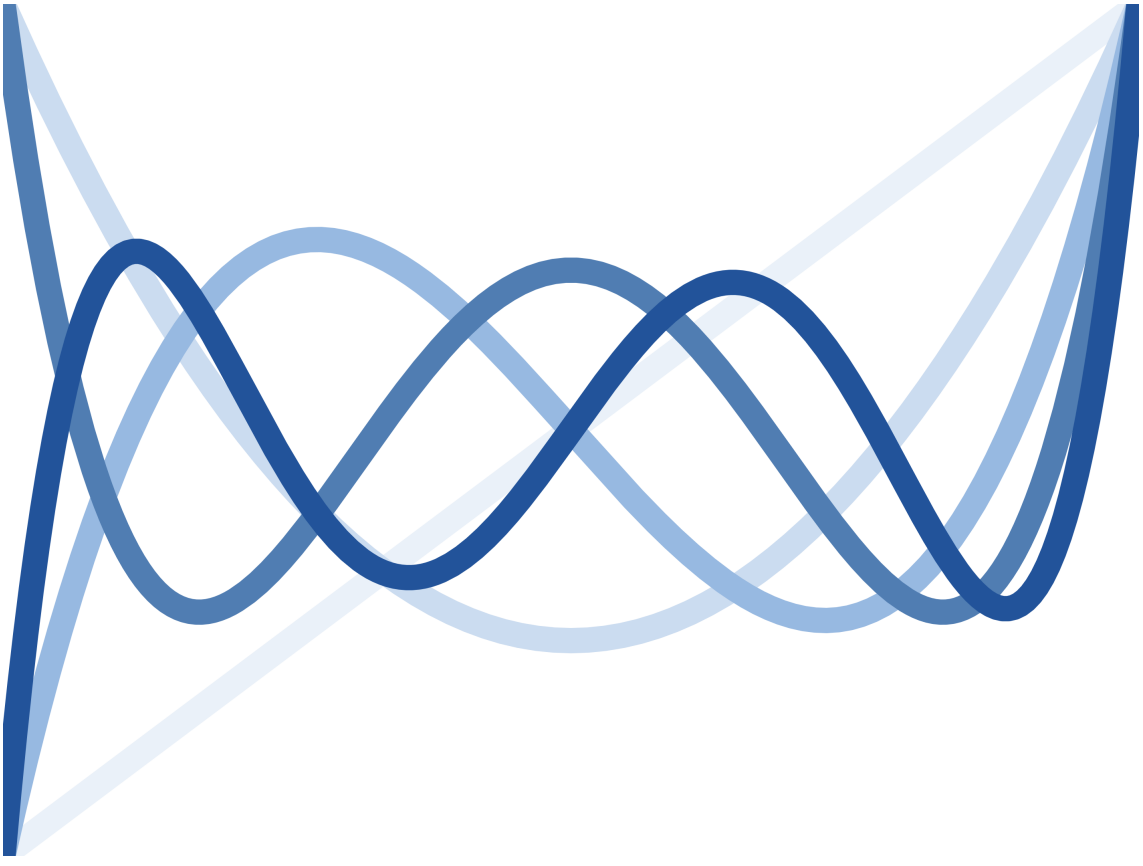
SE | RWTH AACHEN UNIVERSITY

Tobias Schulte to Brinke

Analysis of Information Processing and Memory Prerequisites for Temporal Difference Learning in Cortical Neural Network Models



Aachener Informatik-Berichte,
Software Engineering
Hrsg: Prof. Dr. rer. nat. Bernhard Rumpe
Prof. Dr. rer. nat. Abigail Morrison

[Sch25] T. Schulte to Brinke:
Analysis of Information Processing and Memory Prerequisites for Temporal Difference Learning in Cortical Neural Network Models.
Aachener Informatik-Berichte, Software Engineering, Band 61,
ISBN 978-3-8191-0192-2, Shaker Verlag, Sep. 2025.

Analysis of Information Processing and Memory Prerequisites for Temporal Difference Learning in Cortical Neural Network Models

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M.Sc. TU Ilmenau
Tobias Schulte to Brinke
aus Diepholz, Deutschland

Berichter: Univ.-Prof Dr. rer. nat. Abigail Morrison
Univ.-Prof Dr. rer. nat. Björn Kampa

Tag der mündlichen Prüfung: 16. Juni 2025

Eidesstattliche Erklärung

Tobias Schulte to Brinke

erklärt hiermit, dass diese Dissertation und die darin dargelegten Inhalte die eigenen sind und selbstständig, als Ergebnis der eigenen originären Forschung, generiert wurden.

Hiermit erkläre ich an Eides statt

1. Diese Arbeit wurde vollständig oder größtenteils in der Phase als Doktorand dieser Fakultät und Universität angefertigt;
2. Sofern irgendein Bestandteil dieser Dissertation zuvor für einen akademischen Abschluss oder eine andere Qualifikation an dieser oder einer anderen Institution verwendet wurde, wurde dies klar angezeigt;
3. Wenn immer andere eigene- oder Veröffentlichungen Dritter herangezogen wurden, wurden diese klar benannt;
4. Wenn aus anderen eigenen- oder Veröffentlichungen Dritter zitiert wurde, wurde stets die Quelle hierfür angegeben. Diese Dissertation ist vollständig meine eigene Arbeit, mit der Ausnahme solcher Zitate;
5. Alle wesentlichen Quellen von Unterstützung wurden benannt;
6. Wenn immer ein Teil dieser Dissertation auf der Zusammenarbeit mit anderen basiert,wurde von mir klar gekennzeichnet, was von anderen und was von mir selbst erarbeitet wurde;
7. Teile dieser Arbeit wurden zuvor veröffentlicht, ersichtlich im Abschnitt *Publications and contributions*.

Aachen, Januar 2024

Abstract

This doctoral thesis delves into the computational intricacies of the human brain, exploring the capabilities of cortical microcircuit models, the extent of their information processing capacity, and their role in memory and temporal difference learning through the use of cortico-striatal populations. Central to this exploration is the study of spiking neural networks (SNNs). This research aims to gain a deeper understanding of the structural and neuronal influences on information processing in these networks and at the same time to provide a guideline for their analysis.

In the first part, a network model of a cortical column introduced in a previous paper is reproduced and extended. These analyses show that the specific, data-based connectivity improves computational performance by sharpening the clarity of internal representations rather than increasing the duration of information retention as previously described.

Moving beyond traditional task-based evaluations, the second part introduces a novel application of the information processing capacity (IPC) metric to SNNs. This approach provides a comprehensive profile of the functions computed by SNNs, encompassing memory and nonlinear processing. The study methodically examines various encoding mechanisms and their impact on the IPC and shows that the metric is indicative of the performance in tasks with different demands of nonlinear processing and memory. This exploration not only extends the utility of the IPC metric to more complex neural networks but also offers a deeper insight into their computational capabilities.

The third part of the thesis tests a hypothesis about the computation of temporal difference errors in the brain, focusing on two distinct populations of cortical layer 5 neurons: the crossed corticostriatal (CCS) and corticopontine (CPn) cells. By implementing network models based on these populations and evaluating their memory capabilities through the lens of the IPC, the research supports, at least for continuous rate networks, the proposed role of these neurons in the computation of temporal difference errors. However, the spiking network models pose a greater challenge and exhibit little ability to memorize previous inputs in our experiments.

In summary, this work not only confirms and extends existing research results, but also develops new methods for analyzing SNNs. It lays a solid foundation for future studies of the brain's computational processes and enriches the field of computational neuroscience with advanced tools and methods for exploring the intricate workings of biologically inspired neural network models.

Kurzfassung

Diese Doktorarbeit befasst sich mit den rechnerischen Besonderheiten des menschlichen Gehirns und erforscht die Fähigkeiten kortikaler Mikroschaltkreismodelle, das Ausmaß ihrer Informationsverarbeitungskapazität und ihre Rolle beim Gedächtnis und beim Temporal Difference Learning mithilfe kortiko-striataler Populationen. Im Mittelpunkt dieser Forschung steht die Untersuchung von spikenden neuronalen Netzwerken (SNN). Dabei ist das Ziel, ein tieferes Verständnis der strukturellen und neuronalen Einflüsse auf die Informationsverarbeitung in diesen Netzwerken zu gewinnen und gleichzeitig einen Leitfaden für deren Analyse bereitzustellen.

Im ersten Teil wird ein in einer früheren Arbeit eingeführtes Netzwerkmodell einer kortikalen Säule reproduziert und erweitert. Diese Untersuchungen zeigen, dass die spezifische, datenbasierte Konnektivität die Rechenleistung verbessert, indem sie die Klarheit interner Repräsentationen schärft, anstatt wie zuvor beschrieben die Dauer der Informationsbeibehaltung zu verlängern.

Der zweite Teil geht über die traditionellen aufgabenbasierten Auswertungen hinaus und stellt eine neue Anwendung der Informationsverarbeitungskapazität (IPC) auf SNN vor. Dieser Ansatz liefert ein umfassendes Profil der von SNN berechneten Funktionen, das sowohl das Gedächtnis als auch die nichtlineare Verarbeitung einschließt. Die Studie untersucht methodisch verschiedene Kodierungsmechanismen und ihre Auswirkungen auf die IPC und zeigt, dass die Metrik Rückschlüsse auf die Leistung bei Aufgaben mit unterschiedlichen Anforderungen an nichtlineare Verarbeitung und Gedächtnis zulässt. Diese Untersuchung erweitert nicht nur den Nutzen der IPC-Metrik auf komplexere neuronale Netzwerke, sondern bietet auch einen tieferen Einblick in deren Rechenfähigkeiten.

Der dritte Teil der Arbeit testet eine Hypothese über die Berechnung von Temporal Difference Fehlern im Gehirn und konzentriert sich dabei auf zwei unterschiedliche Populationen von Neuronen der kortikalen Schicht 5: die gekreuzten kortikostriatalen (CCS) und kortikopontinen (CPn) Zellen. Durch die Implementierung von Netzwerkmodellen, die auf diesen Populationen basieren, und die Auswertung ihrer Gedächtnisfähigkeiten mit Hilfe der IPC unterstützt die Arbeit, zumindest für Netzwerke mit kontinuierlicher Rate, die vorgeschlagene Rolle dieser Neuronen bei der Berechnung von Temporal Difference Fehlern. Die spikenden Netzwerkmodelle stellen allerdings eine größere Herausforderung dar und weisen in unseren Experimenten generell kaum Fähigkeiten zum Speichern vorhergehender Eingaben auf.

Zusammenfassend lässt sich sagen, dass diese Arbeit nicht nur bestehende Forschungsergebnisse bestätigt und erweitert, sondern auch neue Methoden für die Analyse von SNN entwickelt. Sie legt eine solide Grundlage für künftige Untersuchungen der Rechenprozesse des Gehirns und bereichert das Feld der Computational Neuroscience mit fortschrittlichen Werkzeugen und Methoden zur Erforschung der komplizierten Funktionsweise biologisch inspirierter neuronaler Netzwerkmodelle.

Publications and contributions

The work presented in this thesis is in part based on the following publications of the author:

Characteristic columnar connectivity caters to cortical computation: Replication, simulation, and evaluation of a microcircuit model
Tobias Schulte to Brinke, Renato Duarte and Abigail Morrison
Frontiers in Integrative Neuroscience 16 (2022)

Chapter 2 and parts of Chapter 1 and Chapter 5 are based on this publication.

Contributions Under the supervision of Renato Duarte and Abigail Morrison, the author performed all parts of the above publication. All authors contributed to the design of the experiments and the writing of the manuscript.

A refined information processing capacity metric allows an in-depth analysis of memory and nonlinearity trade-offs in neurocomputational systems
Tobias Schulte to Brinke, Michael Dick, Renato Duarte and Abigail Morrison
Scientific Reports 13 (2023)

Chapter 3 and parts of Chapter 1 and Chapter 5 are based on this publication.

Contributions Abigail Morrison, Renato Duarte and the author designed the study. Under the supervision of Renato Duarte and Abigail Morrison, the author performed all simulations and analyses of the above publication, except for those concerning the FPUT system. These were carried out by Michael Dick. All authors contributed to the writing of the manuscript.

Temporal difference learning in cortico-striatal populations
Tobias Schulte to Brinke, Barna Zajzon, Renato Duarte and Abigail Morrison
In preparation

Chapter 4 and parts of Chapter 1 and Chapter 5 will constitute the basis of this publication.

Contributions All authors designed the study together. Under the supervision of Renato Duarte and Abigail Morrison, the author performed all simulations and analyses of the study. Barna Zajzon and the author co-implemented the simulation scripts. Tobias Schulte to Brinke is the sole author of the text in Chapter 4.

Contents

I Introduction **1**

1 Introduction **3**
1.1 Background and History . . . 3
1.2 Fundamental principles . . . 8
1.2.1 The human cortex . . . 8
1.2.2 Neurons and synapses . . . 8
1.2.3 Neuron modelling . . . 12
1.2.4 Dynamical systems theory . . . 13
1.2.5 Learning paradigms . . . 14
1.3 Aims and structure of the thesis . . . 17
1.3.1 Computations in cortical microcircuit models . . . 17
1.3.2 Information processing capacity . . . 18
1.3.3 Memory prerequisites for temporal difference learning in cortico-striatal populations . . . 19

II Experiments **21**

2 Information processing in cortical microcircuits **23**
2.1 Introduction . . . 23
2.2 Methods . . . 26
2.2.1 Microcircuit model . . . 26
2.2.2 Tasks . . . 37
2.2.3 Simulation and analysis framework . . . 40
2.3 Results . . . 40
2.3.1 Network activity . . . 40
2.3.2 Task performance for the circuit variants . . . 41
2.3.3 Robustness to neuron model simplifications . . . 46
2.3.4 Detailed memory tasks . . . 48
2.4 Replicability . . . 50
2.5 Conclusion . . . 52

3 Information Processing Capacity **53**
3.1 Introduction . . . 53
3.2 Methods . . . 55
3.2.1 Information processing capacity . . . 55
3.2.2 Tasks . . . 58
3.2.3 Investigated models . . . 59
3.2.4 Capacity chance level and cut-off value . . . 61
3.3 Results . . . 62
3.3.1 Discrete time system: Echo state network . . . 63
3.3.2 Simple continuous time system: Fermi-Pasta-Ulam-Tsingou model 65
3.3.3 Balanced spiking neural network model . . . 67
3.3.4 Biophysical spiking network model . . . 73
3.3.5 Comparative performance on tasks . . . 75
3.4 Conclusion . . . 76

4 Memory prerequisites for temporal difference learning in cortico-striatal populations **77**
4.1 Introduction . . . 77
4.2 Methods . . . 79
4.2.1 Baseline rate-based model . . . 79
4.2.2 Structured rate-based model . . . 81
4.2.3 Spiking baseline model . . . 83
4.2.4 Structured spiking model . . . 83
4.2.5 Network modification experiments . . . 85
4.2.6 Conversion networks . . . 85
4.3 Results . . . 86
4.3.1 How nonlinearities shape the memory in the baseline continuous rate network . . . 86
4.3.2 How weight distributions shape the memory in the baseline continuous rate network . . . 89
4.3.3 From baseline rate network to the structured rate network . . . 91
4.3.4 Structured continuous rate network . . . 96
4.3.5 Spiking neural networks . . . 98
4.3.6 Spiking networks constructed from rate networks . . . 98
4.4 Conclusion . . . 102

III Discussion **105**

5 Discussion **107**
5.1 Cortical microcircuit . . . 107

5.2 Information processing capacity 109
5.3 Temporal difference learning in cortico-striatal populations 111
5.4 Outlook and future work 113
5.5 Conclusion 117

Bibliography **119**

A Microcircuit **135**

B Information processing capacity **139**
B.1 Details on removing the nonlinear encoder effects 139

C Temporal-difference learning in cortico-striatal populations **145**

List of Figures **151**

List of Tables **159**

List of Abbreviations **161**

D Index of Abbreviations **161**

Part I

Introduction

Chapter 1

Introduction

1.1 Background and History

Cardiocentic views

Throughout history, our understanding of the function of the brain has been subject to constant change. The ancient Egyptians, for example, carelessly removed the brains of their dead, whereas they made sure that the heart remained intact in the body (Finger, 2001). Their reason for this was the fact that they considered the heart, rather than the brain, to be the seat of intelligence and emotion, and therefore assumed it would be needed in the afterlife as well. Aristotle (384-322 BCE) attached no more profound significance to the brain either. In his eyes, it was only responsible for cooling the blood and for dissipating the heat generated by the heart (Aristotle, 1911).

The brain as the center of the mind

Gradually, however, the first thinkers, some long before Aristotle, began to recognize the role of the brain in cognitive processes. Alcmaeon of Croton (5th century BCE) was one of the first to identify the brain as an organ for perception and thought, and even the physician Hippocrates of Kos (460-370 BCE) wrote in his Reflections of Epilepsy as a Sacred Disease (Cobb, 2020):

> "And men ought to know that from nothing else but (from the brain) come joys, delights, laughter and sports, and sorrows, griefs, despondency, and lamentations. And by this, in an especial manner, we acquire wisdom and knowledge, and see and hear, and know what are foul and what are fair, what are bad and what are good, what are sweet, and what unsavory; some we discriminate by habit, and some we perceive by their utility. By this we distinguish objects of relish and disrelish, according to the seasons; and the same things do not always please us. And by the same organ we become mad and delirious, and fears and terrors assail us, some by night, and some

> by day, and dreams and untimely wanderings, and cares that are not suitable, and ignorance of present circumstances, desuetude, and unskilfulness." (Hippocrates, 1868)

Furthermore, he stated in the same paper "that the brain exercises the greatest power in the man". Other important findings came from Alexandria after Aristotle's death. In this important center of the Greco-Roman world, it was allowed for a short time to perform dissections on humans, which enabled the anatomists Herophilus (335-280 BCE) and Erasistratus (304-250 BCE) to make important discoveries about the central importance of the brain and its structures (Cobb, 2020). However, even their findings had a hard time holding their own against Aristotle's representations due to his still pronounced reputation, also because people's everyday experiences tended to support a link between the heart and the emotions.

Animal spirits

About 400 years later, the Greek physician Galen (129-216 CE) carried out extensive dissections on animals alongside examinations of wounded Roman gladiators. This led him to develop the theory, based on Herophilus' findings about the brain ventricles, that movements of the body are caused by a kind of gas produced in the brain. This so-called pneuma psychicon flows through the hollow nerves, which according to Galen do not originate from the heart as Aristotle claimed, but all originate from the brain. Although Galen's theory was influential in many respects until the 17th century and was extended after the collapse of the Roman Empire by Arab scholars such as Haly Abbas (10th century) to include the non-material spiritus animalis (Bono, 1984), the encephalocentric picture did not immediately gain full acceptance. Although scholars like Ibn-Sina (980-1037, also known as Avicenna) could be convinced that the nerves originate from the brain, they remained faithful to Aristotle's opinion that the heart was the center of perception and movement (Cobb, 2020).

With *"De Humani Corporis Fabrica Libri Septem"* (On the fabric of the human body in seven books)(Vesalius, 1543), Andreas Vesalius (1514-1564) published one of the most detailed and richly illustrated works on the anatomy of the human body in 1543. He pointed out errors in Galen's work, who had transferred his findings from animal experiments too directly to humans and thus, for example, assigned a central role in the interaction of brain and pneuma psychicon to the rete mirabile, which does not exist in humans. Although Vesalius, like Galen, was an energetic advocate of the hypothesis of the brain as the center of mental faculties, he too was unable to provide an explanation for the exact functioning in the production of thoughts and movements (Catani and Sandrone, 2015).

Descartes' dualism

For the first time, such an explanation was provided by another convinced opponent of cardiocentric views. In his "Tractatus de homine" (1662)(Descartes, 1994), published only after his death, Rene Descartes (1596-1650) postulated that animals and humans functioned fundamentally like machines and that their perceptions and reflexive movements could be explained by hydraulic mechanisms through the gases or fluids flowing in the brain and nerves. In his dualistic worldview, humans differed only in the pineal gland, which served as the point of connection between the spiritual (res cogitans) and the material (res extensa), thus enabling conscious control of the body.

Rise of the scientific method

Apart from the fact that the pineal gland was soon proven in animals, a change of mind in the sciences led thinkers and scholars from the 17th century onwards to question the views of Descartes and previous representatives of the pneuma theory. This change became particularly clear in the "Discourse on the Anatomy of the Brain" by the Dane Nicolaus Steno (1638-1686) in 1665:

> The brain being indeed a machine, we must not hope to find its artifice through other ways than those which are used to find the artifice of the other machines. It thus remains to do what we would do for any other machine; I mean to dismantle it piece by piece and to consider what these can do separately and together. (Cobb, 2020; Steno, 1669)

This new approach was also evident in the experiments of one of Steno's colleagues. The Dutchman Jan Swammerdam (1637-1680) showed with experiments on frog legs that neither gas nor fluid enters contracting muscles. He proved that their volume does not increase in the process. On top of that, he found that contraction could be induced by "irritating" the attached nerve. To do this, for example, he struck them with a pair of scissors.(Cobb, 2002)

Materialism

In the course of this, a purely materialistic view emerged among others through Thomas Hobbes (1588-1679), Margaret Cavendish (1623-1670) and John Locke (1632-1704), which contradicted a spiritual component in the functioning of humans. This view, however, was opposed by thinkers such as Gottfried Wilhelm Leibnitz (1646-1716) and religious conservatives who saw free will and morality as well as the immortality of the soul, and thus Christianity in general, in danger. (Cobb, 2020)

The neuron doctrine

The invention of the light microscope in the early 17th century laid the foundation for the discovery of Pukinje cells in the cerebellum by the Czech anatomist Jan Evangelista Purkinje in 1837 (Purkinje, 1837). In the same century, the Golgi staining technique of the Italian physician Camillo Golgi made it possible to visualize finer structures of neural tissue (Golgi, 1873; López-Muñoz, Boya, and Alamo, 2006). Golgi recognized an interconnected network, or reticulum, in the stained structures and postulated that the entire nervous system was a non-separated whole. The major opponent of this reticulum theory was the Spanish pathologist Santiago Ramón y Cajal. He developed the Golgi stain further and was able to see that the dendrites and axons are separated by a gap. Thus, the brain consisted of individual cells called neurons (López-Muñoz, Boya, and Alamo, 2006; Ramón y Cajal, 1888). Although there was much to support this neuron doctrine, it was not until the later development of the electron microscope in the 1950s that it was finally confirmed (López-Muñoz, Boya, and Alamo, 2006).

The brain as a computer

In the first half of the 20th century, several new movements and insights changed the way we look at how the brain works. The British mathematician and logician Alan Turing published in 1936 in his article *"On Computable Numbers, with an Application to the Entscheidungsproblem"* (Turing, 1936) a mathematical formalization of algorithms and computability by conceiving an idealized computing machine endowed with infinite time and memory that manipulates symbols on a memory tape. He claimed that this abstract but simple machine, known today as the Turing machine, was powerful enough to perform any human-computable calculation that is based on symbolic configurations. Shortly thereafter, Warren McCulloch and Walter Pitts were the first to use Alan Turing's conception of computation to explain neural processing in the brain (McCulloch and Pitts, 1943). Following Cajal's neuron doctrine, they developed a simple neuron model whose output is 1 provided the sum of incoming excitatory signals exceeds a fixed threshold and there is no inhibitory input. Otherwise, these McCulloch-Pitts neurons output 0. From these basic neurons, McCulloch and Pitts were able to construct networks that could among other functions perform the logical operations AND, OR, and NOT. From a conceptual point of view, the Arithmetic Logic Unit (ALU) of the 1945 introduced Von Neumann architecture (von Neumann, 1993), which is implemented in a large number of computers today, can also be seen as a network of gates that perform these same logical operations.

Learning in neural networks

Based upon the research on synaptic plasticity by Donald O. Hebb (Hebb, 1949), Frank Rosenblatt further developed the previously only statically and manually connected Mc-

Culloch Pitts networks into the perceptron by extending them with a learning rule that allows the connections to adapt in a way that the outputs of the network adapt to desired output values (Rosenblatt, 1958). However, this learning rule only works for single-layer perceptrons and therefore can only be applied to solve linearly separable problems.

Artificial intelligence

Although already in the science fiction literature of the 19th and early 20th centuries, such as Samuel Butler's *"Erewhon"* (Butler, 1872), Mary Ann Evans (known as George Eliot) *"Impressions of Theophrastus Such"* (Eliot, 1879) and Karel Čapek's *"R.U. R (Rossum's Universal Robots)"* (Čapek, 1920) machines with intelligence comparable or even superior to humans were conceived (Taylor and Dorin, 2020), this concept found its way into science under the term artificial intelligence (AI) primarily with Alan Turing's work on the "imitation game" (also known as the Turing Test) as a method for recognizing intelligent algorithms (Turing, 1950) and the 1956 Dartmouth Workshop organized by Marvin Minsky, John McCarthy, Claude Shannon, and Nathan Rochester (McCarthy et al., 1955).

Computational theory of mind

Based on the work of McCulloch and Pitts and the artificial intelligence movement, the Computational Theory of Mind emerged in philosophy. In this theory, the mind is viewed as an information-processing system that gives rise to cognition and consciousness as a form of computation (Piccinini and Bahar, 2013). This view was introduced into philosophical discussion primarily by Putnam (1967) and developed and extended by him and his PhD student Jerry Fodor in the following decades (Rescorla, 2020). However, the view about the nature of computation carried out by the brain has changed over time. McCulloch and Pitts were of the opinion that the processes in the brain are based on digital processing, whereas Karl Lashley believed that they were analog in nature. More recent research tends to assume that neural computation is a separate type of computation ("neural computation is sui generis") (Piccinini and Bahar, 2013). Even though the continuous development of methods for the non-invasive analysis of the brain, such as the electroencephalography (EEG) (Berger, 1929), magnetic resonance imaging (MRI) (Lauterbur, 1973) and functional magnetic resonance imaging (fMRI) (Belliveau et al., 1991; Ogawa et al., 1990), has led to new insights and data about its processes, and the steadily increasing computing power allows for ever larger model simulations (Billeh et al., 2020; Häusler and Maass, 2007; Markram et al., 2015; Potjans and Diesmann, 2014) that enable the testing of hypotheses about the functioning of our cognitive center, it is still unclear how neural computations work in detail and how exactly structural and biological properties shape these computations.

1.2 Fundamental principles

1.2.1 The human cortex

The human brain is a marvel of nature that fascinates with its complex structure and dynamic processes. Specifically, the cortex, the outermost shell of the mammalian brain and thus also of the human brain, plays a key role in numerous cognitive functions. For instance, it is substantially involved in the perception of sensory input, the processing of language, the initiation of motor functions, and the planning of complex actions. This outer mantle of the brain consists of a few millimeters thin, but mostly highly folded and thus surface-rich layer of so-called gray matter, which consists mainly of densely packed nerve cell bodies and their local connections to each other (dark outer areas in Figure 1.1 A and B; image of cell bodies in Figure 1.1 C). Underneath this gray matter is a thicker layer of long-range nerve fiber connections, which have a white coloration due to their wrapping with layers of lipid-rich myelin, which serve as insulation (lighter inner areas in Figure 1.1 A and B). For this reason, this layer is called white matter. Each of the approximately 16 billion neurons of the human cortex processes incoming signals from a large number of upstream neurons, the average number of which is on the order of 10^4. The structure of the cortex is characterized by a complex architecture in which different areas appear to be particularly (but not exclusively) specialized for specific parts of information processing. For example, the visual cortex is primarily involved in processing visual impressions, the motor cortex is involved in motor control, and the prefrontal cortex is involved in abstract thought processes related to internal goals (Miller, Freedman, and Wallis, 2002). Despite this division into differently functional areas, the basic microstructure of the cortex is thought to be approximately identical in each of its areas. This local microcircuit architecture spans an area of about 1 mm^2, contains about 80,000 neurons and 0.3 billion connections, and is divided into up to 6 different horizontal layers (Potjans and Diesmann, 2014). These layers differ primarily in the type and distribution of neurons located within them and their interconnections with other cortical and subcortical structures. This division into fundamentally invariant cortical columns, which serve as a basic computational unit in the brain, enables blind people, for example, to use their cortex areas previously used for visual processing for the processing of other modalities such as hearing and touch after their loss of sight (Castaldi, Lunghi, and Morrone, 2020).

1.2.2 Neurons and synapses

On an even smaller level, the human nervous system comprises networks of separate but intercommunicating neurons. As illustrated in Figure 1.2 A, these neurons consist of a cell body (*soma*), *dendrites* and an *axon*. The soma contains the nucleus and is the metabolic center of the cell, while the dendrites and the axon are responsible for receiving and transmitting signals, respectively. The cell membrane of a neuron

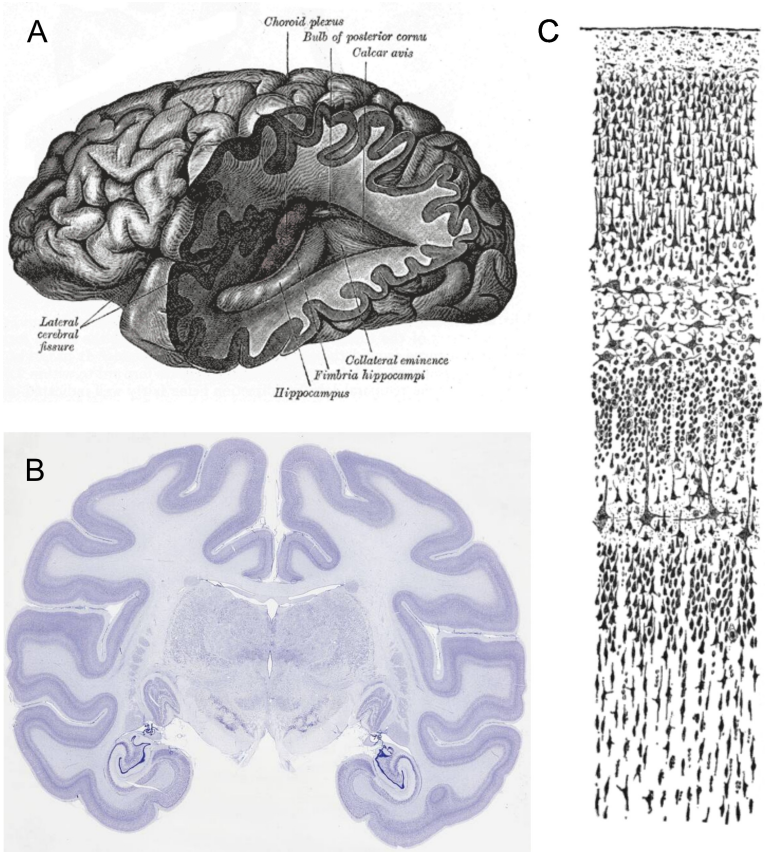


Figure 1.1: **Cortical architecture. A:** Drawing of a human brain from Henry Gray's *Anatomy of the Human Body* (Gray, 2000). **B:** Slice of a macaque monkey brain with Nissl stained cell bodies (retrieved from BrainMaps Atlas (Mikula et al., 2007)). **C:** Historical drawing of the Nissl stained visual cortex of a human adult seen through a microscope by Santiago Ramón y Cajal (Ramón y Cajal, 1899).

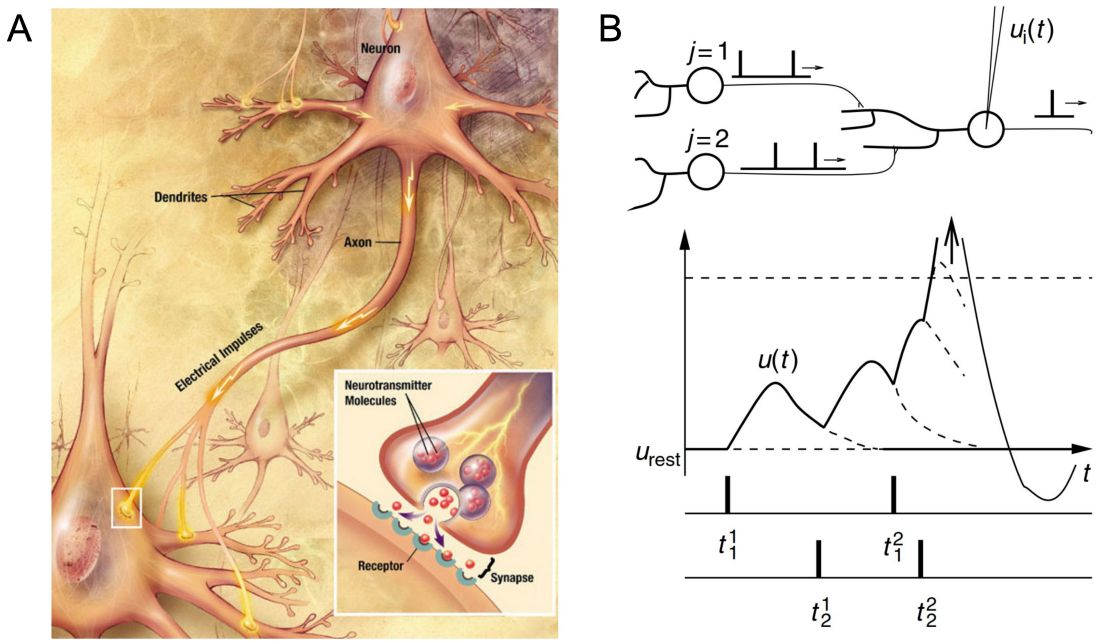


Figure 1.2: **Neuronal action potential generation and transmission. A:** Schematic illustration of a neuron with its different parts and synapses connecting it with a second neuron (Source: Wikimedia Commons, Public Domain, `https://commons.wikimedia.org/wiki/File:Chemical_synapse_schema_cropped.jpg`). **B:** Example illustrating the input integration of a neuron. The incoming action potentials of two presynaptic neurons successively depolarize the postsynaptic neuron until the membrane potential exceeds a threshold value and a new action potential is emitted. The membrane potential then normalizes to the resting potential after a short time. Image is adapted from Gerstner et al. (2014).

serves as an insulator that maintains an electrical potential between the intracellular and extracellular space, called the *membrane potential*. Typically, the inside of the neuron is negatively charged compared to the outside, resulting in a negative *resting potential*. Communication between neurons occurs through the transmission of electrical impulses. These *action potentials* (often referred to as *spikes*) are mainly generated by proteins in the cell membrane that allow positively charged sodium ions (Na^+) to be transported across the cell membrane into the neuron. As a result of the influx of ions allowed by these ion channels, the cell becomes depolarized and the negative resting membrane potential becomes less negative or even positive for a brief moment. However, after this rapid influx of sodium ions, other channels are opened, allowing positively charged potassium ions (K^+) to flow out of the neuron. This positive outflow causes the cell to repolarize again, which restores the resting membrane potential. This precise balance and timing of the ion exchange is essential for the correct generation and propagation of the action potential. A schematic example of the input generation and resulting changes of the membrane potential inside the neuron is visualized in Figure 1.2 B. In addition to the two mentioned channels, there are a myriad of other ion channels that affect the membrane potential, but they will not be discussed further here.

The resulting spike is propagated along the axon and, upon reaching its end, triggers the opening of calcium channels. The resulting influx of calcium ions (Ca^{2+}) triggers the release of chemical messenger molecules called *neurotransmitters*. Those neurotransmitters thus diffusing into the *synaptic cleft* bind to receptor proteins in the cell membrane of the postsynaptic neuron (see inset of Figure 1.2 A), triggering again an opening of ion channels, whose ion flux in turn influences the membrane potential of this second neuron.

Synaptic efficacy is by no means static but can change based on recent activity. Repeatedly arriving action potentials can lead to a depletion of the available vesicles containing the neurotransmitters. This results in a reduction in neurotransmitter release with subsequential stimuli. In addition to this *short-term depression*, there can also be a strengthening of synaptic efficacy, i.e. *short-term facilitation*, which is mainly caused by residual calcium remaining from the previous opening of the corresponding ion channels.

This influence on the postsynaptic neuron can not only be *excitatory*, i.e. bringing the membrane potential closer to the firing threshold value but, depending on the neurotransmitter and receptor type, can also lead to an *inhibition* of the postsynaptic cell. According to *Dale's principle* (Eccles, Fatt, and Koketsu, 1954), however, a single neuron can release only the same set of neurotransmitters at all of its synapses, allowing the classification into excitatory and inhibitory nerve cells. The spike trains at all incoming synapses are integrated by the neuron and, if this depolarizes the cell sufficiently, again lead to the generation of a spike that is sent to further neurons.

1.2.3 Neuron modelling

In computational neuroscience, models represent a valuable tool for the analysis and prediction of neuronal dynamics. These mathematical formulations derived from biological observations can be extensive models that reproduce a highly detailed morphology and physiology. However, the high complexity of these models imposes enormous computational power requirements. This can be reduced by dividing the spatial characteristics of the neuron into different sections or compartments so that we are dealing with so-called *compartmental models*.

Leaky integrate-and-fire model

To further minimize the computational requirements and increase the possibilities of theoretical analyzability, in *point models*, we omit any spatial structure of the neuron and reduce its complexity to a single dimensionless point. In essence, we focus on the temporal dynamics of the membrane potential in these models. The leaky integrate-and-fire (LIF) neuron (Lapicque, 1907; Stein, 1967) represents a fundamental type of these point models. It embodies the fundamental concept of the input-accumulating neuron, which emits an action potential when its potential threshold is reached and thereupon directly resets its membrane potential. It captures the essence of neuronal activity at minimal computational cost, making it a popular choice for large-scale neural network simulations where efficiency and scalability are critical constraints.

Adaptive exponential integrate-and-fire model

A more advanced variant of the LIF model, the adaptive exponential integrate-and-fire (AdEx) neuron (Brette and Gerstner, 2005), adds an exponential term to the dynamics equation of the LIF neuron (Fourcaud-Trocmé et al., 2003), which both better reflects the rapid voltage changes that real neurons exhibit in the vicinity of their threshold and replaces the previously hard threshold with a more gradual and soft spike initiation process. Furthermore, the AdEx neuron model includes an additional dynamic adaptation current that can adjust the responsiveness to incoming signals over time, allowing the model to adapt its behavior based on its recent activity.

Hodgkin-Huxley model

A further increase in the level of detail is provided by the Hodgkin-Huxley point neuron model developed by Alan Hodgkin and Andrew Huxley in the 1950s (Hodgkin and Huxley, 1952). This model is based on the two scientists' studies of the giant axon of the squid, which revealed the essential role of the opening and closing of sodium and potassium channels in the generation and propagation of electrical impulses. The simulation of the nonlinear differential equations defining the complex ion channel dynamics requires

a much higher computational effort compared to the other point models presented. In return, however, it can reproduce the biophysical processes underlying individual neuron behavior with unrivaled fidelity.

1.2.4 Dynamical systems theory

Dynamical systems are systems whose state evolves over time, following specific laws of motion. They are often defined by differential equations that characterize the motion of their elements in a finite state-space (Birkhoff, 1927). The theory of such systems originates in the study of the motions of celestial bodies (Poincaré, 1892), but today finds applications in many research fields. From the double pendulum (Levien and Tan, 1993), to the modeling of population dynamics (May, 1976) and the interacting reactions of chemicals (Dale and Husbands, 2010), to fields such as the study of collective decision-making processes (Yang et al., 2021), infant development (Smith and Thelen, 2003) or brain development (Lefèvre and Mangin, 2010), the theory of dynamical systems is central to many disciplines and offers a mathematical formalism to characterize complex adaptive systems.

The theory also forms the basis of a field of neuroscience (Izhikevich, 2007) and facilitates a better understanding of the activity inside the brain. The dynamics inside the central nervous system give rise to complex behavior, enabling the living being hosting the brain to solve challenging problems, and are therefore of great interest to this research field.

Moreover, there is increasing interest in physical systems that can serve as substrates for brain-like computations. There are numerous dynamical systems that perform non-trivial computations on input signals using the interactions between their parts and the dynamics that arise within them (Grollier et al., 2020; Larger et al., 2012; Lugnan et al., 2020; Sharp et al., 2012).

Balance between excitation and inhibition

As mentioned above, computational neuroscience also deals with biologically inspired dynamical systems. In this context, researchers primarily rely on networks of recurrently connected populations of excitatory and inhibitory neuron models. When characterizing network activity, we pay particular attention to the regularity and synchrony of the spike trains. Of high interest are systems that operate in a state of asynchronous-irregular (AI) activity, typically featuring low firing rates. This state occurs mainly in sufficiently large networks of sparsely interconnected neurons, in which the effect of the excitatory units is approximately balanced by the inhibitory neurons (Brunel, 2000; Van Vreeswijk and Sompolinsky, 1996). On average, the membrane potential thereby shifts to a value just below the threshold and action potentials are evoked in the network mainly by spontaneous fluctuations and external inputs. This tight balance between excitation

and inhibition, as it occurs in these networks, plays a crucial role in the execution of complex neural computations (Denève and Machens, 2016).

1.2.5 Learning paradigms

Sometimes it is sufficient to analyze systems with static structures and look at their activity and dynamics, but often in areas like machine learning and computational neuroscience, we want to study systems that can adapt to their environment or specific tasks. This results in different distinguishable learning paradigms.

Unsupervised learning

If the learning process takes place completely without an external training signal or labeled data, we speak of *unsupervised learning*. This includes, for example, the clustering of data or the reduction of their dimensionality.

Supervised learning

In *supervised learning*, for each point of the input, there is a corresponding correct and explicitly desired output signal that the system to be trained should reproduce. To adapt the system according to these desired outputs, gradient-based methods such as Backpropagation of Errors (Rumelhart, Hinton, and Williams, 1986) can be used to adapt a large number of parameters, such as the weights of a neural network. However, such training of the complete system or major parts of it can lead to substantial computational costs.

Reservoir computing Another form of supervised learning is the concept of *reservoir computing*, which usually requires much less computation. Although exceptions exist (Pyle and Rosenbaum, 2019; Sussillo and Abbott, 2009), the computing system itself is usually not adapted. It typically serves only as a reservoir whose temporal dynamics process the input signals and project them into a higher-dimensional state space from which the outputs required by the task can be extracted using a linear readout mechanism, i.e. a linear combination of the state variables (see Figure 1.3). This readout is the only part of the system that is trained. The reservoirs must exhibit the so-called echo state property, meaning that the effects of previous inputs have to vanish asymptotically from the state representation of the system (Jaeger, 2001b; Yildiz, Jaeger, and Kiebel, 2012).

Although the reservoir computing principle was developed in the early 2000s with the development of the echo state network (ESN) (Jaeger, 2001b) and the liquid state machine (LSM) (Maass, Natschläger, and Markram, 2002) for use with spiking (LSM) and non-spiking (ESN) recurrent neural networks, any other input-driven dynamical system can equivalently be utilized as a reservoir. This opens the possibility to use

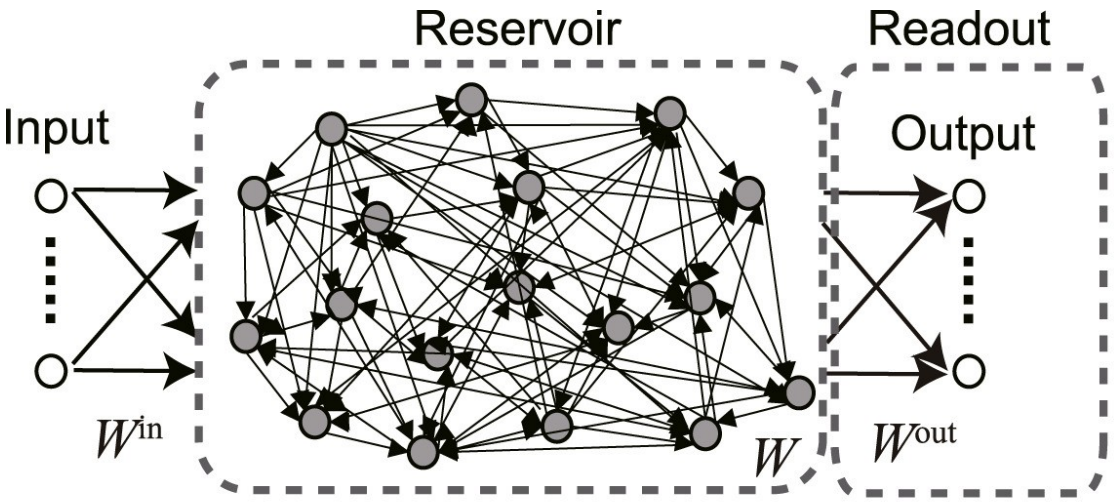


Figure 1.3: **Schematic illustration of the reservoir computing paradigm.** The input is linearly transformed by the weight matrix W^{in} and the reservoir processes these inputs so that the desired outputs can be read out using the linear transformation matrix W^{out}. Adapted from Tanaka et al. (2019).

physical in addition to simulated systems for the computations. Besides more exotic reservoirs such as water buckets (Fernando and Sojakka, 2003) and octopus-arm inspired softbodies (Nakajima et al., 2015), this also enables the use of particularly fast and efficient systems that use, among other phenomena, light (Duport et al., 2012; Paquot et al., 2012; Paquot et al., 2010; Vandoorne et al., 2014) or quantum effects (Chen, Nurdin, and Yamamoto, 2020; Ghosh et al., 2019; Nakajima et al., 2019) for reservoir computations, or rely on mechanical anharmonic nano-oscillators (Coulombe, York, and Sylvestre, 2017) or memristive devices (Bürger et al., 2015; Zhong et al., 2021).

Reservoir computing in computational neuroscience In computational neuroscience, the reservoir computing method is mainly used to study neural network models. In this case, the readout mechanism serves as a translation tool that makes the functions computed naturally by the network usable or at least recognizable to humans. Since the learning mechanism only acts outside of the reservoir that is actually being studied, it is possible to integrate any biologically inspired features such as different plasticity mechanisms or network structures into the neural system without having to adapt the training mechanism.

Reinforcement Learning

The third learning paradigm, *reinforcement learning (RL)*, can be seen as a middle ground between the supervised and unsupervised methods. In reinforcement learning, an agent is trained to maximize a cumulative *reward* by interacting with an *environment*. This reward signal does not specify the absolutely correct output after each step as in supervised learning, but only provides an often sparse and delayed positive (reward) or negative (punishment) evaluation of the current state. Based on this sparse signal, the system must independently find the best actions to maximize this reward by optimizing its policy using a balance between the exploration of unknown actions and the exploitation of steps previously proven useful (*exploration-exploitation dilemma*). This trial-and-error learning goes back, on the one hand, to behaviorism's psychological theories of learning in humans and other animals of classical (Pavlov, 1927) and operant (Skinner, 1938; Thorndike, 1911) conditioning. On the other hand, many basics of reinforcement learning come from the mathematical-technical direction of optimal control theory and dynamic programming (Bellman, 1957).

Temporal difference learning A widely used method of reinforcement learning is temporal difference (TD) learning (Sutton, 1988; Sutton and Barto, 1981). Temporal difference learning combines approaches from *Monte Carlo learning* (Metropolis and Ulam, 1949) and *dynamic programming* (Bellman, 1957). While Monte Carlo learning relies on actual experienced trajectories to estimate the value of a state or action, dynamic programming uses known models of the environment to make predictions. TD Learning combines these approaches and allows estimates based on partial trajectories to be updated without requiring a full model of the environment or the full trajectory. In this approach, the main task of the algorithm is to learn to predict future rewards. Each state s is given a *state value* V that represents the future rewards r expected from that state:

$$V(s_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right]; \gamma \in [0,1] \tag{1.1}$$

where the discount factor γ can be used to implement a different weighting of rewards received over time. For a value of $\gamma = 0$, only the immediate reward is maximized, and the closer γ is to 1, the more the agent tries to maximize the cumulative reward across the entire period. Since the actual state value can only be correctly determined over time by rewards that occur, the agent's task is to minimize a *reward prediction error (RPE)* δ based on the difference between the expected and actual value of the state:

$$\delta(t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{1.2}$$

Temporal difference learning in the brain Such a temporal difference error in the mammalian brain is thought to be signaled by the activity of dopaminergic neurons in

the midbrain (Glimcher, 2011; Montague, Dayan, and Sejnowski, 1996; Schultz, Dayan, and Montague, 1997). Consistent with this theory, activity in the *substantia nigra pars compacta (SNc)* and the *ventral tegmental area (VTA)* has been demonstrated in monkeys (Bayer and Glimcher, 2005; Schultz, Dayan, and Montague, 1997) and also in humans (McClure, Berns, and Montague, 2003; O'Doherty et al., 2003; O'Doherty et al., 2004). Thus, this theory and practical research thereon provide a direct bridge between computational approaches to reinforcement learning and actual neural mechanisms in the brain.

1.3 Aims and structure of the thesis

The scientific community has long been intrigued by the task of comprehending the computational capabilities of dynamical systems, particularly those that aim to simulate and reproduce the intricate processing mechanisms of the human brain. The main objectives of this thesis are to gain a comprehensive understanding of the information processing happening in such systems, with a particular focus on spiking neural networks (SNNs), and to provide other researchers with a guideline on how to analyze such models in the most insightful way possible. For this purpose, we seek to evaluate to what extent structural and neuronal properties derived from biological data shape information processing in neural networks and test computational hypotheses derived from such biological properties.

1.3.1 Computations in cortical microcircuit models

In Chapter 2 *Information processing in cortical microcircuits*, we analyze the influence of the connectivity structure on information processing in a cortical microcircuit model. This first data-based network model we examine was published in the work of Häusler and Maass (2007). In this work, the authors created a microcircuit model using intracellular recordings and compared it to structurally modified but statistically identically connected control models. As Karl Popper already stated in his 1934 book *"The Logic of Scientific Discovery"* that "non-reproducible single occurences are of no significance to science" (Popper, 2005) the scientific progress can only be based on mutual validation and the peer-review process can only be considered the first necessary step on the path of research verification. Therefore we will first reproduce the results from the original paper. Especially in the domain of computational neuroscience, small details and parameter deviations can strongly influence experimental results (Pauli et al., 2018). Therefore, it is not sufficient to run the original code, but only a qualitative reproduction using a reimplementation and additional testing of the robustness of the results ensure that the findings can be trusted. For these reasons, a further part of Chapter 2 deals with subjecting the model we reimplemented to further experiments. Among other aspects, we check whether the results, which mainly emphasize the importance of the

laminar connectivity structure, hold up in the presence of changes in the neuron model and the omission of the intrinsic noise mechanism. In addition, we subject the network models to further tests that provide a more detailed picture of how information is maintained in the systems and which structural properties are of particular importance for this. Furthermore, we verify whether the obtained findings also apply to network sizes that more closely resemble the dimension of the biological equivalent of the network.

1.3.2 Information processing capacity

The evaluation of computational systems commonly focuses on their performance in standard tasks and so we also examined the network models in Chapter 2 using this criterion. Nevertheless, when considering different dynamical systems, specifically spiking neural networks, evaluating task performance alone may provide some insight but lacks the necessary depth to fully comprehend the precise operations performed by these systems. Therefore, in Chapter 3 *Information processing capacity*, we focus on the investigation of a more expressive method for the assessment of the microcircuit model from Chapter 2 and neural networks in general. The concept of information processing capacity (IPC), initially proposed by Dambre et al. (2012), is a measure used to evaluate dynamical systems based on the methodical construction of orthogonal polynomial functions that the system is asked to reconstruct based on random input.

The result is a well-interpretable profile of functions that are computed by the system. However, this approach has predominantly been employed in the context of less complex and primarily discrete-time systems, such as echo state networks and it is not clear how it can be applied to SNNs. In addition to the requirement of converting a discrete signal into continuous time, the optimal method for encoding the signal remains uncertain. There are several criteria to consider. In addition to the objective of optimizing overall capacity, there are other noteworthy considerations such as maximizing memory, attaining optimal nonlinear processing, and ensuring biological plausibility. As a first step, therefore, Chapter 3 will examine the necessary adjustments required for the information processing capacity metric to enable the most insightful analysis of SNNs.

The resulting capacity profile not only provides the total capacity value but also includes information on the maximum delay and maximum degree of the calculated functions. These properties are assumed to be easily interpretable. However, it is unclear to what extent these properties are also associated with the broader understanding of memory and nonlinearity, and how they inform about the system's ability to perform specific tasks. Hence, an additional section inside Chapter 3 addresses this matter by examining the correlation between various aspects of the capacity profile and the ability to successfully complete tasks that explicitly assess the memory and nonlinearity capabilities of the systems.

As previously mentioned, different signal encoding alternatives are also tested. Due to these different encoding methods, the encoder can have a great influence on the calculations in the system. Consequently, the encoder is an integral component of the overall system whose information processing capacity is being measured. Often, however, one is only interested in the computational performance of the main system, excluding the encoder. Therefore, a further part of Chapter 3 searches for ways of subtracting the effects of the encoder from the information processing capacity results to allow for the independent examination of the computational capabilities of the primary system.

With all these investigations, in addition to gaining direct insights into information processing in the studied dynamical systems, our main goal in Chapter 3 is to provide the computational neuroscience community with a method to gain profound insights into the computations of spiking neural networks using the information processing capacity metric.

1.3.3 Memory prerequisites for temporal difference learning in cortico-striatal populations

In Chapter 4 *Memory prerequisites for temporal difference learning in cortico-striatal populations*, we examine a hypothesis put forward by Morita et al. (2012) about the processes involved in computing a reward prediction error in the brain. Specifically, the hypothesis states that two distinct populations of cortical layer 5 neurons, the crossed corticostriatal (CCS) and corticopontine (CPn) cells, respectively encode the current and previous states and make them available to the direct and indirect pathway of the basal ganglia for further processing and computation of a reward prediction error. Morita et al. (2012) base their hypothesis primarily on the experimentally observed structural and neural properties of these two populations. However, this is initially only a theory. To test this theory, we create a network model based on the data known from the literature (Morishima et al., 2011; Morishima and Kawaguchi, 2006; Morishima et al., 2017) and use the information processing capacity prepared in Chapter 3 to examine how long the information about the input state is maintained in the CCS and CPn cells. This allows us to determine if a reward prediction error can be calculated based on the activity of the two populations. Besides the question of whether the tested model can be the basis for temporal difference learning, we also evaluate how strong the influence of the different mainly structural, but also neuronal features on the memory of the two populations is. For this purpose, similar to the approach taken in Chapter 2, we create different control models, each of which lacks a certain feature, and compare their ability to maintain information with that of the full, data-based network.

Part II

Experiments

Chapter 2

Information processing in cortical microcircuits

The neocortex, and with it the mammalian brain, achieves a level of computational efficiency like no other existing computational engine. A deeper understanding of its building blocks (cortical microcircuits), and their underlying computational principles is thus of paramount interest. To this end, we need reproducible computational models that can be analyzed, modified, extended and quantitatively compared. In this chapter, we further that aim by providing a replication of the seminal cortical column model proposed by Häusler and Maass (2007).

2.1 Introduction

Neurons of the neocortex are arranged in layers, forming connectivity structures through their synapses that share many properties across various brain areas. This suggests that diverse cortical areas are likely based on a common microcircuit template (see e.g., DeFelipe, 2012; Harris and Shepherd, 2015; Horton and Adams, 2005; Mountcastle, 1997). These broad commonalities suggest a functional purpose behind this structure that gives networks an information processing advantage over randomly connected circuits.

To investigate this hypothesis, Stefan Häusler and Wolfgang Maass developed a data-based microcircuit model and tested its computational properties in comparison with networks with equivalent dynamics but alternative connectivity structures in their seminal paper *A Statistical Analysis of Information-Processing Properties of Lamina-Specific Cortical Microcircuit Models* (Häusler and Maass, 2007).

After this paper was published as one of the first studies on data-based cortical column models, it was cited hundreds of times and influenced the computational neuroscience community's view on the purpose and benefits of a laminar cortical network structure. Since then, the model has been used by Wolfgang Maass's team to analyze, for example, the distributions of network motifs in its connectivity structure (Häusler, Schuch, and Maass, 2009) and to show how a version of the network with stochastic neurons can exploit noise for computation (Habenschuss, Jonke, and Maass, 20013; Maass, 2014). Rasch et al. (2011) use the model as the basis for a larger network that also includes a

model of the retina and lateral geniculate nucleus (LGN) of the thalamus to analyze its responses to natural stimuli and compare them with in vivo activity. Since the publication of Häusler and Maass (2007), modeling of cortical columns has partially evolved toward large-scale networks of point neurons whose focus is to accurately reproduce the statistical properties of spike activity from in vivo data (e.g. Potjans and Diesmann, 2014). The other recent direction of modeling cortical columns focuses on large networks of biophysically detailed neural compartment models, such as Markram et al. (2015) and Billeh et al. (2020). Nevertheless, smaller network models continue to be relevant because simulating these large-scale models requires huge amounts of computing resources that are beyond the scope of many computational laboratories. Therefore, as a highly influential study uncovering the relationships between structure, dynamics and function, it would be of great benefit to have the computational model introduced by Häusler and Maass (2007) available for further study and quantitative comparison with other models.

Unfortunately, as the model was originally implemented in MATLAB (unknown version, but no later than R2006b) and the C++ simulation plugin *csim* that is no longer maintained, the code can no longer be executed. In this chapter, we present a replication of the original study, which serves the twin purpose of testing the original findings and providing an executable version of the model to the computational neuroscience community. Specifically, we re-implement their model using the open source software *NEST* (Hahne et al., 2021) to simulate the networks, *NESTML* (Babu et al., 2021) to define the neuron model and Python for data analysis, thus ensuring a reusable and maintainable code base.

Here, we use the term *replication* in the R^5 sense described by Benureau and Rougier (2018), i.e. striving to obtain the same results using an independent code base, whereas a *reproduction* (R^3) of the model would have been achieved if we had obtained the results of the original study using the original code. Note that others have argued that these terms should be used the other way around: see Plesser (2018) for an overview and analysis.

Following the structure of the original work, we construct a cortical column model based on data from rat and cat cortical areas published by Thomson et al. (2002). The network consists of spiking Hodgkin-Huxley neurons with an intrinsic conductance-based noise mechanism that represents the incoming currents generated by stochastically releasing synapses and is connected by synapses with short-term plasticity. Using this network model, we investigate the impact of the data-based laminar structure on the computational performance of the system. Besides the data-based model, we implement additional control models that share the global statistics of the microcircuit whilst removing specific network properties. This allows analysis of how different network properties affect the networks' computational performance on various tasks based on input signals that are encoded as precise spike patterns or spike trains with changing firing rates.

These tasks are designed in such a way that they allow us to draw conclusions about

computational abilities of the models under investigation by testing the networks not only on their simple classification capabilities, but also on memory and their nonlinear processing power. Following the reservoir computing paradigm (see paragraph 1.2.5), the synaptic efficacies of the recurrent connections within the network are not trained to improve performance; only the projections from the network to separate readout neurons are learned.

We successfully reproduced the main data-based model and all six control circuit variants. The results on the computational tasks confirm the findings of the original study, most notably that the data-based circuit has superior computational performance to circuits without laminar structure.

Going beyond the experiments of the original study, and demonstrating the value of having executable versions of important models, we further examine the generalizability of the results with respect to the neuron model. Assuming that the laminar structure is the most important component of the model, we hypothesize that the central findings are not dependent on the specific choice of the somewhat complex Hodgkin-Huxley neurons used in the original study. To investigate this hypothesis, we simplify the neuron model by reducing its complexity to basic integrate-and-fire dynamics and show that this simplification not only maintains the superior performance of the data-based circuit but even increases its absolute performance on almost all tasks. The same is true for the removal of the noise mechanism from the Hodgkin-Huxley model. Although noise was added mainly to increase biological plausibility rather than to improve performance, it is not necessarily the case that noise degrades the performance of a neural system, since, for example, effects such as stochastic resonance can improve the detection of weak signals (McDonnell and Ward, 2011; Wiesenfeld and Moss, 1995).

Finally, we extend the original computational tasks to include a more detailed examination of the memory capabilities of the systems under consideration, reflecting the fact that the ability to recall information over time forms the basis for a variety of cognitive processes. Our results reveal a stereotypical memory profile for all tested circuits and demonstrate that the characteristic temporal structure of the stimulus has differential effects on the task performance of the networks receiving it.

Apart from providing a reproducible and re-usable implementation of the cortical microcircuit model in (Häusler and Maass, 2007), our successful replication reduces the likelihood that the original findings were influenced by implementation errors (Benureau and Rougier, 2018; Pauli et al., 2018). Our findings thus lend further support to the hypothesis that the highly nonrandom connectivity structure of cortical columns serves important computational purposes, with the degree distributions, i.e. the distributions of the number of incoming and outgoing connections per neuron, playing the most prominent role. Going beyond the original findings, we further demonstrate that the computational benefits of the laminar structure are not dependent on the complexity of the neuron model. Finally, we discover that the laminar structure does not confer memory benefits in the model - the circuits with laminar structure do not retain stimulus in-

formation for longer than networks with other connectivity assumptions - and conclude that the superior computational performance is achieved primarily by generating more distinct stimulus representations.

2.2 Methods

2.2.1 Microcircuit model

In the following sections we provide details of our implementation of the microcircuit model that is publicly available at Zenodo (Schulte to Brinke, Duarte, and Morrison, 2022) and compare it to the model described in (Häusler and Maass, 2007), whose implementation is available at ModelDB (McDougal et al., 2017; accession number 82385, https://modeldb.science/82385).

Neuron model

The networks consist of single-compartment Hodgkin-Huxley type neurons with three different active currents, as described by Destexhe and Paré (1999), and an intrinsic conductance noise mechanism introduced by Destexhe et al. (2001):

$$C_{\mathrm{m}}\frac{dV_{\mathrm{m}}}{dt} = -g_{\mathrm{L}}(V_{\mathrm{m}} - E_{\mathrm{L}}) - I_{\mathrm{Na}} - I_{\mathrm{K}} - I_{\mathrm{M}} - I_{\mathrm{noise}} \tag{2.1}$$

where V_{m} is the membrane potential, C_{m} is the membrane capacitance, g_{L} is the leak conductance and E_{L} is the leak reversal potential. I_{Na} is a voltage-dependent Na^+ current with the following dynamics:

$$I_{\mathrm{Na}} = g_{\mathrm{Na}} m^3 h\ (V_{\mathrm{m}} - E_{\mathrm{Na}}) \tag{2.2}$$

$$\frac{dm}{dt} = \alpha_m(V_{\mathrm{m}})(1-m) - \beta_m(V_{\mathrm{m}})m \tag{2.3}$$

$$\frac{dh}{dt} = \alpha_h(V_{\mathrm{m}})(1-h) - \beta_h(V_{\mathrm{m}})h \tag{2.4}$$

$$\alpha_m = \frac{-0.32(V_{\mathrm{m}} - V_{\mathrm{T}} - 13)}{\exp[-(V_{\mathrm{m}} - V_{\mathrm{T}} - 13)/4] - 1} \tag{2.5}$$

$$\beta_m = \frac{0.28(V_{\mathrm{m}} - V_{\mathrm{T}} - 40)}{\exp[(V_{\mathrm{m}} - V_{\mathrm{T}} - 40)/5] - 1} \tag{2.6}$$

$$\alpha_h = 0.128\ \exp[-(V_{\mathrm{m}} - V_{\mathrm{T}} - V_{\mathrm{S}} - 17)/18] \tag{2.7}$$

$$\beta_h = \frac{4}{1 + \exp[-(V_{\mathrm{m}} - V_{\mathrm{T}} - V_{\mathrm{S}} - 40)/5]} \tag{2.8}$$

where g_{Na} is the sodium peak conductance, E_{Na} is the sodium reversal potential, V_{S} is a voltage that shifts the inactivation towards hyperpolarized values and V_{T} is a voltage

offset that controls dynamics and adjusts the membrane threshold. Note, the model does not incorporate an explicit threshold; the membrane potential threshold V_{thresh} in Table 2.1 is just the potential at which the peak in the membrane potential is recognized as a spike by the simulator. This is also the reason why a refractory period t_{ref} is needed to avoid the emission of multiple spikes during a peak in the membrane potential. I_{K} is a delayed-rectifier K^+ current:

$$I_{\text{K}} = g_{\text{K}} n^4 (V_{\text{m}} - E_{\text{K}}) \tag{2.9}$$

$$\frac{dn}{dt} = \alpha_n(V_{\text{m}})(1-n) - \beta_n(V_{\text{m}})n \tag{2.10}$$

$$\alpha_n = \frac{-0.032(V_{\text{m}} - V_{\text{T}} - 15)}{\exp[-(V_{\text{m}} - V_{\text{T}} - 15)/5] - 1} \tag{2.11}$$

$$\beta_n = 0.5 \exp[-(V_{\text{m}} - V_{\text{T}} - 10)/40] \tag{2.12}$$

Here, g_{K} is the potassium peak conductance and E_{K} is the potassium reversal potential. The third current is a non-inactivating K^+ current responsible for spike frequency adaptation, which is only activated for excitatory neurons and was first described by Mainen, Huguenard, and Sejnowski (1995):

$$I_{\text{M}} = g_{\text{M}} p (V_{\text{m}} - E_{\text{M}}) \tag{2.13}$$

$$\frac{dp}{dt} = \alpha_p(V_{\text{m}})(1-p) - \beta_p(V_{\text{m}})p \tag{2.14}$$

$$\alpha_p = \frac{r \cdot (V_{\text{m}} + 30)}{1 - \exp[-(V_{\text{m}} + 30)/9]} \tag{2.15}$$

$$\beta_p = \frac{-r \cdot (V_{\text{m}} + 30)}{1 - \exp[-(V_{\text{m}} + 30)/9]} \tag{2.16}$$

where g_{M} is the peak conductance for this additional potassium channel. The factor r is set to 0.0001 by Destexhe et al. (2001), but Häusler and Maass (2007) use a value of 0.001 in their code, and we followed the latter in our implementation. A complete specification of all neuron parameters can be found in Table 2.1.

In addition to these ion channel dynamics, Häusler and Maass (2007) used noisy background currents whose conductances are modeled by an Ornstein-Uhlenbeck process. This stochastic background activity was introduced by Destexhe et al. (2001) to represent the spontaneous activations of incoming synapses as follows:

$$I_{\text{noise}} = g_{\text{ne}}(t)(V_{\text{m}} - E_{\text{ex}}) + g_{\text{ni}}(t)(V_{\text{m}} - E_{\text{in}}) \tag{2.17}$$

where g_{ne} is the time dependent excitatory conductance, E_{ex} is the excitatory synaptic reversal potential and g_{ni} and E_{in} are their inhibitory counterparts.

The conductances are calculated by the following update rules:

$$g_{\mathrm{ne}}(t+h) = g_{\mathrm{e0}} + [g_{\mathrm{ne}}(t) - g_{\mathrm{e0}}\ \exp(-h/\tau_{\mathrm{ne}}) + A_{\mathrm{e}} N_1(0,1)] \tag{2.18}$$

$$g_{\mathrm{ni}}(t+h) = g_{\mathrm{i0}} + [g_{\mathrm{ni}}(t) - g_{\mathrm{i0}}\ \exp(-h/\tau_{\mathrm{ni}}) + A_{\mathrm{i}} N_2(0,1)] \tag{2.19}$$

where g_{e0} and g_{i0} are average conductances, τ_{ne} and τ_{ni} are time constants, h is the integration step, $N_1(0,1)$ and $N_2(0,1)$ are random numbers drawn from a normal distribution with mean 0 and standard deviation 1, and A_{e} and A_{i} are amplitude coefficients given by

$$A_{\mathrm{e}} = \sqrt{\frac{D_{\mathrm{e}}\tau_{\mathrm{ne}}}{2}[1 - \exp(\frac{-2h}{\tau_{\mathrm{ne}}})]} \tag{2.20}$$

$$A_{\mathrm{i}} = \sqrt{\frac{D_{\mathrm{i}}\tau_{\mathrm{ni}}}{2}[1 - \exp(\frac{-2h}{\tau_{\mathrm{ni}}})]} \tag{2.21}$$

D_{e} and D_{i} are noise diffusion coefficients:

$$D_{\mathrm{e}} = \frac{2\sigma_{\mathrm{ne}}^2}{\tau_{\mathrm{ne}}} \tag{2.22}$$

$$D_{\mathrm{i}} = \frac{2\sigma_{\mathrm{ni}}^2}{\tau_{\mathrm{ni}}} \tag{2.23}$$

where σ_{ne} and σ_{ni} are standard deviations of the excitatory and inhibitory noise conductances respectively. All of the parameter values for the noise term can be found in Table 2.3. The neuron model also handles synaptic conductances, which increase immediately with each spike and then decay exponentially with time constants $\tau_{\mathrm{syn_{ex}}}$ for spikes coming from excitatory neurons and $\tau_{\mathrm{syn_{in}}}$ for inhibitory ones. To implement the full neuron model we described it in NESTML (Babu et al., 2021), from which code can be automatically generated for NEST 3.0 (Hahne et al., 2021).

Neuron model variations

We examine the robustness of the model results to simplification of the neuron model described above. The first variation we apply is to disable the intrinsic conductance noise mechanism by setting σ_{ne} and σ_{ni} to 0. This adjustment also allows us to study the susceptibility of the networks to noise in the system. In a second step, we additionally disable the I_{Na}, I_{K}, and I_{M} currents, resulting in leaky (integrate-and-fire (iaf) neurons. We leave all parameters unrelated to these ion channel currents unchanged, but change the membrane potential threshold V_{thresh} of each population (L2/3-E: −52 mV, L2/3-I: −55 mV, L4-E: −49 mV, L4-I: −55 mV, L5-E:−57.0 mV, L5-I: −65.0 mV) such that the means of the population firing rates of the data-based circuit with integrate-and-fire neurons match those of the network with Hodgkin-Huxley neurons as closely as possible (see Supplementary Materials for firing rate distributions of all networks).

Par.	Value	Source	Description
V_{m}	uniformly distributed between -70 and -60 mV	paper	Membrane potential
V_{T}	-63 (-58) mV	code (1)	Voltage offset that controls dynamics
V_{thresh}	-30 mV	code	Membrane potential threshold
V_{S}	-10 mV	(1)	Shifting voltage
E_{L}	-80 mV	(2)	Leak reversal potential
a	34636 µm^2	paper	Membrane area used for all but g_{M}
$\rho_{C_{\mathrm{m}}}$	1 µF/cm^2	(2)	Membrane capacitance density
C_{m}	346.36 pF	$a \cdot \rho_{C_{\mathrm{m}}}$	Capacity of the membrane
$\rho_{g_{\mathrm{L}}}$	0.045 mS/cm^2	(2)	Leak conductance density
g_{L}	15.5862 nS	$a \cdot \rho_{g_L}$	Leak conductance
$\tau_{\mathrm{syn_{ex}}}$	3 ms	code	Time constant of the excitatory synaptic exponential function
$\tau_{\mathrm{syn_{in}}}$	6 ms	code	Time constant of the inhibitory synaptic exponential function
t_{ref}	3 ms	code	Duration of refactory period
E_{ex}	0 mV	(2)	Excitatory synaptic reversal potential
E_{in}	-75 mV	(2)	Inhibitory synaptic reversal potential

Table 2.1: Main neuron Parameters. The source column indicates where the value can be found, searching in the following order: main replicated paper, referenced papers, source code. If a value was given in the paper which differs from the one used in the code, the paper value is written in parenthesis. References: (1) Destexhe and Paré (1999), (2) Destexhe et al. (2001)

Par.	Value	Source	Description
E_{Na}	50 (60) mV	code (3)	Sodium reversal potential
$\rho_{g_{\mathrm{Na}}}$	516 (500) $\frac{pS}{\mu m^2}$	code (paper)	Peak conductance density for I_{Na}
g_{Na}	17872.176 nS (17318 nS)	$a \cdot \rho_{g_{\mathrm{Na}}}$ code (paper)	Sodium peak conductance
E_{K}	-90 mV	code	Potassium reversal potential
$\rho_{g_{\mathrm{K}}}$	100 pS/µm^2	paper	Peak conductance density for I_{K}
g_K	3463.6 nS	$a \cdot \rho_{g_{\mathrm{K}}}$	Potassium peak conductance
a_{M}	10000 µm^2	code	Membrane area used for g_{M}
E_{M}	-80 (-90) mv	code (3)	Potassium reversal potential for I_{M}
$\rho_{g_{\mathrm{M}}}$	10 (5) pS/µm^2	code (paper)	Peak conductance density for I_{M}
$g_{\mathrm{M_{ex}}}$	100 (173.18) nS	$a_{\mathrm{M}} \cdot \rho_{g_{\mathrm{M}}}$ code ($a \cdot \rho_{g_{\mathrm{M}}}$ paper)	Peak conductance of additional potassium ion channel in exc. neurons
$g_{\mathrm{M_{in}}}$	0 nS	paper	Peak conductance of additional potassium ion channel in inh. neurons

Table 2.2: Ion channel parameters. Source definitions as in Table 2.1. References: (3) Mainen, Huguenard, and Sejnowski (1995)

Par.	Value	Source	Description
τ_{ne}	2.7 ms	paper	Time constant for the excitatory noise conductance
τ_{ni}	10.5 ms	paper	Time constant for the inhibitory noise conductance
g_{ne}	12 nS	paper	Mean conductance of the excitatory noise
g_{ni}	57 nS	paper	Mean conductance of the inhibitory noise
σ_{ne}	3 nS	paper	Standard deviation of the excitatory noise conductance
σ_{ni}	6.6 nS	paper	Standard deviation of the inhibitory noise conductance

Table 2.3: Neuronal conductance noise parameters; source definition as in Table 2.1

Synapse model

For the synaptic short-term dynamics, we use the `tsodyks2_synapse` model implemented in NEST. This model implements short-term synaptic plasticity according to Maass and Markram (2002) with the following equations, which are also used in the replicated paper:

$$A_k = w \cdot u_k \cdot R_k \tag{2.24}$$

where A_k is the amplitude of the postsynaptic potential for the kth spike and w is the synaptic weight. The release probability u_k is given by:

$$u_k = U + u_{k-1}(1 - U)\ \exp(-\frac{\Delta_{k-1}}{\tau_{\mathrm{fac}}}) \tag{2.25}$$

where U determines the increase in u with each spike, Δ_k denotes the time since the last spike and τ_{fac} is the time constant for recovery from facilitation. R_k is the fraction of synaptic efficacy available for the kth spike and follows:

$$R_k = 1 + (R_{k-1} - u_{k-1}R_{k-1} - 1)\ \exp(-\frac{\Delta_{k-1}}{\tau_{\mathrm{rec}}}) \tag{2.26}$$

where τ_{rec} is the time constant for recovery from depression. The variables u_k and R_k are initialised with $u_1 = U$ and $R_1 = 1$. The mean values for U, τ_{fac} and τ_{rec} as well as the synaptic delay depend on the type of their source and target neurons and can be found in Table 2.4.

These parameters are not fixed for a given ensemble of synapses between a source population j and a target population i; instead, they are drawn from a Gaussian random distribution with a standard deviation of 50% (b_U for U, b_{rec} for τ_{rec} and b_{fac} for τ_{fac}), 10% (b_d for the delay) or 70% (b_w for the weight) of their mean values. As described by Häusler and Maass (2007), all negative values or values bigger than the upper bound

	Par.			Source	Description
From/to		**E**	**I**		
E	U	0.5 s	0.05 s	paper	Increase of release probability with each spike
	τ_{rec}	1.1 s	0.125 s	paper	Time constant for depression
	τ_{fac}	0.05 s	1.2 s	paper	Time constant for facilitation
	d	1.5 ms	0.8 ms	code	Synaptic delay
I	U	0.25 s	0.32 s	paper	Increase of release probability with each spike
	τ_{rec}	0.7 s	0.144 s	paper	Time constant for depression
	τ_{fac}	0.02 s	0.06 s	paper	Time constant for facilitation
	d	0.8 ms	0.8 ms	code	Synaptic delay

Table 2.4: Population type dependent synaptic parameters; source definition as in Table 2.1

of the range (for U) are replaced by values drawn from a uniform distribution between 0 and two times the mean. Note that using a truncated normal distribution leads to a different network activity with higher firing rates (data not shown).

The mean amplitudes A_{ij} of the postsynaptic potentials for the connections between populations j and i, which are needed to calculate their mean weights, can be found in Figure 2.1. With this, we get the value for their weight w by:

$$w_{\text{ex}} = \frac{A_{ij}}{|E_{\text{ex}} - V_{\text{mean}}|} \tag{2.27}$$

for excitatory synapses and

$$w_{\text{in}} = \frac{A_{ij}}{|E_{\text{in}} - V_{\text{mean}}|} \tag{2.28}$$

for inhibitory ones. E_{ex} and E_{in} are the excitatory and inhibitory synaptic reversal potentials and V_{mean} is the mean membrane voltage of a neuron without input. All values of the synaptic parameters can be found in Tables 2.4 and 2.5.

Network models

In this section, we describe the different network models implemented in the original work and in this replication. All circuits comprise 560 of the Hodgkin-Huxley neurons described above unless otherwise stated. Another common feature shared by six of the seven circuits is that they are connected by synapses with short-term adaptation as described above. The exception is the data-based model variant with static synapses, which helps us examine the effects of synaptic dynamics on task performance. Figure 2.2 shows the histograms of degrees (number of incoming and outgoing synapses) for the different circuits; this serves as the first validation of our work, as they are visually indistinguishable from those presented in Figure 7 of Häusler and Maass (2007), which is the best that can be achieved without access to the original data.

Par.	Value	Source	Description
A	see Figure 2.1	paper	Mean amplitude of PSPs
A_{input}	1.925 mV (1.9 mV)	code (paper)	Mean amplitude of PSPs for input connections
V_{mean}	-65 mV	(2)	Mean membrane voltage of a neuron without input
S_{RW}	66825/N (60000/N)	code (paper)	Scaling parameter for recurrent connections
$S_{\text{RW}_{\text{static}}}$	$S_{RW}/73$	experiment	S_{RW} for data-base circuit with static synapses
S_1	14.85 (14)	code (paper)	Scaling parameter for connections from stream 1
S_2	36.498 (33)	code (paper)	Scaling parameter for connections from stream 2
w_{ex}	$S_{\text{RW}} \cdot \frac{A \cdot g_{\text{L}}}{\lvert E_{\text{ex}} - V_{\text{mean}} \rvert}$	code	Maximum conductance for excitatory synapses
w_{in}	$S_{\text{RW}} \cdot \frac{A \cdot g_{\text{L}}}{\lvert E_{\text{in}} - V_{\text{mean}} \rvert}$	code	Maximum conductance for inhibitory synapses
b_U	0.5	paper	Factor defining the std. for the distribution of U
b_{rec}	0.5	paper	Factor defining the std. for the distribution of τ_{rec}
b_{fac}	0.5	paper	Factor defining the std. for the distribution of τ_{fac}
b_d	0.1	code	Factor defining the std. for the distribution of d
b_w	0.7	paper	Factor defining the std. for the distribution of w_{ex} and w_{in}

Table 2.5: Synapse parameters. Source definitions as in Table 2.1. References: (2) Destexhe et al. (2001)

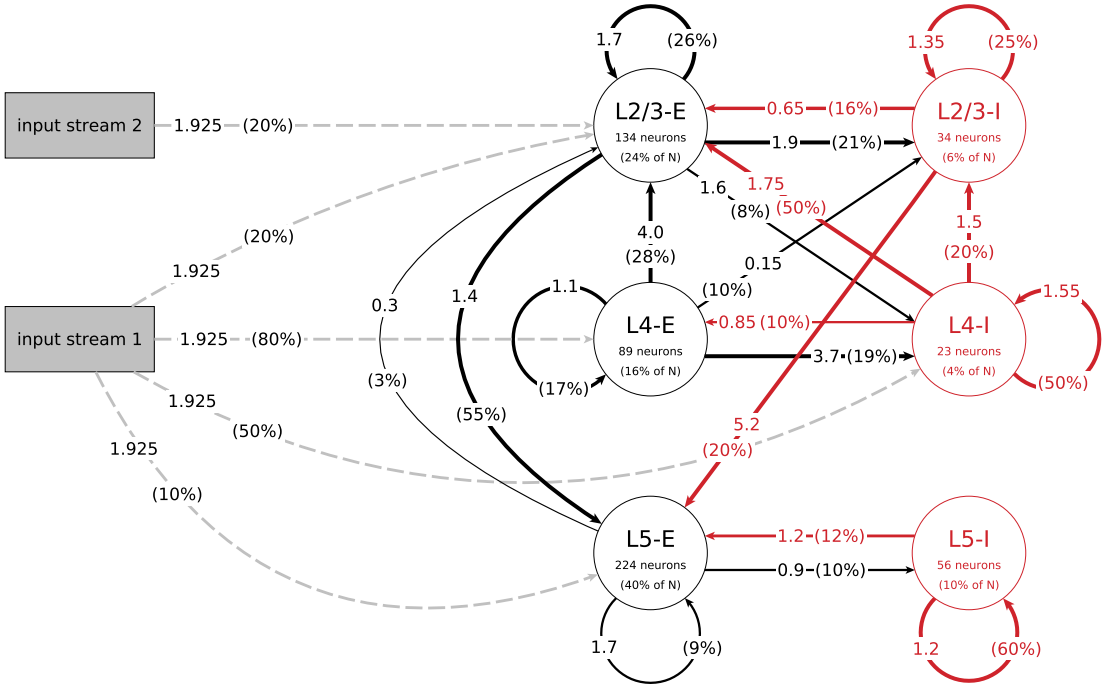


Figure 2.1: Structure of the data-based microcircuit model. The connection arrows are labeled with the connection strength (mean amplitude of post-synaptic potentials (PSPs) in mV, c.f. parameter A in Table 2.5) and the connection probability (in parentheses). Red arrows represent inhibitory connections and excitatory connections are black. The excitatory input connections are represented as grey dashed arrows. Neuron numbers are based on a network size of $N = 560$ neurons. C.f. Figure 1, original publication (Häusler and Maass, 2007).

Data-based circuit The data-based model consists of three layers, each divided into an excitatory and an inhibitory population. Figure 2.1 illustrates the network's connectivity structure; a specification of the parameters can be found in Tables 2.4 and 2.5. Since the data on which the circuit is based comes from biological systems with a much larger number of incoming connections per neuron, the synaptic weights in the model are scaled up by a factor S_{RW} to obtain a reasonable network activity. In the paper, the value of this scaling factor is given as $60000/N$ (about 107 for $N = 560$), but in the published code this parameter is calculated as $66825/N$ (about 119 for $N = 560$). We use the second value in our implementation because it gives a network activity closer to the reported one. The distribution of degrees for this connectivity model (and all following models) can be seen in Figure 2.2.

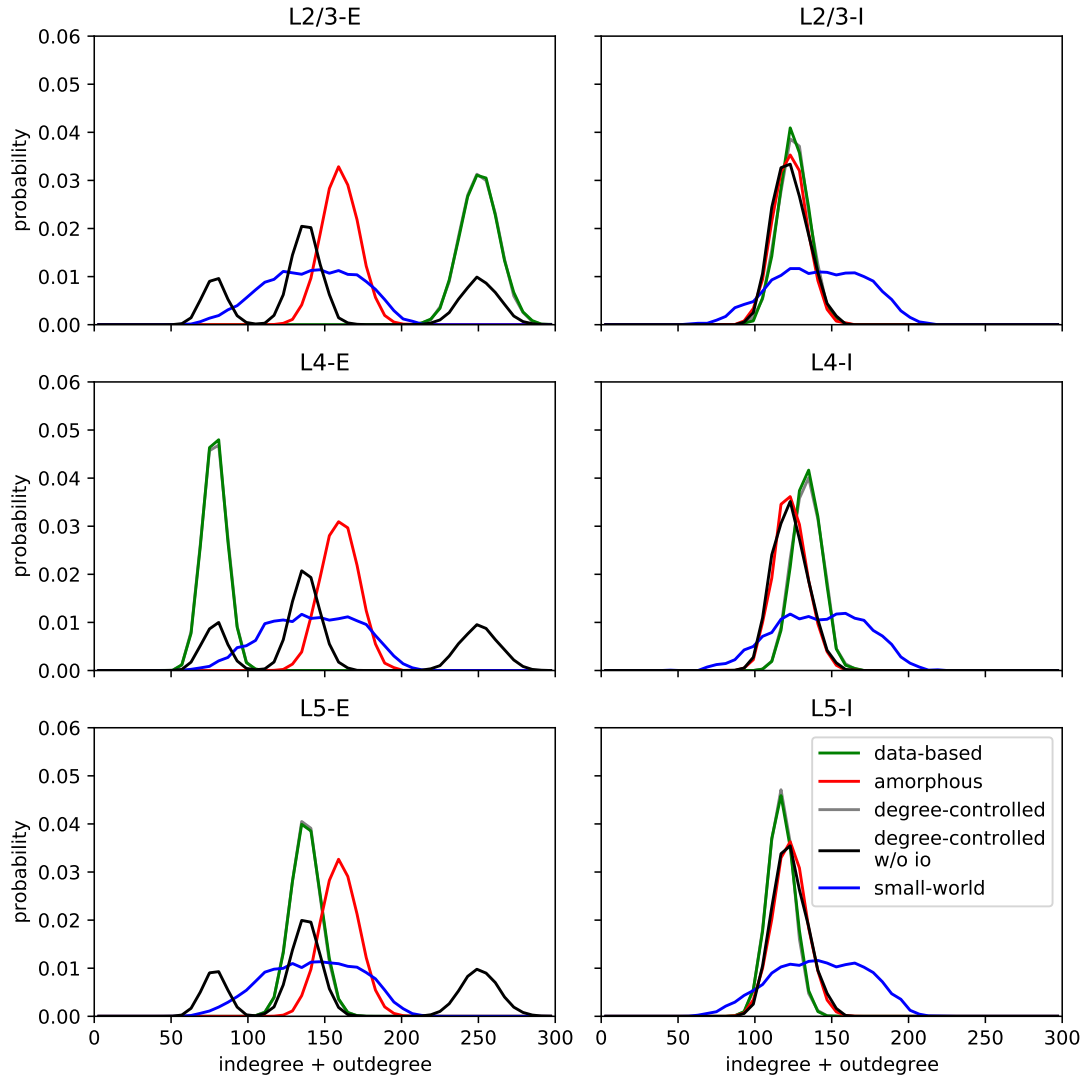


Figure 2.2: Histograms of degrees (number of incoming and outgoing connections per neuron) per population for each circuit. Values are aggregated for 100 runs with different seeds for each model. C.f. Figure 7, original publication (Häusler and Maass, 2007).

Amorphous circuit The amorphous circuit is derived from the data-based circuit by destroying the laminar connectivity structure: for each connection, we replace the source neuron with a random neuron of the same type (excitatory or inhibitory) and also the target neuron with a random neuron of the same type (excitatory or inhibitory), whereby

the new randomly selected neurons are not constrained to belong to the same layer as the ones they replace. Multiple connections between the same neuron pair are excluded. This results in a network that shares most global statistics with the data-based model: number of synapses, their pre- and postsynaptic neuron type and the distribution of all synaptic parameters such as the weights and the parameters defining the short-term dynamics remain unchanged.

Degree-controlled circuit The degree-controlled circuit is also derived from the data-based circuit by scrambling its connections. However, in this network, we ensure that the number of incoming and outgoing connections (the degree) for each neuron remains unchanged. To achieve this, we randomly select two synapses whose source neurons are of the same type (excitatory or inhibitory) and whose target neurons are also of the same type and exchange the target neurons of these synapses. We continue this procedure until none of the original connections remain. Just as in the amorphous circuit, the global statistics of the network are preserved. In addition, the number of incoming and outgoing connections per neuron is the same as in the data-based circuit.

Degree-controlled circuit without input or output specificity The degree-controlled circuit without input or output specificity is derived from the degree-controlled circuit by changing the neurons to which external input is given and from which the states are read out. We implement this by randomly exchanging the layer assignations of neurons of the same type (excitatory or inhibitory) after recurrently connecting the network, but before connecting the external input streams and readouts.

Small-world network As introduced by Watts and Strogatz (1998), the small-world network is one in which the underlying undirected graph has small-world properties. Such networks show a higher clustering coefficient than amorphous circuits while keeping the average shortest path length at a comparable value. Watts and Strogatz define the local clustering coefficient of a node as the fraction of all possible connections between the node's neighbors that actually exist. It represents how close the neighborhood is to being a clique. The global clustering coefficient of a network is the average of all local clustering coefficients. The shortest path length between two nodes measures the separation of nodes and is defined as the minimum number of links required to get from one node to the other. This shortest path length is averaged over all possible node pairs in the network. We generated a small-world network using the spatial growth algorithm proposed by Kaiser and Hilgetag (2004); first we initialize the network by assigning the position (0.5, 0.5) to a random node, then we perform the following steps:

1. Take a new node and assign it a random position (x, y) with coordinate values drawn from the interval [0,1].

2. Connect the new node with all other nodes with probabilities defined by:

$$P(u, v) = \beta e^{-\alpha d(u,v)} \tag{2.29}$$

where $d(u, v)$ is the Euclidian distance between the nodes u and v, β is a general density parameter and α is a spatial range parameter, which regulates the dependence of the connection probability on the distance.

3. Repeat steps 1 and 2 until the desired number of nodes has been reached.

By choosing $\alpha = 4$ and $\beta = 1.32$ we obtain small world networks that have a clustering coefficient around 36% and an average shortest path value of about 1.75 links, comparable to those of data-based circuits.

To get the final connections for the network, we randomly assign a direction to every edge and set the weight and other synaptic parameters according to the neurons' population affiliations (see Figure 2.1 and Table 2.4). If a neuron pair belongs to two populations that aren't connected in the data-based circuit (see Figure 2.1), we randomly draw a weight from connection definitions for the same synapse type (excitatory or inhibitory). For example, given a connection from L5-E to L4-I, we would randomly select another excitatory connection such as L4-E to L4-I and use its weight. Since the other synaptic parameters depend only on the type (excitatory or inhibitory) of the connected neurons and not on the exact population affiliation, we can read them out from Table 2.4 as we did for all other connections.

Data-based circuit with static synapses This network is identical to the data-based circuit but with the dynamic synapse model replaced by static synapses. To achieve a similar network activity we have to adjust the scaling parameter S_{RW}. As this value is not explicitly stated by Häusler and Maass (2007), we tuned this parameter by hand to obtain firing rates of the network as close as possible to the data-based circuit, under the condition that no population is silent. This is achieved at a value $S_{\mathrm{RW}}^{\mathrm{st}} = S_{\mathrm{RW}}/73$ (see Figure A.2 in the appendix for the resulting firing rate histograms of all networks).

Data-based circuit with random synaptic dynamics This network is identical to the data-based circuit, except that the short-term dynamics of the data-based network's connections are scrambled. To do this, for each synapse we randomly select one of the four connection types (EE, EI, IE, II) independently of the actual source and target of the synapse. We then draw values for the parameter values U, τ_{rec}, τ_{fac}, and d according to the corresponding distributions for the selected connection type, with mean values as given in Table 2.4 and standard deviation factors as given in Table 2.5.

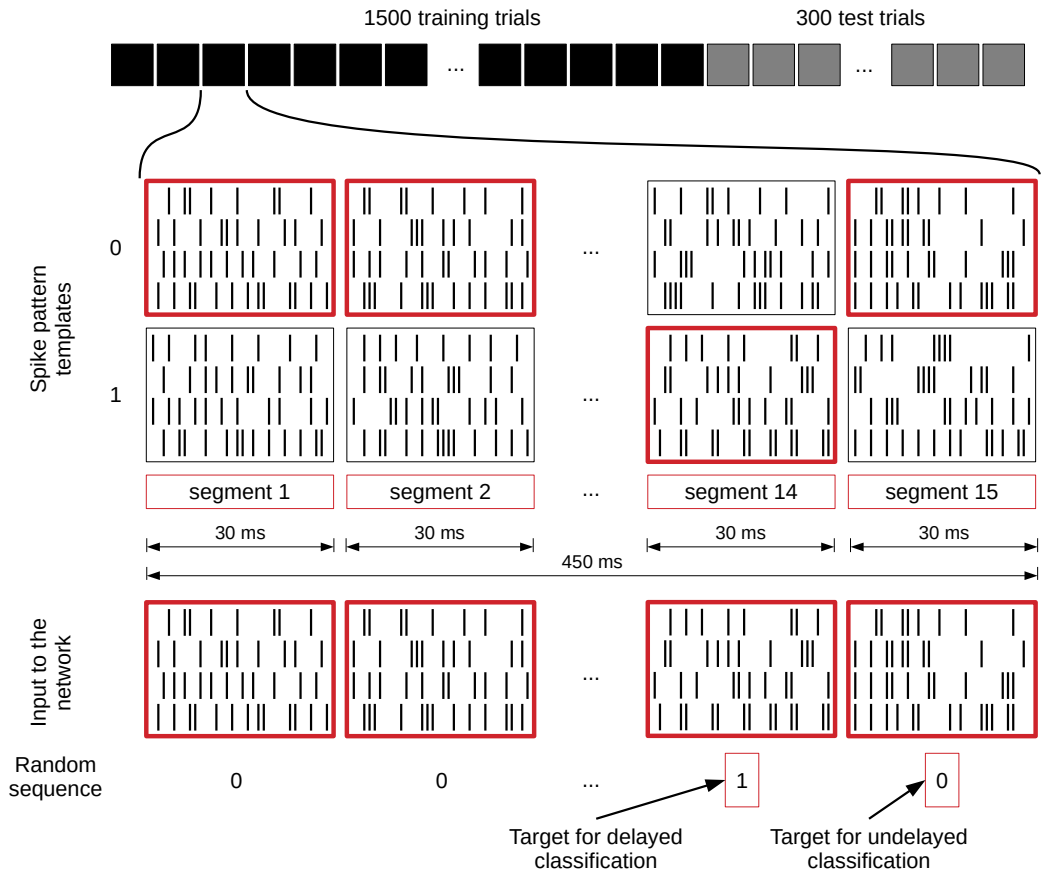


Figure 2.3: Input generation for spike pattern-based tasks. The first row shows the different trials of an experiment and the spike patterns below that represent a zero or one value for each segment of a single trial. These spike pattern templates are identical for all trials. The spike patterns at the bottom are chosen based on and therefore represent the randomly generated sequence of zeros and ones underneath, which also define the target for the readout training (value of segment 14 for delayed classification and segment 15 for the undelayed classification). We give a jittered version of these spike trains to the network (jittering is not shown). C.f. Figure 4, original publication (Häusler and Maass, 2007).

2.2.2 Tasks

Häusler and Maass (2007) implemented several different tasks to evaluate the computational performance of the different network models. Some of the tasks are based on the

classification of precise spatio-temporal spike patterns and for others, the circuits need to perform computations on the input firing rates of input spike trains whose spikes are generated by a Poisson process. All tasks are based on inputs given as two input streams which are connected to the network as shown in Figure 2.1.

Analogously to the scaling of recurrent connections by the factor S_{RW}, the weight values of the synapses from these streams to the circuit are multiplied by their scaling factors S_1 and S_2. As can be seen in Table 2.5 the values given in the paper differ from the values in the code; we use the latter in our implementation.

Depending on the task, the input streams consist of either four (rate-based tasks) or 40 (spike pattern classification tasks) spike trains. Per trial, 15 segments with a duration of 30 ms are generated, resulting in a spike train of 450 ms for each trial. Only the input for the retroactive spike pattern classification with fixed inter-stimulus input and the more finely resolved memory tasks, which are both further explained in the next section, are exceptions to this scheme.

Figure 2.3 illustrates how these input streams are generated for the spike pattern-based tasks. For each segment, two different spike patterns are generated which encode either a zero or a one and the randomly generated input value (bottom row of zeros and ones in the figure) defines which of them is used in the current trial. These two possible patterns for each segment remain identical for each trial. In this way, a sequence of 15 zeros and ones is translated into a set of spike trains of 450 ms length. In addition, each input spike is jittered by a Gaussian distribution with mean 0 ms and a standard deviation of 1 ms. We apply this jittering once per trial to the selected templates and each neuron connected to the input receives the same jittered version of the spike train. Even though some tasks are calculated based on only one of the inputs, both streams are activated and connected all the time, and the tasks are then evaluated for each input separately.

The tasks are performed by the networks using a reservoir computing approach and thus require readout neurons. We connect two different readouts to the systems: the first readout mimics an excitatory neuron of layer 2/3 and sums up the filtered spike trains of its inputs. Exactly like a normal layer 2/3 neuron, it does not receive input from all possible sources in a linked population but is randomly connected to a subset of the units based on the corresponding connection probability. The second readout mimics an excitatory layer 4 neuron in the same way. We filter the spikes with an exponential function using a time constant of 15 ms. Note that inhibitory neurons are connected to the spike filtering devices with a negative weight, resulting in negative values. This is important because the readout weights are trained with a linear least squares method with non-negativity constraints. This results in non-negative readout weights and thus forces the readouts to be in accordance with Dale's principle (Eccles, Fatt, and Koketsu, 1954), which states that a neuron releases the same set of transmitters at all of its synapses. The states on which the readouts are trained and tested are the values of the filtered spike trains at the end of each 450 ms trial. A specification of the task

Pars.	Value	Source	Description
τ_{filter}	15 ms	paper	Time constant for spike filtering
T_{train}	1500	paper	Default number of training trials
T_{test}	300	paper	Number of test trials
n_{seg}	15	paper	Number of segments per trial
t_{seg}	30	paper	Duration of a single input segment

Table 2.6: Task and training parameters.

parameters can be found in Table 2.6.

Spike pattern based tasks We implemented three main spike pattern tasks: spike pattern classification $\text{tcl}_i(t)$, where i denotes the input stream for which the classification is performed, delayed spike pattern classification $\text{tcl}_i(t - \Delta t)$, and the exclusive-or (XOR) task. The inputs for all of these tasks are exactly the same; the difference lies in the task-specific training of the readout weights. For the instantaneous spike pattern classification, the readout weights are trained on the prediction of the value (0 or 1) of the last segment of each trial (segment 15), whereas the target of the delayed classification is the value of the penultimate segment (segment 14), see Figure 2.3.

In a further set of experiments that go beyond the original study, we use a step duration of 5 ms instead of the standard 30 ms and classify the spike patterns of all 15 segments. As these tasks are all based on only one input, we evaluate them for both input streams separately.

In contrast to this, the XOR task is computed based on the value of the last segment of both input streams. To evaluate the task performances we use a threshold of 0.5 to fix the readout predictions to values of zero or one and calculate the kappa coefficient between the target output and the predicted output. The kappa coefficient is calculated as:

$$\kappa = \frac{P_0 - P_C}{1 - P_C} \tag{2.30}$$

where P_0 is the agreement between the target and the observed prediction and P_C is the chance agreement.

In addition to these three task types, we also implemented the retroactive spike pattern classification with fixed inter-stimulus input, which Häusler and Maass (2007) use to evaluate the training convergence in dependence on the number of training trials. The task is to classify spike patterns consisting of four spike trains with a duration of 100 ms after an intervening fixed spike pattern of 100 ms was given to the network in every trial. We implemented this by setting the segment duration to 100 ms, the number of segments per trial to two, the input dimension to four, the second input value of every

trial to zero and the first input value as the target for the readout. We also disabled spike jittering for the fixed spike trains between the target stimuli.

Firing rate tasks In addition to the spike pattern-based tasks, computational tasks were also defined using time-varying firing rates. The structure of the input streams is similar to the one used in the previously described tasks and the visualization in Figure 2.3. However, instead of taking one of two pre-generated spike patterns, we define a target firing rate between 15 and 25 spks/s for each of the 15 segments in both input streams and generate the spike trains based on this rate. The firing rate of the last 15 ms of each trial is used as the target for the readouts. To avoid errors resulting from a division by zero, we ensure that at least one spike is placed in this last 15 ms window of the input streams. The tasks we implemented are the quotient of the two input streams r_1/r_2 and the square of their difference $(r_1 - r_2)^2$. As the kappa coefficient can't be used for these analog prediction tasks, we evaluate the performance on the basis of the Pearson correlation coefficient between the prediction and the target.

2.2.3 Simulation and analysis framework

We simulate the spiking neural network experiments with a timestep of 0.2 ms using NEST 3.0 (Hahne et al., 2021). Since the neuron model we describe above is not included in NEST, we implement it using NESTML 4.0.0 (Babu et al., 2021). For all other data analysis and plotting we use Python 3.8.8 and a modified version of the Functional Neural Architectures library (Duarte et al., 2021).

2.3 Results

2.3.1 Network activity

After establishing that the degree distributions of the various networks were visually indistinguishable from the published distributions (see Figure 2.2), we then examined the activity of the data-based network. Figure 2.4A shows a raster plot for the network with input stream two becoming active at 100 ms, and Figure 2.4B provides the corresponding firing rate histograms for the six populations and, combined, the three layers (c.f. Figure 2.1). These plots can be compared with Figure 2B,C of the original publication.

Note that whereas the firing rate histograms in Figure 2.4B are very similar to those shown in the original paper, the raster plot in Figure 2.4A exhibits some discrepancies. Most notably, the latency of network activity is longer in our implementation than in the original. Only a few inhibitory layer 4 neurons show earlier activity, and although both figures are based on trials of only 450 ms, this behavior is consistent in our experiments. Less consistent is the measured firing rate of layer 5. In contrast to the original study,

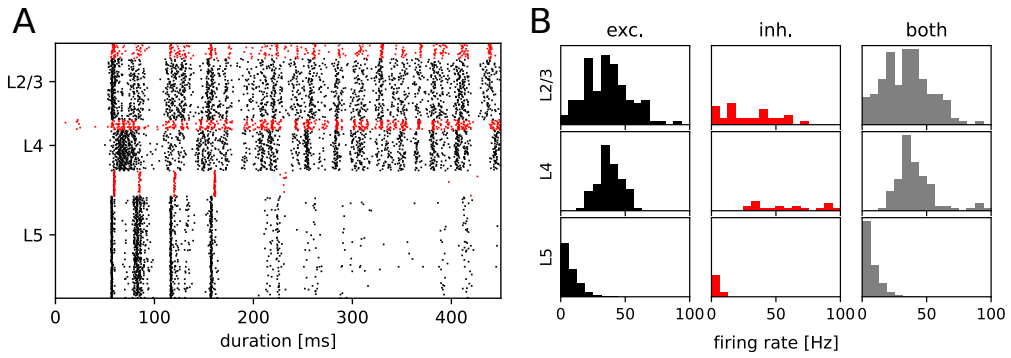


Figure 2.4: Activity of the data-based circuit. **A**: Raster plot for the data-based circuit after input stream two was activated at 100 ms. Excitatory neurons are black and inhibitory neurons are red. **B**: The corresponding firing rate histograms for each population and layer. C.f. Figure 2B and 2C, original publication (Häusler and Maass, 2007).

which reports a stable firing rate of around 8.5 spks/s in this layer, we observe a range of firing rates between 3 and 9 spks/s for differently seeded runs. A possible explanation for these discrepancies is that in the original code, the values of E_{M} and V_{m} are transformed into a different simulation voltage range to compute the non-inactivating K^+ current I_{M}. For this transformation, values of -70 mV for the resting potential and -40 mV for the threshold potential were used, rather than the values used in the rest of the study (-80 mV, -30 mV, see Table 2.1). In our implementation, we elected not to include these transformations in the neuron model, as we could determine neither a biological basis nor a computational advantage for so doing; as shown in the following sections, a qualitative reproduction of the task performances is achieved without such transformations.

2.3.2 Task performance for the circuit variants

Figure 2.5 shows the results of the seven main tasks for the data-based and the amorphous circuits. Although the performance values are not identical to the ones in the original study, the values are close and qualitatively reproduce the key finding that the data-based circuit outperforms the amorphous control circuit in every task. The main difference between our results and those reported in the original study is our comparatively low performance at rate-based tasks and the delayed spike pattern classification of input stream one (for both circuits).

Additionally, Figure 2.6A shows the performance of the data-based and amorphous circuit for the retroactive spike pattern classification task with fixed inter-stimulus input

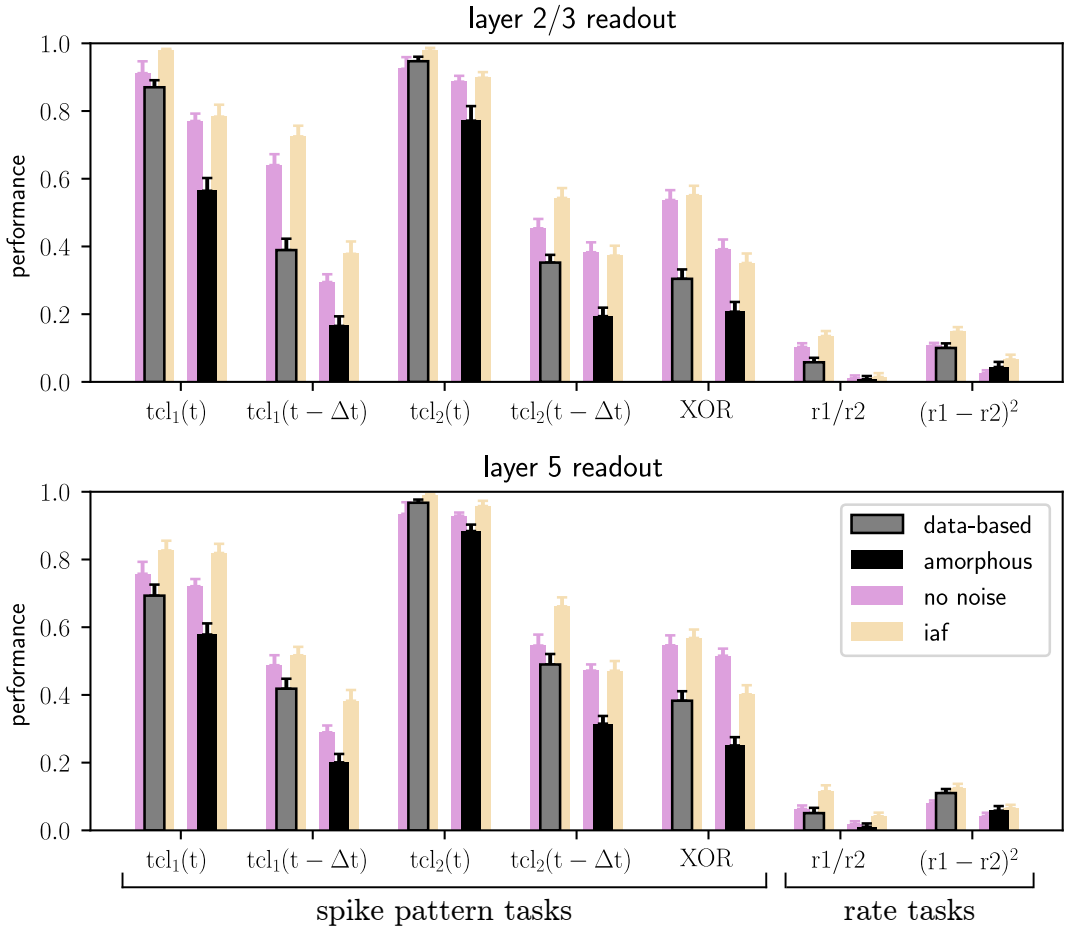


Figure 2.5: Performance of trained linear readout neurons in layers 2/3 and layer 5 for the classification tasks on spike patterns and computations performed on the rates of the two input streams (see Section 2.2.2), both for data-based laminar microcircuit models (gray bars) and for the amorphous control circuits (scrambled laminar structure; black bars). Light purple bars represent the results for networks with neurons without conductance noise and light orange bars networks consisting of integrate-and-fire neurons. Error bars are the standard errors of mean. All values are averaged over 20 runs. C.f. Figure 5, original publication (Häusler and Maass, 2007), likewise averaged over 20 runs.

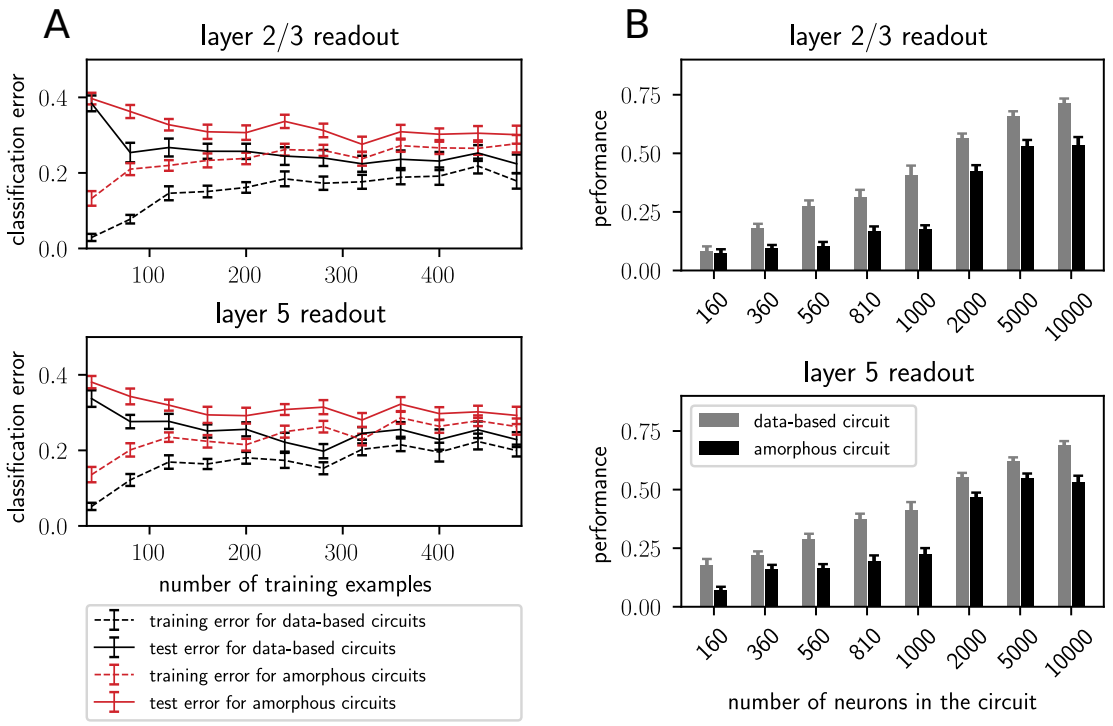


Figure 2.6: **A**: Training and testing error of readouts from data-based and amorphous circuit models as functions of the size of the training set. 300 trials are always used for testing. Error bars indicate the standard error of means. Values are averaged over 30 runs. C.f. Figure 8, original publication (Häusler and Maass, 2007), averaged over 20 runs. **B**: Performance (kappa coefficient) on the XOR task of projection neurons in layers 2/3 and layer 5 for different circuit sizes, with and without a data-based laminar structure. All values are averaged over 30 runs. Error bars indicate the standard error of the means. C.f. Figure 6, original publication (Häusler and Maass, 2007), averaged over 10 runs.

Tasks/circuits	Data-based	Amor-phous	Small-world	DC	DC (no io)	Rand. dyn.	Static syn.
$\mathrm{tcl}_1(\mathrm{t})$ (L23)	**0.87**	-0.305	-0.17	-0.106	-0.235	-0.117	0.013
$\mathrm{tcl}_2(\mathrm{t})$ (L23)	**0.947**	-0.175	-0.171	-0.097	-0.168	-0.44	-0.067
$\mathrm{tcl}_1(\mathrm{t})$ (L5)	**0.693**	-0.115	-0.145	-0.106	-0.049	-0.407	-0.069
$\mathrm{tcl}_2(\mathrm{t})$ (L5)	**0.968**	-0.084	-0.117	-0.065	-0.191	-0.469	-0.159
$\mathrm{tcl}_1(\mathrm{t}-\Delta t)$ (L23)	0.389	-0.224	-0.024	-0.061	-0.131	**+0.021**	-0.056
$\mathrm{tcl}_2(\mathrm{t}-\Delta t)$ (L23)	0.352	-0.158	-0.02	**+0.071**	-0.081	-0.185	-0.091
$\mathrm{tcl}_1(\mathrm{t}-\Delta t)$ (L5)	**0.418**	-0.217	-0.073	-0.117	-0.101	-0.136	-0.258
$\mathrm{tcl}_2(\mathrm{t}-\Delta t)$ (L5)	**0.49**	-0.175	-0.119	-0.007	-0.187	-0.259	-0.196
XOR (L23)	**0.304**	-0.097	-0.145	-0.053	-0.114	-0.174	-0.061
r1/r2 (L23)	0.058	-0.05	**+0.083**	+0.043	+0.076	+0.073	-0.007
$(\mathrm{r1}-\mathrm{r2})^2$ (L23)	0.1	-0.056	+0.026	**+0.028**	+0.012	+0.006	-0.013
XOR (L5)	**0.383**	-0.132	-0.206	-0.102	-0.198	-0.206	-0.095
r1/r2 (L5)	0.051	-0.04	+0.069	+0.064	**+0.116**	+0.029	-0.025
$(\mathrm{r1}-\mathrm{r2})^2$ (L5)	0.11	-0.052	**+0.026**	+0.017	+0.014	-0.02	-0.067
memory	**0.43**	-0.183	-0.07	-0.001	-0.147	-0.17	-0.135
nonlinear	**0.168**	-0.072	-0.025	-0.001	-0.016	-0.049	-0.045
other	**0.938**	-0.162	-0.144	-0.083	-0.196	-0.373	-0.093
all	**0.463**	-0.129	-0.072	-0.024	-0.105	-0.176	-0.084

Table 2.7: Performance measures for all networks and all tasks. Spike-based tasks are evaluated with the kappa coefficient and rate-based tasks with the correlation coefficient. The data-based column gives the absolute value and the other columns show the difference from this value. The best performance per task/row is marked in bold. Grey/blue shading denotes tasks from the categories memory/nonlinear. All values are averaged over 20 runs (10 runs in the original paper).

for different numbers of training examples. The original study does not specify which input stream was used to generate the corresponding figure in their work (Figure 8); we therefore tested both of them. Our experiments show more similar results for input stream one, and so we use those results as the basis for Figure 2.6A. As in the original study, the data-based circuit has a lower test and training error than the amorphous circuit for all sizes of training set. Taken together, Figure 2.5 and Figure 2.6A support the argument put forward by the original study that a laminar structure has a positive effect on the computational performance of a circuit.

The quantitative performance measures for all circuits (Section 2.2.1) and all tasks (Section 2.2.2) can be found in Table 2.7. These results can also be expressed as percentage difference from the performance of the data-based circuit; this analysis is given in Table 2.8.

The last four rows of both tables show results averaged over both readouts and over a category of tasks. The *memory* row averages over all tasks for which the networks need to memorize earlier inputs ($\mathrm{tcl}_1(t-\Delta t)$ and $\mathrm{tcl}_2(t-\Delta t)$), the *nonlinear* row averages

Task/circuits	Amor-phous	Small-world	DC	DC (no io)	Rand. dyn.	Static syn.
tcl_1(t) (L23)	-35.1	-19.6	-12.2	-27.0	-13.4	1.5
tcl_2(t) (L23)	-18.5	-18.1	-10.2	-17.8	-46.5	-7.1
tcl_1(t) (L5)	-16.6	-20.9	-15.3	-7.2	-58.7	-10.0
tcl_2(t) (L5)	-8.6	-12.1	-6.6	-19.6	-48.4	-16.4
$tcl_1(t-\Delta t)$ (L23)	-57.5	-6.2	-15.6	-33.8	**5.4**	-14.4
$tcl_2(t-\Delta t)$ (L23)	-44.9	-5.9	**20.1**	-23.1	-52.6	-25.9
$tcl_1(t-\Delta t)$ (L5)	-52.0	-17.5	-28.1	-24.3	-32.5	-61.8
$tcl_2(t-\Delta t)$ (L5)	-35.7	-24.3	-1.4	-38.2	-52.9	-40.0
XOR (L23)	-31.9	-47.8	-17.6	-37.5	-57.4	-20.2
r1/r2 (L23)	-86.8	**142.6**	**74.1**	**130.8**	**126.5**	-11.2
$(r1 - r2)^2$ (L23)	-56.1	**26.3**	**28.2**	**11.8**	**6.3**	-12.7
XOR (L5)	-34.5	-53.9	-26.7	-51.8	-53.9	-24.8
r1/r2 (L5)	-79.2	**136.1**	**126.4**	**227.8**	**57.4**	-48.6
$(r1 - r2)^2$ (L5)	-47.1	**23.7**	**15.5**	**12.3**	-18.0	-61.2
memory	-42.5	-16.4	-0.2	-34.1	-39.7	-31.3
nonlinear	-41.7	-14.7	-0.3	-9.5	-29.0	-26.6
other	-17.3	-15.3	-8.8	-20.9	-39.8	-9.9
all	-27.9	-15.5	-5.2	-22.6	-38.1	-18.2

Table 2.8: Performance measures for all control networks and all tasks expressed as the average difference (in percent) from the performance of the data-based circuit. Positive values indicating a performance improvement with respect to the data-based circuit are marked in bold. All values are averaged over 20 runs. C.f. Table 2 in original publication (Häusler and Maass, 2007).

the results over the tasks based on nonlinear computations (XOR, $r1/r2$ and $(r1-r2)^2$) and the *other* row summarizes all other tasks ($\text{tcl}_1(t)$ and $\text{tcl}_2(t)$). The last row of the tables averages the results over all tasks. In the original paper only the last four rows of Table 2.8 were presented (Table 2 in Häusler and Maass, 2007).

According to the averaged results, the data-based circuit outperforms all other circuits on all tasks. Here we have broad agreement with the original study, in which only the degree-controlled network had slightly superior performance in two task categories.

Examining the disaggregated data in Table 2.8, we observe that there are 17 instances where a control circuit exhibited superior performance to the data-based circuit. In particular, most networks surpass the data-based circuit on all rate-based tasks. However, as Table 2.7 shows, the performance for these tasks is very low for all networks, which means that even a small absolute increase in the correlation coefficient results in a substantial percentage increase. We therefore consider this partial contradiction of the original study to be neuroscientifically uninteresting.

In addition to the rigorous analysis of the effect of circuit connectivity, the original study also considered the influence of network size by increasing the number of neurons within each population of the circuit. Figure 2.6B shows the dependence of the XOR task performance as a function of the number of neurons in the circuit. As with the original study, our results show a systematically better performance for the data-based circuit over the amorphous circuit, and an increase in performance for both circuit types with increasing circuit size up to 5000 neurons. While the data-based network still benefits from increasing the network size to 10000, the performance of the amorphous circuit reaches its maximum value at 5000 neurons. This effect could not be observed in the original study, as the maximum size of the network examined was 1000 neurons.

2.3.3 Robustness to neuron model simplifications

The original study demonstrated the computational benefits of lamina-specific connectivity using a fairly complex neuron model. We therefore hypothesize that the details of the neuron model are not relevant to this key finding. To test this hypothesis, we examine the robustness of our dynamical and computational results to variations in the neuron model (see Section 2.2.1). First, we consider the intrinsic noise mechanism. As shown by the raster plots and firing rate histograms in Figure 2.7, the firing activity in the networks does not change significantly for the data-based and amorphous circuits in the absence of intrinsic noise. Moreover, we observe that the data-based connectivity structure is still superior to all other connectivity patterns in all task types, which Figure 2.5 (light purple bars) illustrates for the comparison with the amorphous circuit (see Table A.1 in the appendix for the summarized performance measures for all circuit types). Figure 2.5 also shows that networks without noise (both data-based and amorphous circuits) perform slightly better than their noisy counterparts in most of the tasks performed, and even considerably better in the nonlinear XOR task.

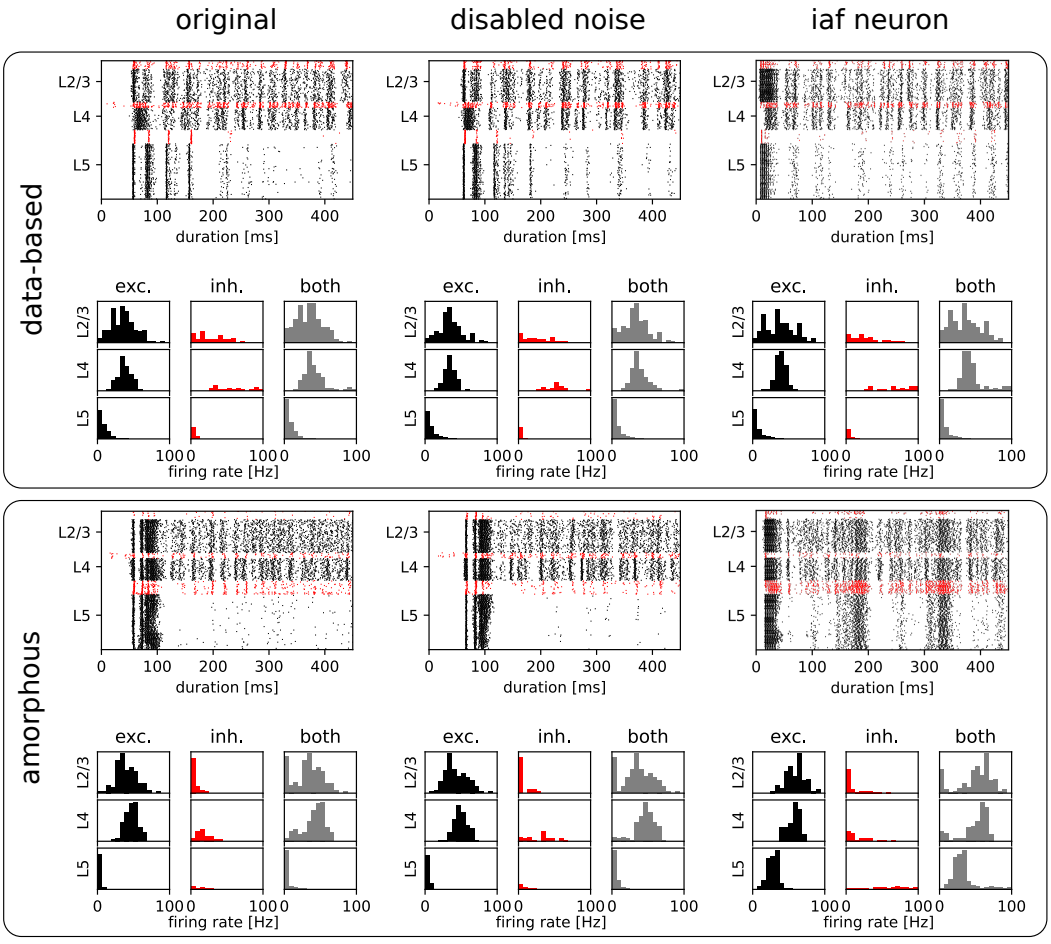


Figure 2.7: Raster plots and firing rate histograms for the data-based and amorphous circuits for the three different neuron types (original: Hodgkin-Huxley neurons that were used in the original publication, disabled-noise: Hodgkin-Huxley neurons without intrinsic conductance noise, iaf neuron: integrate-and-fire neurons). The spikes of the inhibitory populations are colored red, while those of the excitatory populations are shown in black. As in Figure 2B and Figure 2C, original publication (Häusler and Maass, 2007), for the raster plots input stream two starts at 100 ms.

Second, we reduce the neuron model from a Hodgkin-Huxley to a much simpler integrate-and-fire neuron model. To obtain firing rate ranges in the data-based circuit as close as possible to those of the network with Hodgkin-Huxley neurons, we adjust V_{thresh} of the integrate-and-fire neuron model to a different value for each population

(see Section 2.2.1 for the parameter values and Figure A.3 in the appendix for the firing rate distributions). The top right part of Figure 2.7 shows the corresponding raster and firing rate plots.

Also among the circuits consisting of integrate-and-fire units, the data-based network has the best task performance (see Table A.2 of the appendix for the summarized performance measures). Moreover, Figure 2.5 shows similar results for the Hodgkin-Huxley neural networks without noise (light purple bars) and the iaf circuits (light orange bars), with the XOR task values of the amorphous circuits showing the most noticeable difference.

We conclude that these results confirm our hypothesis that the superiority of the data-based connectivity structure does not depend on the specifics of the neuron model. Moreover, they reveal that a reduction in complexity even leads to an increase in performance on the tasks conducted. We hypothesize that the dynamics of the Hodgkin-Huxley model, which are much more intricate than those of integrate-and-fire neurons, may effectively act as an additional noise source that reduces task performance.

2.3.4 Detailed memory tasks

To get further insight into the memory capabilities of the networks, we devised a modification of the retroactive spike pattern classification task, namely reducing the step size from 30 ms to 5 ms and classifying the spike patterns of all 15 segments (see Section 2.2.2). This gives our view on retroactive spike pattern classification a six times higher resolution with one data point for every 5 ms interval instead of only every 30 ms, allowing us to determine the memory profile for each circuit variant. Our results for the more detailed memory tasks are summarized in in Figure 2.8.

We observe that all network variants require some processing time to reach their peak performance, see Figure 2.8A. For all combinations of network, input stream, and readout location, the maximum kappa coefficient is reached after a delay of two steps (10 ms), i.e. the networks have the greatest accuracy in identifying the stimulus inserted two steps before the current one. The only exception is the layer 5 classification of input stream 1 of the data-based network with random synaptic dynamics, which reaches its maximum after a delay of three steps (15 ms). However, in general, the performance increases steeply up to delay 2 (10 ms) and then decreases more slowly until all circuits reach a value close to zero at delay 10 (50 ms).

Notably, the performance of the undelayed classification is worse than that of the short-term delayed classification, in contrast to the results presented in Figure 2.5 for a step of 30 ms. This can be understood by considering that the networks need more than 5 ms to process the input and generate an informative response from the few neurons on which the readouts are based. One reason for this is synaptic delays since for example excitatory-to-excitatory synapses already require an average of 1.5 ms to transmit a single spike from a presynaptic to a postsynaptic neuron. With the longer step of 30 ms,

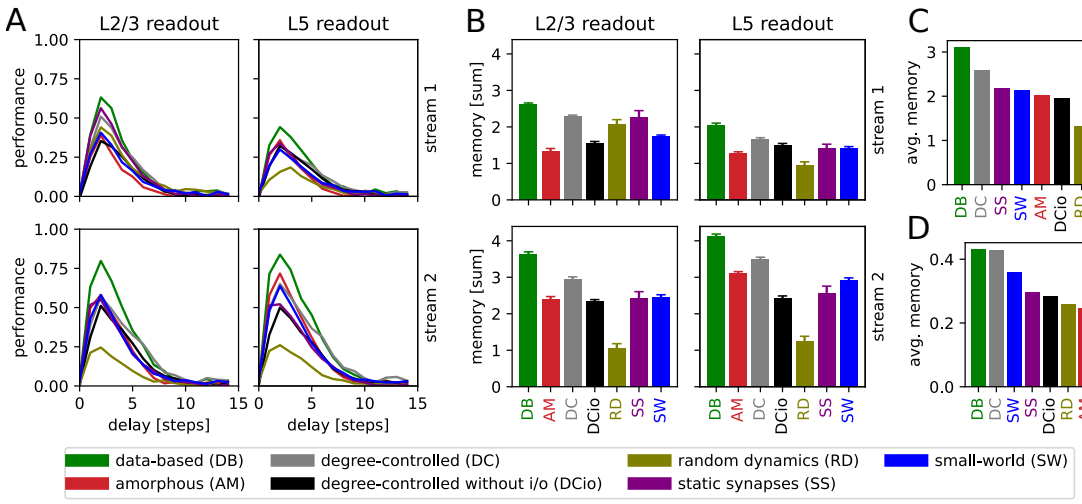


Figure 2.8: Results of retroactive spike pattern classification tasks for all network types. **A**: Performances (kappa coefficient) for the classification of spike patterns with a duration of 5 ms at different delays, separated by input stream and readout (averaged over 40 trials). **B**: Bars representing the sum of task performances over all delays for the same task as in A. Error bars represent the standard error of mean. **C**: Values from B averaged across all input streams and readouts. **D**: Averaged results of the delayed classification of 30 ms spike patterns (data of *memory* row in Table 2.7).

the network has plenty of time to respond informatively to the undelayed stimulus, whereas the effects of the previous stimulus have already faded considerably. Likewise, the longer step duration provides greater possibilities for readout weights to be learned that accurately distinguish between stimuli, resulting in a better peak performance for the 30 ms task.

The heights of the bars in Figure 2.8B indicate the sum of the values for all delays in Figure 2.8A. This illustrates that not only does the peak performance of the circuit with data-based connectivity surpass that of all other systems at the optimal delay, as shown in the previous line graph, but also that a more general view encompassing the task results for all delays reveals the superiority of this circuit. As the data-based circuit does not retain stimulus information for longer than the other circuits, we conclude that its superior performance must be due to the laminar connectivity enabling it to generate more distinct representations of the input stimuli.

Figure 2.8C generalizes this view even further by averaging the results shown in Figure 2.8B over the input stream and readout location and sorting the networks by performance. Here, the degree-controlled circuit follows the best-performing data-based circuit

and the network with random synaptic dynamics has the lowest result. The next four systems (SS, SW, AM, DCio; Figure 2.8C) are comparatively at the same level with only minor differences. Although one can not directly compare the averaged sum over multiple delays with the performance based on only a single delay, it is interesting to note that this graph does not display the same order as the results of the previous memory tasks with the longer step of 30 ms shown in Figure 2.8D and Table 2.7. While in both cases the data-based circuit is best and the degree-controlled circuit second best, the difference is much smaller for the original memory tasks and the order of the remaining networks is different. For both step durations, the amorphous and the degree-controlled circuit without input and output specificity show results at similar levels to the network with static synapses. For the 30 ms memory task type the small-world network performs comparatively better than this group, positioning itself in third place, whereas for the detailed memory tasks with 5 ms resolution, the network with random synaptic dynamics performs noticeably worse than the other circuits.

Clearly, the stimulus step duration has an effect on the computational capabilities of the networks and can move a network's peak performance to different delays. Moreover, the ordering differences in Figure 2.8C and D suggest that the optimal step duration for a network depends on the connectivity structure. However, the good performance of the degree-controlled circuit and especially the data-based circuit for both step durations tested show that both of these systems have a lower dependence on this duration parameter and can more robustly handle stimuli of different lengths.

2.4 Replicability

Our results in this chapter demonstrate that we could replicate the circuits of the original study and both confirm and strengthen their key findings. However, we encountered significant challenges during this process and it would have been essentially impossible if we had only had the paper as a source of information. As can be seen in the parameter tables 2.1, 2.4 and 2.5, several of the neuron and synapse parameters were not given in the paper, or a different value was reported than was used in the code. For example, only by examining the code could we determine the specification of the synaptic delay, namely that it is drawn from a random distribution with a mean value that depends on the connection type (EE, EI, IE, II).

Likewise, the synaptic time constants were only given in the code and the parameterization of the neuron ion channels showed some discrepancies. In addition to minor differences in the parameters of the Na^+ ion channel, there was greater variation in the definition of the K^+ ion channel, which is responsible for the current I_M. Häusler and Maass (2007) provide peak conductance densities for the different ion channels and a single membrane area, which can be used to calculate the peak conductance of every channel. Examining the code reveals that for the standard sodium and potassium ion

channels, the reported membrane area is used to calculate the conductivity, however, it is based on a different (and unreported) area value for the other potassium ion channel (responsible for I_{M}). In addition, the conductance density of the latter channel is twice as large as that stated in the paper and different factor r was used in equations 2.15 and 2.16 defining the channel dynamics, which is a deviation that is easy to overlook. These differences in parameters significantly alter the activity of the network and led us to disable the M ion channel in early experiments to obtain approximately comparable network responses.

Besides the difficulty in specifying basic synaptic and neuronal parameters, the scaling parameters of the synaptic weights SRW, S_1 and S_2 also caused some problems. All of them were different from the values reported in the paper and it was hard to track them down in the MATLAB code because the final scaling parameter values were not set directly, but defined by somewhat convoluted calculations distributed over multiple source code files. This was a particularly challenging example of a general problem: as the code was not executable due to its age, it was not possible to simply output the final values of variables, or examine the parameters and dynamic variables of the neurons and synapses. Instead, calculations had to be painstakingly reconstructed by analyzing the source code and replicating the logic.

As a final example of the nature of the replication challenges, the input scaling parameters in the code are approximately - but not exactly - 1000 times smaller than given in the paper, because the input weight to be scaled is approximately - but not exactly - 1000 times bigger than reported. In contrast to the weights inside the network, which are defined in the code as amplitudes of post-synaptic potentials in the same way they are given in the paper, the input weights are defined as a post-synaptic current of 30 nA. This results in the following PSP-amplitude:

$$PSP_{\mathrm{input}} = \frac{PSC_{\mathrm{input}}}{g_{\mathrm{L}}} = 1.924779612734342\ \mathrm{V} \tag{2.31}$$

instead of the reported 1.9 mV. However, the combination of these differences results in a scaled input weight with the same order of magnitude as the one reported in the paper.

These hurdles to replicating the paper provide a good demonstration of the argument presented in Pauli et al. (2018): whereas provision of source code is the absolute minimum requirement for replicating a study in computational neuroscience, the process is rendered much simpler if appropriate care is taken with the code implementation, e.g. writing modular, encapsulated, well-commented code with separation of parameters and program logic. Moreover, many of the hurdles we encountered would have been substantially reduced if the code had been executable. To foster reproducibility, we therefore recommend - again following Pauli et al. (2018) - that models should not be expressed in homebrewed code, as this is unlikely to be maintained. Instead, developing the model using a simulator that is actively developed by a community reduces the maintenance

load and increases the likelihood that the model will remain accessible, executable, and part of the scientific discourse for years to come.

2.5 Conclusion

In this chapter, we have reproduced the cortical column model of Häusler and Maass (2007) and their experiments with it. By doing so, we provide the scientific community with a reusable and customizable model implementation that can be leveraged for further experiments to investigate the computations performed by such neural microcircuits. The model consists of noisy Hodgkin-Huxley neurons connected by dynamic synapses, whose connectivity scheme is based on empirical findings from intracellular recordings. Our analysis confirms the key original finding that the specific, data-based connectivity structure enhances the computational performance compared to a variety of alternatively structured control circuits. For this comparison, we use tasks based on spike patterns and rates that require the systems not only to have simple classification capabilities but also to retain information over time and to be able to compute nonlinear functions. Going beyond the scope of the original study, we demonstrate that this finding is independent of the complexity of the neuron model, which further strengthens the argument that it is the connectivity that is crucial. Finally, a detailed analysis of the memory capabilities of the circuits reveals a stereotypical memory profile common across all circuit variants. Notably, the circuit with laminar structure does not retain a stimulus any longer than any other circuit type. We therefore conclude that the model's computational advantage lies in a sharper representation of the stimuli.

Chapter 3

Information Processing Capacity

In the previous chapter, we implemented and analyzed a cortical microcircuit model and control models derived from it. However, we based this analysis purely on the evaluation of computational tasks. One metric that enables a much deeper analysis that reveals in detail the functions dynamical systems compute is the information processing capacity. This method not only provides us with information about the complexity of a system's computations in an interpretable form but also indicates its different processing modes with different requirements on memory and nonlinearity. In the following chapter, we provide a guideline for adapting the application of this metric to continuous-time systems in general and spiking neural networks as well as the microcircuit model of Chapter 2 in particular.

3.1 Introduction

Evaluating the functional capabilities of input-driven systems is crucial for the study of neural and neuromorphic systems alike. Specifically, it is vital to compare systems quantitatively, to uncover relationships between computations and structural or dynamical features, and to optimize their performance. For this purpose, benchmarking a system's performance on standard tasks is illustrative but insufficiently informative, as it cannot give a comprehensive profile of the functions conducted.

Therefore, in this chapter, we investigate a system-agnostic metric to quantify the capacity of a dynamical system to perform arbitrary transformations on an input, i.e. its information processing capacity (IPC). Originally proposed in Dambre et al. (2012), we extend the notion of passive fading memory (Jaeger, 2001a) to nonlinear transformations of increasing degrees of complexity. In the original formulation, the evaluation of the processing capacity is based on discrete inputs and discrete system states. For continuous-time systems we expand discrete inputs to continuous values and compress sections of continuous system states into discrete steps. Based on this, we study the effects the duration of these sections has on the capacity functions computed by the system. Since we need to impose few restrictions on systems when evaluating this metric, it is ideal for analyzing biology-inspired systems, such as spiking neural networks.

We apply the method to systems of gradually increasing complexity and investigate how different parameterizations of the signal encoding influence the outcomes. First, we analyse the time-discrete echo state network (ESN), before we consider the continuous-time Fermi-Pasta-Ulam-Tsingou (FPUT) oscillator chain. With a balanced random network (BRN), we take the step to spiking neural networks (SNNs), in order to consider a more complex representative of these systems in the next step with a model of a cortical column.

Since there is no generally accepted way to pass real numbers to SNNs, we evaluate different methods to achieve this. The signal can be represented by rates of spike trains or by changes in an injected current, it can be spatially encoded or given to all neurons simultaneously, either as a fixed value or multiplied by randomly drawn weights.

Using a direct current signal is motivated by the fact that it represents a simple and precise way of input. However, this mechanism is biologically implausible, since a large part of the communication in the brain takes place via sequences of spikes. To stay close to the biology of the brain and thus increase the probability of revealing insights into its function, it is desirable to rely on signal transmission via spikes. The reasoning behind the encoding methods is similar. The simplest option is to apply the same signal to every neuron. If this input causes too little variability in the network, the next possibility is to weight the inputs to the neurons unequally. However, methods, in which all neurons are stimulated simultaneously, stand in contrast to the biological implementation, in which in most cases only a small part of the neurons is activated. Therefore, from a biological point of view, spatial encoding, which always excites only a small part of the network, is preferable to the other methods.

SNNs commonly receive spike trains as background input to bring them to a suitable working point in terms of firing activity and spiking statistics. We need this background activity because we simulate networks that represent a very small part of the brain and cannot be considered as completely independent components. They receive spikes from their environment, which we approximate by random Poisson spike trains. This background noise reduces the processing capacity (Dambre et al., 2012), as it introduces an additional input that is not taken into account when evaluating the computed transformations of the main inputs. Therefore, we investigate ways to reach an appropriate working point without introducing an untreated input signal.

The information processing capacity (Dambre et al., 2012) is based on linear encoded inputs. Any nonlinearity in the input encoding distorts the capacity profile and complicates its interpretability. However, there are reasons, such as the desire for biological plausibility, to analyze the system using nonlinear inputs. For these cases, we provide a method to remove the effects of such encodings and provide a lower bound on the actual processing capacity of the dynamical system. This extraction of the encoder capacity allows to investigate parts of complex dynamical systems separately by considering all signals entering the subsystem as encoded by the rest of the system. For example, computations of parts of a comprehensive brain model could be studied separately without

isolating them from the overall system.

As a last step, we show that the capacity profile correlates with the performances on tasks of different complexity and with different memory requirements and thereby emphasize the usefulness of this metric in the characterization of dynamical systems.

3.2 Methods

3.2.1 Information processing capacity

The empirical estimation of the information processing capacity of a dynamical system quantifies the different processing modes it can employ by determining the number of linearly independent functions of its input that the system can compute. Figure 3.1 shows a schematic representation of the metric. The dynamical system under investigation is passively driven by a time-dependent input signal $u(k)$ of finite total length T. In this work the discrete values of $u(k)$ are independent and identically drawn (i.i.d.) from a uniform distribution over the interval [-1, +1]. The aim of the analysis is to quantify the system's ability to carry out computations on u. For that purpose, we gather the system's states in response to the input $x[u(k)]$, train linear estimators using the Moore-Penrose pseudoinverse to reconstruct a set of L target functions y_l for $l = 1, ..., L$ from these states and evaluate how well the system is able to reconstruct each function by calculating the squared correlation coefficient between the target functions y_l and the reconstructed functions z_l:

$$C_l = \frac{\mathrm{cov}(y_l, z_l)^2}{\mathrm{var}(y_l) \cdot \mathrm{var}(z_l)} \tag{3.1}$$

The sum of the single capacities C_l over all target functions is the total processing capacity of the system.

We choose the target functions to be orthogonal to the input distribution. This orthogonality of the target functions ensures that each independent measurement reflects an independent transformation that the system can perform. Furthermore, we compute target functions not only based on $u(k)$, but also on time-delayed versions $u(k-i), \forall i \in [0, k]$, in order to measure the amount of linear and nonlinear memory the system can maintain. The basic components of the orthogonal target functions that we use in this work are Legendre polynomials, which are defined by

$$\mathcal{P}_d(s) = \frac{1}{2^d} \sum_{j=0}^{d} \left(\tfrac{d}{j}\right)^2 (s-1)^{d-j} (s+1)^j \tag{3.2}$$

where d is the degree of the polynomial. Each unique target function is composed by selecting a degree d_i for each considered delay i and building the product of the thereby specified polynomials:

$$y_l = \prod_i \mathcal{P}_{d_i}(u[k-i])) \tag{3.3}$$

where $\mathcal{P}_{d_i}$ are the Legendre polynomials of degree d_i. Thus the degree tuple $D_l = (d_{l,0}, d_{l,1}, d_{l,2}, \ldots, d_{l,m})$ completely defines a target function y_l having a maximum delay $i = m$. The sum of all degrees in D_l determines the total degree of the corresponding target function y_l. For example, the degree tuple $(2, 1, 0, 4)$ corresponds to the target function:

$$y = \mathcal{P}_2(u[k]) \cdot \mathcal{P}_1(u[k-1]) \cdot \mathcal{P}_4(u[k-3]) \tag{3.4}$$

which has a total degree of 7. Note that we do not explicitly include the polynomial for delay $i = 2$, because its degree is 0 and therefore corresponds to the constant polynomial $\mathcal{P}_0 = 1$.

The total degree of a target function specifies the complexity of the required computation (nonlinearity), whereas the delays i specify the memory requirements of the investigated system. For further details, see Dambre et al. (2012) and Duarte and Morrison (2019).

Since the number of functions in the capacity space is infinite and the computational resources are finite, we cannot evaluate the capacities for all basis functions. Moreover, we do not know in advance which functions the dynamical systems can compute, and therefore we use an exploration strategy to guide our search for nonzero capacities in the space of target functions, rather than determining in advance which ones we will evaluate. The basis of this strategy is the assumption that capacities decrease with the complexity of the target functions and the memory required to compute them. We increase the degree and delay separately and stop computing more functions of higher degree or longer delays when the capacity falls below a threshold whose value is on the order of $O(\frac{N}{T})$, where N is the number of readout states per input value. For more information on the calculation of this threshold, see the supplementary material of Dambre et al. (2012).

Input encoding for spiking networks

Different dynamical systems may impose very different (physical) constraints on how a stimulus is encoded. To pass the signal $u(k)$ to continuous time systems, we thus need to encode this input appropriately depending on the system's specifications. To do so, we first define which sub-set of units are to receive input, i.e. we specify the density of input connections as a fixed connection probability p (see below for model-specific definitions of the set of potential input units). This results in a random subset of $N_{\text{inp}} = pN$ units that are effectively input-driven. We then define a continuous version of the discrete signal $u(k)$ in a similar way as in Appeltant et al. (2011) and Röhm and Lüdge (2018), i.e.

$$u(t) = u(k) \text{ for } (k-1) \cdot \Delta s \leq t < k \cdot \Delta s \tag{3.5}$$

where Δs is the stimulus step size and $0 \leq t < T\Delta s$, see Figure 3.1A.

This continuous, time-varying signal is delivered to the system following three different types of encoding schemes. These were chosen due to the constraints of the systems under investigation, but are general enough to cover a wide variety of possible encoding mechanisms in input-driven, continuous, dynamical systems. For the *amplitude-value* scheme (Figure 3.1B), the value of $u(t)$ is encoded by amplitude, i.e. $u(t)$ to values in the range $[0, a_{\max}]$ resulting in the scaled amplitude signal:

$$a(t) = \frac{(u(t)+1)}{2} \cdot a_{\max} \tag{3.6}$$

This amplitude signal $a(t)$ is also the basis for the *distributed-value* scheme illustrated in Figure 3.1C. Here, encoding variability is introduced by drawing the input weights from a uniform distribution in $[-1, 1]$, i.e. $a(t)$ gets multiplied by a randomly drawn, target-specific weight w_j such that each target neuron j receives a slightly different input.

$$a_j(t) = a(t) \cdot w_j \tag{3.7}$$

The *spatial-value* scheme (Figure 3.1D) assumes the input can be spatially encoded, i.e. the target neurons have amplitude-specific receptive fields. The scheme thus considers different subsets of target neurons are responsive to stimuli of different amplitudes. The value of $u(t)$ is considered the centre of a Gaussian profile which determines which units in $[1, \ldots, N_{\text{inp}}]$ receive the input. This encoding scheme is illustrated in the bottom panel of Figure 3.1. Under this encoding scheme, each input neuron j receives an input of amplitude:

$$a_j(t) = a_{\max} \frac{\exp(-j^2/2)}{\sigma\sqrt{2\pi}} \left(\frac{j - \mu(t)}{\sigma} \right) \tag{3.8}$$

where $\mu(t)$ is the distribution's mean

$$\mu(t) = \frac{(u(t)+1)}{2} \cdot N_{\text{inp}} \tag{3.9}$$

and the standard deviation σ is an experimental parameter, which determines the spatial spread (and overlap) of the input encoding. Note that we do not use periodic boundaries, in order to be able to distinguish the encoding of $u(k) = 1$ and $u(k) = -1$.

Note that for this encoding scheme to target a sufficiently large subset of neurons, we clip the input connection density to $p = 1$.

The continuous activity signals $a_{\text{j}}(t)$ are converted into input for spiking neural networks in two ways (Duarte et al., 2018). In the first method, we interpret $a_{\max}$ as a maximum current, and supply each neuron with a corresponding scaled direct current $a_j(t)$. In the second method, $a_{\max}$ is interpreted as the maximum firing rate of a Poisson generator. Thus, $a(t)$ is converted into the piece-wise rates of an inhomogeneous Poisson generator providing independent spiking input to the input neurons.

3.2.2 Tasks

We let the dynamical systems solve various tasks and encode the required inputs as close as possible to the way we described for the capacity in the previous section. Continuous inputs (see NARMA task in Section 3.2.2) are encoded exactly like the inputs for the capacity calculations. For multidimensional binary inputs, we need to adjust the encoding. In distributed-value encoding, n-dimensional inputs are split into n input streams that are separately connected to the system (each with its own connectivity matrix). The value of the activated input streams is set to $a_{\max}$ and the value of the inactivated inputs is set to 0. In spatial-encoding, each active input is encoded by a Gaussian profile (see Equation 3.8) with a fixed value for its mean. The mean values of the n inputs are evenly distributed over the input space and inactive streams do not cause any activation of the system.

XOR task variants

As representatives of tasks requiring nonlinear computations, we use exclusive-or (XOR) and variations thereof. XOR is based on two binary inputs and results in 1 if the inputs are different and in 0 if they are not. A more complex version is the nested calculation of XOR tasks (XORXOR). Unlike the normal XOR task, we need four input signals to perform this task:

$$\mathrm{XORXOR}(\mathrm{inp}_1, \mathrm{inp}_2, \mathrm{inp}_3, \mathrm{inp}_4) = \mathrm{XOR}(\mathrm{XOR}(\mathrm{inp}_1, \mathrm{inp}_2), \mathrm{XOR}(\mathrm{inp}_3, \mathrm{inp}_4)) \tag{3.10}$$

The temporal exclusive-or (tXOR) task is based on a single input stream and requires additional memory. We define tXOR as the XOR of the current input and the input of the previous time step. We evaluate all XOR variants with Cohen's kappa score.

Delayed classification

In this task, we use ten different input signals, only one of which is active in each step. We then evaluate for how many delay steps in the past a system can identify above chance level which of these ten one-hot encoded signals was active.

NARMA time series

The nonlinear autoregressive moving average (NARMA) task tests systems for both nonlinear computation and memory, and its evolution is described by the following equation:

$$y(t+1) = \alpha y(t) + \beta y(t-1) \left(\sum_{i=0}^{n-1} y(t-i) \right) + \gamma u(t-n+1) u(t) + \epsilon \tag{3.11}$$

We use $(\alpha, \beta, \gamma, \epsilon) = (0.2, 0.004, 1.5, 0.001)$ and time lag $n = 5$ as parameter values and evaluate the results with the squared correlation coefficient.

3.2.3 Investigated models

Echo state network model

The echo state network (ESN), which Dambre et al. (2012) also used as an example, is a simple, discrete-time recurrent neural network. The evolution of its $N = 50$ state variables x_i is given by:

$$x_i(k+1) = \tanh(\rho \sum_{j=1}^{N} w_{ij} x_j(k) + \iota \nu_i u(k)) \tag{3.12}$$

where ρ is the feedback gain that scales the recurrent weights w_{ij} from unit j to unit i, and ι is the input gain that scales the weights ν_i from input $u(k)$ to unit i. We initialize both the recurrent weight matrix and the input weights with values drawn from a uniform distribution between -1 and 1. In addition, we orthogonalize w and scale it to unit spectral radius. To calculate the processing capacity, and given that this system is used as a baseline, we consider all N units receive the input directly, without further scaling or input encoding, as shown in Equation 3.12. For the capacity experiments in this work we use $T = 100,000$ input steps.

Fermi–Pasta–Ulam–Tsingou model (FPUT)

The Fermi-Pasta-Ulam-Tsingou (FPUT) model describes a one-dimensional string of coupled oscillators and was originally studied to investigate the equipartition of energy among its degrees of freedom (Fermi et al., 1955). As linear forces between neighboring oscillators lead to an analytically solvable system, several nonlinear interactions were studied in the original paper. For our purposes, it is sufficient to consider quadratic nonlinearities, as they appear in the α-FPUT model.

In contrast to the original paper, however, we are not interested in a closed system, but an input-forced system. We drive the system via an external time-dependant field $a(t)$ coupling to all oscillators equally. Different inputs are then represented via different values of a, whereby we ensure $\langle a(t) \rangle_t = 0$, i.e. we consider an amplitude-value encoding scheme and a dense input connectivity, $p = 1$. To deal with the resulting energy put into the system we also introduce a dampening term with decay constant τ.

These considerations lead to the following differential equation for the oscillators

$$\ddot{x}_i = (x_{i+1} + x_{i-1} - 2x_i) + \alpha((x_{i+1} - x_i)^2 - (x_{i-1} - x_i)^2) - \frac{\dot{x}_i}{\tau} - a(t), \tag{3.13}$$

where x_i is the deflection of oscillator i from the rest position. For the experiments we used $\alpha = 0.25$, $\tau = 10$ and $T = 100,000$ input steps.

Balanced random network model

The simpler spiking neural network we use in this study is a balanced random network (BRN) (Brunel, 2000) consisting of $N = 1250$ leaky integrate-and-fire neurons, sub-divided into a population E of $N_{\text{exc}} = 1000$ excitatory neurons and a population I of $N_{\text{inh}} = 250$ inhibitory neurons. The connections between these neurons are crucial for the functionality and dynamics of the network. Excitatory neurons propagate activation to other neurons, while inhibitory neurons function to regulate this activation, preventing potential runaway excitation within the network. The interplay between these two types of neurons contributes to the balance that allows the network to perform complex computations while maintaining stability. After the initialisation of the membrane potential V_{m} with values drawn from a uniform distribution between V_{min} and V_{max}, the neuron dynamics follows

$$\tau_{\text{m}} \frac{dV_{\text{m}}}{dt} = -(V_{\text{m}} - E_{\text{L}}) + \frac{\tau_{\text{m}}}{C_{\text{m}}} I(t) \tag{3.14}$$

where τ_{m} is the membrane time constant, C_{m} is the membrane capacitance, E_{L} is the resting membrane potential and $I(t)$ is the synaptic current. Synaptic transmission in this model is considered to elicit a delta-shaped post-synaptic current:

$$I(t) = \Sigma w_{ij} \delta(t - t_j^{\text{sp}} + d) \tag{3.15}$$

where t_j^{sp} is the time at which neuron j spikes and d is the synaptic delay. When the membrane potential reaches a fixed threshold V_{th}, it is set back to the reset potential V_{reset} and stays at this value for a refractory period τ_{ref}. Recurrent connections among excitatory and inhibitory populations are established to maintain fixed in-degrees of C_{exc} (excitatory synapses) and C_{inh} (inhibitory synapses). To compensate for the effect of the larger excitatory population, the weight w_{inh} of inhibitory synapses is g times stronger than the excitatory one w_{exc}.

In order to obtain responsive networks that operate in biologically meaningful regimes (see Brunel (2000)), an additional background input is necessary. Besides the input signal $u(t)$ described in Section 3.2.1 that we potentially give to all excitatory neurons (based on connection probability p), every neuron in this model is driven by time-varying spikes that are connected with the same weight w_{exc} as excitatory recurrent synapses and exhibit a firing rate ν_{noise}. We use two different methods to generate these background spike trains. In the changing noise version, we use random Poisson spike trains whose spike times vary for each input step. Since we must consider each source of randomness that varies for each step as an additional input to the system that reduces the maximum possible processing capacity, in the frozen noise experiments we generate a single Poisson spike pattern of length Δs per neuron and repeat it for each step. As readout values for the capacity calculation, we use the membrane potentials of all 1000 excitatory neurons

to represent the state of the system after each of the $T = 200,000$ inputs. Table B.1 in the appendix lists all parameter values for this network.

Cortical microcircuit model

The more complex spiking neural network used in our analysis is the cortical microcircuit model introduced by Häusler and Maass (2007) and subjected to additional investigation in Chapter 2 In the first experiments in Chapter 2, the network consists of conductance-based Hodgkin-Huxley neurons equipped with an additional intrinsic noise mechanism. However, we have shown that even significantly simplified integrate-and-fire neurons do not harm the computational performance of the network and can even increase it (see Section 2.3.3 and Figure 2.5). We take advantage of this finding and use this simpler neuron model in the current chapter, which gives us the benefit of a significantly reduced simulation time.

We use all neurons that are not part of the inhibitory populations of layers 2/3 and 5 as input neurons and feed the signal described in Section 3.2.1 into these units. The reason for this choice is that in the original model description all but the two mentioned populations are connected to external inputs. Just as with the balanced random network described above, we also drive this system using spike generators that generate either changing Poisson spike trains or frozen noise. In this case, we use two input streams consisting of 40 spike trains connected to the network in the same way, i.e. using the same synaptic weights and connection probabilities, as the two input streams in Chapter 2. Since we have an additional signal driving the activity as described above, we halved the firing rate of the spike trains to 10 spikes per second compared to the original inputs. To calculate the capacity, we use the membrane potentials of all 447 excitatory neurons at the end of each of the $T = 100,000$ inputs.

3.2.4 Capacity chance level and cut-off value

The theory of the information processing capacity is based on inputs of infinite length. Since we have to use finite inputs in our experiments, we must account for a systematic error in the measured capacities. The chance level of the capacity is given by a chi-squared distribution $\chi^2(N)$ with mean $\frac{mN}{T}$ and variance $\frac{2vN}{T^2}$ (for details see Dambre et al. (2012)). m and v are equal to 1 for independent state variables. However, since we cannot assume the independence of the state variables, we do not know the correct values for m and v. To define a suitable threshold c below which we set all capacities to 0, we calculate the value c_{ind} for which the probability $P(\chi^2(N) \leq c_{\text{ind}}) = 10^{-4}$ for $m = 1$ and $v = 1$. We account for the unknown factors m and n by multiplying c_{ind} by a constant factor. For the experiments in this chapter, we use the factor 6:

$$c = 6 \cdot c_{\text{ind}} \tag{3.16}$$

3.3 Results

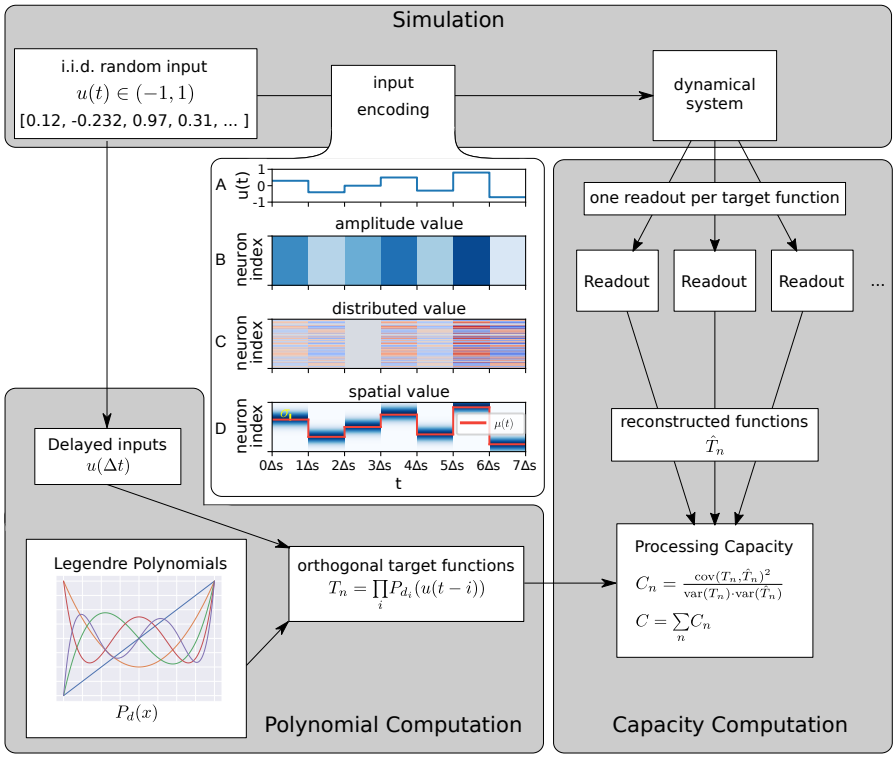


Figure 3.1: Schematic of processing capacity. **Simulation**: real-valued random inputs between -1 and 1 are encoded and fed into the dynamical system. **A**: temporally unfolded signal. **B**: amplitude-value encoding where each neuron receives the same input. **C**: distributed encoding where the encoder is connected to the system by randomly drawn weights. **D**: spatially encoded signal. **Polynomial computation**: Products of Legendre polynomials with delayed inputs are calculated as target functions. **Capacity computation**: the reconstructed functions are evaluated against the target functions using the squared correlation coefficient.

We investigate systems of varying complexity (see Section 3.2.3) to uncover relationships between input and system parameter configurations and their processing capacity. The schematic representation in Figure 3.1 shows how the information processing capacity tests the systems' ability to calculate orthogonal polynomial functions based on different delayed inputs. To provide a comprehensive view of the systems' computations, we look not only at the total capacities of the dynamical systems, but also at their

composition, based on the maximum polynomial degree and maximum input delay (i.e. nonlinearity and memory, see Section 3.2.1).

3.3.1 Discrete time system: Echo state network

Our first case study is a discrete-time recurrent neural network, an *echo state network*. This model was chosen to validate the metric implementation because it is neuro-inspired at a fundamental level and due to the simplicity of the system, it allows fast numerical simulations and, consequently, detailed analyses.

We extend the ESN experiments conducted in Dambre et al. (2012) by performing a comprehensive parameter scan, evaluating the maximum degree and delay in addition to the total capacity, looking at more detailed capacity profiles and comparing the capacity properties with task performance. Figure 3.2 shows the results of the ESN with varying levels of detail and focus on different aspects of the capacity. Panel A shows the total capacity of the ESN as a function of input gain ι and feedback gain ρ (see Section 3.2.3), which scale the input and recurrent weights, respectively. For feedback gain values below 1, the total capacity is close to the maximum of 50 for all ι. The capacity decreases for higher values of ρ and low values of ι. However, the processing capacity metric (see Section 3.2.1) provides more information about the computations performed by the system. First, we consider the complexity of the computations, expressed by the ability to reconstruct polynomials with higher degrees. Although the changes in the left part of the capacity heat map A ($\rho \leq 1$) seem to be minimal, the degree heat map B shows that the ability to perform nonlinear computations increases with ι. This can be accounted for by the shape of the tanh nonlinearity. For input values close to zero, it is almost linear, while higher and lower values reach the saturating nonlinearity of the transfer function. Charts D and E illustrate how increasing values of the input gain ι shift the computational capacity away from linear to increasingly nonlinear computations.

With regards to the delays that are representable by the ESN, i.e. the system's memory, panel C shows that even in the parameter range with low ι and $\rho > 1$, which corresponds to low total capacity, there are notable changes in the computations. Compared to the homogeneous values for the maximum delay at higher input gains, the maximum delay increases dramatically with the feedback gain up to $\rho = 1.13$. Thus, these results demonstrate that the system can operate at two extremes - linear, high-memory processing, and nonlinear, low memory processing - as depicted in F and G, i.e. there is a tradeoff between memory and nonlinearity.

The method allows us to evaluate the system's ability to compute nonlinear functions with different memory requirements. To analyze whether these insights hold for structured instead of random input signals, we evaluate the ability of the systems to solve tasks requiring varying degrees of nonlinearity and memory and compare the results with properties of the capacity profiles. Therefore, we compare the maximum capacity degree with the performance in the nonlinear exclusive-or (XOR) task and test the system on a

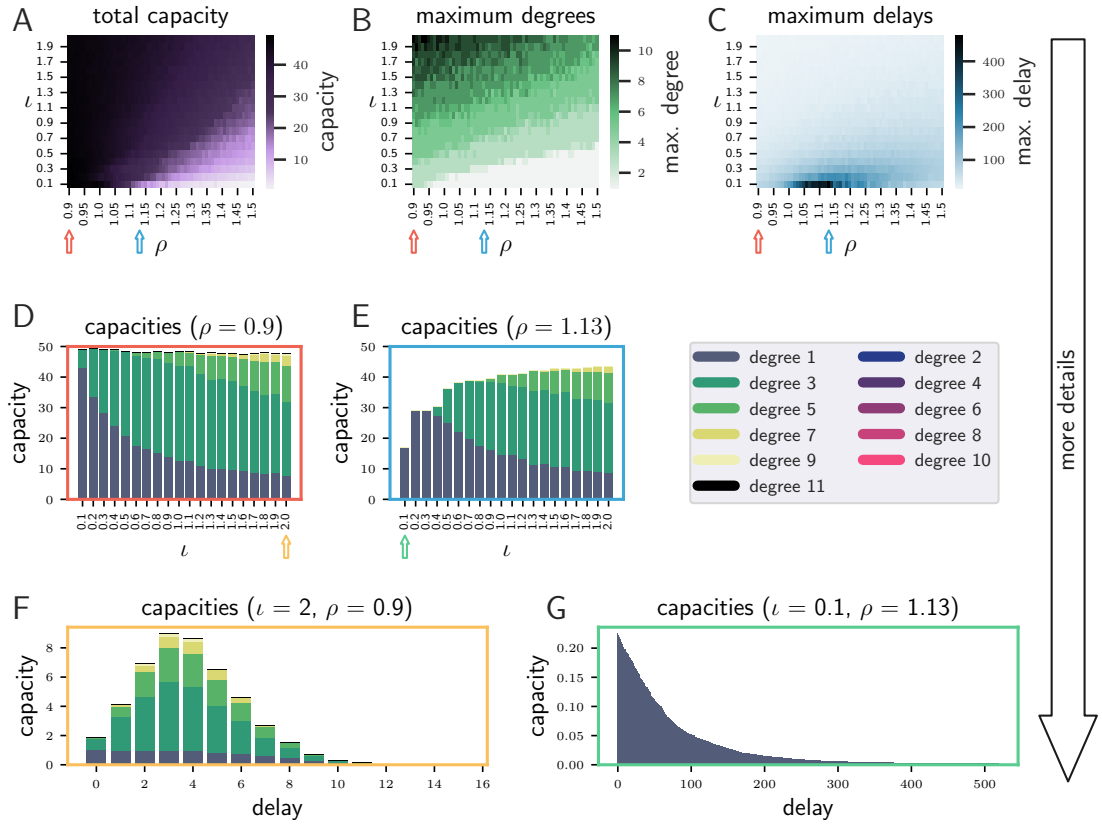


Figure 3.2: ESN results with increasing level of detail from top to bottom. **A-C**: Heatmaps of the total capacity, maximum degree and maximum delay for a parameter scan over ρ and ι and based on all capacity functions (over all degrees and delays). **D** and **E**: Total and degree-specific capacities as a function of ι, for the values of ρ indicated by the correspondingly coloured arrows in **A-C**. Values are summed over all delays for each bar. Note that the ESN only exhibits capacities of odd degree, because of the odd nature of the tanh nonlinearity. **F** and **G**: Capacity profiles showing total and degree-specific capacities as a function of delay for the parameter configurations indicated by the correspondingly coloured arrows in **D,E**

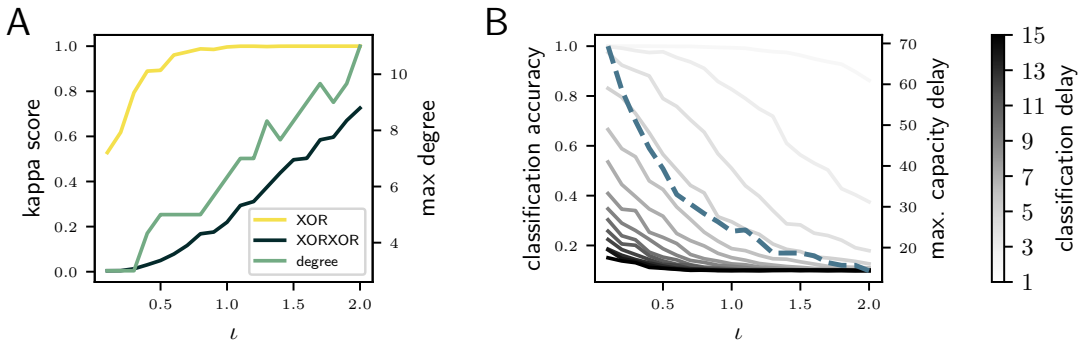


Figure 3.3: Comparison of ESN task results with the processing capacity as a function of ι. **A**: Performance for nonlinear tasks (yellow and blue curves; left y-axis) and the maximum capacity degrees (green curve; right y-axis). **H**: Performance for memory tasks (gray curves; left y-axis) and maximum capacity delay (blue dashed curve; right y-axis).

delayed classification task that requires deciding which of ten streams was active before a given number of input steps. We set ρ to 0.9 and examine the performance of the task with varying ι.

Figure 3.3 visualizes the comparison between the results of the tasks performed by the ESN and the corresponding capacity properties. Panel A shows how the performance of the XOR task (yellow curve) increases along with ι until it saturates near the maximum value of one. Since the performance is saturated for a large portion of the parameter range, we also evaluate the more complex nested exclusive-or (XORXOR) task. The blue curve in A shows that the XORXOR performance of ESNs is also strongly correlated with the maximum degree.

Panel B compares the maximum capacity delay (dashed curve) with the classification performance at different delays (indicated by shades of gray). Increased maximum capacity delay corresponds to an increase in the delayed classification performance. However, the graph shows that the maximum capacity delay is larger than the delay for which the system can still perform the classification. One explanation is that the capacity at the maximum delay is usually small, as shown in Figure 3.2 F and G. In addition, for the classification not only one but ten input streams with different connection weights are fed into the network. Nevertheless, the results show that the capacity delay is a good indicator for the memory in ESNs.

3.3.2 Simple continuous time system: Fermi-Pasta-Ulam-Tsingou model

The previous section demonstrates that the processing capacity of a simple discrete time system, with respect to complexity and memory, depends on the relative strengths of

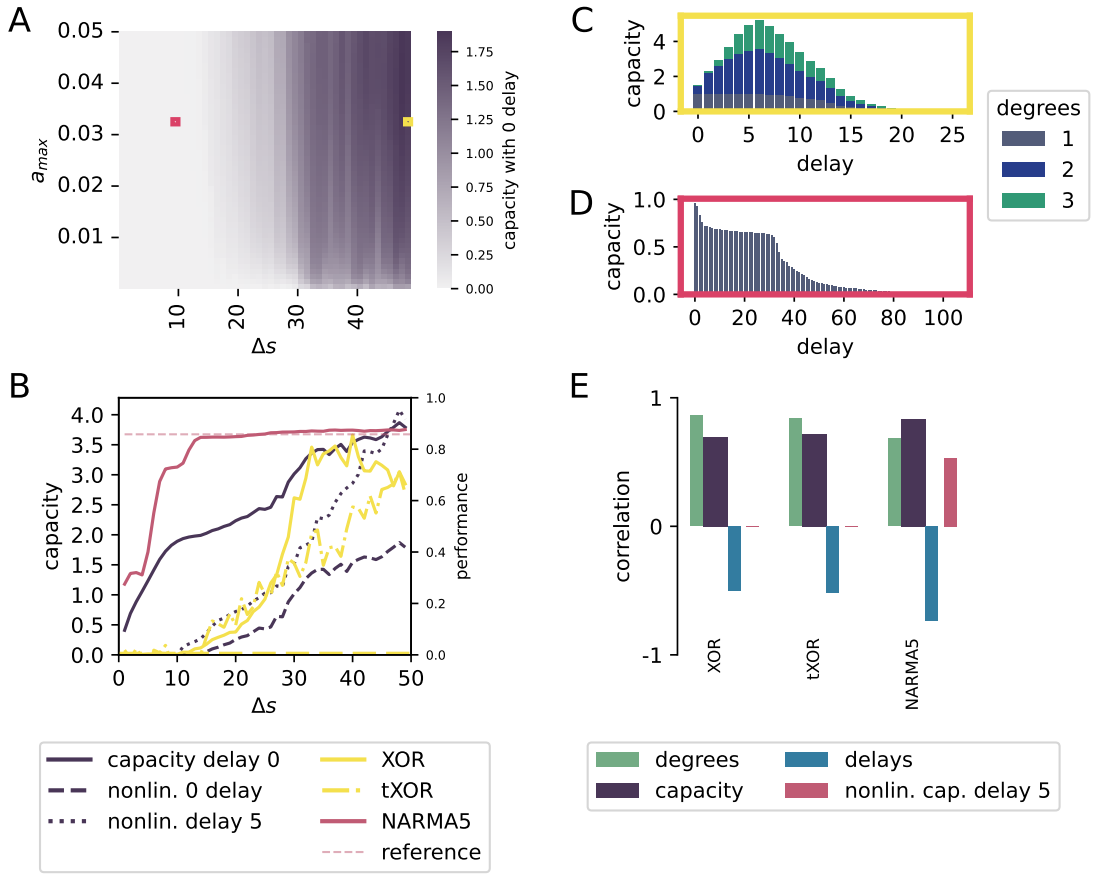


Figure 3.4: FPUT results. **A**: Heatmap of nonlinear capacity with zero delay, where the maximum amplitude a_{max} is a force given in arbitrary units, the input duration is given in units of simulation time steps. **B**: Total nonlinear capacity, nonlinear capacity with zero delay (dashed) and with at least delay 5 (dotted) as well as performance on XOR (yellow), time delayed XOR (yellow, dash-dotted), NARMA5 (maroon) and NARMA5 evaluated on a linear reference system with perfect memory (light maroon, dashed) over input duration for fixed $a_{max} = 0.033$. The performance of both XOR tasks is measured using the Cohen's kappa score and for NARMA it is the squared Pearson's correlation coefficient. **C, D**: Capacity profiles showing total and degree-specific capacities as a function of delay for the parameter configurations indicated by the correspondingly coloured squares in **A**. **E**: Correlations between capacity and task performances for the same parameter ranges of amplitude and duration as shown in **A**.

input and recurrent weights. Here, we extend the analysis to a simple continuous time system, the Fermi-Pasta-Ulam-Tsingou (FPUT) model. We investigate a chain of 64 nonlinearly coupled oscillators; input is applied to all equally. To give discretized input to the chain, we need to decide for what time period we present each input. Panel A of Figure 3.4 illustrates the impact of different input amplitudes and durations on the nonlinear capacity with zero delay. Panel B shows the nonlinear instantaneous capacity for fixed $a_{max} = 0.033$ as a function of Δs, compared to total instantaneous capacity and capacity with delay 5. While the total capacity with delay 0 is already non-zero for small step sizes, the nonlinear capacities both remain zero for small Δs. This is consistent with the system's improved performance in solving XOR and time-delayed XOR tasks, which exceed chance level ($\kappa = 0$) as soon as nonlinear capacities become non-zero.

The NARMA5 performance starts to get close to the performance of a reference implementation of a linear system with perfect memory when the delay 5 capacity becomes non-zero. For large step sizes, the FPUT marginally exceeds the performance of the linear reference system. Interestingly, the nonlinear capacity with delay 5 becomes nonzero for smaller Δs than its instantaneous counterpart. This can be understood via panels C and D, which show capacity profiles resolved by delay, revealing that nonlinear capacities grow with time lag before declining again. Even for large Δs as in Panel C the quadratic capacity at zero time lag is small compared to the one for delays up to 10, and the cubic capacity remains zero. These higher nonlinear capacities come at the cost of losing ability to reconstruct linear signals with large delays, as seen in the long tail of panel D. Finally, panel E shows the correlation between capacities and task performance. Especially interesting is the negative correlation with delayed capacity for all tasks. This can be again explained by the shape of the capacity profiles: the ability to reconstruct signals with larger delays contribute to neither of the XOR tasks, but yield larger capacities. The NARMA5 task is only sensitive to dynamics with a delay of 5, which is also reflected in the capacity.

3.3.3 Balanced spiking neural network model

Spiking neural networks are continuous time systems that communicate via pulses. Applying the random signal needed for measuring the computational capacity to spiking neurons requires extra steps: we encode the signal with an amplitude-value scheme; a distributed-value scheme or with a spatial-value scheme (see Section 3.2.1 and Figure 3.1). We use balanced random network (BRN) consisting of two populations of neurons as an entry model for spiking neural networks. We choose this system because it is one of the simplest SNN architectures, while still providing rich dynamics through the interplay of excitation and inhibition (see e.g. Brunel (2000)).

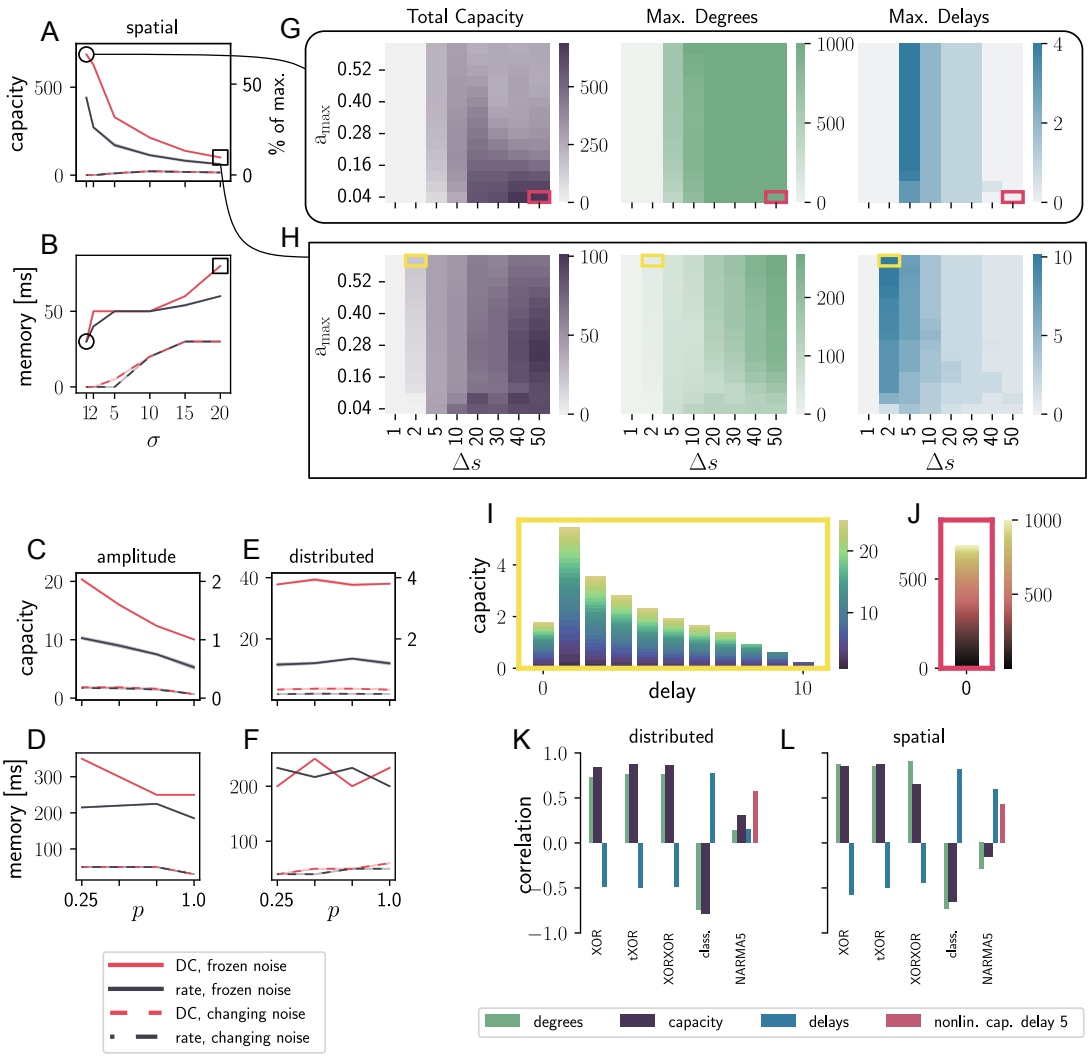


Figure 3.5: BRN results for different encoding schemes. **A,B**: maximum capacity and memory (product of maximum delay and step duration) for the spatial encoding scheme averaged over five trials with different network and input initializations. The standard deviation across trials is too small to be visible. **C,D**: as for **A,B**, but with amplitude encoding. **E,F**: as for **A,B**, but with distributed encoding. **G, H**: Heat maps for total capacity, maximum degree and maximum delay for the parameter scans marked with a circle and a square in A and B. **I, J**: Capacity profiles showing total and degree-specific capacities as functions of delay for the parameter configurations indicated by the magenta and yellow boxes in **G** and **H**. Note the different axis and color scales in these panels. **K-L**: Correlations between capacity statistic and task performances for the same parameter ranges of $a_{\max}$ and Δ_s as shown in **G** and **H**, using DC input and frozen noise; $p = 1$ in **K** and $\sigma = 20$ in **L**.

Encoding schemes distribute memory and nonlinearity

Figure 3.5 visualizes the results of the BRN with different encoding schemes. The different panels focus on distinct properties and details of the processing capacity and its connection to task performance. Panels A-F show the maximum capacity and maximum memory measured for the BRN whilst varying the proportion p of input neurons for the amplitude-value and distributed-value scheme and the standard deviation σ of the spatial-value scheme, respectively. Each data point in A, C and E corresponds to the maximum capacity resulting from a parameter scan over the maximum amplitude $a_{\max}$ and the step duration Δs of the input signal. B, D and F show the same analysis performed on the maximum memory the network can retain, i.e. the maximum number of delays multiplied by Δs.

Comparing the positions of the curves in all six plots, we observe that frozen background noise (solid curves) increases maximum capacity and maximum memory over changing noise (dashed curves), see Section 3.2.3. This is a consequence of the reduced randomness in the system. Likely due to the same cause, the performance is higher for a direct current than a rate input in the frozen noise condition, whereas the input modality plays little role in the changing noise condition. Comparing A, C and E, we conclude that spatial-value encoding results in higher maximum capacities than the amplitude-value and distributed-value schemes. In contrast, the maximum memory (B, D and F) is higher for the amplitude-value and distributed-value encodings.

Within A and C, reducing the number of neurons receiving input in a given input step, i.e. p for amplitude-value encoding and σ for spatial-value encoding, increases the maximum capacity for the tested parameter range. Conversely, a wider distribution across the input neurons is beneficial for the maximum memory when using spatial-value encoding (B) but plays little role for the amplitude-value encoding (D). However, distributed-value encoding does not show comparable trends. The input connection probability p neither significantly changes the total capacity (E) nor the maximum memory (F). The fluctuations in the results are so minor that their standard deviations are too small to be visible in these panels.

In summary, these results suggest that spatial encoding maximizes the system's ability to compute nonlinear functions whereas amplitude-value encoding maximizes memory, while both perform best with frozen background noise, direct input current and a smaller number of input neurons. Motivated by its superior overall capacity, we show a detailed analysis of the spatial-value approach in G-J. The first row of heat maps shows the total capacity, the maximum degree and the maximum delay as a function of the maximum amplitude $a_{\max}$ and step duration Δs, assuming a standard deviation $\sigma = 1$ for the distribution of the input over the neurons, which results in the maximum capacity observed in A (circle). The next row shows the equivalent results, but with a standard deviation $\sigma = 20$, which leads to the maximum memory in B (square).

The capacity heat maps for the two input configurations exhibit their maximum ca-

pacity at the maximum step duration Δs of 50 ms. However, the best values for the maximum amplitude $a_{\max}$ are different for each σ. While the system with $\sigma = 1$ has its maximum at a low input amplitude of 0.04 pA, we need to increase $a_{\max}$ to 0.24 pA for $\sigma = 20$ to reach the maximum capacity. The green heat maps show that longer step durations cause higher maximum degrees. The heat maps displaying the maximum delays show similar trends, i.e. a decrease in the maximum delay as we increase the step duration, which we attribute to the increase of absolute time the network has to maintain the information of a previous step with prolonged step durations.

I and J illustrate capacity profiles for a high capacity parameter set and a high delay parameter set. Together with their positions in G and H (yellow and magenta markers), they show that these networks obtain their total capacity value based on very different compositions of target functions. The yellow framed profile consist of many functions based on delayed inputs, while in J the computations are invariably based on the undelayed signal and consist instead of very high degree nonlinear functions.

Correlation with task performance

To assess whether we can infer information about task performance from processing capacity, we calculate the correlations between task scores and total capacity, maximum delay and maximum degree, respectively. The bar graphs at the bottom of Figure 3.5 show the results for distributed encoding with $p = 1.0$ (K) and spatial-valued encoding with $\sigma = 20.0$ (L). XOR, tXOR and XORXOR (see Section 3.2.2) show high correlations with maximum degree and total capacity for both encoding types, while the correlation with the performance of the XORXOR task and total capacity is lowest for the spatial encoding setup. The maximum delay is positively correlated with the maximum classification delay in both cases. The opposite correlation values for the maximum degree and maximum delay (also observed in the other systems) indicate a trade-off between nonlinear computation and memory.

The NARMA5 task (Atiya and Parlos, 2000) requires both memory and nonlinear computation. The bars for this task in K and L show lower correlations with capacity, delay and degree compared to other tasks. To account for the dependence on the combination of memory and nonlinear computation, we also evaluate the correlation of NARMA5 task performance with nonlinear capacity at delay 5, resulting in a higher correlation for the distributed encoding setup.

Encoding effects

Despite being an important component of signal processing in (bio-) physically meaningful systems, the encoding scheme can bias the capacity estimation. As Panel A of Figure 3.6 illustrates, the effectively measured system includes the effects of the encoder, although we are only interested in isolating the system itself. The main system can have

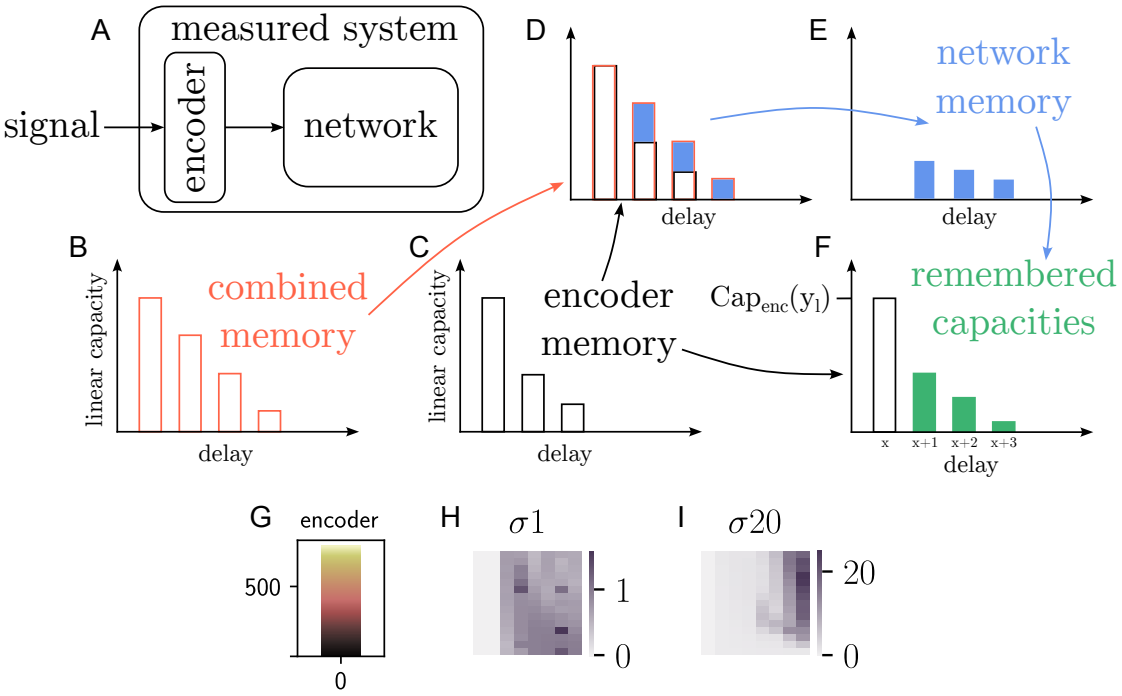


Figure 3.6: Schematic for the calculation of remembered encoder capacities. **A**: Diagram emphasizing that the combination of encoder and network acts as the measured system. **B**: Linear memory of the combined measured system. **C**: Linear memory of the encoder. **D, E**: Difference between **B** and **C** that results in the actual linear memory of the network. **F**: Possibly nonlinear encoder capacity of target function y_l (black border) and its versions remembered by the network (green). **G**: capacity profile for the encoder used in Figure 3.5J. **H-I**: Capacity heat maps as in Figure 3.5G,H, but with all possible encoder capacities subtracted. Note the different scales of the color bars.

nonlinear capacities without calculating nonlinear functions itself by merely remembering the nonlinear or delayed inputs from the encoder, i.e. the encoder can introduce nonlinearity and memory which bias the estimate. One solution are linear encoders without memory. However, when we prefer more complex methods to encode the signal, for example to increase biological plausibility, we want to exclude encoder effects from the measured capacity. Figure 3.6B-F outline a way to calculate the encoder effects that need to be subtracted.

We first calculate the capacity of the encoder output by building the state matrix from the encoded signals that the neurons get as inputs, instead of taking the membrane potentials to evaluate the processing capacity. In the case of the spatial value scheme, this encoder state matrix is built from the Gaussian profiles representing the inputs (see Figure 3.1D for a time-unfolded version of such a state matrix). With these capacities and the capacities of the overall system, we calculate the main system's effective linear memory by subtracting the encoder memory (degree 1 encoder capacities) from the combined memory (based on the membrane potentials) for each delay (B-E). Based on the system and encoder memory values, we calculate the fraction of the encoder input the system can memorize after each delay. Using these fractions, we compute how much of a capacity value for a target function can be based on remembering a target function that is already computed by the encoder. These remembered capacities (F) result from the linear memory of the system and the nonlinearity and memory of the encoder. Therefore we subtract them from the capacities of the combined system to obtain the effective capacity of the main system for all objective functions (see Section B.1 in the appendix).

With this method we cannot get information about the precise functions the system computes as with linearly encoded input, because the system computes the function $F_{\text{sys}}(F_{\text{enc}}(u))$ instead of $F_{\text{sys}}(u)$. Therefore a system capacity $C^{\text{sys}}(y_l) > 0$ does not mean that the system computes the specific target function y_l with the degree d_{y_l}. However, it tells us that the system computes a function that goes beyond remembering the input.

A problem with this procedure is that the capacity values cannot necessarily be subtracted, divided and multiplied, but need to be transformed first to get the correct system capacity. These transformations can be different for every target function and we need additional information to calculate them correctly (see Section B.1 in the appendix). However, we can subtract all encoder capacities and their delayed versions completely, i.e. we fully remove all capacities of target functions that can be partly a result of the network remembering the encoder inputs. This results in a lower limit for the total capacity that can be calculated for any type of encoder and dynamical system. However, it depends on the scientific question whether it makes sense to remove the encoder capacities. For example, if the aim is to compare the information processing capacity with the performance on tasks whose inputs are encoded in the same way, it makes more sense to analyze the capacity of the entire system. If the intention is to examine the calculations of the dynamical system independently of the encoder calculations, then

the encoder capacities should be subtracted. For more details on the procedure and the different ways to remove the encoder capacities, see the corresponding section in the appendix (Section B.1).

Panel G shows that for the spatially encoded signals the encoder alone has higher capacities than the network in Figure 3.5J. H and I show the lower limits for the capacity functions actually calculated by the network. Removing the encoder capacities leaves only a fraction of the original results for $\sigma = 1$. In contrast, networks given a wider input with $\sigma = 20$ compute target functions beyond those remembered from the input. Apart from that, the delays remain unchanged because the spatial value encoding does not introduce any additional memory into the system.

3.3.4 Biophysical spiking network model

We analyze the processing capacity from the biophysically more detailed spiking microcircuit model of Chapter 2 by using two encoding schemes: distributed encoding and spatial-value encoding. This network combines neurobiological properties such as a data-based connectivity and synaptic plasticity in a network size that still allows for extensive simulations and analyses. Figure 3.7 compares the different encoding schemes for the microcircuit model at different levels of processing capacity detail. Panels A-E show the summarized results for the different encoding schemes. There are strong similarities to the BRN, in particular: 1) the use of frozen instead of changing noise increases computational capacity and memory; 2) spatial coding is superior to distributed coding when we do not subtract the encoder effects, but gives lower capacities otherwise; 3) DC input leads to higher capacities in most cases than rate encoded inputs. There is one exception to this last point, namely the spatial-value encoded signals with very short stimulus duration, and this effect disappears when we remove all remembered encoder capacities (C). Although there are some larger standard deviations (shaded areas) than in the BRN, the overall variation in capacity results is still small.

Although the microcircuit is less uniformly structured and includes synaptic short-term dynamics, the maximum total capacity is lower than for the BRN, not only in absolute values (BRN: 686, MC: 49.5), which could be explained by the smaller network size, but also when we consider the normalized capacity values (right axis of A-C; BRN: 68.6 %, MC: 11 %). These values are the fraction of the number of readout neurons, since this is the upper limit for the capacity. Each of these maximum values are based on the spatial encoding scheme. However, if we compare only the normalized capacities for the distributed encoding, they are around 4 % for both networks. Moreover, if we subtract the remembered encoder capacities from the spatial encoding values, the microcircuit (4%, Figure 3.7C) outperforms the BRN (2.5%, Figure 3.5L). In addition, the microcircuit can store information about past inputs longer than the BRN (D and E; BRN: 350 ms, MC: 483 ms). This is likely due to the synaptic plasticity and the biologically inspired connectivity structure, as these features were shown to lead to

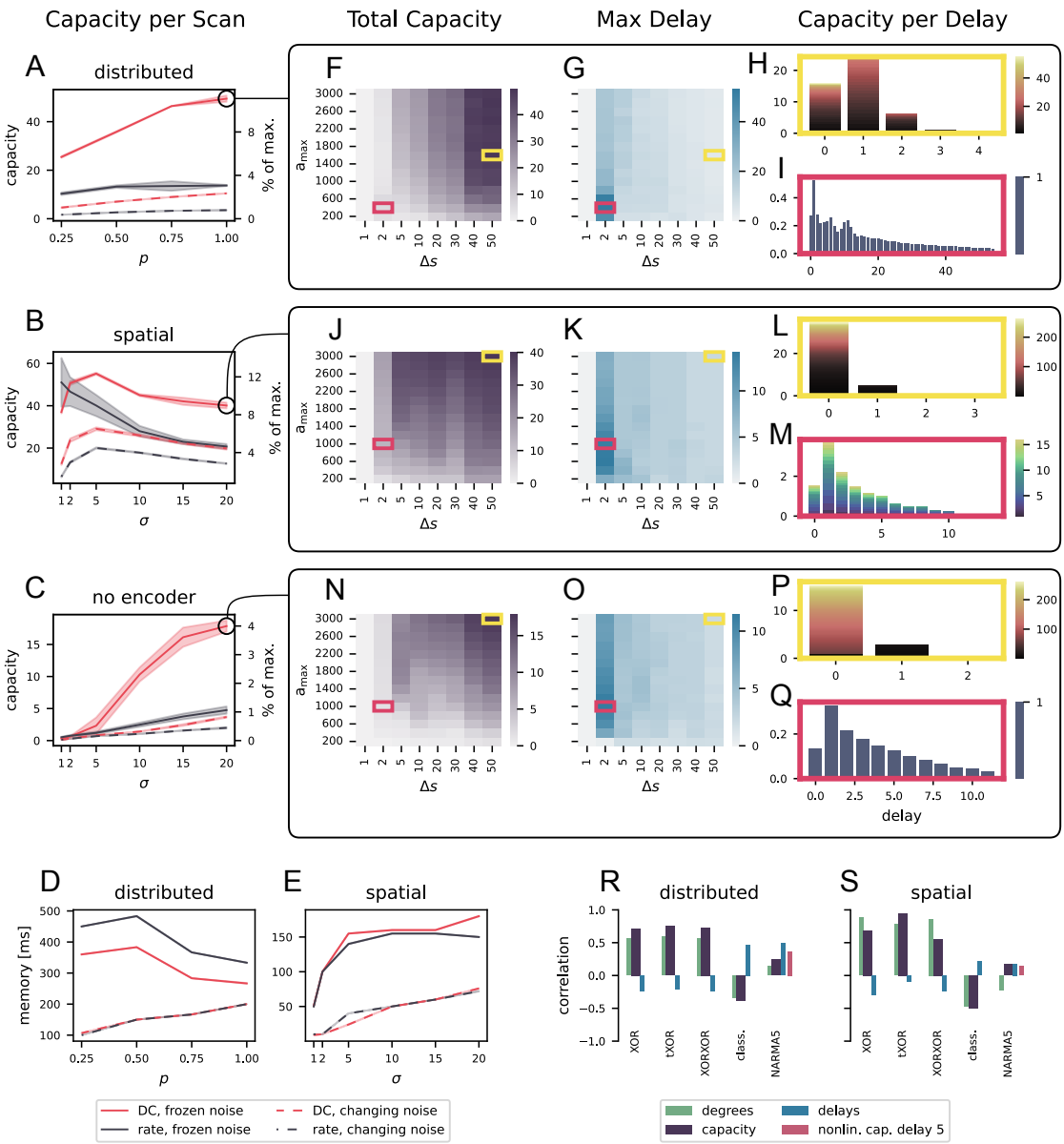


Figure 3.7: Microcircuit results for different encoding schemes. **A-C**: Maximum capacity per input parameter p or σ respectively for distributed encoding (**A**), spatial encoding (**B**) and spatial encoding without remembered encoder capacities (**C**) averaged over five trials with different network and input initialisations. Shaded areas indicate the standard deviation across trials. **D, E**: Maximum memory (product of maximum delay and step duration) for distributed and spatial encoding. **F-Q**: Capacity (**F, J, N**) and delay (**G, K, O**) heat maps of the parameter scans corresponding to the markers in **A-C** together with capacity profiles for a high capacity (yellow) and high delay (magenta) parameter configuration. Note the different axis and color scales in the bar graphs. **R-S**: Correlations between capacity statistic and task performances for the same parameter ranges of $a_{\max}$ and Δ_s used in **F** and **J**, both with DC input and frozen noise; $p = 1$ in **R** and $\sigma = 20$ in **S**.

longer memory in Häusler and Maass (2007) and Chapter 2.

Figure 3.7 F-I, J-M and N-Q give details of the parameter scans for distributed encoding, spatial-value encoding and spatial-value encoding with removed encoder effects, respectively. The heat maps for total capacity and maximum delay show that longer input durations tend to correspond to higher capacities based only on short delays and thus higher degrees, while shorter steps allow longer delays. We show detailed capacity profiles with high degrees but small maximum delays (yellow frames) and lower maximum degrees but longer delays (magenta frames) in the bar graphs on the right of Figure 3.7. The differences between the two calculation modes are clearly visible, but in contrast to the results for the BRN, here even the configurations with high degrees exhibit memory.

As for the BRN, we evaluate the correlation between the different capacity statistics and the performance on tasks. Figure 3.7R-S shows the two configurations marked in A and B. As expected, capacity and degree are positively correlated and the delay is negatively correlated with the performances of the nonlinear tasks (XOR, tXOR, XORXOR). The opposite is true for the memory based task, i.e. delayed classification (class.). Overall, however, the correlations with the spatially encoded tasks (S) are lower for the microcircuit than for the BRN, especially for the memory tasks, and even the nonlinear capacity for the NARMA5 specific delay 5 (dark pink) does not improve the correlations.

3.3.5 Comparative performance on tasks

	ESN ρ 0.9	FPUT α 0.25	BRN distr. p 1	BRN spatial σ 20	MC distr. p 1	MC spatial σ 20
readout units	50	64	1000	1000	447	447
XOR	1.	0.85	1.	0.99	1.	0.99
XORXOR	0.73		0.17	0.62	0.21	0.22
tXOR		0.72	1.	0.99	1.	0.99
classification			26	18	42	50
NARMA5		0.88	0.29	0.2	0.43	0.17
capacity	49	55	38	100	49	40
max. degree	11	3	35	285	71	255
max. delay	69	688	32	10	53	14

Table 3.1: Performance on tasks and capacity measures for different dynamical systems. Given as Cohen's kappa score for XOR, tXOR and XORXOR, maximum delay up to which accuracy is above chance level for delayed classification, and squared correlation coefficient for NARMA5.

Table 3.1 shows the maximum task performances of each system, and its corresponding capacity measures. XOR and tXOR are almost perfectly solvable for the investigated systems except for the FPUT, whereas XORXOR is difficult for all systems. The ESN and the BRN with spatial encoding stand out with significantly higher XORXOR values than the other systems. The microcircuit benefits from its connectivity and short-term plasticity in delayed classification, as its measured delay is significantly higher compared to the BRN. The results of the NARMA5 task in combination with the maximum delays and the maximum degrees show that long memory is more important than nonlinear computations for solving this task. Therefore, especially the FPUT oscillator chain and the microcircuit with distributed encoding perform better than the other models.

3.4 Conclusion

In order to examine in greater depth the microcircuit model considered in Chapter 2 and also spiking neural networks in general, in this chapter we have adapted the application of the information processing capacity which was previously used mainly for simpler discrete-time dynamical systems. In contrast to the analysis based on computational task evaluations, the IPC offers a detailed profile of functions that are calculated by the input-driven dynamical system under investigation and require memory in addition to nonlinear processing. To obtain a capacity profile for SNN that is as meaningful as possible, we first looked at various dynamical systems with gradually increasing complexity. In particular, we analyzed different ways of encoding the input signal and their effects on the information processing capacity. Depending on these different encoding mechanisms, considerably different information processing was observed in the analyzed systems, which revealed a trade-off between non-linear processing and the memory of the corresponding system. Also due to this strong influence of the encoding mechanism used, especially when it already performed non-linear transformations, and because the encoder must therefore also be regarded as part of the system under investigation, we have presented a method for extracting the polynomial functions calculated by the encoder. Thus, the processing performed purely by the main system can be analyzed. In further experiments, we were able to show that the total capacity and important markers such as the maximum delay and the maximum degree correlate with the performance in different tasks, suggesting that the computational profile determined by the IPC provides insight into what is generally seen as information retention and nonlinear processing. Thus, in this chapter, we have created a comprehensive guide for the application of the information processing capacity to spiking neural networks and can now build on this to analyze further network models.

Chapter 4

Memory prerequisites for temporal difference learning in cortico-striatal populations

In the previous chapter, we prepared the information processing capacity for its utilization on spiking neural networks and also applied it to the microcircuit model from Chapter 2. In the following chapter, we use the findings from these two chapters to examine the memory capabilities of two cortical populations to see if they can form the basis for the computation of a temporal difference (TD) error in the brain.

4.1 Introduction

It is assumed that dopamine neurons encode a temporal difference error in the brain and thus play an important role in learning by trial and error (see paragraph 1.2.5 and paragraph 1.2.5). Many of the theoretical models presented (Doya, 2002; Joel, Niv, and Ruppin, 2002; Kawato and Samejima, 2007; Wörgötter and Porr, 2005) mainly consider the processes that take place in the basal ganglia and largely ignore the upstream cortical circuits. Morita et al. (2012) have therefore proposed a mechanism for computing a TD error that takes the encoding of the current and previous state in two separate cortical neuron populations from layer 5 as the basis. These populations are the crossed corticostriatal (CCS) cells and the corticopontine (CPn) cells (also called pyramidal tract (PT) cells).

Experiments by Morishima and Kawaguchi (2006) and Morishima et al. (2011) have shown that these populations have characteristically different properties that should ensure that they maintain input information in their activity over different periods of time. A subset of the CCS cells is supposed to encode the current state at each time step and transmit it to the corresponding subset of CPn cells via unidirectional connections. However, the CCS cells can only maintain this information for a short time because, in addition to their fast spike frequency adaptation, their recurrent synaptic connections are weaker compared to those of the CPn population and they exhibit lower reciprocity and mostly short-term depression. In contrast, the activity of CPn neurons

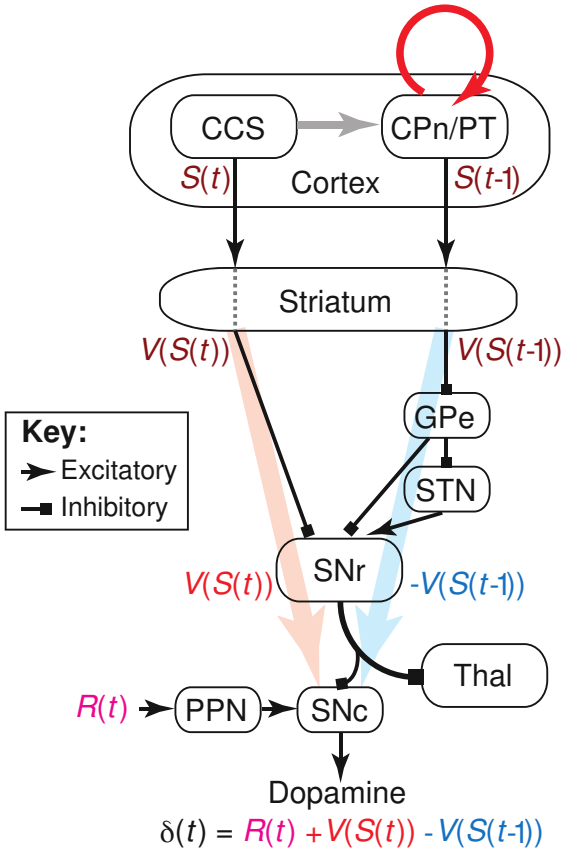


Figure 4.1: Hypothetical mechanism of the computation of TD error in the brain. The activity of the CCS cells encodes the current state $S(t)$ and unidirectional connections propagate the information to CPn cells, where it is preserved longer such that their activity represents the previous state $S(t-1)$. The state values of the current and previous state are calculated through connections to the striatum. The value of the current state $V(S(t))$ is transported via the direct pathway (red arrow) to the substantia nigra pars reticulata (SNr). The previous state value $V(S(t-1))$ is transported over the indirect pathway (blue arrow) via the external segment of the globus pallidus (GPe) and the subthalamic nucleus (STN) to the SNr. The SNr passes this information on to the SNc. The $V(S(t))$ activity causes a positive modulation on the SNc by disinhibition and the $V(S(t-1))$ activity causes a net negative modulation by triple inhibition. Together with a reward signal coming from the pedunculopontine tegmental nucleus (PPN), the SNc computes the TD error. Figure adapted from Morita et al. (2012).

shows non-adaptive repetitive firing and recurrent connections within the population are significantly stronger, exhibit increased reciprocity and show short-term facilitation. These properties suggest that the activity of the CPn population can be maintained over a longer period of time by recurrent synaptic reverberations and thus can potentially store and transmit previous state information received from the CCS population. In addition to this representation of the current state $S(t)$ and the previous state $S(t-1)$ by these two populations, Figure 4.1 also shows how the further calculation of the temporal difference error is carried out according to Morita et al. (2012). Through the corticostriatal projection of the CCS neurons to medium spiny neurons, the instantaneous state value is calculated in the striatum and then passed on to the substantia nigra pars reticulata (SNr) via the direct pathway through the substantia nigra pars compacta (SNc). Due to the inhibitory connections between striatum and SNr and also between SNr and SNc, the direct pathway has a net positive effect on the activity of the dopamine neurons in the SNc. On the other hand, the previous state transferred from the CCS population to the CPn neurons and sustained there reaches the SNc neurons via the indirect pathway. Here, the state value calculated in the striatum has a net negative effect on the activity in the SNc through the triple inhibitory connection via the external segment of the external segment of the globus pallidus (GPe) and the SNr. In combination with the reward signal represented by a subset of the pedunculopontine tegmental nucleus (PPN), the dopamine neurons calculate the TD error.

In this chapter, we investigate whether the basic assumptions of this hypothesis hold: (1) The activity of CCS cells can represent the current state, (2) the state information is transmitted from the CCS population to the CPn population and (3) CPn neurons can retain information about previous states. We first analyze this using network models consisting of continuous rate neurons and then move on to spiking networks. As in Chapter 2, we first create data-based networks and then compare them with modified control circuits in order to investigate the effects of individual network properties. We apply the linear part of the information processing capacity presented in Chapter 3 to analyze the memory capabilities of the CCS and CPn populations of the respective networks.

4.2 Methods

4.2.1 Baseline rate-based model

In the first set of experiments, we analyze a continuous-variable firing rate model of a network of $N = 4,000$ units. The dynamics of the firing rate r are given by:

$$\tau \frac{dr}{dt} = -r + f(W^{\text{rec}} r + g(u) + b^{\text{rec}}) \tag{4.1}$$

where τ is the decay time constant of the firing rate, W^{rec} is the matrix of synaptic weights of the recurrent connections, and the function g encodes the external input u before it is given to the input neurons. We use the *spatial-value* encoding scheme that is described in Section 3.2.1. The function f is the nonlinear activation function of the units and in this model.

In the first experiments, we test the effect of different activation functions on the memory capabilities of the network. Besides the standard tanh function that also exhibits biologically non-plausible negative rates, we adjust this function to be always positive by shifting it one unit to the top (tanh+1). Then we scale this shifted tanh function to values between zero and one:

$$\text{scaled-tanh}(x) = 0.5 \cdot (\tanh(x) + 1) \tag{4.2}$$

The last tanh-based function is the rectified-tanh function. Here we take the tanh function and replace all negative outputs with zero:

$$\text{rectified-tanh}(x) = \begin{cases} \tanh(x) & \text{if } \tanh(x) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Besides these tanh-based functions, we also test the sigmoid and the Heaviside function:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{4.4}$$

$$\text{heaviside}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Following Masse et al. (2019), we use the first-order Euler approximation to simulate the network. The dynamics of the network are then given by:

$$r_t = (1 - \alpha) r_{t-1} + \alpha f\left(W^{\text{rec}} r_{t-1} + g(u_t)\right) \tag{4.6}$$

where $\alpha = \Delta t / \tau$ is the Euler integration step size.

Since we want to model two different equally sized groups of units ($N_{\text{pop}} = \frac{N}{2}$) representing the CCS and CPn populations, we structure the weight matrix W^{rec} as follows:

$$W^{\text{rec}} = \begin{bmatrix} W^{\text{CCS}\to\text{CCS}} & W^{\text{CCS}\to\text{CPn}} \\ W^{\text{CPn}\to\text{CCS}} & W^{\text{CPn}\to\text{CPn}} \end{bmatrix} \tag{4.7}$$

where the matrices $W^{\text{CCS}\to\text{CCS}}$ and $W^{\text{CPn}\to\text{CPn}}$ define the recurrent connections within the CCS and CPn populations, the matrix $W^{\text{CCS}\to\text{CPn}}$ defines the forward connection weights from CCS to CPn and $W^{\text{CPn}\to\text{CCS}}$ the backward connections in the opposite direction. Unless otherwise stated, input is only given to the first population, i.e. the

Connection densities			
Name	**Value**	**Description**	**Source**
$p_{\mathrm{CPn}\to\mathrm{CPn}}$	0.13(50/381)	connection density inside CPn population	[2], [3]
$p_{\mathrm{CCS}\to\mathrm{CCS}}$	0.14(38/273) $*$0.1(31/308)	connection density inside CCS population	[2], [3] $*$[1]
$p_{\mathrm{CCS}\to\mathrm{CPn}}$	0.11(11/98)	connection density from CCS to CPn	[3]
$p_{\mathrm{CPn}\to\mathrm{CCS}}$	0.01(1/96)	connection density from CPn to CCS	[3]
$r_{\mathrm{CPn}\leftrightarrow\mathrm{CPn}}$	0.32(16/50)	reciprocity of recurrent CPn connections	[2]
$r_{\mathrm{CCS}\leftrightarrow\mathrm{CCS}}$	0.13(4/30) $*$0.11(4/34)	reciprocity of recurrent CCS connections	[2] ($*$[1])

Table 4.1: Parameters for connection densities and reciprocities. Sources: [1] Morishima and Kawaguchi (2006); [2] Morishima et al. (2011); [3] Morita et al. (2012)

CCS population. Table C.1 shows the mean and standard deviation for each of the weight matrices together with the literature references from which these values originate. We draw the weights from a log-normal distribution and in order to produce a distribution with the desired mean and standard deviation, we transform the mean μ_{ln} and standard deviations σ_{ln} from Table C.1 to the corresponding parameters μ_{gauss} and σ_{gauss} of the underlying Gaussian of the log-normal distribution:

$$\mu_{\mathrm{gauss}} = \log\left(\frac{\mu_{\mathrm{ln}}^2}{\sqrt{\sigma_{\mathrm{ln}}^2 + \mu_{\mathrm{ln}}^2}}\right) \tag{4.8}$$

$$\sigma_{\mathrm{gauss}} = \sqrt{\log\left(\frac{\sigma_{\mathrm{ln}}^2}{\mu_{\mathrm{ln}}^2} + 1\right)} \tag{4.9}$$

The populations are sparsely connected with each other. Table 4.1 shows the densities of connections within and between the different populations and the fraction of reciprocal connections inside the recurrent weight matrices $W^{\mathrm{CCS}\to\mathrm{CCS}}$ and $W^{\mathrm{CPn}\to\mathrm{CPn}}$. The reciprocity parameter defines the fraction of mutual connections between two neurons inside the respective population, i.e. the probability that a connection from neuron i to neuron j is accompanied by a connection from neuron j to neuron i. Similar to the weights of the echo state network in Section 3.2.3, we scale W^{rec} such that the spectral radius of the matrix is equal to 0.9 to ensure stable network dynamics.

4.2.2 Structured rate-based model

Based on the rate-based baseline model, we now extend the model to follow Dale's principle and divide the CCS and CPn populations into inhibitory and excitatory subpopulations (see Figure 4.2). The size of these separate excitatory and inhibitory subpopulations is defined by the proportion of inhibitory neurons β_{inh} in the network. Therefore,

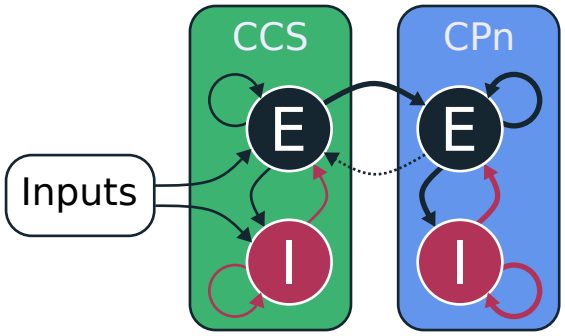


Figure 4.2: Scheme of the structured network model. The network is separated into a CCS (green) and a CPn population (blue). Both populations are further divided into an excitatory (black) and an inhibitory subpopulation (red). The input is only given to the CCS population. The connections from CCS to CPn and the recurrent connections inside the CPn population are stronger (indicated by thicker arrows).

both the CCS and CPn populations consist of $\beta_{\text{inh}} \cdot N_{\text{pop}}$ inhibitory and $(1-\beta_{\text{inh}}) \cdot N_{\text{pop}}$ excitatory neurons. The weight matrix W^{rec} is then structured as follows:

$$W^{\text{rec}} = \begin{bmatrix} W^{\text{CCS}_\text{E}\to\text{CCS}_\text{E}} & W^{\text{CCS}_\text{E}\to\text{CCS}_\text{I}} & W^{\text{CCS}_\text{E}\to\text{CPn}_\text{E}} & W^{\text{CCS}_\text{E}\to\text{CPn}_\text{I}} \\ W^{\text{CCS}_\text{I}\to\text{CCS}_\text{E}} & W^{\text{CCS}_\text{I}\to\text{CCS}_\text{I}} & W^{\text{CCS}_\text{I}\to\text{CPn}_\text{E}} & W^{\text{CCS}_\text{I}\to\text{CPn}_\text{I}} \\ W^{\text{CPn}_\text{E}\to\text{CCS}_\text{E}} & W^{\text{CPn}_\text{E}\to\text{CCS}_\text{I}} & W^{\text{CPn}_\text{E}\to\text{CPn}_\text{E}} & W^{\text{CPn}_\text{E}\to\text{CPn}_\text{I}} \\ W^{\text{CPn}_\text{I}\to\text{CCS}_\text{E}} & W^{\text{CPn}_\text{I}\to\text{CCS}_\text{I}} & W^{\text{CPn}_\text{I}\to\text{CPn}_\text{E}} & W^{\text{CPn}_\text{I}\to\text{CPn}_\text{I}} \end{bmatrix} \tag{4.10}$$

where the subscripts E and I denote the excitatory and inhibitory subpopulations, respectively. Weight matrices corresponding to connections from inhibitory populations are multiplied by the factor γ to ensure a strong inhibitory effect from the activity of these populations. We scale W^{rec} in the same way as we describe above in Section 4.2.1 for the baseline continuous rate network. In addition, to ensure only non-negative rates, we use a sigmoid function as the activation function f (unless stated otherwise) instead of the tanh non-linearity. To further enhance the biological plausibility of the model, we also incorporate a dynamic modulation of the synaptic efficacies through short-term plasticity (STP) in some of the experiments (see Masse et al. (2019) and Mongillo, Barak, and Tsodyks (2008)). STP is modeled by the interactions between the fraction of available neurotransmitters x and the neurotransmitter utilization u. The dynamics of these two variables evolve according to the following equations:

$$\frac{dx(t)}{dt} = \frac{1 - x(t)}{\tau_D} - u(t)x(t)r(t)\Delta t \tag{4.11}$$

$$\frac{du(t)}{dt} = \frac{U - u(t)}{\tau_F} + U(1 - u(t))r(t)\Delta t \tag{4.12}$$

where τ_D and τ_F are the time constants of the recovery from depression and facilitation, respectively, and U is the neurotransmitter increment. The synaptic input I to postsynaptic neurons is then given by:

$$I(t) = W^{\text{rec}}x(t)u(t)r(t) \tag{4.13}$$

4.2.3 Spiking baseline model

The spiking baseline model is based on the rate-based model described in Section 4.2.1. The main difference is that the units are now modeled as spiking neurons. We use the leaky integrate and fire neuron model with synaptic transmissions that elicit a delta-shaped post-synaptic current as described in Section 3.2.3, Equation 3.14 and Equation 3.15. The neuron parameters can be found in Table C.2 in the appendix. As a state variable for the readout in all spiking networks, we use the filtered spike trains of the neurons (filter time constant $\tau = 50$ ms).

4.2.4 Structured spiking model

Just as the spiking baseline model is structured like the continuous-rate baseline model, the structured spiking model shares the main structure with its continuous rate-based counterpart described in Section 4.2.2. In addition to dividing each population into excitatory and inhibitory subpopulations, we also consider short-term synaptic plasticity and use an adaptive-exponential integrate-and-fire model whose parameters are fitted to the activity reported in Morishima and Kawaguchi (2006). The resulting parameters for the CCS and CPn populations are listed in Table C.3 and Table C.4. The spiking behavior of the two models is shown in Figure 4.3. To obtain a firing behavior that closely matches the experimental recordings, we measured the spike times from Figure 2 in Morishima and Kawaguchi (2006) and optimized the membrane capacitance C_{m}, the firing threshold V_{th}, the reset potential V_{reset} and the spike frequency adaptation parameters a, b and τ_{w} using the Python package Optuna (Akiba et al., 2019). We also use conductance-based exponential synapses with short-term plasticity (Mongillo, Barak, and Tsodyks, 2008) with parameters from Morishima et al. (2011) (see Table C.6 in the appendix) instead of the current-based models used in the spiking baseline network.

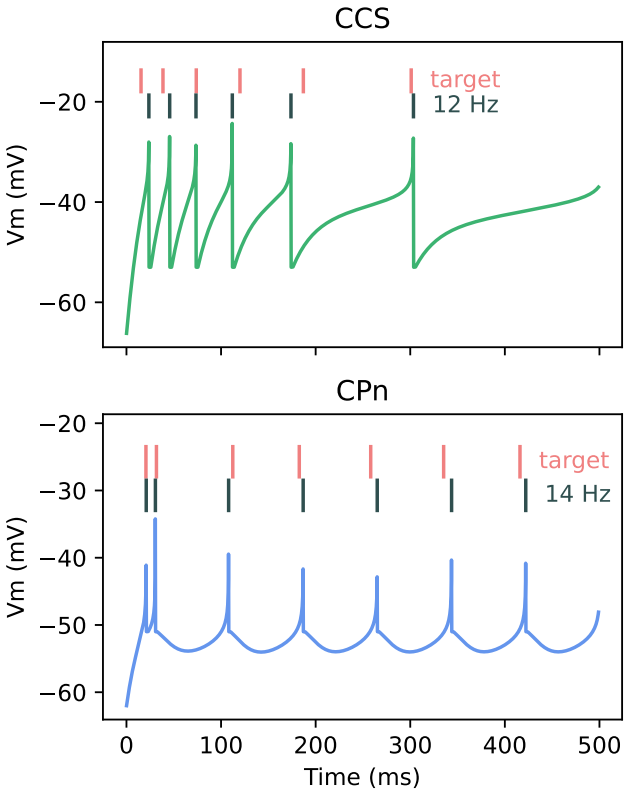


Figure 4.3: Fitted neuron models. Voltage traces in green (CCS) and blue (CPn) in response to a step depolarization of 0.5 nA over 500 ms. Black bars represent the spike times of the fitted models and red bars the spike times of the experimental data from Morishima and Kawaguchi (2006).

4.2.5 Network modification experiments

To investigate the effects of the different properties of the network model, we perform a series of experiments in which we deactivate or homogenize parts of the connectivity structure or the neuron parameters and measure the linear memory part of the information processing capacity (see Section 3.2.1) of the CCS and CPn populations separately. We evaluate the sum of capacities over all delays and the maximum delay before the capacity drops below a chance level threshold. As we have already described in Section 3.2.4, we cannot exactly define the chance level in advance. We assume that there won't be memory longer than 3000 ms in any of the systems and therefore, we evaluate the capacity for up to a maximum delay of 100 with a stimulus duration of 50 ms, resulting in a maximum memory of 5000 ms. We then calculate the capacities for the last 40 stimuli (after 3000 ms) and use the maximum value multiplied by a factor of 1.05 as the chance level cut-off. Based on this we evaluate the different network modification experiments.

Three of these experiments involve adjustments to the connection weights in the networks. In the first experiment, we set the mean of the synaptic weights to the same value for all connections, while leaving the connection type-specific standard deviation untouched. The next adjustment of the network is to homogenize the standard deviations of the weights but leave the mean values of the weights at their data-based values. In the last adjustment, we combine the two previous changes and homogenize the means and standard deviations of the weights together.

In the next two experiments, we change the connectivity structure of the network by (1) connecting each subpopulation with the same connection probability and (2) removing the data-based reciprocity properties. Then we test the effect of short-term plasticity by disabling it, and eventually, for the spiking neural networks, we test the effect of spike frequency adaptation also by disabling it. In a final experiment, we remove all connections between the CCS and CPn populations and provide the input to both to remove the effects of information transfer between the populations and test the memory of the CPn population under the condition that it has unaltered information about the input signal.

4.2.6 Conversion networks

The two preceding spiking neural networks are constructed from first principles and data from the literature (Morishima et al., 2011; Morishima and Kawaguchi, 2006; Morita et al., 2012). As an intermediate step between these networks and the rate-based networks described in Section 4.2.1 and Section 4.2.2, we create two additional spiking networks by transforming the continuous rate-based networks according to Kim, Li, and Sejnowski (2019). Just like Kim, Li, and Sejnowski (2019), we use leaky integrate-and-fire neurons following the Equation 3.14 of Section 3.2.3 and parameterised as in Nicola and Clopath

(2017) (see Table C.5). Incoming spikes trigger an exponentially shaped postsynaptic current. To obtain a spiking network functionally equivalent to the rate network, we scale the weights in the SNN by $1/\lambda$. We determine the correct value of λ by performing a parameter scan from 20 to 75 with a step size of 5 as in Kim, Li, and Sejnowski (2019). In this parameter scan, we look for the value of λ that maximizes the linear memory capacity of the CCS and CPn populations.

4.3 Results

4.3.1 How nonlinearities shape the memory in the baseline continuous rate network

In the first set of experiments, we investigate the effect the nonlinear transfer function f has on the memory in the baseline rate network described in Section 4.2.1. We test the six different nonlinearities shown in Figure 4.4A and described in Section 4.2.1.

The main results for these transfer function experiments are shown in Figure 4.4B. The upper panel shows the sum of capacities above the random threshold for each network variant, separated into CCS (green) and CPn (blue) populations. The bottom panel shows the maximum delay before the capacity falls below this threshold (in milliseconds, as it is multiplied by the stimulus duration of 50 ms). As the first row of bars and panel C show, the model with tanh-nonlinearity has the largest memory capacities and, especially, the memory in the CPn population is much higher than in the other systems. Even the shift to exclusively positive output values (tanh+1) lowers CPn memory significantly (panel D). In contrast, the differences between tanh+1, scaled-tanh and sigmoid are only small (panels D-F). The two functions that set all negative inputs to zero, rectified-tanh and Heaviside, perform worse than the rest, especially when looking at the capacity sum (panels G and H). There, the CCS population even shows a higher capacity sum than the CPn population. In all other cases, however, the CPn memory is higher than the memory of the CCS population, regardless of the shape of the transfer function.

These observations can be explained by two different aspects of the transfer function f; 1) how close to linear the transfer function is and 2) which output value it has at zero input. First of all, a linear function results in the optimal memory in the system because then the information processing capacity only consists of linear memory and is not distributed over multiple higher-degree capacity functions. The slope of the linear function is effectively scaling the spectral radius of the weights. Therefore, if we want our effective spectral radius to be 0.9 we have to use the identity function as f. As Figure 4.5 shows, the deviation from the identity function can be indicative of how long the information can be preserved by the system. Here we use the integral of the absolute difference between the identity function and f between -1 and 1 as a measure for this deviation from the identity function (Figure 4.5B). We chose these limits because in the baseline rate network, each neuron is on average excitatory and inhibitory to the

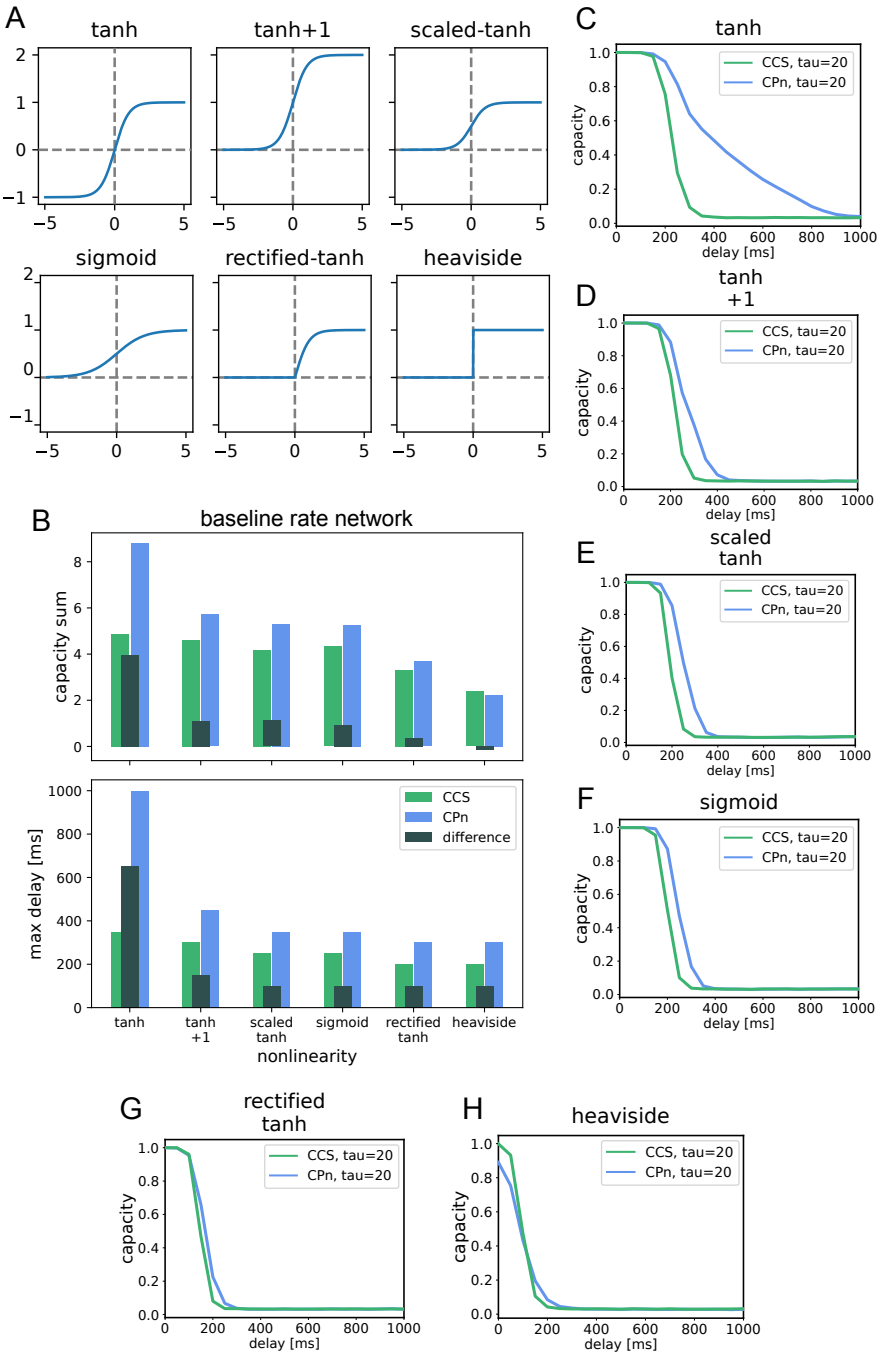


Figure 4.4: Baseline rate network with different nonlinearities. **A:** Shapes of the tested nonlinearities. **B:** Sum of capacities above the chance level threshold for the nonlinearities shown in A. **C-H:** Memory curves for the networks with the different nonlinearities.

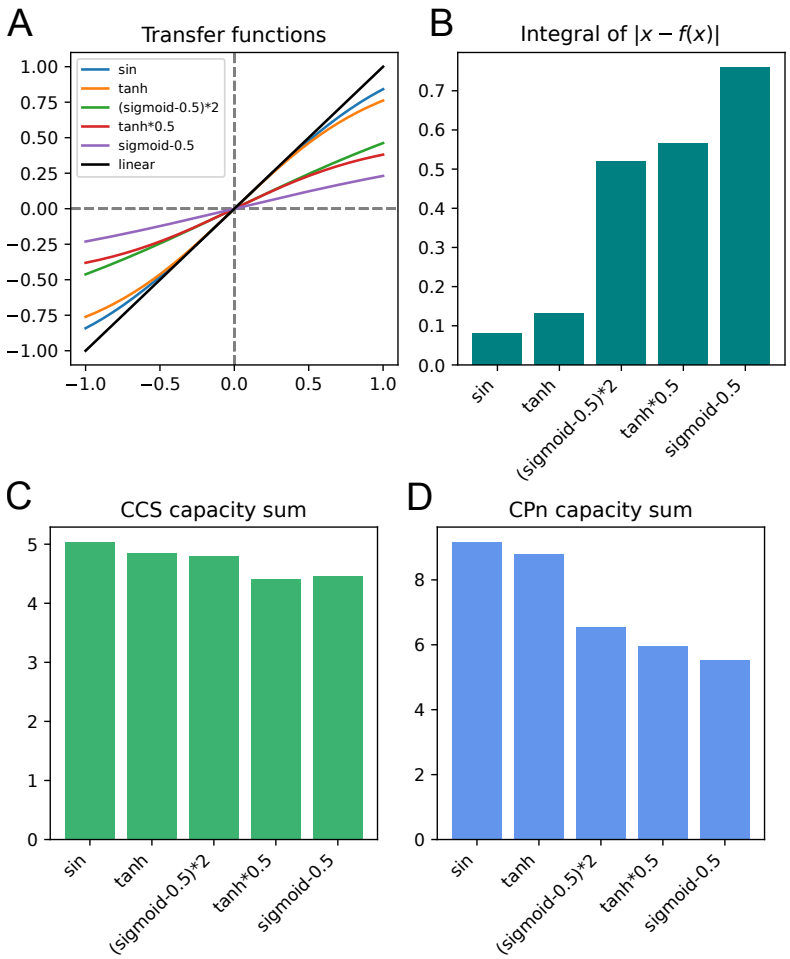


Figure 4.5: Deviation of nonlinear transfer functions from the linear identity function. **A:** Shapes of the tested nonlinearities between -1 and 1. In addition to the previously described functions also the trigonometric sine function (sin) is used. **B:** Area between the transfer functions and the identity function as a measure for deviation from the identity function. **C and D:** Sum of capacities above the chance level threshold for the different transfer functions for the CCS (A) and CPn (B) populations.

same extent, resulting in a mean recurrent input to each neuron of 0. Therefore, it is most important for the transfer function to be linear around 0 inputs to achieve a good memory performance. This also explains why functions that have their most non-linear part at zero (rectified-tanh and Heaviside) exhibit the lowest memory. Figure 4.5B-D show that the amount of deviation from the identity function is anti-correlated with the memory capacity sum in the CCS and CPn populations. However, this still does not explain the difference between tanh and tanh+1 because, for the average zero input to each neuron, tanh+1 is just as linear as tanh. The difference lies in the amount of variation in activity and therefore in the recurrent inputs that the neurons receive. Transfer functions that result in a higher output at zero input result in higher variability in the recurrent inputs because the weight matrix scatters higher values more widely. If the inputs to the individual neurons are more widely spread, a broader part of the transfer function is used. This inclusion of the strongly non-linear parts of f reduces memory. We test this by evaluating networks with differently shifted versions of the tanh and sigmoid nonlinearity. Figure 4.6 shows the results for the tanh and sigmoid nonlinearities.

In both cases, the trend can be clearly observed: the more the transfer function is shifted away from $f(0) = 0$ the higher the standard deviation of the activity and the lower the memory in both populations of the system.

4.3.2 How weight distributions shape the memory in the baseline continuous rate network

After evaluating the transfer functions, we study the effects of the different structural properties of the baseline network by homogenizing certain features of the connections (see Section 4.2.5). For this purpose, we analyze the tanh network and the network with sigmoid nonlinearity. We chose the first system because of its superior performance and the second because its non-negative outputs are more compatible with the concept of firing rates in biological neural networks. Figure 4.7 shows the results of these studies. In panel A, are the sum of the capacities (top) and the maximum capacity delay (bottom) for the different tanh networks visible. These bar graphs show that there is little difference between the memory of the unmodified model (rightmost bar and panel B), the network with the same density for all connections, the network that does not incorporate the data-based reciprocity features, the network that uses the same standard deviation for all connections (but different mean values), and the network constructed with the same weight mean for all synapses (but different standard deviations). However, the simultaneous change in mean and standard deviation of the weight distributions has a more pronounced effect. This change increases CCS memory and decreases CPn memory, with CPn memory remaining higher than memory in the CCS population (leftmost bars and panel C). Therefore, the memory results are robust to changes in network

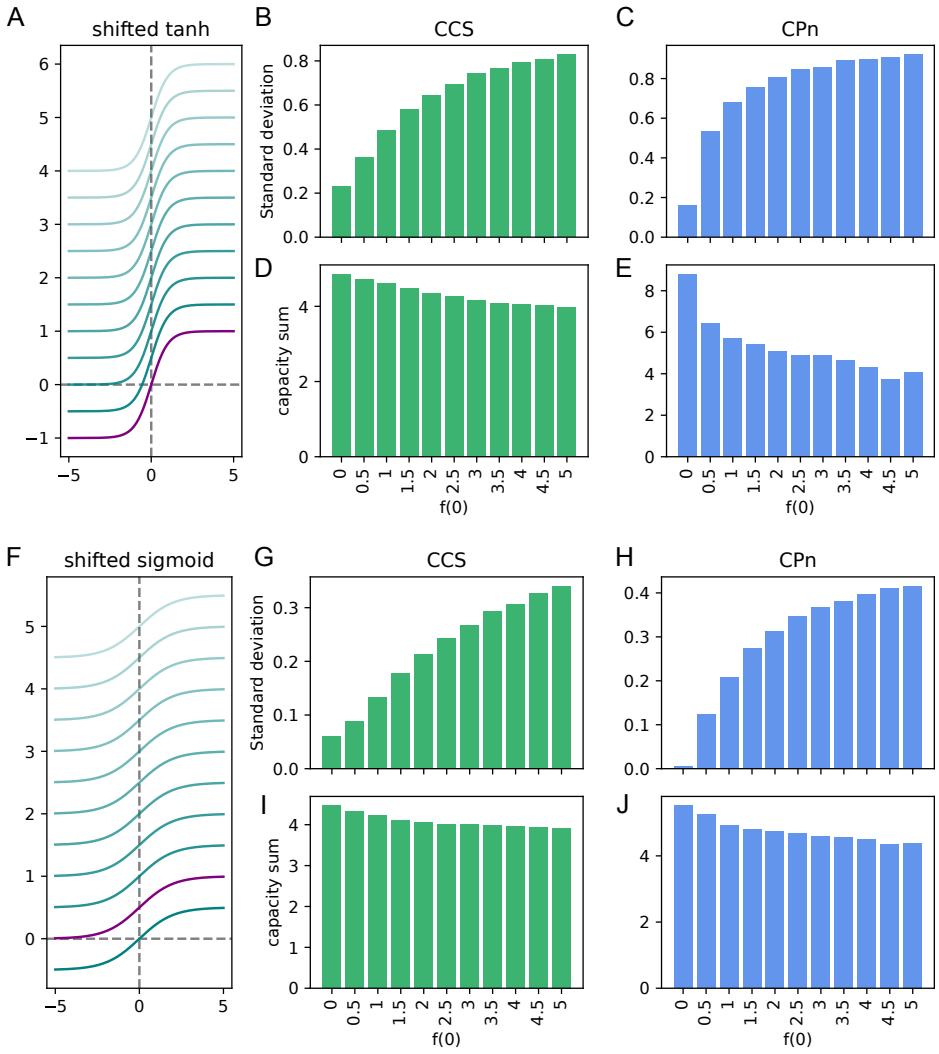


Figure 4.6: Effect of different shifts of the tanh and sigmoid transfer functions on memory. **A:** Shifted versions of the tanh transfer function. **B and C:** Standard deviation of the activity in the CCS (B) and CPn (C) populations for the different shifted tanh transfer functions. **D and E:** Sum of linear capacities above the chance level threshold for the different shifted tanh transfer functions for the CCS (D) and CPn (E) populations. **F:** Shifted versions of the sigmoid transfer function. **G and H:** Standard deviation of the activity in the CCS (G) and CPn (H) populations for the different shifted sigmoid transfer functions. **I and J:** Sum of linear capacities above the chance level threshold for the different shifted sigmoid transfer functions for the CCS (I) and CPn (J) populations.

structure, and the specificity of the weights plays the most important role. However, even these changes do not significantly affect the memory profiles in the two populations unless they are fully homogenized. In the final experiment with the tanh network, we remove all connections between the CCS and CPn populations and provide both with input. Thus, we eliminate the effects of information transfer between the populations and test the memory of the CPn population under the condition that it has unchanged information about the input signal. This modification significantly reduces the memory of the CPn population (penultimate set of bars and panel D), demonstrating that information transfer between the populations is a critical factor in maintaining information in the CPn population. The signal of the CCS population already contains information about previous inputs and passes this on to the CPn population. The CPn population stores this information just like the information about the current input. This allows an accumulation of the memory of both populations in the CPn neurons.

The memory differences between the CCS and CPn populations are less pronounced in all variations of the sigmoid network compared to the tanh networks, as shown in the bottom half of Figure 4.7. Overall, memory capacity is also lower in these sigmoid networks. Similar to the tanh networks, the complete homogenization of weights (using the same mean and standard deviation for all connections) slightly increases the CCS capacity sum while decreasing the CPn memory compared to the unmodified network (full model). Further, homogenizing the weight means notably increases the memory capacity difference between the two populations, primarily due to a decrease in the CCS population's memory (panel G). The separation of the two populations results in a decrease in the CPn population's memory, reinforcing the importance of information transfer from the CCS to the CPn population for memory retention (panel H). Other adjustments to the sigmoid network, such as homogenization of weight standard deviations, connection density, and reciprocity, show minimal impact on memory performance. In summary, both the tanh and sigmoid baseline rate networks display high robustness against most structural changes, with significant result alterations primarily occurring in response to modifications in connection weights.

4.3.3 From baseline rate network to the structured rate network

In the previous section, we analyzed the baseline rate networks. In the next step, we want to bring these artificial networks closer to biological neural networks by accounting for Dale's principle (Eccles, Fatt, and Koketsu, 1954) and separating the populations into exclusively excitatory and exclusively inhibitory neurons. For this to be a meaningful change we also cannot use the tanh transfer function anymore because the negative rates could cause an inhibitory effect from excitatory neurons and vice versa. Instead, we only use the sigmoid function as nonlinearity and first search for the adequate ratio between excitation and inhibition by adjusting the fraction of inhibitory neurons β_{inh}

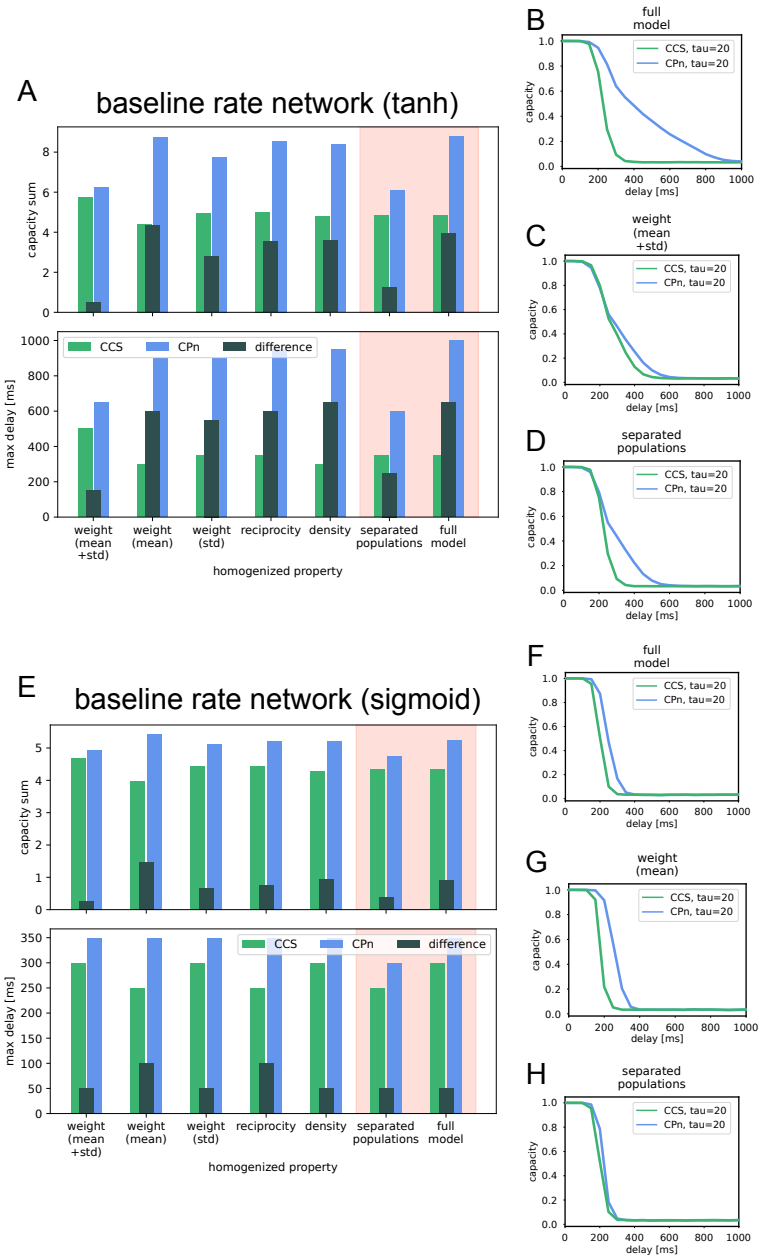


Figure 4.7: Network modification studies for baseline rate networks with tanh and sigmoid nonlinearity. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks with tanh nonlinearity. **B-D:** Memory curves for the full model and the two networks with the biggest difference to the full network model. **E-H:** The same figures as in A-D but for the networks with sigmoid nonlinearity.

and the inhibitory weight factor γ in the network. Figure 4.8 shows the results of these experiments. These heatmaps show us that especially when the inhibition is exactly balanced with the excitation the memory capacity for both populations is high. This is the case when the following equation is true:

$$\gamma \cdot \beta_{\text{inh}} = 1 - \beta_{\text{inh}} \tag{4.14}$$

However, this effect is weakly pronounced in the CCS population. In contrast, the performance of the CPn population seems to depend more strongly on the exact balance between excitation and inhibition since especially the γ and β_{inh} combinations that perfectly fit Equation 4.14, i.e. $(\gamma, \beta_{\text{inh}}) \in ((1, 0.5), (4, 0.2), (9, 0.1))$, result in higher capacity sums that can easily be distinguished from the surrounding parameter combinations. In the CPn population, the area of high memory capacity sums is slightly wider. In general, most of the values in the second column are higher (lighter colors) than in the first column. Therefore, for most parameter combinations the CPn can retain information for longer than the CCS population.

In Figure 4.9A and B the activity of two of the above-described networks (A: $\gamma = 4, \beta_{\text{inh}} = 0.2$; B: $\gamma = 1, \beta_{\text{inh}} = 0.5$) can be seen. In these configurations excitation and inhibition are perfectly balanced. This leads to very similar activity in the two configurations, both when comparing the CCS populations and when comparing the CPn populations. On the one hand, this can be seen in the similar color distributions in the status matrices in the upper two subpanels of A and B, and on the other hand in the corresponding histograms in the lower row. The latter shows that the inhibitory and excitatory parts of the CCS population have a very similar and symmetrical activity distribution around a value close to 0.5. However, the activity of the two subpopulations of CPn neurons exhibits higher values around approximately 0.8, which is likely caused by the additional excitatory feedforward input coming from the CCS population.

The enhanced memory at a distribution near 0.5 in the CCS populations makes sense in that the sigmoid function is almost linear in this range, favoring the maintenance of input information rather than the computation of non-linear functions, similar to the echo state networks in Section 3.3.1. However, the utilization of the higher parts of the nonlinearity in the CPn population is not optimal for memory.

Panel C gives us more information about the activity distributions of the different network configurations. These heatmaps look qualitatively very similar to the capacity sum heatmaps in Figure 4.8. The transition between mean activities above and below 0.5 is along the line where Equation 4.14 is true, which is no surprise since excitation and inhibition balance out here. The fact that the mean activity of the CCS population for the marked configurations is slightly above 0.5 is probably due to the additional positive input signal. The mean activity of the CPn population is also above 0.5 in these configurations because of the additional excitatory feedforward input. However, the line of balanced activity is also visible as a line of enlarged standard deviations in

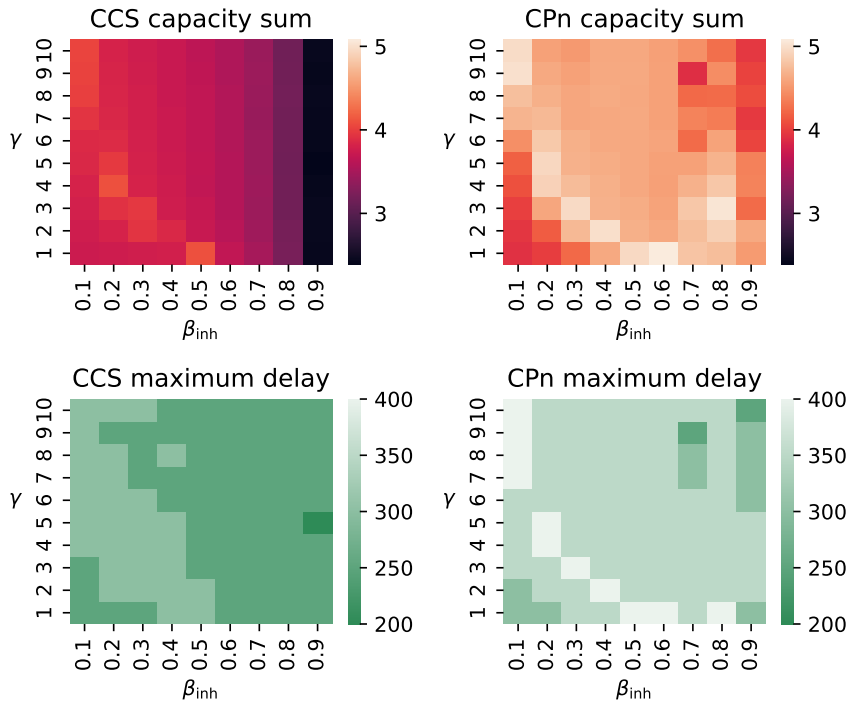


Figure 4.8: Effect of balance between excitation and inhibition on memory. **Top:** Heatmaps of linear capacity sum for a parameter scan of inhibitory weight factor γ and fraction of inhibitory neurons β_{inh} for the CCS (left) and CPn (right) populations. **Bottom:** Maximum capacity delay for the same parameter scan for the CCS (left) and CPn (right) populations.

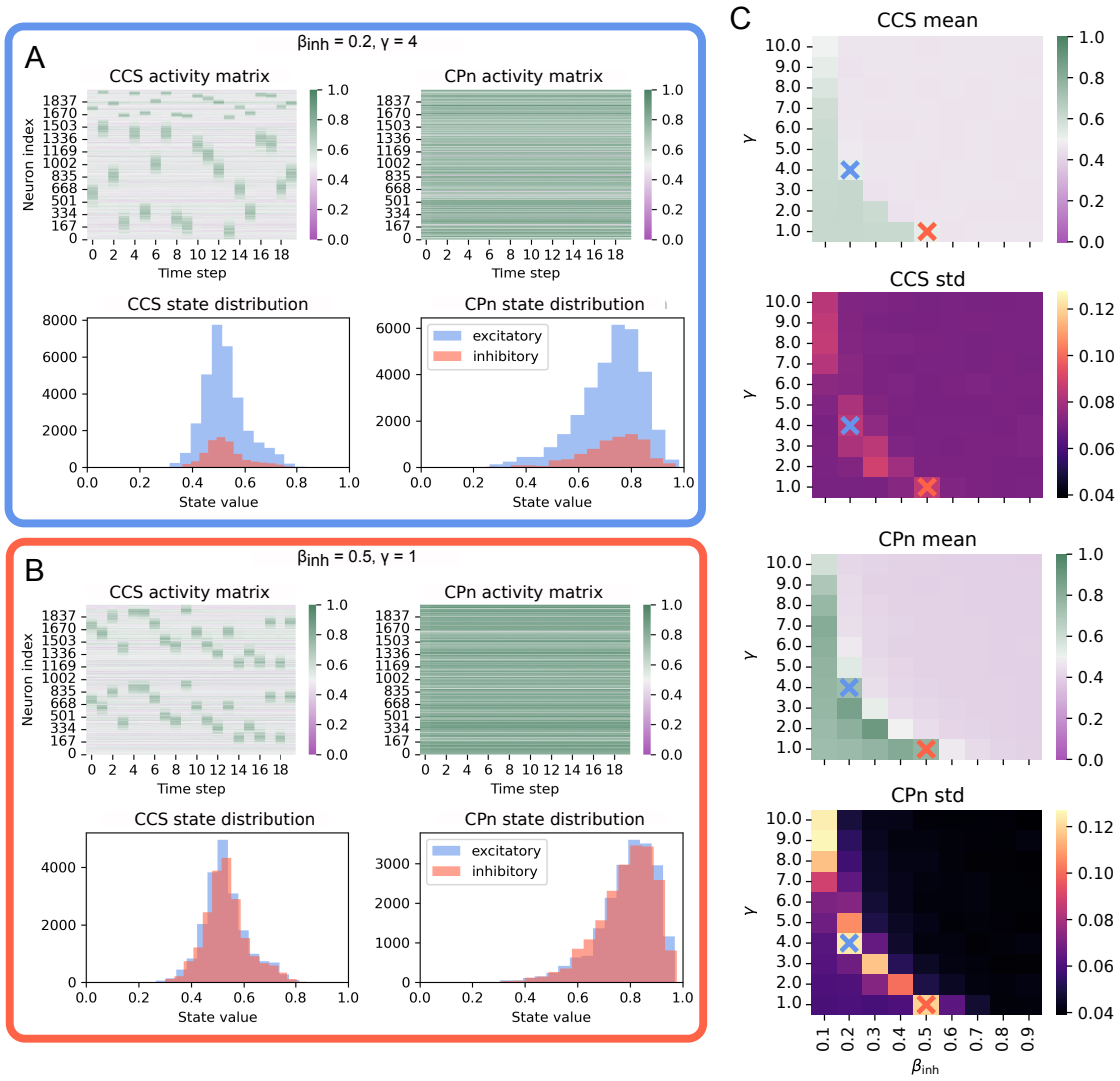


Figure 4.9: State matrix data of networks with different EI-ratios. **A:** State matrices of the CCS (left) and CPn (right) population of a network with $\gamma = 4$ and $\beta_{\text{inh}} = 0.2$ at the top and their corresponding state distributions at the bottom (based on the same data as the activity matrices above). **B:** The same figure as in A but for a network with $\gamma = 1$ and $\beta_{\text{inh}} = 0.5$. **C:** Heatmaps of mean activity and standard deviation of the activity for the parameter scan over γ and β_{inh}. The blue mark corresponds to the network in A and the red mark to the network in B.

the corresponding standard deviation heatmaps. The reason for this is likely the fact that the sigmoid function has its steepest slope at an input value of 0.5. Therefore, the activity of the neurons is more strongly affected by small changes and this results in a broader distribution of activities.

Since the activity of the CPn population seems to be too high for optimal information retention, we test whether increased inhibition in this population has a positive effect on CPn memory. We adjusted the factor for inhibitory weights in the CPn population γ_{CPn}, keeping $\gamma_{\mathrm{CCS}} = 4$ and $\beta_{\mathrm{inh}} = 0.8$ constant. Figure 4.10 shows the results of this experiment. The first panel shows that the capacity sum in the CPn population increases up to a γ_{CPn} value of 4 and then does not change significantly. However, the optimal capacity sum shows up at a slightly higher γ_{CPn} value of 4.8 (green bar). In panels B-D it can be seen that the activity mean of the CPn population is still above 0.5 for this configuration. While the mean is already close to the most linear part of the sigmoid nonlinearity, the distribution of activities in this configuration is wider than in the surrounding configurations, as can be seen from the higher bar in the standard deviation subfigure in panel E and also from the wider histogram in panel C. With this larger standard deviation, much of the almost linear component of the sigmoid function is used, although the mean activity is above 0.5. The reason for the lower storage capacity sum for higher values of γ_{CPn} is probably that the storage capacity of the CCS population decreases for these higher values (panel F). This is likely due to the higher inhibition weights in the CPn population, so the entire weight matrix must be scaled by a higher factor to achieve the same spectral radius. This scaling also reduces the weights within the CCS population and may reduce its memory. If the signal from the CCS population to the CPn population no longer contains the information of previous inputs as long, the CPn population will also not be able to retain this information to the same extent.

4.3.4 Structured continuous rate network

Based on the above studies, we choose the parameters $\beta_{\mathrm{inh}} = 0.2$ and $\gamma = 4$ for the further experiments. We avoided strengthening the inhibitory weights of the two populations with different scaling factors to avoid increasing the complexity of the network and because this separation did not lead to significantly different results. Just as for the baseline rate network in Section 4.3.2, we analyze which aspect of the network structure has the strongest effect on memory in the two populations by successively omitting data-based structural aspects of the network. The results of these structural reduction experiments are presented in Figure 4.11. Panel A shows that most structural changes have negligible effects on the sum of memory capacity in the two populations. Only when we adjust the mean values of the weight distributions do the results change. Homogenizing the mean weights while retaining the individual standard deviations has little effect on CCS memory but increases memory in the CPn population (second set

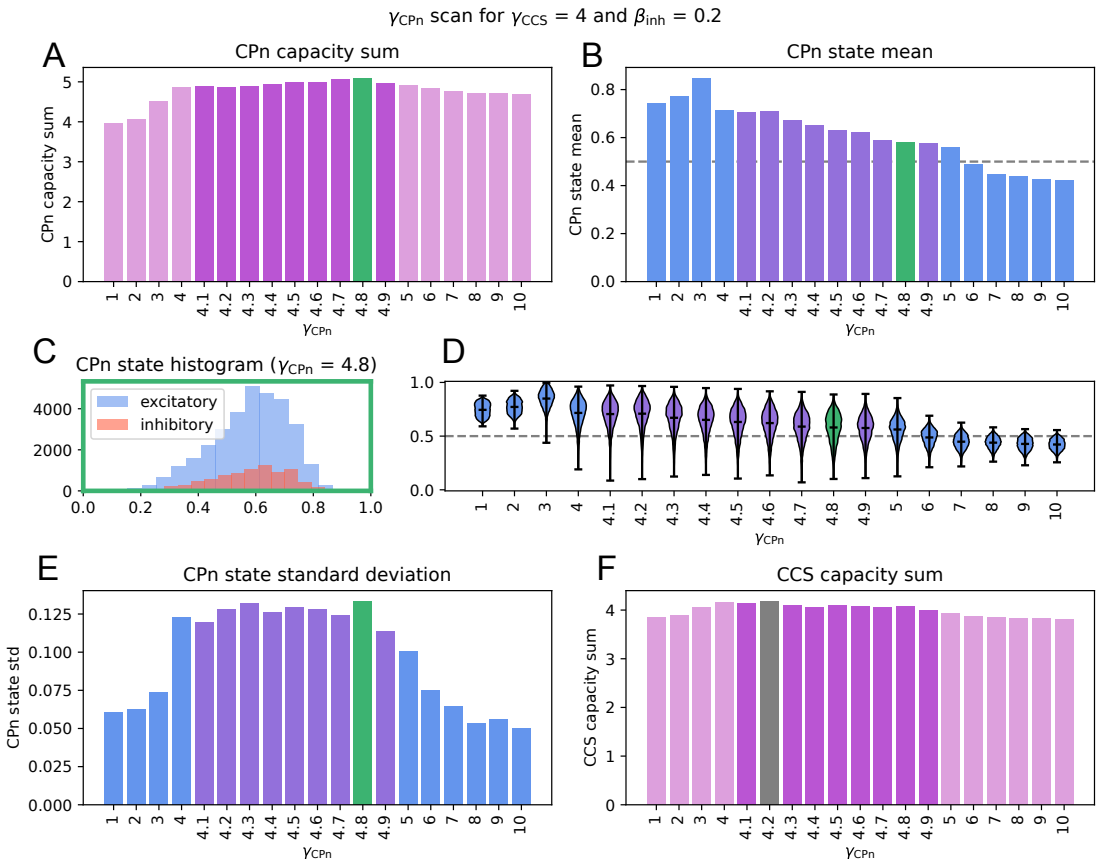


Figure 4.10: Scan over the inhibitory weight factor for the CPn population γ_{CPn} for fixed parameters for the CCS population ($\gamma_{\mathrm{CCS}} = 4$ and $\beta_{\mathrm{inh,CCS}} = 0.2$). **A:** Linear capacity sum of the CPn population for different values of γ_{CPn}. Bars with darker color mark a shift from step size 1 to step size 0.1. The green bar marks the highest capacity sum. **B:** Mean activity of the CPn population for the same configurations as in A. Again a color change marks the change in step size and the green bar marks the network with the highest capacity sum in the CPn population. **C:** Histogram of the activity of the CPn population for the network with the highest capacity sum, separated into excitatory and inhibitory neurons. **D:** Violin plot showing the state distributions of the CPn populations for each γ_{CPn}. Colors mean the same as in B. **E:** Standard deviation of the activity of the CPn population for each network. Colors mean the same as in B. **F:** Linear capacity sum of the CCS population (instead of the CPn population) for the same configurations as in A. The grey bar marks the maximum capacity for the CCS population.

of bars and panel C). The additional homogenization of the standard deviations reduces the storage capacity in the CPn population while increasing the storage capacity in the CCS population. This leads to almost equal linear capacity sums in the two populations (first set of bars and panel B). Just as in the baseline rate network, the specificity of the weights is the most important factor for the memory properties of the CCS and CPn populations in the full data-based network model (rightmost set of bars and panel D), and the results are robust to other changes.

4.3.5 Spiking neural networks

We have investigated the continuous rate-based networks in detail and now make the step to spiking neural networks. Also for these networks, we start with a baseline model that does not incorporate Dale's principle and then separate both populations into excitatory and inhibitory subpopulations in a second step. Both types of populations are made of neurons of the fitted adaptive-exponential model described in Section 4.2.2. In addition to the structural properties we have changed in the rate-based networks, we add two new property reduction experiments, namely we disable the plasticity and the firing rate adaptation of the neurons. The results of these experiments can be seen in Figure 4.12 (baseline network) and Figure 4.13 (structured network). Both of these sets of experiments give us qualitatively the same results.

Neither in the baseline network nor in the structured network is information being transferred from the CCS population to the CPn population. Already the memory in the CCS population is significantly reduced compared to the results in the rate-based networks (see Figure 4.7F and Figure 4.11D), but in the CPn population not even the undelayed input signal can be reconstructed from the state matrix. Only when we separate the populations and give input to both we can retrieve information from the CPn population (penultimate set of bars in Figure 4.12A and C and Figure 4.13A and C). However, the memory in the CPn population is still lower than in the CCS population.

4.3.6 Spiking networks constructed from rate networks

The above experiments show that we cannot extract a long memory from spiking networks if we construct them according to first principles. However, since the continuous rate networks show results that support our underlying hypothesis of longer memory in the CPn population, we test the procedure from Kim, Li, and Sejnowski (2019) to construct spiking networks from existing rate networks. To do so, as explained in Section 4.2.6, we use less complex leaky integrate-and-fire neurons, perform a scan over the weight scaling parameter λ, and calculate the memory capacity of both populations for each configuration. Figure 4.14 shows the memory capacity curves of the best configurations for the spiking networks based on the baseline rate network (Panel A) and the structured rate network (Panel B). The line graphs show that also in these spiking

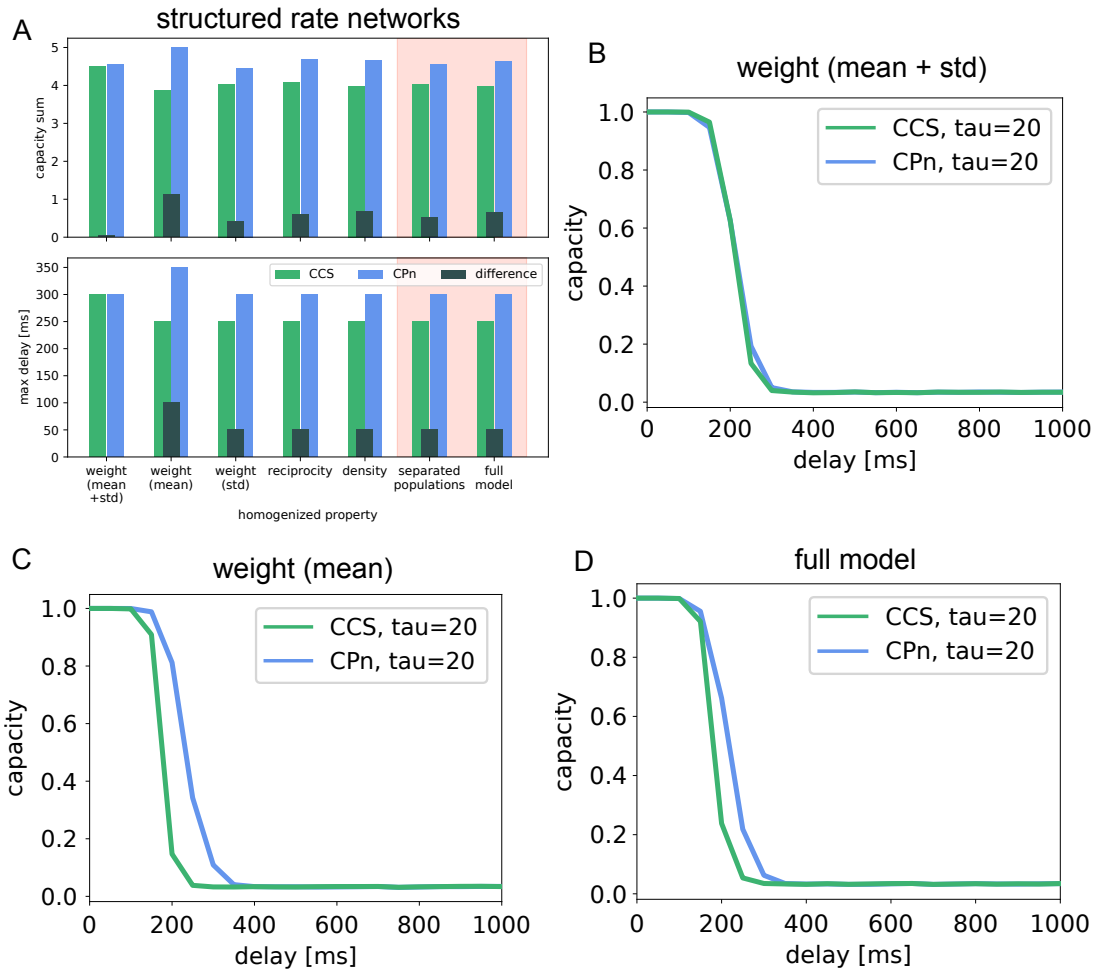


Figure 4.11: Modified network studies for structured rate networks with sigmoid nonlinearity. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B-D:** Memory curves for the full model and the two networks with the biggest difference to the full network model.

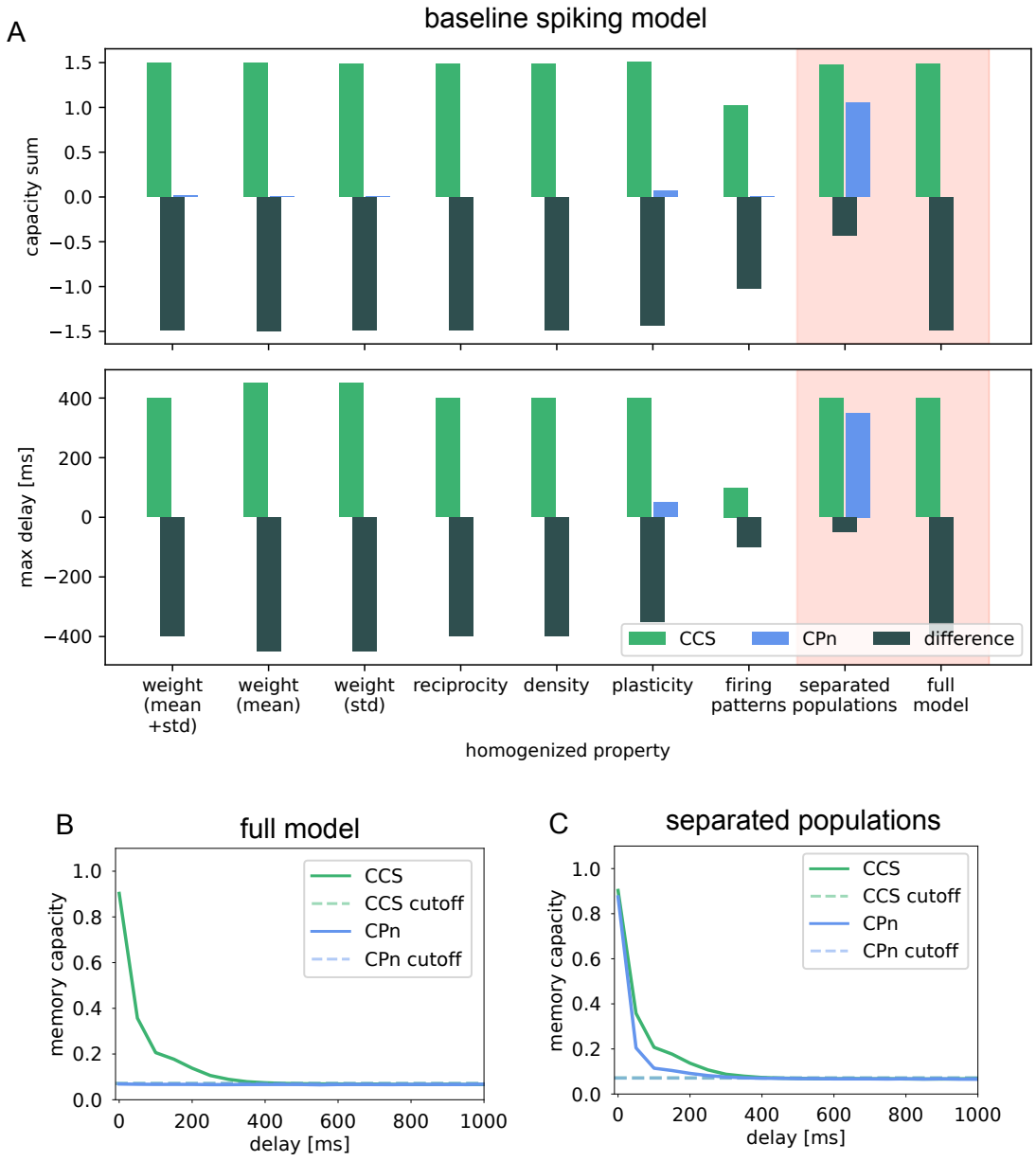


Figure 4.12: Modified network studies for baseline spiking network. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B:** Memory curves for the full model. **C:** Memory curves for the network with separated populations and input to both populations.

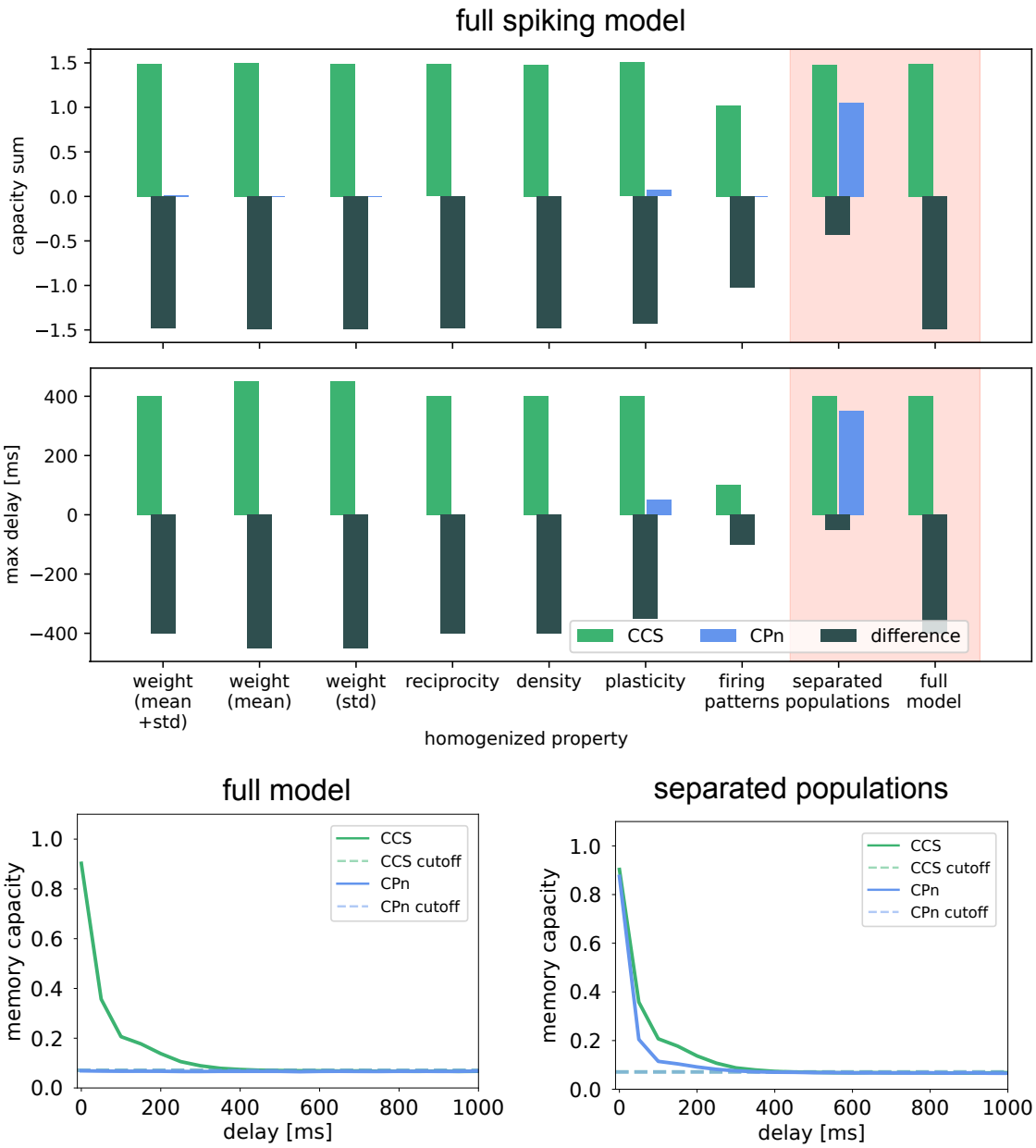


Figure 4.13: Modified network studies for full data-based spiking network. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B:** Memory curves for the full model. **C:** Memory curves for the network with separated populations and input to both populations.

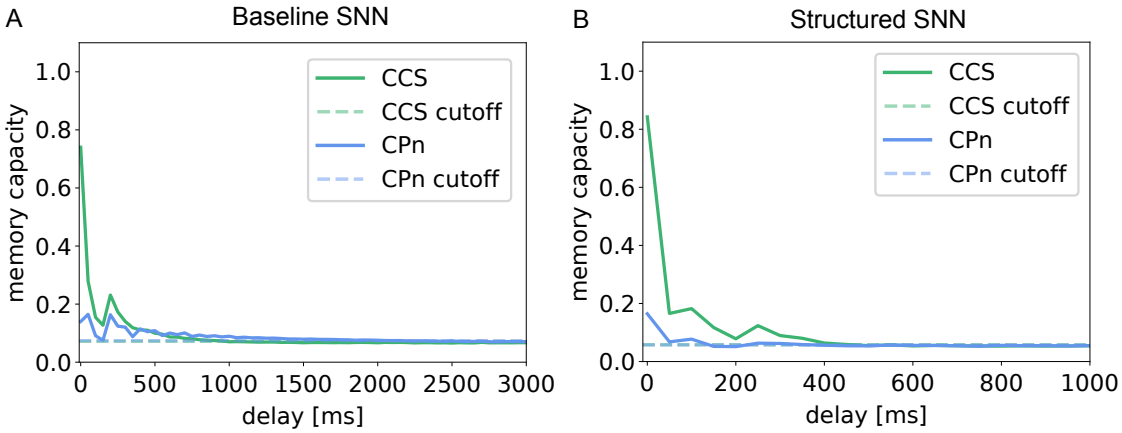


Figure 4.14: Modified network studies for spiking networks constructed from rate networks. **A:** Memory curves for the best configuration of a spiking network that is based on the baseline rate network ($\lambda = 20$). **B:** Memory curves for the best configuration of a spiking network that is based on the structured rate network ($\lambda = 40$).

networks, even the undelayed input signal can hardly be reconstructed from the CPn population state. The reconstruction of the undelayed signal in the CCS population is also not perfect. Although the memory capacity for all delays in the CPn population is very small, it remains above the random value threshold for more than two seconds in the baseline network. Thus, there appears to be a positive memory effect, but it is unlikely that these small memory capacities are sufficient to calculate a temporal difference error signal for reinforcement learning based on them.

4.4 Conclusion

In this chapter, we built on the methods developed in Chapter 2 and 3 to test the hypothesis proposed by Morita et al. (2012) on the computation of temporal difference errors in the brain by investigating the memory properties of a neural network model representing two populations of cortical layer 5 neurons: the crossed corticostriatal (CCS) cells and the corticopontine (CPn) cells. We collected data on neuronal and structural properties of these populations from the literature and implemented different network models. Similar to Chapter 3, we used progressively more complex network types as the basis for our models by starting with the study of continuous rate networks, then adapting them to follow Dale's principle in a second step, and finally moving on to spiking models. Following the approach from Chapter 2, we then created different control circuits based on these network types to investigate the effects of the individual

biological network properties. To evaluate the memory capabilities of the systems, we applied the linear part of the information processing capacity presented in Chapter 3 with the spatial input encoding scheme. Our experiments with continuous rate networks support the hypothesis of Morita et al. (2012) that these two populations of neurons can represent the current state (CCS population) and the previous state (CPn population) and therefore can be used as a basis for computing a temporal difference error in the brain. In our experiments, we found that the distribution of the weights, i.e. their mean and standard deviation, plays the most important role in the population difference in memory performance. Against other parameter changes, the results show a high robustness. In contrast, the spiking networks show different behavior. There was hardly any information transfer from the CCS neurons to the CPn population and their ability to maintain information about input signals is very limited. Even the method of Kim, Li, and Sejnowski (2019) to convert the earlier successfully tested rate networks into equivalent spiking counterparts did not result in satisfactory transfer and retention of inputs and therefore does not seem to work properly to transform untrained rate networks into spiking versions.

Part III

Discussion

Chapter 5

Discussion

The main focus of this thesis was to shed light on the information processing in neural network models as a step towards an understanding of the computations performed in the mammalian cortex. In Chapter 2 we built a model of a cortical column, analyzed its ability to solve tasks with different demands in non-linear processing and memory and tested which aspects of its connectivity structure play the most important role in the computations of the network. In order to deepen the analysis of this network model and facilitate analyses of further spiking networks in the future, Chapter 3 deals in detail with the information processing capacity. In this chapter, we first examine in increasingly complex steps which adjustments are necessary in order to be able to apply this metric not only to discrete-time dynamical systems but also to simple continuous-time systems and finally to spiking neural networks in such a way that it provides the most meaningful insights possible. In Chapter 4, we use the insights from the previous two chapters to investigate the hypothesis proposed by Morita et al. (2012) that two separate populations of neurons in layer 5 of the neocortex, the crossed corticostriatal (CCS) and corticopontine (CPn), respectively encode the current and previous state and thus provide the basis of biological temporal difference learning. Similar to Chapter 2, we have implemented network models based on biological data from the literature, initially based on rate neurons and finally also consisting of spiking units. Using the processing capacity from Chapter 3, we then examined the memory of these networks and structurally adapted control models to determine how structural and neural properties affect the ability of the two populations to maintain information.

5.1 Cortical microcircuit

In Chapter 2, we analyzed how the laminar structure of a cortical column model affects the computational capabilities of spiking neural networks. In a first step, we replicated the models and experiments described by Häusler and Maass (2007). The similarity in network activity and the degree histograms convincingly demonstrate the success of our replication, although the task results were not absolutely identical. Our findings on the tasks defined in the original study validate their key result. Specifically, we confirm that the degree distribution exhibited by the laminar structure of the data-

based circuit confers a computational advantage over circuits with modified connectivity patterns that destroy the laminar connectivity whilst maintaining the global statistics of the network. We reach this conclusion by training readout weights to solve tasks based on spike patterns and firing rates that require linear and non-linear computations on two separate input signals and the memorization of prior information.

The microcircuit model at the heart of the project shares many properties with biological microcircuits. In addition to its data-based structure (based on intracellular recordings from rats and cats by Thomson et al. (2002)), it consists of Hodgkin-Huxley neurons with multiple different ion channel dynamics and a conductance-based background noise mechanism, and its synapses exhibit short-term plasticity. For further biological plausibility, its readouts receive only inputs restricted to layer 2/3 and layer 5 specific connections. The readout weights obey Dale's principle (Eccles, Fatt, and Koketsu, 1954): excitatory neurons contribute only positive values to the activity function of the readout neuron, and inhibitory neurons only negative values.

Given the superior performance of the data-based circuit, we formulated the hypothesis that the results should be robust with respect to the specifics of the neuron model. We extended the analysis of the original study by decreasing the complexity of the neuron model, first removing the intrinsic noise and then reducing the dynamics to that of an integrate-and-fire neuron. The results confirmed our hypothesis that neuron model details were not important to the key result: in both cases, the data-based circuit continued to exhibit superior performance over all other variants. Our results also rule out the possibility that a complex neuron model is necessary for the data-based circuit to reach a good performance since the two simpler neuron types tested outperform it in the majority of tasks.

To obtain a higher temporal resolution in the examination of the memory capabilities of the circuit variants, we additionally extended the original analysis to include retrospective classification of spike patterns of much shorter segments with a duration of 5 ms instead of 30 ms and classifying all of the segments rather than just the last two. Here we observe a stereotypical memory profile for all circuit types, where the data-based circuit beats the other networks in peak performance and summed reconstruction capability across all delays, with comparable performance for the degree-controlled circuit. However, there is no significant difference between the various networks when comparing the maximum delay up to which the signal can still be at least partially reconstructed. Thus, we conclude that the advantage of the laminar connectivity structure lies primarily in the clarity of the internal representation rather than in significantly longer information retention. These results also highlight the characteristic time scale of the input as a relevant parameter for determining the computational capacities of a spiking neural network.

In future work, the scientific community can use our NEST implementation of the data-based microcircuit model, which is now freely available to all researchers, to lay the groundwork for further experiments to investigate the computational properties of corti-

cal columns and to make quantitative comparisons with alternative microcircuit models. In this context it would also be reasonable to tune the network to obtain biologically more realistic long-tailed firing rate distributions with a mean below 1 spks/s instead of the comparatively high activity currently exhibited by the model (about 40 spks/s for layers 2/3 and 4). For the simplified network with integrate-and-fire neurons, this can probably be achieved by adjusting the firing threshold per population based on in-vivo data rather than using the activity of the original model as a basis. Similarly, for the network with Hodgkin-Huxley neurons, it is likely that population-level tuning of neuron parameters and probably adjusted scaling of recurrent weights will be required to achieve the intended firing rates.

From here it is also possible to add other biological details such as additional or different plasticity mechanisms, or to investigate the computational capacities of larger networks using this microcircuit as a basic building block for systems representing the meso- or macroscopic level. This could be achieved, for example, by adjusting the weight scaling parameters along with the network size and connecting multiple differently parameterized instances of the microcircuit using inter-area connectivity that is based on experimental findings.

5.2 Information processing capacity

The information processing capacity enables thorough investigations of dynamical systems in terms of the functions they can compute. It produces a comprehensive computational profile with intuitively interpretable indicators of complexity (polynomial degree) and required memory (maximum delay).

We explored ways of applying the information processing capacity to dynamical systems with increasing complexity, culminating in and focusing on biologically inspired spiking neural networks. Our initial experiments extend the analysis of the (discrete-time) ESN used in Dambre et al. (2012). By investigating the FPUT model, we expand the scope of our study to continuous-time systems, and then make the step into SNNs with a balanced random network. Finally, we apply the measure to the cortical column model developed in Chapter 2.

We evaluate the effects of different input parameterizations to provide a guide for future application of the capacity measure to similar systems and especially to spiking neural networks. Although the metric is highly informative, it can be computationally expensive and any restriction on the parameter search space drastically saves computational costs.

As reported in previous work such as Verstraeten et al. (2010) and Dambre et al. (2012), we found no single optimal parameterization that simultaneously maximizes nonlinearity and memory capacity. All dynamical systems show a trade-off between memory and nonlinear processing since their processing capacity is bounded by the

number of readout units. Therefore, for a given use case, one would have to tune the parameters appropriately to achieve optimal results. For continuous-time systems, an increase in step duration is accompanied by a shift to the nonlinear regime. Our explanation is that the systems have more time to transform a single input step, but in return, they also need to retain information for a longer period to process previous signals. The ESN has a small parameter range that results in particularly long memory. In this range, the input scaling is so small that the signal is transformed mainly by the linear part of the tanh activation function, while a spectral radius slightly larger than one ensures a strong influence of previous inputs.

A reduction of capacity by introducing noise into the dynamical system has previously been reported by Dambre et al. (2012). Consistent with these findings, we also find that sources of randomness commonly used in spiking neural networks, such as driving the network with Poisson spike trains and encoding the inputs as firing rates of such spike trains, reduce the processing capacity. We, therefore, propose to first operate the systems deterministically using frozen background noise and direct current as inputs, and later to analyze the robustness to random perturbations as a separate property.

While the ESN for most parameters has near-maximum capacity, there are both high and low capacity configurations for the FPUT. In contrast, the SNNs achieve only a fraction of the possible capacity. We note that in our study, the information processing capacity or different aspects of it, such as the maximum degree and delay, generally show strong correlations with task performance. Whereas even systems whose processing capacity is low can solve tasks such as XOR or tXOR almost perfectly, higher capacity systems such as the ESN have a clear advantage for more demanding tasks such as XORXOR.

An explanation for the low capacity of SNNs may be the frequent reset of the membrane potential after a spike, especially since we use the membrane potential as a state variable for the readout. If these results are indicative of the computational power of biological neural networks, and under the assumption that the information processing capacity captures a large portion of the relevant information processing that dynamical systems, including the brain, can perform, this suggests a limitation. It indicates that the reservoir computing approach may not be an effective tool for determining, or a model for understanding, the computational capabilities of cortical systems.

To prevent a distortion of the capacity results we must encode the signal linearly. With nonlinear encoding, systems can reconstruct polynomial functions by remembering the nonlinear inputs over several time steps without transforming them. Thus, we advise to use only linear encoders if possible.

However, there are reasons to use nonlinear inputs such as the desire for biological realism in models in computational neuroscience. Therefore, we presented a procedure to remove nonlinear encoder effects and provide a lower bound on the main system's actual capacity. The procedure can also handle encoders with memory. This enables, for example, to analyze specific parts of larger systems, for example models of the brain, sep-

arately. For this purpose, we consider all components feeding signals into the subsystem as part of the encoder and remove their capacity from the results.

Overall, we have laid the foundation for detailed analyses of various systems, encompassing discrete and continuous-time systems as well as biologically inspired neural networks. This method can now be used, for example, to test computational hypotheses on spiking neural networks and even in-vitro experiments, or to optimize the parameter configuration and input encoding of neuromorphic hardware for a given computational goal.

5.3 Temporal difference learning in cortico-striatal populations

After having mainly dealt with the information processing of a scaled-down model of an entire cortical column among other models in Chapter 2 and Chapter 3, we have investigated a more granular subset of neurons in cortical layer 5 in Chapter 4. This part of the doctoral thesis deals with the hypothesis of Morita et al. (2012), which states that the activities of the crossed corticostriatal (CCS) and corticopontine (CPn) cells in layer 5 of the cortex represent the current and previous states, respectively, and are thus fundamental for the calculation of a reward prediction error downstream in the basal ganglia and dopaminergic neurons. Therefore, it must be possible to read out current inputs to the CCS population from this group of neurons and to extract previous signals that were transferred from the CCS to the CPn population from the activity of the CPn neurons. Thus, there should be a signal transmission from the CCS to the CPn population and the CPn population must be able to store this information over a sustained period of time. To test this hypothesis, we created data-based models as in Chapter 2 and measured their linear memory with the information processing capacity from Chapter 3. We started with continuous rate networks and initially focused only on the connectivity properties of the populations in order to start with models of lower complexity, as in Chapter 3, and then gradually move on to more complex systems. As a first step, we tested different activation functions and found that the use of the tanh function, which is not particularly biologically plausible due to its partly negative outputs, leads to significantly longer memory, especially in the CPn population, than the other non-linearities tested. Overall, the results of these experiments suggest that for the persistent maintenance of input information, it is important that the utilized part of the activation function is approximately linear and that the slope in combination with the connectivity matrix yields the appropriate spectral radius close to 1. In the next step, we used the data-based model as a starting point and, following the procedure in Chapter 2, created control circuits in which certain properties were systematically adapted in order to determine their influence on memory in both populations. This showed that the population-specific distributions of the synaptic weights, i.e. their mean and standard deviations, have the decisive effect on the measured memory, both

when using the tanh function as non-linearity and the consistently positive and thus biologically more plausible sigmoid function. Other factors such as the reciprocity and density of the connections only marginally influence the results. In general, however, it can be seen that the information in all tested configurations is retained longer in the CPn population than in the CCS neurons, as assumed by the hypothesis.

The next set of experiments divides the populations of the network into excitatory and inhibitory subpopulations according to Dale's principle. Since this division does not make sense for the tanh activation function, because even neurons with positive (negative) output weights can have an inhibitory (excitatory) effect on subsequent neurons due to the partially negative output values, we restricted ourselves to the sigmoid non-linearity in these models. The first part of these experiments showed that the correct ratio between the strength of inhibitory weights and the proportion of inhibitory neurons in the network plays a decisive role for the memory capacity. In particular, configurations that ensure a balance between excitation and inhibition exhibit an increased memory performance. Also for this well-balanced network, we created control circuits whose analysis, similar to the systems in which we did not differentiate between excitatory and inhibitory populations, showed that the weight distributions in particular have a prominent influence on memory retention. In addition, here it also became apparent that the CPn population represents previous states better than the CCS neurons. Thus, the results of the rate networks can be seen as a reinforcement of the hypothesis of Morita et al. (2012), regardless of whether we use separate excitatory and inhibitory subpopulations or not.

In order to further increase the biological plausibility of our models and to better address the characteristics of the neurons themselves, we next investigated networks of spiking units and extended them to include plastic synapses and population-specific adaptation behavior. Again, we first examined a homogeneous network and its control circuits and then moved on to networks with a separation between excitatory and inhibitory units. However, in both cases, there was hardly any transfer of information between CCS and CPn and only a direct signal input into the CPn population made it possible to extract information from this population. Moreover, the reconstruction of the unmodified input signal in the CCS population was already not optimal. Based on these results, we transformed our previously successfully investigated rate models into spiking equivalents via the method presented by Kim, Li, and Sejnowski (2019). However, these models also showed little improvement in information transfer between the populations. Even though the memory curve of the CPn population had a long tail with values just above the threshold, such a minimal reconstruction accuracy is not sufficient to be used as a basis for calculating reward prediction errors. The method of Kim, Li, and Sejnowski (2019) therefore does not appear to be fully applicable to all types of networks, even if the same parameters are used as in the original publication for the rate and spiking networks. One reason for this could be the difference in the network sizes used. Kim, Li, and Sejnowski (2019) test networks with a maximum size of 400 units,

but use 250 neurons in the majority of the experiment because this leads to the smallest deviation between the performance of spiking and rate networks. The networks we use are an order of magnitude larger. In addition, unlike those of Kim, Li, and Sejnowski (2019), our connections are not trained to solve a specific task but are comparatively unstructured and only defined by random distributions. It is therefore possible that these specific network structures resulting from the training are an important factor for the success of this transformation from rate networks to functionally equivalent spiking networks.

In general, the results from Chapter 4 are a bit ambiguous. The rate network experiments support the hypothesis that the activity of the CCS and CPn populations can be used as a basis for calculating a reward prediction error. However, we cannot say the same for the SNN experiments. Despite strict adherence to biological conditions and a later less biologically inspired transformation from rate to spiking networks, the SNN did not show an adequate memory capacity. This suggests that we might have overlooked key aspects of computation in biological neural systems and need to investigate potentially important properties in future work.

5.4 Outlook and future work

There are several directions in which research could be conducted to reduce the discrepancy between the comparatively poor performance of spiking network models and that of biological neural systems. On the one hand, it may be necessary to integrate other biological properties that are fundamentally important for calculations with spikes into the models. For example, previous work has shown that heterogeneity in synaptic and especially neuronal parameters can have a positive effect on the ability to process information (Duarte and Morrison, 2019). Furthermore, in reality, neurons are not dimensionless points but have a complex morphology that can enable even a single neuron to perform complex computations. One example is the processing in dendrites and the associated generation of long-lasting depolarizations through the activation of glutamate-sensitive N-methyl-D-aspartate (NMDA) receptors, the so-called NMDA spikes. In general, there are also many other ion channels and receptors that are ignored here and in most other studies on spiking neuronal networks in order to reduce complexity. There is also the possibility that non-neuronal glial cells make an important contribution to computations in the brain or that various often neglected synaptic connection properties such as gap junctions, which can bridge the gap between cell membranes, can have a significant influence. In addition to properties such as the density of connections between populations, more localized connection motifs (in addition to reciprocity) can also have an impact on the processes in the networks (Häusler, Schuch, and Maass, 2009; Perin, Berger, and Markram, 2011; Song et al., 2005). Of course, it is also possible that the spiking network-specific computational advantages only occur in systems that have

a critical minimum size that is significantly larger than the ones we tested. However, there are biological organisms that already require considerably smaller nervous systems of spiking neurons. For example, the nematode Caenorhabditis elegans has only 302 or 385 neurons, depending on sex (Jarrell et al., 2012; Sammut et al., 2015). In most cases, however, an increase in biological plausibility is also accompanied by an increase in complexity and the number of free parameters in the system. On the one hand, this makes it more complex to simulate these systems and, on the other hand, it can reduce the interpretability and make it more difficult to derive functional concepts from the experiments with these networks. At a certain point, the models lose the advantages that modeling is supposed to bring.

Considering these potential problems in bringing the models closer to biology, research based on this thesis could go in a different direction and further adapt the measurement methods, such as the information processing capacity, to biology instead of extending the systems under investigation. The encodings we use, including those based on spike trains, convert the signals in such a way that the information is not contained in the precise spike times, but in the firing rates. It is therefore plausible that systems that are fundamentally based on the processing of firing rates can process the information encoded in this way more effectively than those that react differently to differently timed spikes. Moreover, there is good reason to believe that communication in the brain is not fundamentally based on firing rates (Brette, 2015). In the future, encoding methods that make use of the timing of the spikes should therefore be tested. For example, temporal kernels could be used instead of spatial coding (see Chapter 3) or for each input neuron long spike trains could be pre-generated, of which a different section serves as a spike pattern depending on the input value (see Figure 5.1 A). By spreading and compressing pre-generated spike trains, it is also possible to create an encoding that carries the information both in the rates and in the precise spike times (see Figure 5.1 B). To use an encoding with a constant rate, two spike trains per input neuron can be pre-generated. Then the first spike train is warped using the normal signal (e.g., 0.7) and the second using the inverted signal (e.g., -0.7). These resulting spike trains are then combined into one. These temporal encoding methods can of course also be combined with the spatial encoding scheme.

Another possible point of improvement could be that the biological plausibility of the input signal itself could also be increased, as it is unlikely that the brain often has to directly process floating point numbers in the range between -1 and +1. Therefore, it can be further investigated how sounds, visual impressions or other stimuli based on the sensory system of humans or other animals can be used as input signals. In connection with the information processing capacity, a continuous transition is required. For example, the individual images of a video could represent values between -1 and +1. This initially only enables a discrete resolution of the signal, as the number of images is limited. However, videos with a high number of images can mitigate this effect and it can even be completely avoided by interpolating between the individual frames.

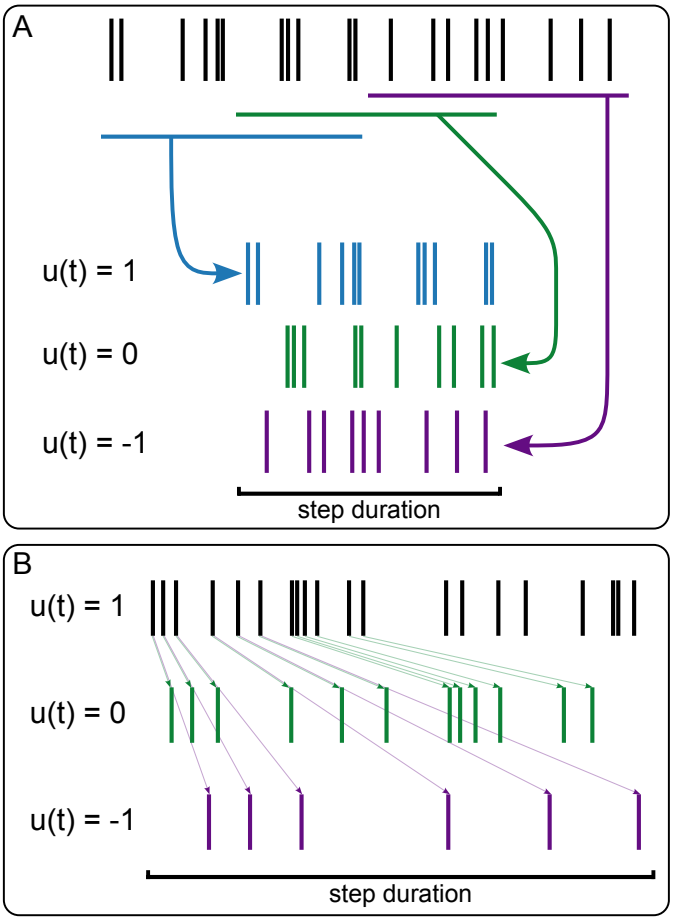


Figure 5.1: **Encodings with precise spike times. A:** Input encoding with a constant rate (on average). A window with the duration of one step (colored horizontal bars) can be moved across a fixed pre-generated spike train (black) to select the subsection that is used as input. The position of the window is defined by the input $u(t)$. **B:** Input encoding with information in the rate and precise spike times. A spike train with the maximum firing rate (black spikes) is pre-generated. Depending on the input value $u(t)$, the spikes times are stretched (multiplied with a corresponding factor) and all spikes that are outside the step duration window are cut off. This results in the final input spike trains (colored spikes).

Another similar possibility would be the rendering of a 3D model in which a parameter is changed according to the input signal. A slightly more flexible method could be to traverse the latent state space of trained variational autoencoders (VAEs) or generative adversarial networks (GANs) according to the input signal to generate images (Klys, Snell, and Zemel, 2018; Shen and Zhou, 2021; Shen et al., 2022; Winant, Schreurs, and Suykens, 2021), sounds (Madhu and Kumaraswamy, 2019) or speech (Saito, Takamichi, and Saruwatari, 2018) that differ only in one continuously changing property. You could also use an audio signal whose pitch is changed according to the input value. However, these stimuli should still be encoded as biologically inspired as possible. For example, one can use the method of Smith and Lewicki (2006) for encoding audio and a retina model (Kenyon et al., 2003) for the encoding of visual stimuli. Even if such inputs are much closer to stimuli occurring in biology, these encodings are very non-linear and therefore influence the result of the information processing capacity. If the focus is solely on memory, as in Chapter 4, this is not a problem. Otherwise, the method presented in Chapter 3 must be used to subtract the encoder capacities. However, in its current form, this method only provides a lower bound of the capacity. Therefore, further research could also develop a more precise removal of the encoder effects that includes additional metrics gathered during the reconstruction of the individual target functions.

In addition to these suggestions for adapting the input encoding, it is also possible to make adjustments to the state variable for the linear readout. In order to rely on precise spike times here too, the relative time of the first (or last) spike within an input step could be used as a state variable. With low firing rates and comparatively short step durations, the probability of more than one spike occurring per step is very low and therefore very little information would be lost. The information about the first spike time can also be used, for example, in backpropagation-based learning algorithms to solve complex tasks (Göltz et al., 2021). Of course, the state could also be extended to the times of the first two or more spikes (for example, to reflect bursts). However, this would lead to a significantly larger dimension of the state space and thus require more inputs (to minimize the noise cut-off), longer simulations and thus more computing power and time.

In general, however, experiments could also be carried out to find out to what extent a state matrix that has been reduced in size through different dimensionality reduction mechanisms can lead to meaningful results for spiking neural networks. The possibility of reducing the size of the state space would be particularly desirable with regard to significantly larger network models in order to reduce the number of simulation steps required.

5.5 Conclusion

In conclusion, in this doctoral thesis we have achieved many of the goals set out in Chapter 1. In addition to confirming and consolidating previous research results by reproducing them in Chapter 2, we also have gained new insights into information processing in cortical networks. For example, we have shown that the connectivity structures inside a cortical column do not extend the maximum duration of information retention, as suggested by Häusler and Maass (2007), but rather sharpen the clarity of internal representations. We were also able to confirm in Chapter 4, at least in part by our experiments with rate networks, that the specific properties of cortical CCS and CPn populations give rise to the memory characteristics required for the computation of temporal difference errors in the brain. However, most importantly, we have laid the foundation for further research. The reimplementation of the microcircuit model using the highly performant NEST simulator (Hahne et al., 2021), which is constantly being developed further by an active community, and the extension and publication of the model also enable other research groups to carry out further experiments and give them the possibility to extend the microcircuit model further. With the work in Chapter 3, we have created even more extensive possibilities for future research into the analysis of computational processes in the brain by working out which adjustments are needed to apply the information processing capacity to spiking neuronal networks. This paves the way for the future creation and analysis of detailed profiles of the computations performed by biology-inspired dynamical systems.

Bibliography

Čapek, Karel (1920). *RUR-Rossum's Universal Robots: Rossumovi univerzln roboti*. Aventinum.

Akiba, Takuya et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Appeltant, Lennert et al. (2011). "Information processing using a single dynamical node as complex system". In: *Nature communications* 2.1, p. 468.

Aristotle (1911). *De partibus animalium*. Trans. by William Ogle. Oxford: Clarendon Press.

Atiya, A.F. and A.G. Parlos (2000). "New results on recurrent network training: unifying the algorithms and accelerating convergence". In: *IEEE Transactions on Neural Networks* 11.3, pp. 697–709. DOI: 10.1109/72.846741.

Babu, Pooja Nagendra et al. (June 2021). "NESTML 4.0". Version 4.0. In: DOI: 10.5281/zenodo.4740083. URL: https://doi.org/10.5281/zenodo.4740083.

Bayer, Hannah M and Paul W Glimcher (2005). "Midbrain dopamine neurons encode a quantitative reward prediction error signal". In: *Neuron* 47.1, pp. 129–141.

Belliveau, Jack W et al. (1991). "Functional mapping of the human visual cortex by magnetic resonance imaging". In: *Science* 254.5032, pp. 716–719.

Bellman, Richard (1957). *Dynamic Programming*. Dover Publications. ISBN: 9780486428093.

Benureau, Fabien C. Y. and Nicolas P. Rougier (2018). "Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions". In: *Frontiers in Neuroinformatics*. ISSN: 1662-5196. DOI: 10.3389/fninf.2017.00069. URL: https://www.frontiersin.org/article/10.3389/fninf.2017.00069.

Berger, Hans (1929). "Über das elektroenkephalogramm des menschen". In: *Archiv für psychiatrie und nervenkrankheiten* 87.1, pp. 527–570.

Billeh, Yazan N. et al. (2020). "Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex". In: *Neuron* 106.3, 388–403.e18. ISSN: 08966273. DOI: 10.1016/j.neuron.2020.01.040. URL: https://linkinghub.elsevier.com/retrieve/pii/S0896627320300672.

Birkhoff, George David (1927). *Dynamical systems*. Vol. 9. American Mathematical Soc.

Bono, James J (1984). "Medical spirits and the medieval language of life". In: *Traditio* 40, pp. 91–130.

Brette, Romain (2015). "Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain". In: *Frontiers in Systems Neuroscience* 9. ISSN: 1662-5137. URL: https://www.frontiersin.org/articles/10.3389/fnsys.2015.00151 (visited on 11/21/2023).

Brette, Romain and Wulfram Gerstner (Nov. 2005). "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". en. In: *Journal of Neurophysiology* 94.5, pp. 3637–3642. ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.00686.2005. URL: https://www.physiology.org/doi/10.1152/jn.00686.2005 (visited on 04/03/2023).

Brunel, Nicolas (2000). "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons". In: *Journal of computational neuroscience* 8, pp. 183–208.

Bürger, Jens et al. (July 2015). "Hierarchical composition of memristive networks for real-time computing". en. In: *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH´15)*. Boston, MA, USA: IEEE, pp. 33–38. ISBN: 978-1-4673-7849-9. DOI: 10.1109/NANOARCH.2015.7180583. URL: http://ieeexplore.ieee.org/document/7180583/ (visited on 10/11/2023).

Butler, Samuel (1872). *Erewhon, or, Over the range*. London: Trübner & Co. URL: https://archive.org/details/ErewhonOverrang00Butl/page/ii/mode/2up.

Castaldi, Elisa et al. (2020). "Neuroplasticity in adult human visual cortex". In: *Neuroscience & Biobehavioral Reviews* 112, pp. 542–552.

Catani, Marco and Stefano Sandrone (2015). *Brain renaissance: from Vesalius to modern neuroscience*. Oxford University Press.

Chen, Jiayin et al. (2020). "Temporal Information Processing on Noisy Quantum Computers". In: *Phys. Rev. Appl.* 14 (2), p. 024065. DOI: 10.1103/PhysRevApplied.14.024065. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.14.024065.

Cobb, Matthew (2002). "Exorcizing the animal spirits: Jan Swammerdam on nerve function". In: *Nature Reviews Neuroscience* 3.5, pp. 395–400.

– (2020). *The idea of the brain: The past and future of neuroscience*. Hachette UK.

Coulombe, Jean C. et al. (June 2017). "Computing with networks of nonlinear mechanical oscillators". en. In: *PLOS ONE* 12.6. Ed. by Gennady Cymbalyuk, e0178663. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0178663. URL: https://dx.plos.org/10.1371/journal.pone.0178663 (visited on 10/11/2023).

Dale, Kyran and Phil Husbands (Jan. 2010). "The Evolution of Reaction-Diffusion Controllers for Minimally Cognitive Agents". In: *Artificial Life* 16.1, pp. 1–19. ISSN: 1064-5462. DOI: 10.1162/artl.2009.16.1.16100. eprint: https://direct.mit.edu/artl/article-pdf/16/1/1/1662621/artl.2009.16.1.16100.pdf. URL: https://doi.org/10.1162/artl.2009.16.1.16100.

Dambre, Joni et al. (2012). "Information processing capacity of dynamical systems". In: *Scientific reports* 2.1, pp. 1–7.

DeFelipe, Javier (June 2012). "The neocortical column". In: *Frontiers in Neuroanatomy* 6, p. 22. ISSN: 16625129. DOI: 10.3389/fnana.2012.00022. URL: http://journal.frontiersin.org/article/10.3389/fnana.2012.00022/abstract.

Denève, Sophie and Christian K Machens (2016). "Efficient codes and balanced networks". In: *Nature neuroscience* 19.3, pp. 375–382.

Descartes, René (1994). *Tractatus de homine*. apud Danielem Elseverium.

Destexhe, A et al. (Nov. 2001). "Fluctuating synaptic conductances recreate in vivo-like activity in neocortical neurons". In: *Neuroscience* 107.1, pp. 13–24. ISSN: 03064522. DOI: 10.1016/S0306-4522(01)00344-X. URL: https://www.sciencedirect.com/science/article/pii/S030645220100344X.

Destexhe, Alain and Denis Paré (1999). "Impact of network activity on the integrative properties of neocortical pyramidal neurons in vivo". In: *Journal of neurophysiology* 81.4, pp. 1531–1547.

Doya, Kenji (2002). "Metalearning and neuromodulation". In: *Neural Networks* 15.4, pp. 495–506. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(02)00044-8. URL: https://www.sciencedirect.com/science/article/pii/S0893608002000448.

Duarte, Renato and Abigail Morrison (Apr. 2019). "Leveraging heterogeneity for neural computation with fading memory in layer 2/3 cortical microcircuits". In: *PLOS Computational Biology* 15.4, pp. 1–43. DOI: 10.1371/journal.pcbi.1006781. URL: https://doi.org/10.1371/journal.pcbi.1006781.

Duarte, Renato et al. (2018). "Encoding symbolic sequences with spiking neural reservoirs". In: *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489114.

Duarte, Renato et al. (2021). "Functional Neural Architectures". In: DOI: 10.5281/zenodo.5752597.

Duport, François et al. (Sept. 2012). "All-optical reservoir computing". In: *Optics Express* 20.20, p. 22783. ISSN: 1094-4087. DOI: 10.1364/OE.20.022783. URL: https://www.osapublishing.org/oe/abstract.cfm?uri=oe-20-20-22783.

Eccles, J. C. et al. (1954). "Cholinergic and Inhibitory Synapses in a Pathway from Motor-Axon Collaterals to Motoneurones". In: *The Journal of Physiology* 126.3, pp. 524–562. ISSN: 00223751. DOI: 10.1113/jphysiol.1954.sp005226. URL: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1954.sp005226.

Eliot, George (1879). *Impressions of theophrastus such*. Collection of british and american authors. tex.lccn: 07000301. Harper & brothers. ISBN: 978-1-4142-8858-1. URL: https://books.google.cg/books?id=1xIOAAAAYAAJ.

Fermi, E. et al. (1955). *studies of the nonlinear problems I*. Tech. rep. LA-1940. Los Alamos National Lab. (LANL), Los Alamos, NM (United States). DOI: 10.2172/4376203. URL: https://www.osti.gov/biblio/4376203.

Fernando, Chrisantha and Sampsa Sojakka (2003). "Pattern Recognition in a Bucket". In: *Advances in Artificial Life*. Ed. by Wolfgang Banzhaf et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 588–597. ISBN: 978-3-540-39432-7.

Finger, Stanley (2001). *Origins of neuroscience: a history of explorations into brain function*. Oxford University Press, USA.

Fourcaud-Trocmé, Nicolas et al. (Dec. 2003). "How Spike Generation Mechanisms Determine the Neuronal Response to Fluctuating Inputs". en. In: *The Journal of Neuroscience* 23.37, pp. 11628–11640. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.23-37-11628.2003. URL: https://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.23-37-11628.2003.

Göltz, J. et al. (Sept. 2021). "Fast and energy-efficient neuromorphic deep learning with first-spike times". en. In: *Nature Machine Intelligence* 3.9. Number: 9 Publisher: Nature Publishing Group, pp. 823–835. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00388-x. URL: https://www.nature.com/articles/s42256-021-00388-x (visited on 11/29/2023).

Gerstner, Wulfram et al. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

Ghosh, Sanjib et al. (Apr. 2019). "Quantum reservoir processing". en. In: *npj Quantum Information* 5.1, p. 35. ISSN: 2056-6387. DOI: 10.1038/s41534-019-0149-8. URL: https://www.nature.com/articles/s41534-019-0149-8 (visited on 10/10/2023).

Glimcher, Paul W (2011). "Understanding dopamine and reinforcement learning: the dopamine reward prediction error hypothesis". In: *Proceedings of the National Academy of Sciences* 108.supplement_3, pp. 15647–15654.

Golgi, C (1873). "Sulla struttura della sostanza grigia del cervelo. Gazzetta Medica Italiana". In: *Lombardia* 33, p. 244.

Gray, Henry (2000). *Anatomy of the human body, by Henry Gray. 20th ed., thoroughly rev. and re-edited by Warren H. Lewis.* Vol. 8. Lea & Febiger, 1918. URL: www.bartleby.com/107/.

Grollier, Julie et al. (2020). "Neuromorphic spintronics". In: *Nature electronics* 3.7, pp. 360–370.

Habenschuss, Stefan et al. (20013). "Stochastic Computations in Cortical Microcircuit Models". In: *PLoS Computational Biology* 9.11. Ed. by Olaf Sporns, e1003311. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003311. URL: https://dx.plos.org/10.1371/journal.pcbi.1003311 (visited on 08/10/2022).

Hahne, Jan et al. (2021). "NEST 3.0". In: DOI: 10.5281/zenodo.4739103.

Harris, Kenneth D. and Gordon M.G. Shepherd (Jan. 2015). "The neocortical circuit: Themes and variations". In: *Nature Neuroscience* 18 (2), pp. 170–181. ISSN: 15461726. DOI: 10.1038/nn.3917.

Häusler, Stefan and Wolfgang Maass (Feb. 2007). "A Statistical Analysis of Information-Processing Properties of Lamina-Specific Cortical Microcircuit Models". In: *Cerebral Cortex* 17.1, pp. 149–162. ISSN: 1047-3211. DOI: 10.1093/cercor/bhj132. eprint: https://academic.oup.com/cercor/article-pdf/17/1/149/797744/bhj132.pdf. URL: https://doi.org/10.1093/cercor/bhj132.

Häusler, Stefan et al. (2009). "Motif Distribution, Dynamical Properties, and Computational Performance of Two Data-Based Cortical Microcircuit Templates". In: *Journal of Physiology-Paris* 103.1-2, pp. 73–87. ISSN: 09284257. DOI: 10.1016/j.jphysparis.2009.05.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0928425709000266.

Hebb, Donald O. (June 1949). *The organization of behavior: A neuropsychological theory.* New York: Wiley. ISBN: 0-8058-4300-0.

Hippocrates (1868). *On the Sacred Disease.* Trans. by Charles Darwin Adams. Oxford: Clarendon Press. URL: http://www.perseus.tufts.edu/hopper/text?doc=urn:cts:greekLit:tlg0627.tlg027.perseus-eng1:1.

Hodgkin, Alan L and Andrew F Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4, p. 500.

Horton, Jonathan C and Daniel L Adams (Apr. 2005). "The cortical column: a structure without a function." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 360 (1456), pp. 837–862. ISSN: 0962-8436. DOI: 10.1098/rstb.2005.1623. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1569491&tool=pmcentrez&rendertype=abstract.

Izhikevich, Eugene M (2007). *Dynamical systems in neuroscience*. MIT Press.

Jaeger, Herbert (2001a). "Short term memory in echo state networks". In: URL: https://publica.fraunhofer.de/handle/publica/291107.

– (2001b). "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note". In: 148.34, p. 13.

Jarrell, Travis A. et al. (2012). "The Connectome of a Decision-Making Neural Network". In: *Science* 337.6093, pp. 437–444. DOI: 10.1126/science.1221762. eprint: https://www.science.org/doi/pdf/10.1126/science.1221762. URL: https://www.science.org/doi/abs/10.1126/science.1221762.

Joel, Daphna et al. (2002). "Actor–critic models of the basal ganglia: new anatomical and computational perspectives". In: *Neural Networks* 15.4, pp. 535–547. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(02)00047-3. URL: https://www.sciencedirect.com/science/article/pii/S0893608002000473.

Kaiser, Marcus and Claus C. Hilgetag (Mar. 2004). "Spatial growth of real-world networks". In: *Physical Review E* 69.3, p. 036103. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.69.036103. URL: https://link.aps.org/doi/10.1103/PhysRevE.69.036103.

Kawato, Mitsuo and Kazuyuki Samejima (2007). "Efficient reinforcement learning: computational theories, neuroscience and robotics". In: *Current Opinion in Neurobiology* 17.2. Cognitive neuroscience, pp. 205–212. ISSN: 0959-4388. DOI: https://doi.org/10.1016/j.conb.2007.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0959438807000372.

Kenyon, Garrett T. et al. (Sept. 2003). "A model of high-frequency oscillatory potentials in retinal ganglion cells". en. In: *Visual Neuroscience* 20.5, pp. 465–480. ISSN: 0952-5238, 1469-8714. DOI: 10.1017/S0952523803205010. URL: https://www.cambridge.org/core/product/identifier/S0952523803205010/type/journal_article (visited on 11/29/2023).

Kim, Robert et al. (2019). "Simple framework for constructing functional spiking recurrent neural networks". In: *Proceedings of the national academy of sciences* 116.45, pp. 22811–22820.

Klys, Jack et al. (Dec. 2018). "Learning latent subspaces in variational autoencoders". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., pp. 6445–6455. (Visited on 11/21/2023).

López-Muñoz, Francisco et al. (Oct. 2006). "Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal". en. In: *Brain Research Bulletin* 70.4-6, pp. 391–405. ISSN: 03619230. DOI: 10.1016/j.brainresbull.2006.07.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0361923006002334 (visited on 10/31/2023).

Lapicque, L (1907). "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation". In: *J Physiol Pathol Gen* 9, pp. 620–635.

Larger, Laurent et al. (2012). "Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing". In: *Optics express* 20.3, pp. 3241–3249.

Lauterbur, Paul C (1973). "Image formation by induced local interactions: examples employing nuclear magnetic resonance". In: *nature* 242.5394, pp. 190–191.

Lefèvre, Julien and Jean-François Mangin (Apr. 2010). "A Reaction-Diffusion Model of Human Brain Development". In: *PLOS Computational Biology* 6.4, pp. 1–10. DOI: 10.1371/journal.pcbi.1000749. URL: https://doi.org/10.1371/journal.pcbi.1000749.

Levien, R. B. and S. M. Tan (Nov. 1993). "Double pendulum: An experiment in chaos". In: *American Journal of Physics* 61.11, pp. 1038–1044. DOI: 10.1119/1.17335. URL: https://doi.org/10.1119/1.17335.

Lugnan, Alessio et al. (2020). "Photonic neuromorphic information processing and reservoir computing". In: *APL Photonics* 5.2, p. 020901.

Maass, Wolfgang (2014). "Noise as a Resource for Computation and Learning in Networks of Spiking Neurons". In: *Proceedings of the IEEE* 102.5, pp. 860–880. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2014.2310593. URL: http://ieeexplore.ieee.org/document/6797856/.

Maass, Wolfgang and Henry Markram (Mar. 2002). "Synapses as dynamic memory buffers". In: *Neural Networks* 15.2, pp. 155–161. ISSN: 08936080. DOI: 10.1016/S0893-6080(01)00144-7. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608001001447 (visited on 10/27/2021).

Maass, Wolfgang et al. (2002). "Real-time computing without stable states: A new framework for neural computation based on perturbations". In: *Neural computation* 14.11, pp. 2531–2560.

Madhu, Aswathy and Suresh Kumaraswamy (Sept. 2019). "Data Augmentation Using Generative Adversarial Network for Environmental Sound Classification". In: *2019 27th European Signal Processing Conference (EUSIPCO)*. ISSN: 2076-1465, pp. 1–5. DOI: 10.23919/EUSIPCO.2019.8902819. URL: https://ieeexplore.ieee.org/abstract/document/8902819 (visited on 11/22/2023).

Mainen, Zachary F et al. (1995). "A Model of Spike Initiation in Neocortical Pyramidal Neurons". In: *Neuron* 15.6. Publisher: Cell Press, pp. 1427–1439. ISSN: 0896-6273. DOI: 10.1016/0896-6273(95)90020-9. URL: http://dx.doi.org/10.1016/0896-6273(95)90020-9.

Markram, Henry et al. (2015). "Reconstruction and Simulation of Neocortical Microcircuitry". In: *Cell* 163.2, pp. 456–492. ISSN: 00928674. DOI: 10.1016/j.cell.2015.09.029. URL: https://linkinghub.elsevier.com/retrieve/pii/S0092867415011915.

Masse, Nicolas Y et al. (2019). "Circuit mechanisms for the maintenance and manipulation of information in working memory". In: *Nature neuroscience* 22.7, pp. 1159–1167.

May, Robert M (1976). "Simple mathematical models with very complicated dynamics". In: *Nature* 261.5560, pp. 459–467.

McCarthy, John et al. (1955). "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955". In: *AI magazine* 27.4, pp. 12–12.

McClure, Samuel M et al. (2003). "Temporal prediction errors in a passive learning task activate human striatum". In: *Neuron* 38.2, pp. 339–346.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5, pp. 115–133.

McDonnell, Mark D. and Lawrence M Ward (2011). "The Benefits of Noise in Neural Systems: Bridging Theory and Experiment". In: *Nature Reviews Neuroscience* 12.7, pp. 415–425. ISSN: 1471-003X. DOI: 10.1038/nrn3061. pmid: 21685932. URL: http://dx.doi.org/10.1038/nrn3061.

McDougal, Robert A et al. (2017). "Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience". In: *Journal of computational neuroscience* 42.1, pp. 1–10.

Metropolis, Nicholas and Stanislaw Ulam (1949). "The monte carlo method". In: *Journal of the American statistical association* 44.247, pp. 335–341.

Mikula, Shawn et al. (2007). "Internet-enabled high-resolution brain mapping and virtual microscopy". In: *Neuroimage* 35.1, pp. 9–15.

Miller, Earl K et al. (2002). "The prefrontal cortex: categories, concepts and cognition". In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 357.1424, pp. 1123–1136.

Mongillo, Gianluigi et al. (2008). "Synaptic theory of working memory". In: *Science* 319.5869, pp. 1543–1546.

Montague, P Read et al. (1996). "A framework for mesencephalic dopamine systems based on predictive Hebbian learning". In: *Journal of neuroscience* 16.5, pp. 1936–1947.

Morishima, M. et al. (July 2011). "Highly Differentiated Projection-Specific Cortical Subnetworks". en. In: *Journal of Neuroscience* 31.28, pp. 10380–10391. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.0772-11.2011. URL: https://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.0772-11.2011.

Morishima, Mieko and Yasuo Kawaguchi (2006). "Recurrent Connection Patterns of Corticostriatal Pyramidal Cells in Frontal Cortex". In: *Journal of Neuroscience* 26.16, pp. 4394–4405. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.0252-06.2006. eprint: https://www.jneurosci.org/content/26/16/4394.full.pdf. URL: https://www.jneurosci.org/content/26/16/4394.

Morishima, Mieko et al. (Dec. 2017). "Segregated Excitatory–Inhibitory Recurrent Subnetworks in Layer 5 of the Rat Frontal Cortex". en. In: *Cerebral Cortex* 27.12, pp. 5846–5857. ISSN: 1047-3211, 1460-2199. DOI: 10.1093/cercor/bhx276. URL: https://academic.oup.com/cercor/article/27/12/5846/4555263.

Morita, Kenji et al. (Aug. 2012). "Reinforcement learning: computing the temporal difference of values via distinct corticostriatal pathways". en. In: *Trends in Neurosciences* 35.8, pp. 457–467. ISSN: 01662236. DOI: 10.1016/j.tins.2012.04.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166223612000719 (visited on 11/09/2021).

Mountcastle, Vernon B. (1997). "The columnar organization of the neocortex". In: *Brain* 120 (4), pp. 701–722. ISSN: 00068950. DOI: 10.1093/brain/120.4.701. URL: http://www.ncbi.nlm.nih.gov/pubmed/9153131.

Nakajima, Kohei et al. (2015). "Information processing via physical soft body". In: *Scientific reports* 5.1, p. 10487.

Nakajima, Kohei et al. (Mar. 2019). "Boosting Computational Power through Spatial Multiplexing in Quantum Reservoir Computing". en. In: *Physical Review Applied* 11.3, p. 034021. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.11.034021. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.11.034021 (visited on 10/10/2023).

Nicola, Wilten and Claudia Clopath (2017). "Supervised learning in spiking neural networks with FORCE training". In: *Nature communications* 8.1, p. 2208.

O'Doherty, John P et al. (2003). "Temporal difference models and reward-related learning in the human brain". In: *Neuron* 38.2, pp. 329–337.

O'Doherty, John et al. (2004). "Dissociable roles of ventral and dorsal striatum in instrumental conditioning". In: *science* 304.5669, pp. 452–454.

Ogawa, Seiji et al. (1990). "Brain magnetic resonance imaging with contrast dependent on blood oxygenation." In: *proceedings of the National Academy of Sciences* 87.24, pp. 9868–9872.

Paquot, Y. et al. (Feb. 2012). "Optoelectronic Reservoir Computing". en. In: *Scientific Reports* 2.1, p. 287. ISSN: 2045-2322. DOI: 10.1038/srep00287. URL: https://www.nature.com/articles/srep00287 (visited on 05/21/2023).

Paquot, Yvan et al. (Apr. 2010). "Reservoir computing: a photonic neural network for information processing". en. In: ed. by Benjamin J. Eggleton et al. Brussels, Belgium, 77280B. DOI: 10.1117/12.854050. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.854050 (visited on 10/10/2023).

Pauli, Robin et al. (2018). "Reproducing Polychronization: A Guide to Maximizing the Reproducibility of Spiking Network Models". In: *Frontiers in Neuroinformatics* 12. ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00046. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00046.

Pavlov, Ivan Petrovich (1927). *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*. Oxford University Press: Humphrey Milford.

Perin, Rodrigo et al. (2011). "A synaptic organizing principle for cortical neuronal groups". In: *Proceedings of the National Academy of Sciences* 108.13, pp. 5419–5424. DOI: 10.1073/pnas.1016051108. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1016051108. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1016051108.

Piccinini, Gualtiero and Sonya Bahar (2013). "Neural Computation and the Computational Theory of Cognition". In: *Cognitive Science* 37.3, pp. 453–488. DOI: https://doi.org/10.1111/cogs.12012. eprint: https://onlinelibrary.wiley.

com/doi/pdf/10.1111/cogs.12012. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12012.

Plesser, Hans E. (2018). "Reproducibility vs. Replicability: A Brief History of a Confused Terminology". In: *Frontiers in Neuroinformatics* 11. ISSN: 1662-5196. DOI: 10.3389/fninf.2017.00076. URL: https://www.frontiersin.org/article/10.3389/fninf.2017.00076.

Poincaré, Henri (1892). *Les méthodes nouvelles de la mécanique céleste*. Vol. 3. Gauthier-Villars et fils.

Popper, Karl (2005). *The logic of scientific discovery*. Routledge.

Potjans, Tobias C. and Markus Diesmann (2014). "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model". In: *Cerebral Cortex* 24.3, pp. 785–806. ISSN: 1460-2199. DOI: 10.1093/cercor/bhs358. pmid: 23203991. URL: https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhs358.

Purkinje, Johannes Evangelista (1837). "Neueste Untersuchungen aus der Nerven und Hirn Anatomie". In: *Bericht über die Versammlung deutscher Naturforscher und Aerzte in Prag im September* 1883, pp. 177–180.

Putnam, Hilary (1967). "Psychological Predicates". In: *Art, Mind, and Religion*. Ed. by W. H. Capitan and D. D. Merrill. University of Pittsburgh Press, pp. 37–48.

Pyle, Ryan and Robert Rosenbaum (July 2019). "A Reservoir Computing Model of Reward-Modulated Motor Learning and Automaticity". In: *Neural Computation* 31.7, pp. 1430–1461. ISSN: 0899-7667. DOI: 10.1162/neco_a_01198. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1430/1053175/neco_a_01198.pdf. URL: https://doi.org/10.1162/neco_a_01198.

Röhm, André and Kathy Lüdge (2018). "Multiplexed networks: reservoir computing with virtual and real nodes". In: *Journal of Physics Communications* 2.8, p. 085007. DOI: 10.1088/2399-6528/aad56d. URL: https://dx.doi.org/10.1088/2399-6528/aad56d.

Ramón y Cajal, Santiago (1888). *Estructura de los centros nerviosos de las aves*. Vol. 1.

– (1899). *Comparative study of the sensory areas of the human cortex*. Clark University.

Rasch, Malte J. et al. (2011). "Statistical Comparison of Spike Responses to Natural Stimuli in Monkey Area V1 With Simulated Responses of a Detailed Laminar Network Model for a Patch of V1". In: *Journal of Neurophysiology* 105.2, pp. 757–778. ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.00845.2009. URL: https://www.physiology.org/doi/10.1152/jn.00845.2009.

Rescorla, Michael (2020). "The Computational Theory of Mind". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2020. Metaphysics Research Lab, Stanford University.

Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rumelhart, David E et al. (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Saito, Yuki et al. (Jan. 2018). "Statistical Parametric Speech Synthesis Incorporating Generative Adversarial Networks". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.1. Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing, pp. 84–96. ISSN: 2329-9304. DOI: 10.1109/TASLP.2017.2761547. URL: https://ieeexplore.ieee.org/abstract/document/8063435 (visited on 11/22/2023).

Sammut, Michele et al. (2015). "Glia-derived neurons are required for sex-specific learning in C. elegans". In: *Nature* 526.7573, pp. 385–390.

Schulte to Brinke, Tobias et al. (2022). "Source code for "Characteristic columnar connectivity caters to cortical computation: replication, simulation and evaluation of a microcircuit model"". In: DOI: 10.5281/zenodo.7037649.

Schultz, Wolfram et al. (1997). "A neural substrate of prediction and reward". In: *Science* 275.5306, pp. 1593–1599.

Sharp, Thomas et al. (2012). "Power-efficient simulation of detailed cortical microcircuits on SpiNNaker". In: *Journal of neuroscience methods* 210.1, pp. 110–118.

Shen, Yujun and Bolei Zhou (2021). "Closed-Form Factorization of Latent Semantics in GANs". en. In: pp. 1532–1540. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Shen_Closed-Form_Factorization_of_Latent_Semantics_in_GANs_CVPR_2021_paper.html (visited on 11/21/2023).

Shen, Yujun et al. (Apr. 2022). "InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.4. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 2004–2018. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3034267. URL: https://ieeexplore.ieee.org/abstract/document/9241434 (visited on 11/21/2023).

Skinner, BF (1938). *The behavior of organisms: an experimental analysis.* Appleton-Century.

Smith, Evan C. and Michael S. Lewicki (Feb. 2006). "Efficient auditory coding". en. In: *Nature* 439.7079. Number: 7079 Publisher: Nature Publishing Group, pp. 978–982. ISSN: 1476-4687. DOI: 10.1038/nature04485. URL: https://www.nature.com/articles/nature04485 (visited on 11/21/2023).

Smith, Linda B. and Esther Thelen (Aug. 2003). "Development as a dynamic system". en. In: *Trends in Cognitive Sciences* 7.8, pp. 343–348. ISSN: 13646613. DOI: 10.1016/S1364-6613(03)00156-6. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364661303001566 (visited on 01/05/2023).

Song, Sen et al. (Mar. 2005). "Highly Nonrandom Features of Synaptic Connectivity in Local Cortical Circuits". In: *PLOS Biology* 3.3, null. DOI: 10.1371/journal.pbio.0030068. URL: https://doi.org/10.1371/journal.pbio.0030068.

Stein, R.B. (1967). "Some Models of Neuronal Variability". In: *Biophysical Journal* 7.1, pp. 37–68. ISSN: 0006-3495. DOI: https://doi.org/10.1016/S0006-3495(67)86574-3. URL: https://www.sciencedirect.com/science/article/pii/S0006349567865743.

Steno, Nicolas (1669). *Discours de Monsieur Stenon, sur l'anatomie du cerveau. A messieurs de l'Assemblée que se fait chez Monsieur Thevenot.* French. A Paris, chez Robert de Ninville. URL: https://archive.org/details/BIUSante_31863/page/n25/mode/2up.

Sussillo, David and Larry F Abbott (2009). "Generating coherent patterns of activity from chaotic neural networks". In: *Neuron* 63.4, pp. 544–557.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3, pp. 9–44.

Sutton, Richard S and Andrew G Barto (1981). "Toward a modern theory of adaptive networks: expectation and prediction." In: *Psychological review* 88.2, p. 135.

Tanaka, Gouhei et al. (2019). "Recent advances in physical reservoir computing: A review". In: *Neural Networks* 115, pp. 100–123.

Taylor, Tim and Alan Dorin (2020). "Rise of the Self-Replicators". In: *Cham, Switzerland: Springer.*

Thomson, Alex M. et al. (Sept. 2002). "Synaptic Connections and Small Circuits Involving Excitatory and Inhibitory Neurons in Layers 2–5 of Adult Rat and Cat Neocortex: Triple Intracellular Recordings and Biocytin Labelling In Vitro". In: *Cerebral Cortex* 12.9, pp. 936–953. ISSN: 1047-3211. DOI: 10.1093/cercor/12.9.936. eprint: https://academic.oup.com/cercor/article-pdf/12/9/936/9752427/1200936.pdf. URL: https://doi.org/10.1093/cercor/12.9.936.

Thorndike, Edward Lee (1911). *Animal intelligence: Experimental studies*. Transaction Publishers.

Turing, A. M. (Oct. 1950). "I.—COMPUTING MACHINERY AND INTELLIGENCE". In: *Mind* LIX.236, pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. URL: https://doi.org/10.1093/mind/LIX.236.433.

Turing, Alan M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 2.42, pp. 230–265. URL: http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf.

Van Vreeswijk, Carl and Haim Sompolinsky (1996). "Chaos in neuronal networks with balanced excitatory and inhibitory activity". In: *Science* 274.5293, pp. 1724–1726.

Vandoorne, Kristof et al. (2014). "Experimental demonstration of reservoir computing on a silicon photonics chip". In: *Nature communications* 5.1, p. 3541.

Verstraeten, David et al. (2010). "Memory versus non-linearity in reservoirs". In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2010.5596492.

Vesalius, Andreas (1543). *De Humani Corporis Fabrica Libri Septem*. Basel: Joannes Oporinus.

von Neumann, J. (1993). "First draft of a report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.4, pp. 27–75. DOI: 10.1109/85.238389.

Wörgötter, Florentin and Bernd Porr (2005). "Temporal Sequence Learning, Prediction, and Control: A Review of Different Models and Their Relation to Biological Mechanisms". In: *Neural Computation* 17.2, pp. 245–319. DOI: 10.1162/0899766053011555.

Watts, Duncan J and Steven H Strogatz (1998). "Collective dynamics of 'small-world'networks". In: *nature* 393.6684, pp. 440–442.

Wiesenfeld, Kurt and Frank Moss (1995). "Stochastic Resonance and the Benefits of Noise: From Ice Ages to Crayfish and SQUIDs". In: *Nature* 373.6509, pp. 33–36. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/373033a0. URL: http://www.nature.com/articles/373033a0.

Winant, David et al. (May 2021). "Latent Space Exploration Using Generative Kernel PCA". In: arXiv:2105.13949 [cs, stat]. DOI: 10.48550/arXiv.2105.13949. URL: http://arxiv.org/abs/2105.13949 (visited on 11/21/2023).

Yang, Vicky Chuqiao et al. (2021). "Dynamical system model predicts when social learners impair collective performance". In: *Proceedings of the National Academy of Sciences* 118.35, e2106292118. DOI: 10.1073/pnas.2106292118. eprint: https:

//www.pnas.org/doi/pdf/10.1073/pnas.2106292118. URL: https://www.pnas.org/doi/abs/10.1073/pnas.2106292118.

Yildiz, Izzet B. et al. (2012). "Re-visiting the echo state property". In: *Neural Networks* 35, pp. 1–9. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608012001852.

Zhong, Yanan et al. (Jan. 2021). "Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing". en. In: *Nature Communications* 12.1, p. 408. ISSN: 2041-1723. DOI: 10.1038/s41467-020-20692-1. URL: https://www.nature.com/articles/s41467-020-20692-1 (visited on 10/11/2023).

Appendix A

Microcircuit

Tasks/circuits	Amorphous	Small-world	DC	DC (no io)	Rand. dyn.	Static syn.
memory	-25.8	-25.8	-7.9	-28.6	-46.7	-17.3
nonlinear	-30.1	-31.6	-22.8	-29.2	-52	-18.9
other	-5.1	-12.2	-4	-17	-39.7	-1.6
all	-16.2	-20	-8.9	-22.8	-44.2	-9.7

Table A.1: Performance measures for all control networks consisting of Hodgkin-Huxley neurons without intrinsic conductance noise expressed as average difference (in percent) from the performance of the data-based circuit. All values are averaged over 20 runs.

Tasks/circuits	Amorphous	Small-world	DC	DC (no io)	Rand. dyn.	Static syn.
memory	-34.6	-23.6	-9.5	-31.3	-59.8	-24.8
nonlinear	-42.8	-21.8	-28.8	-28.8	-62.3	-27
other	-8.6	-9.9	-6.6	-18.1	-46.3	-8
all	-23.7	-16.6	-12	-24.5	-53.8	-17.2

Table A.2: Performance measures for all control networks consisting of integrate-and-fire neurons expressed as average difference (in percent) from the performance of the data-based circuit. All values are averaged over 20 runs.

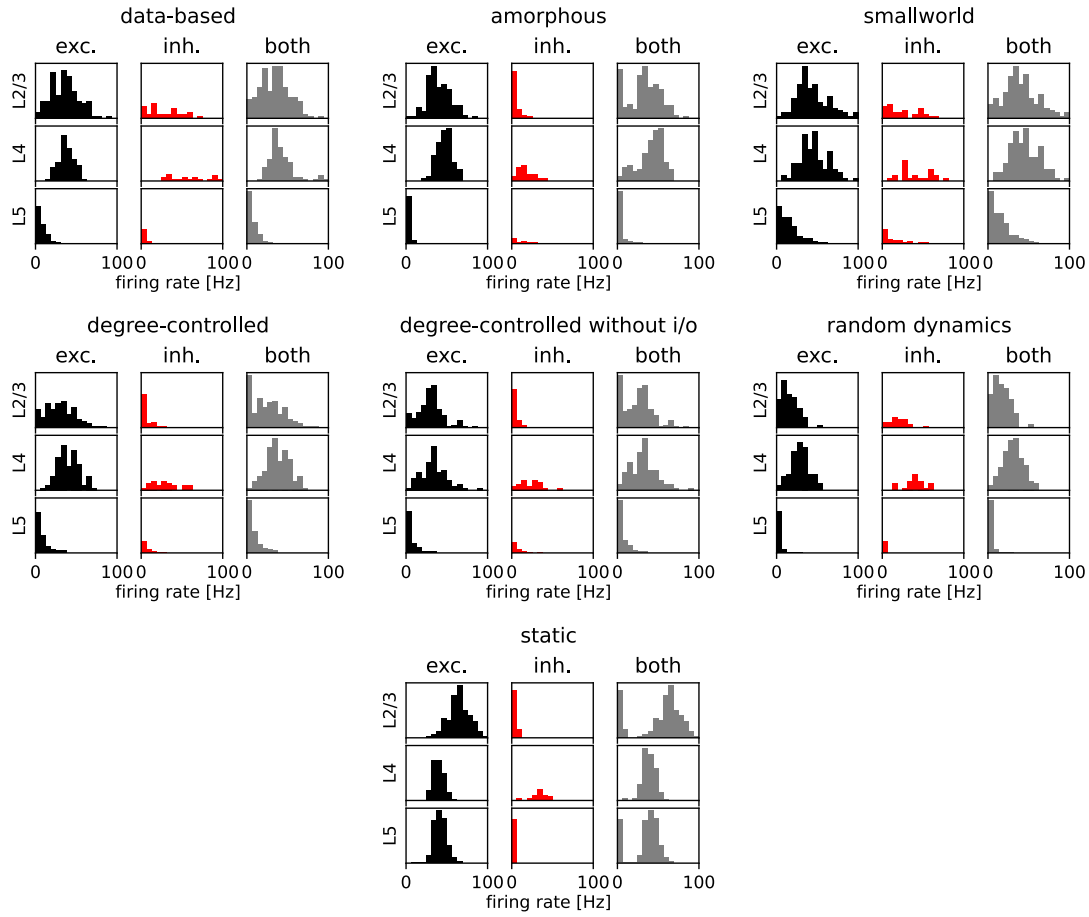


Figure A.1: Firing rate histograms of the control circuits consisting of Hodgkin-Huxley neurons with conductance noise.

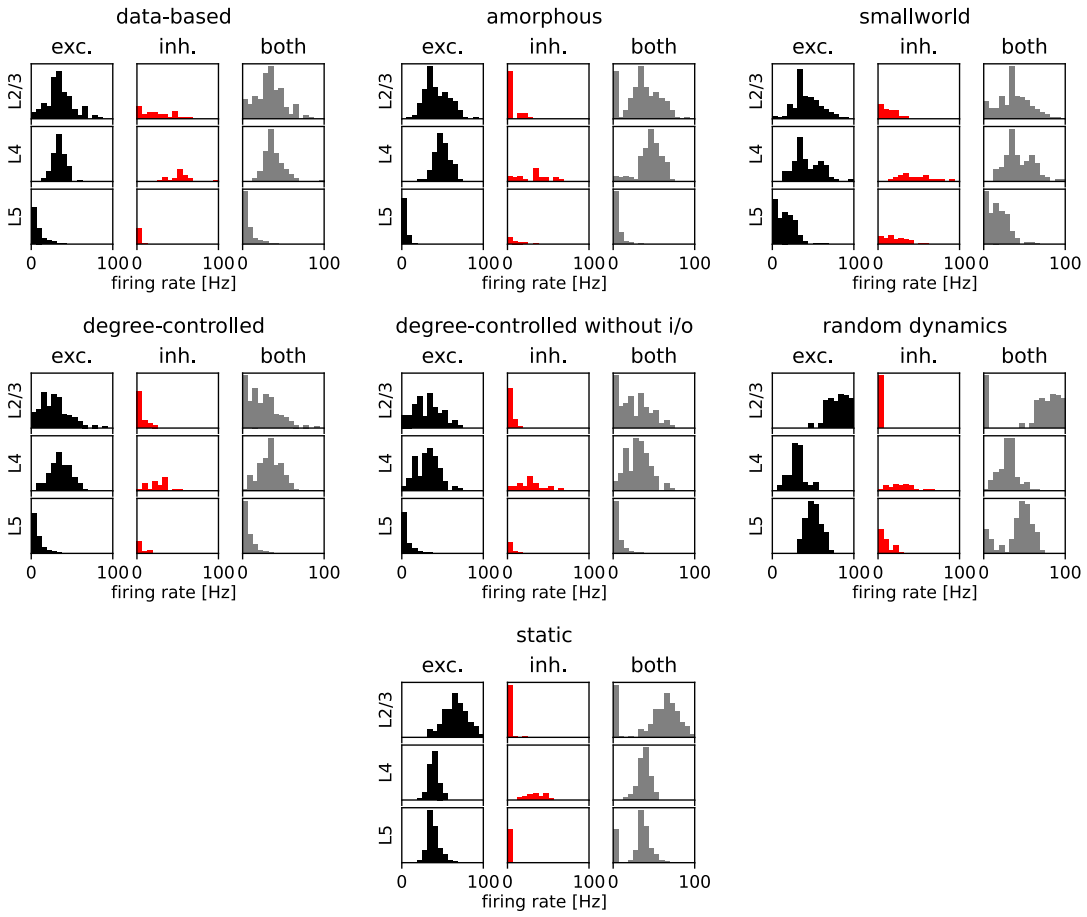


Figure A.2: Firing rate histograms of the control circuits consisting of Hodgkin-Huxley neurons without conductance noise.

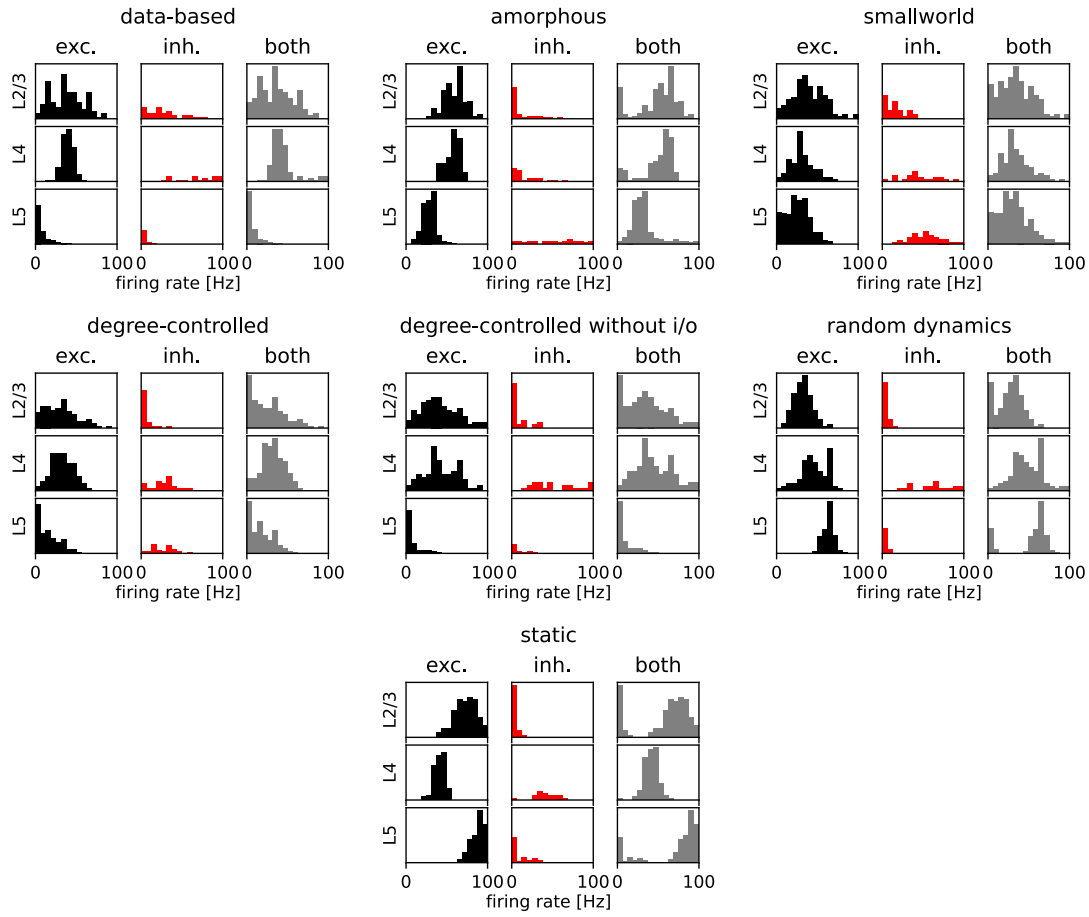


Figure A.3: Firing rate histograms of the control circuits consisting of integrate-and-fire neurons.

Appendix B

Information processing capacity

par.	value	description
$V_{\min}$	0 mV	maximum of the membrane potential distribution
$V_{\max}$	20 mV	minimum of the membrane potential distribution
τ_{m}	20 ms	membrane time constant
C_{m}	1 pF	membrane capacitance
E_{L}	0 mV	resting membrane potential
d	1.5 ms	synaptic delay
V_{th}	20 mV	threshold potential
V_{reset}	10 mV	reset potential
τ_{ref}	2 ms	refractory period
N	1250	network size
N_{exc}	1000	number of excitatory neurons
N_{inh}	250	number of inhibitory neurons
C_{exc}	100	number of incoming excitatory synapses
C_{inh}	25	number of incoming inhibitory synapses
g	5	ratio of inhibitory to excitatory weight
w_{exc}	0.2 pA	excitatory synaptic weight
s_{inh}	$-gw_{\mathrm{exc}} = -1$ pA	inhibitory synaptic weight
ν_{noise}	4000 spk/sec	rate of background noise

Table B.1: Parameters for balanced random network

The lower half of Figure B.1 shows the capacity heat maps for a fixed γ value of 1. We have thus removed all encoder capacities and their delayed versions, resulting in a lower capacity bound. The fact that there are no differences between the results of the different transformation functions tells us that the networks do not compute target functions y_l with an encoder capacity $C^{enc}(y_l) > 0$ better than the encoder does.

B.1 Details on removing the nonlinear encoder effects

First, we take a closer look at the reconstructions of the target signal and the squared correlation coefficient, which forms the basis for the capacity evaluation. The reconstructed function z_l is a weighted sum of y_l and an uncorrelated noise signal b_l:

$$z_l = \alpha_l \cdot y_l + (1 - \alpha_l) \cdot b_l \tag{B.1}$$

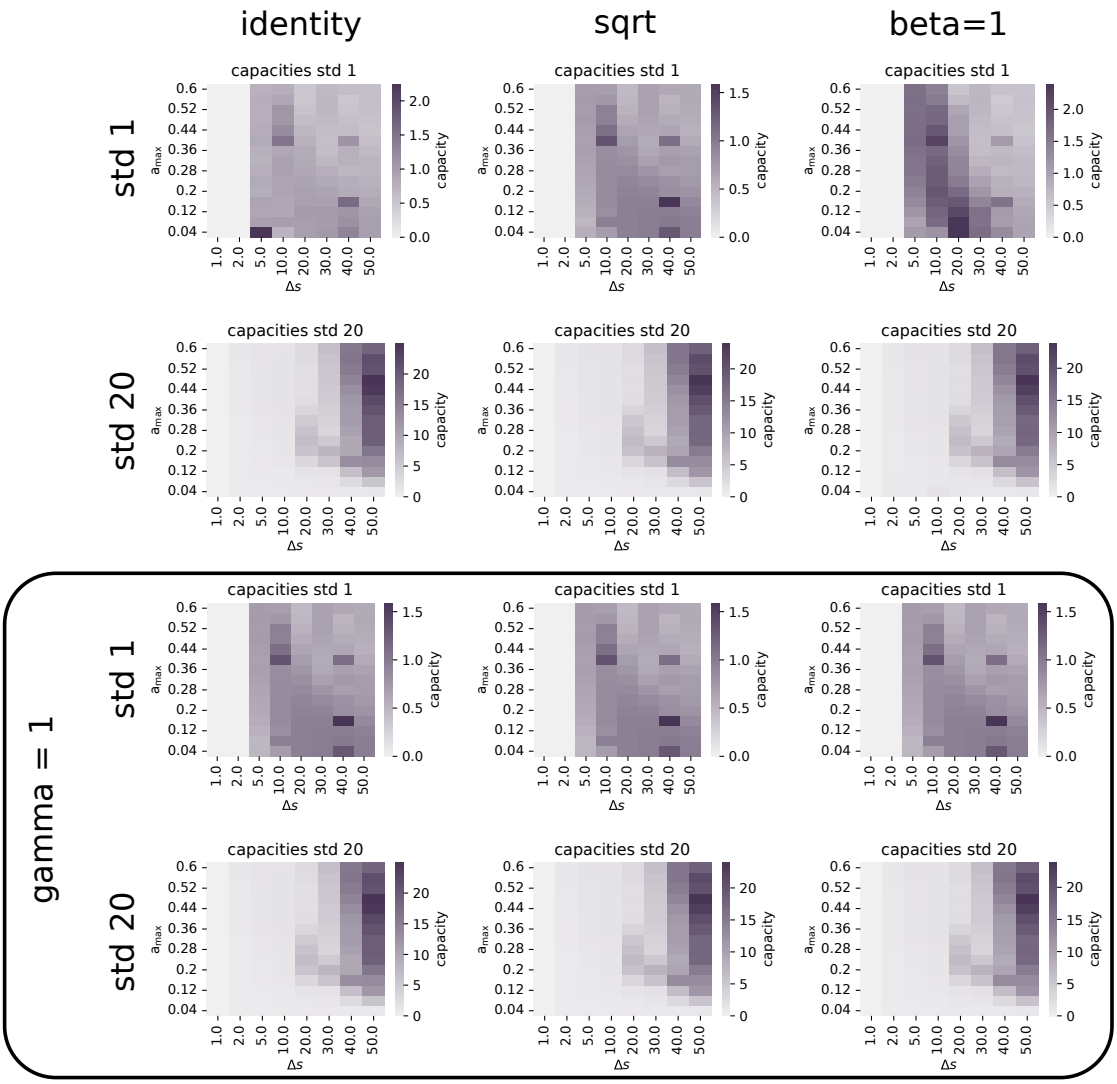


Figure B.1: BRN results, spatial encoding, removed remembered encoder capacities

where α_l is the relative weight of y_l compared to b_l and is related to the capacity:

$$
\begin{aligned}
C_l &= \frac{\mathrm{cov}(y_l, z_l)^2}{\mathrm{var}(y_l) \cdot \mathrm{var}(z_l)} \\
&= \frac{\alpha_l^2 \mathrm{var}(y_l)^2}{\mathrm{var}(y_l)(\alpha_l^2 \cdot \mathrm{var}(y_l) + (1-\alpha_l)^2 \cdot \mathrm{var}(b_l))} \\
&= \frac{\mathrm{var}(y_l)}{\mathrm{var}(y_l) + \mathrm{var}(b_l)\frac{(1-\alpha_l)^2}{\alpha_l^2}} \\
&= \frac{1}{1 + \frac{\mathrm{var}(b_l)}{\mathrm{var}(y_l)}\frac{(1-\alpha_l)^2}{\alpha_l^2}}
\end{aligned}
\tag{B.2}
$$

Therefore, the relationship between capacity C_l and α_l is nonlinear and depends on the ratio between the variances of b_l and y_l, as Equation B.2 and also Figure B.2A show. Note that this ratio can be different for each target function.

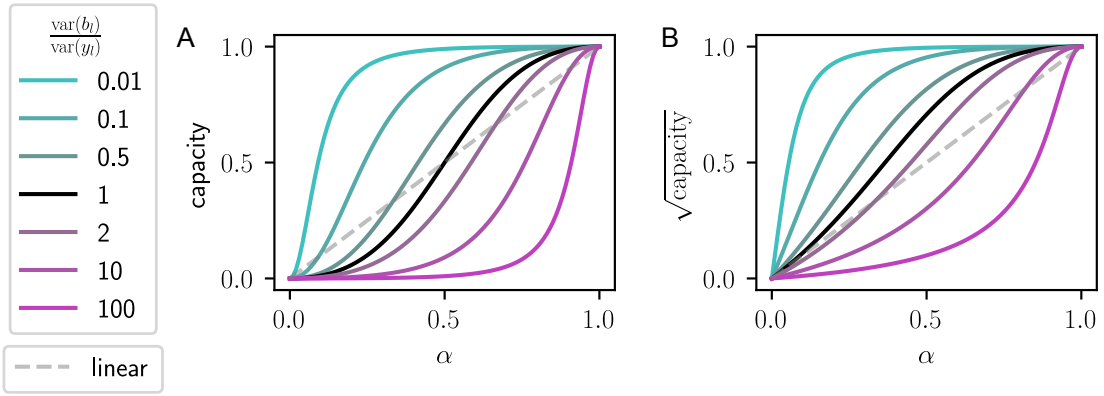


Figure B.2: capacity transformation

Following Equation B.2 we can define the function f_T that calculates α_l based on the capacity C_l and the ratio of variances β_l of b_l and y_l:

$$
f_T(C_l, \beta_l) = \alpha_l = \frac{1}{\sqrt{\frac{\frac{1}{C_l} - 1}{\beta_l}} + 1} \tag{B.3}
$$

To remove the encoder effects, we first calculate the capacity of the encoder output. We use the resulting capacity profile together with the capacities of the overall system to calculate the effective linear memory that the main system introduces in addition to the encoder memory. To do this, we subtract the encoder memory M^{enc} from the combined memory M^{comb} for each delay i (main Figure 6 B-E):

$$M_i^{\text{sys}} = M_i^{\text{comb}} - M_i^{\text{enc}} \tag{B.4}$$

M_i is the capacity with input u delayed by i steps as target function.

$$M_i = f_T(C[u(k-i)], \beta_{u_i}) \tag{B.5}$$

The transformation f_T allows us to obtain meaningful results when we add, subtract or divide the M values under the assumption that we know β_{u_i}.

Based on the system and encoder memory values, we calculate the memory ratio γ for all delays, i.e., the fraction of the encoder input that the system can memorize after a delay i:

$$\gamma_i = \frac{M_i^{\text{sys}}}{M_0^{\text{enc}}} \tag{B.6}$$

Using these memory ratios, we compute the remembered encoder capacities C^{rem}, i.e., how much a capacity value for a target function can be based on remembering a previous target function that is already computed by the encoder:

$$C^{\text{rem}}(y_n) = f_T^{-1}(f_T(C_0^{\text{enc}}(y_m), \beta_m) \cdot \gamma_i, \beta_m) \tag{B.7}$$

where $C^{\text{rem}}(y_n)$ is the remembered capacity for the target function y_n, which corresponds to the target function y_m delayed by i steps. These remembered capacities are the result of the linear memory of the system and the nonlinearity and memory of the encoder, and therefore we must subtract them from the capacities of the combined system to obtain the effective capacity of the main system for all objective functions y_l:

$$C^{\text{sys}}(y_l) = f_T^{-1}\left(f_T(C^{\text{comb}}(y_l), \beta_l) - f_T(max(C^{\text{enc}}(y_l), max(C^{rem}(y_l)), \beta_l)\right) \tag{B.8}$$

where C^{comb} is the measured capacity of the combined system including the encoder and the main system. With this method we can not get information about the precise functions the system can compute as with a linearly encoded input, because the system does not compute the function $F_{\text{sys}}(u)$, but the function $F_{\text{sys}}(F_{\text{enc}}(u))$. Therefore a system capacity $C^{\text{sys}}(y_l) > 0$ does not mean that the system computes the specific target function y_l with the degree d_{y_l}. However, it tells us that the system computes a function which goes beyond remembering the input signal.

The problem with the transformation f_T is that we do not know the variance ratio β and therefore cannot remove the exact encoder effects. However, we test different ways to approximate f_T. The simplest approximation is to use the identity function and not transform the capacities before calculating γ and subtracting the encoder values. Other possibilities are to take the square root of the capacities to obtain the correlation coefficient instead of its squared value or to set β to a fixed value (e.g. 1) for each target

function. To obtain a lower bound for the capacities, we can set γ to 1 for all linear capacities $C[u(k-i)] > 1$. This leads to a complete subtraction of all encoder capacities and their delayed versions and thus to a lower limit for the capacity.

Appendix C

Temporal-difference learning in cortico-striatal populations

Weights			
Name	**Value**	**Description**	**Source**
$w_{\mathrm{CPn}\to\mathrm{CPn}}$	32.3 ± 27.4 pA, median: 22.8	weight from CPn to CPn	[2]
$w_{\mathrm{CCS}\to\mathrm{CCS}}$	17.0 ± 13.7 pA, median: 13.9 (17.8 ± 15.4 pA)	weight from CCS to CCS	[2], ([1])
$w_{\mathrm{CCS}\to\mathrm{CPn}}$	14.7 ± 9.2 pA, median: 11.1	weight from CCS to CPn	[2]
$w_{\mathrm{CPn}\to\mathrm{CCS}}$	$w_{\mathrm{CCS}\to\mathrm{CPn}}$	weight from CPn to CCS	-

Table C.1: Synaptic weights. References: [1] Morishima and Kawaguchi (2006) [2] Morishima et al. (2011)

Neuron Model baseline SNN		
Name	**Value**	**Description**
C_{m}	250 pF	Membrane capacitance
E_L	-70 mV	Resting membrane potential
τ_{m}	10 ms	Membrane time constant
V_{th}	-55 mV	Membrane potential threshold for action-potential firing
V_{reset}	-70 mV	Reset potential
τ_{ref}	2 ms	Absolute refractory period

Table C.2: Neuron model parameters for baseline spiking neuronal networks

Neuron Model CCS		
Name	**Value**	**Description**
C_{m}	236 pF (Optuna)	Membrane capacitance
Δ_T	2.0	
E_L	-66.4 mV	Resting membrane potential
E_{ex}	0 mV	
E_{in}	-80 mV	
τ_{m}	22.6 ms	Membrane time constant
V_{th}	-40 mV (Optuna)	Membrane potential threshold for action-potential firing
V_{reset}	-53 mV (Optuna)	Reset potential
τ_{ref}	2 ms	Absolute refractory period
g_{L}	10.44nS (C_m / τ_{m})	Leak conductance
a	5. (Optuna)	
b	52. (Optuna)	
τ_{w}	536 ms (Optuna)	

Table C.3: Neuron model parameters for the CCS population.

Neuron Model CPn		
Name	**Value**	**Description**
C_{m}	496 pF (Optuna)	Membrane capacitance
Δ_T	2.0	
E_L	-62.1 mV	Resting membrane potential
E_{ex}	0 mV	
E_{in}	-80 mV	
τ_{m}	15.5 ms	Membrane time constant
V_{th}	-53 mV (Optuna)	Membrane potential threshold for action-potential firing
V_{reset}	-51 mV (Optuna)	Reset potential
τ_{ref}	2 ms	Absolute refractory period
g_{L}	32.nS (C_m / τ_{m})	Leak conductance
a	6. (Optuna)	
b	194. (Optuna)	
τ_{w}	88 ms (Optuna)	

Table C.4: Neuron model parameters for the CPn population.

Neuron Model SNN from Rate Network		
Name	**Value**	**Description**
C_{m}	20 pF	Membrane capacitance
E_L	0 mV	Resting membrane potential
τ_{m}	20 ms	Membrane time constant
V_{th}	−40 mV	Membrane potential threshold for action-potential firing
V_{reset}	−65 mV	Reset potential
τ_{ref}	2 ms	Absolute refractory period
I_{e}	−40 mA	External input current

Table C.5: Neuron model parameters spiking neuronal networks that are reconstructed based on the procedure proposed in Kim, Li, and Sejnowski (2019)

Synapse Model (CPn)			
Name	**Value**	**Description**	**Source**
U	0.37 ± 0.14	Increase of release probability with each spike	(1)
τ_{D}	$317\,\mathrm{ms} \pm 146$	Time constant for depression	(1)
τ_{F}	$519\,\mathrm{ms} \pm 981$	Time constant for facilitation	(1)
Synapse Model (CCS)			
Name	**Value**	**Description**	**Source**
U	0.41 ± 0.19	Increase of release probability with each spike	(1)
τ_{D}	$532\,\mathrm{ms} \pm 244$	Time constant for depression	(1)
τ_{F}	$65\,\mathrm{ms} \pm 110$	Time constant for facilitation	(1)

Table C.6: Synapse short-term plasticity parameters for the CPn and CCS populations. References: (1) Morishima et al. (2011)

C: Synapse Model		
τ_{E}	5 ms	Synaptic decay time constant for excitatory synapses
τ_{I}	10 ms	Synaptic decay time constant for inhibitory synapses
V_{E}	0 mV	Excitatory reversal potential
V_{I}	−80 mV	Inhibitory reversal potential

Table C.7: Synapse parameters for the CPn and CCS populations.

delays			
$d_{\mathrm{CPn}\to\mathrm{CPn}}$	1.6 ± 0.3	delay between CPn to CPn	[2]
$d_{\mathrm{CCS}\to\mathrm{CCS}}$	1.5 ± 0.5 ms (1.6 ± 0.6)	delay between CCS to CCS	[2] ([1])
$d_{\mathrm{CCS}\to\mathrm{CPn}}$	1.8 ± 0.5 ms	delay between CCS to CPn	[2], [1]
C: Synapse Model			
$\tau_{\mathrm{E,CPn}\to\mathrm{CPn}}$	6.9 ± 1.9 ms	Synaptic decay time constant for excitatory synapses	[2]
$\tau_{\mathrm{E,CCS}\to\mathrm{CCS}}$	6.8 ± 1.9 ms (6.0 ± 2.0 ms)	Synaptic decay time constant for excitatory synapses	[2] ([1])
$\tau_{\mathrm{E,CCS}\to\mathrm{CPn}}$	6.3 ± 2.5 ms	Synaptic decay time constant for excitatory synapses	[1], [2]
τ_{I}	10 ms	Synaptic decay time constant for inhibitory synapses	
V_{E}	0 mV	Excitatory reversal potential	
V_{I}	-80 mV	Inhibitory reversal potential	

Table C.8: Synapse parameters for the CPn and CCS populations. References: [1] Morishima and Kawaguchi (2006) [2] Morishima et al. (2011)

Neuronal parameters			
Name	**Value**	**Description**	**Source**
$E_{\mathrm{L,CCS}}$	-66.4 ± 5.4 mV	Resting membrane potential	[1]
$E_{\mathrm{L,CPn}}$	-62.1 ± 4.1 mV	Resting membrane potential	[1]
$\tau_{\mathrm{m,CCS}}$	22.6 ± 11.5 ms	Membrane time constant	[2]
$\tau_{\mathrm{m,CPn}}$	15.5 ± 4.6 ms	Membrane time constant	[2]

Table C.9: Neuronal parameters for the CPn and CCS populations. References: [1] Morishima et al. (2011) [2] Morishima and Kawaguchi (2006)

List of Figures

1.1 **Cortical architecture. A:** Drawing of a human brain from Henry Gray's *Anatomy of the Human Body* (Gray, 2000). **B:** Slice of a macaque monkey brain with Nissl stained cell bodies (retrieved from BrainMaps Atlas (Mikula et al., 2007)). **C:** Historical drawing of the Nissl stained visual cortex of a human adult seen through a microscope by Santiago Ramón y Cajal (Ramón y Cajal, 1899). 9
1.2 Neuronal action potential generation and transmission 10
1.3 **Schematic illustration of the reservoir computing paradigm.** The input is linearly transformed by the weight matrix W^{in} and the reservoir processes these inputs so that the desired outputs can be read out using the linear transformation matrix W^{out}. Adapted from Tanaka et al. (2019). 15

2.1 Structure of the data-based microcircuit model. The connection arrows are labeled with the connection strength (mean amplitude of PSPs in mV, c.f. parameter A in Table 2.5) and the connection probability (in parentheses). Red arrows represent inhibitory connections and excitatory connections are black. The excitatory input connections are represented as grey dashed arrows. Neuron numbers are based on a network size of $N = 560$ neurons. C.f. Figure 1, original publication (Häusler and Maass, 2007). 33
2.2 Histograms of degrees (number of incoming and outgoing connections per neuron) per population for each circuit. Values are aggregated for 100 runs with different seeds for each model. C.f. Figure 7, original publication (Häusler and Maass, 2007). 34
2.3 Input generation for spike pattern-based tasks. The first row shows the different trials of an experiment and the spike patterns below that represent a zero or one value for each segment of a single trial. These spike pattern templates are identical for all trials. The spike patterns at the bottom are chosen based on and therefore represent the randomly generated sequence of zeros and ones underneath, which also define the target for the readout training (value of segment 14 for delayed classification and segment 15 for the undelayed classification). We give a jittered version of these spike trains to the network (jittering is not shown). C.f. Figure 4, original publication (Häusler and Maass, 2007). 37

2.4 Activity of the data-based circuit. **A**: Raster plot for the data-based circuit after input stream two was activated at 100 ms. Excitatory neurons are black and inhibitory neurons are red. **B**: The corresponding firing rate histograms for each population and layer. C.f. Figure 2B and 2C, original publication (Häusler and Maass, 2007). 41
2.5 Performance of trained linear readout neurons in layers 2/3 and layer 5 for the classification tasks on spike patterns and computations performed on the rates of the two input streams (see subsection 2.2.2), both for data-based laminar microcircuit models (gray bars) and for the amorphous control circuits (scrambled laminar structure; black bars). Light purple bars represent the results for networks with neurons without conductance noise and light orange bars networks consisting of integrate-and-fire neurons. Error bars are the standard errors of mean. All values are averaged over 20 runs. C.f. Figure 5, original publication (Häusler and Maass, 2007), likewise averaged over 20 runs. 42
2.6 **A**: Training and testing error of readouts from data-based and amorphous circuit models as functions of the size of the training set. 300 trials are always used for testing. Error bars indicate the standard error of means. Values are averaged over 30 runs. C.f. Figure 8, original publication (Häusler and Maass, 2007), averaged over 20 runs. **B**: Performance (kappa coefficient) on the XOR task of projection neurons in layers 2/3 and layer 5 for different circuit sizes, with and without a data-based laminar structure. All values are averaged over 30 runs. Error bars indicate the standard error of the means. C.f. Figure 6, original publication (Häusler and Maass, 2007), averaged over 10 runs. 43
2.7 Raster plots and firing rate histograms for the data-based and amorphous circuits for the three different neuron types (original: Hodgkin-Huxley neurons that were used in the original publication, disabled-noise: Hodgkin-Huxley neurons without intrinsic conductance noise, iaf neuron: integrate-and-fire neurons). The spikes of the inhibitory populations are colored red, while those of the excitatory populations are shown in black. As in Figure 2B and Figure 2C, original publication (Häusler and Maass, 2007), for the raster plots input stream two starts at 100 ms. 47
2.8 Results of retroactive spike pattern classification tasks for all network types. **A**: Performances (kappa coefficient) for the classification of spike patterns with a duration of 5 ms at different delays, separated by input stream and readout (averaged over 40 trials). **B**: Bars representing the sum of task performances over all delays for the same task as in A. Error bars represent the standard error of mean. **C**: Values from B averaged across all input streams and readouts. **D**: Averaged results of the delayed classification of 30 ms spike patterns (data of *memory* row in Table 2.7). . 49

3.1 Schematic of processing capacity. **Simulation**: real-valued random inputs between -1 and 1 are encoded and fed into the dynamical system. **A**: temporally unfolded signal. **B**: amplitude-value encoding where each neuron receives the same input. **C**: distributed encoding where the encoder is connected to the system by randomly drawn weights. **D**: spatially encoded signal. **Polynomial computation**: Products of Legendre polynomials with delayed inputs are calculated as target functions. **Capacity computation**: the reconstructed functions are evaluated against the target functions using the squared correlation coefficient. 62

3.2 ESN results with increasing level of detail from top to bottom. **A-C**: Heatmaps of the total capacity, maximum degree and maximum delay for a parameter scan over ρ and ι and based on all capacity functions (over all degrees and delays). **D** and **E**: Total and degree-specific capacities as a function of ι, for the values of ρ indicated by the correspondingly coloured arrows in **A-C**. Values are summed over all delays for each bar. Note that the ESN only exhibits capacities of odd degree, because of the odd nature of the tanh nonlinearity. **F** and **G**: Capacity profiles showing total and degree-specific capacities as a function of delay for the parameter configurations indicated by the correspondingly coloured arrows in **D,E** . 64

3.3 Comparison of ESN task results with the processing capacity as a function of ι. **A**: Performance for nonlinear tasks (yellow and blue curves; left y-axis) and the maximum capacity degrees (green curve; right y-axis). **H**: Performance for memory tasks (gray curves; left y-axis) and maximum capacity delay (blue dashed curve; right y-axis). 65

3.4 FPUT results. **A**: Heatmap of nonlinear capacity with zero delay, where the maximum amplitude a_{max} is a force given in arbitrary units, the input duration is given in units of simulation time steps. **B**: Total nonlinear capacity, nonlinear capacity with zero delay (dashed) and with at least delay 5 (dotted) as well as performance on XOR (yellow), time delayed XOR (yellow, dash-dotted), NARMA5 (maroon) and NARMA5 evaluated on a linear reference system with perfect memory (light maroon, dashed) over input duration for fixed $a_{max} = 0.033$. The performance of both XOR tasks is measured using the Cohen's kappa score and for NARMA it is the squared Pearson's correlation coefficient. **C, D**: Capacity profiles showing total and degree-specific capacities as a function of delay for the parameter configurations indicated by the correspondingly coloured squares in **A**. **E**: Correlations between capacity and task performances for the same parameter ranges of amplitude and duration as shown in **A**. . . 66

3.5 BRN results for different encoding schemes. **A,B**: maximum capacity and memory (product of maximum delay and step duration) for the spatial encoding scheme averaged over five trials with different network and input initializations. The standard deviation across trials is too small to be visible. **C,D**: as for **A,B**, but with amplitude encoding. **E,F**: as for **A,B**, but with distributed encoding. **G, H**: Heat maps for total capacity, maximum degree and maximum delay for the parameter scans marked with a circle and a square in A and B. **I, J**: Capacity profiles showing total and degree-specific capacities as functions of delay for the parameter configurations indicated by the magenta and yellow boxes in **G** and **H**. Note the different axis and color scales in these panels. **K-L**: Correlations between capacity statistic and task performances for the same parameter ranges of $a_{\max}$ and Δ_s as shown in **G** and **H**, using DC input and frozen noise; $p = 1$ in **K** and $\sigma = 20$ in **L**. 68

3.6 Schematic for the calculation of remembered encoder capacities. **A**: Diagram emphasizing that the combination of encoder and network acts as the measured system. **B**: Linear memory of the combined measured system. **C**: Linear memory of the encoder. **D, E**: Difference between **B** and **C** that results in the actual linear memory of the network. **F**: Possibly nonlinear encoder capacity of target function y_l (black border) and its versions remembered by the network (green). **G**: capacity profile for the encoder used in Figure 3.5J. **H-I**: Capacity heat maps as in Figure 3.5G,H, but with all possible encoder capacities subtracted. Note the different scales of the color bars. 71

3.7 Microcircuit results for different encoding schemes. **A-C**: Maximum capacity per input parameter p or σ respectively for distributed encoding (**A**), spatial encoding (**B**) and spatial encoding without remembered encoder capacities (**C**) averaged over five trials with different network and input initialisations. Shaded areas indicate the standard deviation across trials. **D, E**: Maximum memory (product of maximum delay and step duration) for distributed and spatial encoding. **F-Q**: Capacity (**F, J, N**) and delay (**G, K, O**) heat maps of the parameter scans corresponding to the markers in **A-C** together with capacity profiles for a high capacity (yellow) and high delay (magenta) parameter configuration. Note the different axis and color scales in the bar graphs. **R-S**: Correlations between capacity statistic and task performances for the same parameter ranges of $a_{\max}$ and Δ_s used in **F** and **J**, both with DC input and frozen noise; $p = 1$ in **R** and $\sigma = 20$ in **S**. 74

4.1 Hypothetical mechanism of the computation of TD error in the brain. The activity of the CCS cells encodes the current state $S(t)$ and unidirectional connections propagate the information to CPn cells, where it is preserved longer such that their activity represents the previous state $S(t-1)$. The state values of the current and previous state are calculated through connections to the striatum. The value of the current state $V(S(t))$ is transported via the direct pathway (red arrow) to the SNr. The previous state value $V(S(t-1))$ is transported over the indirect pathway (blue arrow) via the GPe and the STN to the SNr. The SNr passes this information on to the SNc. The $V(S(t))$ activity causes a positive modulation on the SNc by disinhibition and the $V(S(t-1))$ activity causes a net negative modulation by triple inhibition. Together with a reward signal coming from the PPN, the SNc computes the TD error. Figure adapted from Morita et al. (2012). 78

4.2 Scheme of the structured network model. The network is separated into a CCS (green) and a CPn population (blue). Both populations are further divided into an excitatory (black) and an inhibitory subpopulation (red). The input is only given to the CCS population. The connections from CCS to CPn and the recurrent connections inside the CPn population are stronger (indicated by thicker arrows). 82

4.3 Fitted neuron models. Voltage traces in green (CCS) and blue (CPn) in response to a step depolarization of 0.5 nA over 500 ms. Black bars represent the spike times of the fitted models and red bars the spike times of the experimental data from Morishima and Kawaguchi (2006). 84

4.4 Baseline rate network with different nonlinearities. **A:** Shapes of the tested nonlinearities. **B:** Sum of capacities above the chance level threshold for the nonlinearities shown in A. **C-H:** Memory curves for the networks with the different nonlinearities. 87

4.5 Deviation of nonlinear transfer functions from the linear identity function. **A:** Shapes of the tested nonlinearities between -1 and 1. In addition to the previously described functions also the trigonometric sine function (sin) is used. **B:** Area between the transfer functions and the identity function as a measure for deviation from the identity function. **C and D:** Sum of capacities above the chance level threshold for the different transfer functions for the CCS (A) and CPn (B) populations. 88

4.6 Effect of different shifts of the tanh and sigmoid transfer functions on memory. **A:** Shifted versions of the tanh transfer function. **B and C:** Standard deviation of the activity in the CCS (B) and CPn (C) populations for the different shifted tanh transfer functions. **D and E:** Sum of linear capacities above the chance level threshold for the different shifted tanh transfer functions for the CCS (D) and CPn (E) populations. **F:** Shifted versions of the sigmoid transfer function. **G and H:** Standard deviation of the activity in the CCS (G) and CPn (H) populations for the different shifted sigmoid transfer functions. **I and J:** Sum of linear capacities above the chance level threshold for the different shifted sigmoid transfer functions for the CCS (I) and CPn (J) populations. 90

4.7 Network modification studies for baseline rate networks with tanh and sigmoid nonlinearity. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks with tanh nonlinearity. **B-D:** Memory curves for the full model and the two networks with the biggest difference to the full network model. **E-H:** The same figures as in A-D but for the networks with sigmoid nonlinearity. 92

4.8 Effect of balance between excitation and inhibition on memory. **Top:** Heatmaps of linear capacity sum for a parameter scan of inhibitory weight factor γ and fraction of inhibitory neurons β_{inh} for the CCS (left) and CPn (right) populations. **Bottom:** Maximum capacity delay for the same parameter scan for the CCS (left) and CPn (right) populations. 94

4.9 State matrix data of networks with different EI-ratios. **A:** State matrices of the CCS (left) and CPn (right) population of a network with $\gamma = 4$ and $\beta_{\text{inh}} = 0.2$ at the top and their corresponding state distributions at the bottom (based on the same data as the activity matrices above). **B:** The same figure as in A but for a network with $\gamma = 1$ and $\beta_{\text{inh}} = 0.5$. **C:** Heatmaps of mean activity and standard deviation of the activity for the parameter scan over γ and β_{inh}. The blue mark corresponds to the network in A and the red mark to the network in B. 95

4.10 Scan over the inhibitory weight factor for the CPn population γ_{CPn} for fixed parameters for the CCS population ($\gamma_{\mathrm{CCS}} = 4$ and $\beta_{\mathrm{inh,CCS}} = 0.2$). **A:** Linear capacity sum of the CPn population for different values of γ_{CPn}. Bars with darker color mark a shift from step size 1 to step size 0.1. The green bar marks the highest capacity sum. **B:** Mean activity of the CPn population for the same configurations as in A. Again a color change marks the change in step size and the green bar marks the network with the highest capacity sum in the CPn population. **C:** Histogram of the activity of the CPn population for the network with the highest capacity sum, separated into excitatory and inhibitory neurons. **D:** Violin plot showing the state distributions of the CPn populations for each γ_{CPn}. Colors mean the same as in B. **E:** Standard deviation of the activity of the CPn population for each network. Colors mean the same as in B. **F:** Linear capacity sum of the CCS population (instead of the CPn population) for the same configurations as in A. The grey bar marks the maximum capacity for the CCS population. 97

4.11 Modified network studies for structured rate networks with sigmoid non-linearity. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B-D:** Memory curves for the full model and the two networks with the biggest difference to the full network model. 99

4.12 Modified network studies for baseline spiking network. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B:** Memory curves for the full model. **C:** Memory curves for the network with separated populations and input to both populations. 100

4.13 Modified network studies for full data-based spiking network. **A:** Linear capacity sums (top) and maximum capacity delay (bottom) for modified networks. **B:** Memory curves for the full model. **C:** Memory curves for the network with separated populations and input to both populations. . 101

4.14 Modified network studies for spiking networks constructed from rate networks. **A:** Memory curves for the best configuration of a spiking network that is based on the baseline rate network ($\lambda = 20$). **B:** Memory curves for the best configuration of a spiking network that is based on the structured rate network ($\lambda = 40$). 102

5.1 **Encodings with precise spike times. A:** Input encoding with a constant rate (on average). A window with the duration of one step (colored horizontal bars) can be moved across a fixed pre-generated spike train (black) to select the subsection that is used as input. The position of the window is defined by the input $u(t)$. **B:** Input encoding with information in the rate and precise spike times. A spike train with the maximum firing rate (black spikes) is pre-generated. Depending on the input value $u(t)$, the spikes times are stretched (multiplied with a corresponding factor) and all spikes that are outside the step duration window are cut off. This results in the final input spike trains (colored spikes). 115

A.1 Firing rate histograms of the control circuits consisting of Hodgkin-Huxley neurons with conductance noise. 136
A.2 Firing rate histograms of the control circuits consisting of Hodgkin-Huxley neurons without conductance noise. 137
A.3 Firing rate histograms of the control circuits consisting of integrate-and-fire neurons. 138

B.1 BRN results, spatial encoding, removed remembered encoder capacities . 140
B.2 capacity transformation 141

List of Tables

2.1 Main neuron Parameters. The source column indicates where the value can be found, searching in the following order: main replicated paper, referenced papers, source code. If a value was given in the paper which differs from the one used in the code, the paper value is written in parenthesis. References: (1) Destexhe and Paré (1999), (2) Destexhe et al. (2001) 29

2.2 Ion channel parameters. Source definitions as in Table 2.1. References: (3) Mainen, Huguenard, and Sejnowski (1995) 29

2.3 Neuronal conductance noise parameters; source definition as in Table 2.1 . 30

2.4 Population type dependent synaptic parameters; source definition as in Table 2.1 31

2.5 Synapse parameters. Source definitions as in Table 2.1. References: (2) Destexhe et al. (2001) 32

2.6 Task and training parameters. 39

2.7 Performance measures for all networks and all tasks. Spike-based tasks are evaluated with the kappa coefficient and rate-based tasks with the correlation coefficient. The data-based column gives the absolute value and the other columns show the difference from this value. The best performance per task/row is marked in bold. Grey/blue shading denotes tasks from the categories memory/nonlinear. All values are averaged over 20 runs (10 runs in the original paper). 44

2.8 Performance measures for all control networks and all tasks expressed as the average difference (in percent) from the performance of the data-based circuit. Positive values indicating a performance improvement with respect to the data-based circuit are marked in bold. All values are averaged over 20 runs. C.f. Table 2 in original publication (Häusler and Maass, 2007). 45

3.1 Performance on tasks and capacity measures for different dynamical systems. Given as Cohen's kappa score for XOR, tXOR and XORXOR, maximum delay up to which accuracy is above chance level for delayed classification, and squared correlation coefficient for NARMA5. 75

4.1 Parameters for connection densities and reciprocities. Sources: [1] Morishima and Kawaguchi (2006); [2] Morishima et al. (2011); [3] Morita et al. (2012) 81

A.1 Performance measures for all control networks consisting of Hodgkin-Huxley neurons without intrinsic conductance noise expressed as average difference (in percent) from the performance of the data-based circuit. All values are averaged over 20 runs. 135
A.2 Performance measures for all control networks consisting of integrate-and-fire neurons expressed as average difference (in percent) from the performance of the data-based circuit. All values are averaged over 20 runs. 135

B.1 Parameters for balanced random network 139

C.1 Synaptic weights. References: [1] Morishima and Kawaguchi (2006) [2] Morishima et al. (2011) 146
C.2 Neuron model parameters for baseline spiking neuronal networks 146
C.3 Neuron model parameters for the CCS population. 147
C.4 Neuron model parameters for the CPn population. 147
C.5 Neuron model parameters spiking neuronal networks that are reconstructed based on the procedure proposed in Kim, Li, and Sejnowski (2019) 148
C.6 Synapse short-term plasticity parameters for the CPn and CCS populations. References: (1) Morishima et al. (2011) 148
C.7 Synapse parameters for the CPn and CCS populations. 148
C.8 Synapse parameters for the CPn and CCS populations. References: [1] Morishima and Kawaguchi (2006) [2] Morishima et al. (2011) 149
C.9 Neuronal parameters for the CPn and CCS populations. References: [1] Morishima et al. (2011) [2] Morishima and Kawaguchi (2006) 150

Appendix D

Index of Abbreviations

AdEx adaptive exponential integrate-and-fire

AI artificial intelligence

BRN balanced random network

CCS crossed corticostriatal

CPn corticopontine

EEG electroencephalography

ESN echo state network

fMRI functional magnetic resonance imaging

FPUT Fermi-Pasta-Ulam-Tsingou

GAN generative adversarial network

GPe external segment of the globus pallidus

iaf integrate-and-fire

IPC information processing capacity

LGN lateral geniculate nucleus

LIF leaky integrate-and-fire

LSM liquid state machine

MRI magnetic resonance imaging

NARMA nonlinear autoregressive moving average

NMDA N-methyl-D-aspartate

PPN pedunculopontine tegmental nucleus

PSP post-synaptic potential

PT pyramidal tract

RL reinforcement learning

RPE reward prediction error

SNc substantia nigra pars compacta

SNr substantia nigra pars reticulata

SNN spiking neural network

STN subthalamic nucleus

STP short-term plasticity

TD temporal difference

tXOR temporal exclusive-or

VAE variational autoencoder

VTA ventral tegmental area

XOR exclusive-or

XORXOR nested exclusive-or

Related Interesting Work from the SE Group, RWTH Aachen

The following section gives an overview of related work done at the SE Group, RWTH Aachen. More details can be found on the website `www.se-rwth.de/topics/` or in [HMR+19]. The work presented here mainly has been guided by our mission statement:

Our mission is to define, improve, and industrially apply *techniques*, *concepts*, and *methods* for *innovative and efficient development* of software and software-intensive systems, such that *high-quality* products can be developed in a *shorter period of time* and with *flexible integration* of *changing requirements*. Furthermore, we *demonstrate the applicability* of our results in various domains and potentially refine these results in a domain specific form.

Agile Model Based Software Engineering

Agility and modeling in the same project? This question was raised in [Rum04c]: "Using an executable, yet abstract and multi-view modeling language for modeling, designing and programming still allows to use an agile development process.", [JWCR18] addresses the question of how digital and organizational techniques help to cope with the physical distance of developers and [RRSW17] addresses how to teach agile modeling.

Modeling will increasingly be used in development projects if the benefits become evident early, e.g with executable UML [Rum02] and tests [Rum03]. In [GKR+06], for example, we concentrate on the integration of models and ordinary programming code. In [Rum11, Rum12] and [Rum16, Rum17], the UML/P, a variant of the UML especially designed for programming, refactoring, and evolution is defined.

The language workbench MontiCore [GKR+06, GKR+08, HKR21] is used to realize the UML/P [Sch12]. Links to further research, e.g., include a general discussion of how to manage and evolve models [LRSS10], a precise definition for model composition as well as model languages [HKR+09], and refactoring in various modeling and programming languages [PR03]. To better understand the effect of an agile evolving design, we discuss the need for semantic differencing in [MRR10].

In [FHR08] we describe a set of general requirements for model quality. Finally, [KRV06] discusses the additional roles and activities necessary in a DSL-based software development project. In [CEG+14] we discuss how to improve the reliability of adaptivity through models at runtime, which will allow developers to delay design decisions to runtime adaptation. In [KMA+16] we have also introduced a classification of ways to reuse modeled software components.

Artifacts in Complex Development Projects

Developing modern software solutions has become an increasingly complex and time consuming process. Managing the complexity, the size, and the number of artifacts developed and used during a project together with their complex relationships is not trivial [BGRW17].

To keep track of relevant structures, artifacts, and their relations in order to be able, e.g., to evolve or adapt models and their implementing code, the *artifact model* [GHR17, Gre19] was introduced. [BGRW18] and [HJK+21] explain its applicability in systems engineering based on MDSE projects and [BHR+18] applies a variant of the artifact model to evolutionary development, especially for CPS.

An artifact model is a meta-data structure that explains which kinds of artifacts, namely code files, models, requirements files, etc. exist and how these artifacts are related to each other. The artifact model, therefore, covers the wide range of human activities during the development down to fully automated, repeatable build scripts. The artifact model can be used to optimize parallelization during the development and building, but also to identify deviations of the real architecture and dependencies from the desired, idealistic architecture, for cost estimations, for requirements and bug tracing, etc. Results can be measured using metrics or visualized as graphs.

Artificial Intelligence in Software Engineering

MontiAnna is a family of explicit domain specific languages for the concise description of the architecture of (1) a neural network, (2) its training, and (3) the training data [KNP+19]. We have developed a compositional technique to integrate neural networks into larger software architectures [KRRW17] as standardized machine learning components [KPRS19]. This enables the compiler to support the systems engineer by automating the lifecycle of such components including multiple learning approaches such as supervised learning, reinforcement learning, or generative adversarial networks.

For analysis of MLOps in an agile development, a software 2.0 artifact model distinguishing different kinds of artifacts is given in [AKK+21].

According to [MRR11g] the semantic difference between two models are the elements contained in the semantics of the one model that are not elements in the semantics of the other model. A smart semantic differencing operator is an automatic procedure for computing diff witnesses for two given models. Such operators have been defined for Activity Diagrams [MRR11d], Class Diagrams [MRR11b], Feature Models [DKMR19], Statecharts [DEKR19], and Message-Driven Component and Connector Architectures [BKRW17, BKRW19]. We also developed a modeling language-independent method for determining syntactic changes that are responsible for the existence of semantic differences [KR18a].

We apply logic, knowledge representation, and intelligent reasoning to software engineering to perform correctness proofs, execute symbolic tests, or find counterexamples using a theorem prover. We have defined a core theory in [BKR+20], which is based on the core concepts of Broy's Focus theory [RR11, BR07], and applied it to challenges in intelligent flight control systems and assistance systems for air or road traffic management [KRRS19, KMP+21, HRR12].

Intelligent testing strategies have been applied to automotive software engineering [EJK+19, DGH+19, KMS+18], or more generally in systems engineering [DGH+18]. These methods are realized for a variant of SysML Activity Diagrams (ADs) and Statecharts.

Machine Learning has been applied to the massive amount of observable data in energy management for buildings [FLP+11, KLPR12] and city quarters [GLPR15] to optimize operational efficiency and prevent unneeded CO_2 emissions or reduce costs. This creates a structural and behavioral system theoretical view on cyber-physical systems understandable as essential parts of digital twins [RW18, BDH+20].

Generative Software Engineering

The UML/P language family [Rum12, Rum11, Rum16] is a simplified and semantically sound derivate of the UML designed for product and test code generation. [Sch12] describes a flexible generator for the UML/P, [Hab16] for MontiArc is used in domains such as cars or robotics

[HRR12], and [AMN+20a] for enterprise information systems based on the MontiCore language workbench [KRV10, GKR+06, GKR+08, HKR21].

In [KRV06], we discuss additional roles necessary in a model-based software development project. [GKR+06, GHK+15, GHK+15a] discuss mechanisms to keep generated and handwritten code separated. In [Wei12, HRW15, Hoe18], we demonstrate how to systematically derive a transformation language in concrete syntax and, e.g., in [HHR+15, AHRW17] we have applied this technique successfully for several UML sub-languages and DSLs.

[HNRW16] presents how to generate extensible and statically type-safe visitors. In [NRR16], we propose the use of symbols for ensuring the validity of generated source code. [GMR+16] discusses product lines of template-based code generators. We also developed an approach for engineering reusable language components [HLN+15, HLN+15a].

To understand the implications of executability for UML, we discuss the needs and the advantages of executable modeling with UML in agile projects in [Rum04c], how to apply UML for testing in [Rum03], and the advantages and perils of using modeling languages for programming in [Rum02].

Unified Modeling Language (UML) & the UML-P Tool

Starting with the early identification of challenges for the standardization of the UML in [KER99] many of our contributions build on the UML/P variant, which is described in the books [Rum16, Rum17] and is implemented in [Sch12].

Semantic variation points of the UML are discussed in [GR11]. We discuss formal semantics for UML [BHP+98] and describe UML semantics using the "System Model" [BCGR09], [BCGR09a], [BCR07b] and [BCR07a]. Semantic variation points have, e.g., been applied to define class diagram semantics [CGR08]. A precisely defined semantics for variations is applied when checking variants of class diagrams [MRR11e] and object diagrams [MRR11c] or the consistency of both kinds of diagrams [MRR11f]. We also apply these concepts to activity diagrams [MRR11a] which allows us to check for semantic differences in activity diagrams [MRR11d]. The basic semantics for ADs and their semantic variation points are given in [GRR10].

We also discuss how to ensure and identify model quality [FHR08], how models, views, and the system under development correlate to each other [BGH+98b], and how to use modeling in agile development projects [Rum04c], [Rum03] and [Rum02].

The question of how to adapt and extend the UML is discussed in [PFR02] describing product line annotations for UML and more general discussions and insights on how to use meta-modeling for defining and adapting the UML are included in [EFLR99a], [FEL+98] and [SRVK10].

The UML-P tool was conceptually defined in [Rum16, Rum17, Rum12, Rum11], got the first realization in [Sch12], and is extended in various ways, such as logically or physically distributed computation [BKRW17a]. Based on a detailed examination [JPR+22], insights are also transferred to the SysML 2.

Domain Specific Languages (DSLs)

Computer science is about languages. Domain Specific Languages (DSLs) are better to use than general-purpose programming languages but need appropriate tooling. The MontiCore language workbench [GKR+06, KRV10, Kra10, GKR+08, HKR21] allows the specification of an integrated abstract and concrete syntax format [KRV07b, HKR21] for easy development.

New languages and tools can be defined in modular forms [KRV08, GKR+07, Voe11, HLN+15, HLN+15a, HRW18, BEK+18b, BEK+19, Sch12] and can, thus, easily be reused. We discuss the roles in software development using domain specific languages already in [KRV06] and elaborate on the engineering aspect of DSL development in [CFJ+16].

[Wei12, HRW15, Hoe18] present an approach that allows the creation of transformation rules tailored to an underlying DSL. Variability in DSL definitions has been examined in [GR11, GMR+16]. [BDL+18] presents a method to derive internal DSLs from grammars. In [BJRW18], we discuss the translation from grammars to accurate metamodels. Successful applications have been carried out in the Air Traffic Management [ZPK+11] and television [DHH+20] domains. Based on the concepts described above, meta modeling, model analyses, and model evolution have been discussed in [LRSS10] and [SRVK10]. [BJRW18] describes a mapping bridge between both. DSL quality in [FHR08], instructions for defining views [GHK+07] and [PFR02], guidelines to define DSLs [KKP+09], and Eclipse-based tooling for DSLs [KRV07a] complete the collection.

A broader discussion on the *global* integration of DSMLs is given in [CBCR15] as part of [CCF+15a], and [TAB+21] discusses the compositionality of analysis techniques for models.

The MontiCore language workbench has been successfully applied to a larger number of domains, resulting in a variety of languages documented, e.g., in [AHRW17, BEH+20, BHR+21, BPR+20, HHR+15, HJRW20, HMR+19, HRR12, PBI+16, RRW15] and Ph.D. theses like [Ber10, Gre19, Hab16, Her19, Kus21, Loo17, Pin14, Plo18, Rei16, Rot17, Sch12, Wor16].

Software Language Engineering

For a systematic definition of languages using a composition of reusable and adaptable language components, we adopt an engineering viewpoint on these techniques. General ideas on how to engineer a language can be found in the GeMoC initiative [CBCR15, CCF+15a]. As said, the MontiCore language workbench provides techniques for an integrated definition of languages [KRV07b, Kra10, KRV10, HR17, HKR21, HRW18, BPR+20, BEK+19].

In [SRVK10] we discuss the possibilities and the challenges of using metamodels for language definition. Modular composition, however, is a core concept to reuse language components like in MontiCore for the frontend [Voe11, Naz17, KRV08, HLN+15, HLN+15a, HNRW16, HKR21, BEK+18b, BEK+19] and the backend [RRRW15b, NRR16, GMR+16, HKR21, BEK+18b, BBC+18]. In [GHK+15, GHK+15a], we discuss the integration of handwritten and generated object-oriented code. [KRV10] describes the roles in software development using domain specific languages.

Language derivation is to our belief a promising technique to develop new languages for a specific purpose, e.g., model transformation, that relies on existing basic languages [HRW18].

How to automatically derive such a transformation language using a concrete syntax of the base language is described in [HRW15, Wei12] and successfully applied to various DSLs.

We also applied the language derivation technique to tagging languages that decorate a base language [GLRR15] and delta languages [HHK+15, HHK+13] that are derived from base languages to be able to constructively describe differences between model variants usable to build feature sets.

The derivation of internal DSLs from grammars is discussed in [BDL+18] and a translation of grammars to accurate metamodels in [BJRW18].

Modeling Software Architecture & the MontiArc Tool

Distributed interactive systems communicate via messages on a bus, discrete event signals, streams of telephone or video data, method invocation, or data structures passed between software services.

We use streams, statemachines, and components [BR07] as well as expressive forms of composition and refinement [PR99, RW18] for semantics. Furthermore, we built a concrete tooling infrastructure called MontiArc [HRR10, HRR12] for architecture design and extensions for states [RRW13c, BKRW17a, RRW14a, Wor16]. In [RRW13], we introduce a code generation framework for MontiArc. [RRRW15b] describes how the language is composed of individual sublanguages.

MontiArc was extended to describe variability [HRR+11] using deltas [HRRS11, HKR+11] and evolution on deltas [HRRS12]. Other extensions are concerned with modeling cloud architectures [PR13], security in [HHR+15], and the robotics domain [AHRW17, AHRW17b]. Extension mechanisms for MontiArc are generally discussed in [BHH+17].

[GHK+07] and [GHK+08] close the gap between the requirements and the logical architecture and [GKPR08] extends it to model variants.

[MRR14b] provides a precise technique for verifying the consistency of architectural views [Rin14, MRR13] against a complete architecture to increase reusability. We discuss the synthesis problem for these views in [MRR14a]. An experience report [MRRW16] and a methodological embedding [DGH+19] complete the core approach.

Extensions for co-evolution of architecture are discussed in [MMR10], for powerful analyses of software architecture behavior evolution provided in [BKRW19], techniques for understanding semantic differences presented in [BKRW17], and modeling techniques to describe dynamic architectures shown in [HRR98, HKR+16, BHK+17, KKR19].

Compositionality & Modularity of Models

[HKR+09, TAB+21] motivate the basic mechanisms for modularity and compositionality for modeling. The mechanisms for distributed systems are shown in [BR07, RW18] and algebraically grounded in [HKR+07]. Semantic and methodical aspects of model composition [KRV08] led to the language workbench MontiCore [KRV10, HKR21] that can even be used to develop modeling tools in a compositional form [HKR21, HLN+15, HLN+15a, HNRW16, NRR16, HRW18, BEK+18b, BEK+19, BPR+20, KRV07b]. A set of DSL design guidelines incorporates reuse through this form of composition [KKP+09].

[Voe11] examines the composition of context conditions respectively the underlying infrastructure of the symbol table. Modular editor generation is discussed in [KRV07a]. [RRRW15b] applies compositionality to robotics control.

[CBCR15] (published in [CCF+15a]) summarizes our approach to composition and remaining challenges in form of a conceptual model of the "globalized" use of DSLs. As a new form of decomposition of model information, we have developed the concept of tagging languages in [GLRR15, MRRW16]. It allows the description of additional information for model elements in separated documents, facilitates reuse, and allows typing tags.

Semantics of Modeling Languages

The meaning of semantics and its principles like underspecification, language precision, and detailedness is discussed in [HR04]. We defined a semantic domain called "System Model" by

using mathematical theory in [RKB95, BHP+98] and [GKR96, KRB96, RK96]. An extended version especially suited for the UML is given in [GRR09], [BCGR09a] and in [BCGR09] its rationale is discussed. [BCR07a, BCR07b] contain detailed versions that are applied to class diagrams in [CGR08] or sequence diagrams in [BGH+98a].

To better understand the effect of an evolved design, detection of semantic differencing, as opposed to pure syntactical differences, is needed [MRR10]. [MRR11d, MRR11a] encode a part of the semantics to handle semantic differences of activity diagrams. [MRR11f, MRR11f] compare class and object diagrams with regard to their semantics. And [BKRW17] compares component and connector architectures similar to SysML' block definition diagrams.

In [BR07, RR11], a precise mathematical model for distributed systems based on black-box behaviors of components is defined and accompanied by automata in [Rum96]. Meta-modeling semantics is discussed in [EFLR99]. [BGH+97] discusses potential modeling languages for the description of exemplary object interaction, today called sequence diagram. [BGH+98b] discusses the relationships between a system, a view, and a complete model in the context of the UML.

[GR11] and [CGR09] discuss general requirements for a framework to describe semantic and syntactic variations of a modeling language. We apply these to class and object diagrams in [MRR11f] as well as activity diagrams in [GRR10].

[Rum12] defines the semantics in a variety of code and test case generation, refactoring, and evolution techniques. [LRSS10] discusses the evolution and related issues in greater detail. [RW18] discusses an elaborated theory for the modeling of underspecification, hierarchical composition, and refinement that can be practically applied to the development of CPS.

A first encoding of these theories in the Isabelle verification tool is defined in [BKR+20].

Evolution and Transformation of Models

Models are the central artifacts in model driven development, but as code, they are not initially correct and need to be changed, evolved, and maintained over time. Model transformation is therefore essential to effectively deal with models [CFJ+16].

Many concrete model transformation problems are discussed: evolution [LRSS10, MMR10, Rum04c, MRR10], refinement [PR99, KPR97, PR94], decomposition [PR99, KRW20], synthesis [MRR14a], refactoring [Rum12, PR03], translating models from one language into another [MRR11e, Rum12], systematic model transformation language development [Wei12, HRW15, Hoe18, HHR+15], repair of failed model evolution [KR18a].

[Rum04c] describes how comprehensible sets of such transformations support software development and maintenance [LRSS10], technologies for evolving models within a language and across languages, and mapping architecture descriptions to their implementation [MMR10]. Automaton refinement is discussed in [PR94, KPR97] and refining pipe-and-filter architectures is explained in [PR99]. This has e.g. been applied for robotics in [AHRW17, AHRW17b].

Refactorings of models are important for model driven engineering as discussed in [PR01, PR03, Rum12]. [HRRS11, HRR+11, HRRS12] encode these in constructive Delta transformations, which are defined in derivable Delta languages [HHK+13].

Translation between languages, e.g., from class diagrams into Alloy [MRR11e] allows for comparing class diagrams on a semantic level. Similarly, semantic differences of evolved activity diagrams are identified via techniques from [MRR11d] and for Simulink models in [RSW+15].

Variability and Software Product Lines (SPL)

Products often exist in various variants, for example, cars or mobile phones, where one manufacturer develops several products with many similarities but also many variations. Variants are managed in a Software Product Line (SPL) that captures product commonalities as well as differences. Feature diagrams describe variability in a top down fashion, e.g., in the automotive domain [GHK+08, GKPR08] using 150% models. Reducing overhead and associated costs is discussed in [GRJA12].

Delta modeling is a bottom up technique starting with a small, but complete base variant. Features are additive, but also can modify the core. A set of commonly applicable deltas configures a system variant. We discuss the application of this technique to Delta-MontiArc [HRRS11, HRR+11] and to Delta-Simulink [HKM+13]. Deltas can not only describe special variability but also temporal variability which allows for using them for software product line evolution [HRRS12]. [HHK+13, HHK+15] and [HRW15] describe an approach to systematically derive delta languages.

We also apply variability modeling languages to describe syntactic and semantic variation points, e.g., in UML for frameworks [PFR02] and generators [GMR+16]. Furthermore, we specified a systematic way to define variants of modeling languages [CGR09], leverage features for their compositional reuse [BEK+18b, BEK+19], and applied it as a semantic language refinement on Statecharts in [GR11].

Digital Twins and Digital Shadows in Engineering and Production

The digital transformation of production changes the life cycle of the design, the production, and the use of products [BDJ+22]. To support this transformation, we can use Digital Twins (DTs) and Digital Shadows (DSs). In [DMR+20] we define: "A digital twin of a system consists of a set of models of the system, a set of digital shadows, and provides a set of services to use the data and models purposefully with respect to the original system."

We have investigated how to synthesize self-adaptive DT architectures with model-driven methods [BBD+21a] and have applied it e.g. on a digital twin for injection molding [BDH+20]. In [BDR+21] we investigate the economic implications of digital twin services.

Digital twins also need user interaction and visualization, why we have extended the infrastructure by generating DT cockpits [DMR+20]. To support the DevOps approach in DT engineering, we have created a generator for low-code development platforms for digital twins [DHM+22] and sophisticated tool chains to generate process-aware digital twin cockpits that also include condensed forms of event logs [BMR+22].

[BBD+21b] describes a conceptual model for digital shadows covering the purpose, relevant assets, data, and metadata as well as connections to engineering models. These can be used during the runtime of a DT, e.g. when using process prediction services within DTs [BHK+21].

Integration challenges for digital twin systems-of-systems [MPRW22] include, e.g., the horizontal integration of digital twin parts, the composition of DTs for different perspectives, or how to handle different lifecycle representations of the original system.

Modeling for Cyber-Physical Systems (CPS)

Cyber-Physical Systems (CPS) [KRS12, BBR20] are complex, distributed systems that control physical entities. In [RW18], we discuss how an elaborated theory can be practically applied to

the development of CPS. Contributions for individual aspects range from requirements [GRJA12], complete product lines [HRRW12], the improvement of engineering for distributed automotive systems [HRR12, KRRW17], autonomous driving [BR12b, KKR19], and digital twin development [BDH+20] to processes and tools to improve the development as well as the product itself [BBR07].

In the aviation domain, a modeling language for uncertainty and safety events was developed, which is of interest to European avionics [ZPK+11]. Optimized [KRS+18a] and domain specific code generation [AHRW17b], and the extension to product lines of CPS [RSW+15, KRR+16, RRS+16] are key for CPS.

A component and connector architecture description language (ADL) suitable for the specific challenges in robotics is discussed in [RRW13c, RRW14a, Wor16, RRSW17, Wor21]. In [RRW12], we use this language for describing requirements and in [RRW13], we describe a code generation framework for this language. Monitoring for smart and energy efficient buildings is developed as an Energy Navigator toolset [KPR12, FPPR12, KLPR12].

Model-Driven Systems Engineering (MDSysE)

Applying models during Systems Engineering activities is based on the long tradition of contributing to systems engineering in automotive [FND+98] and [GHK+08a], which culminated in a new comprehensive model-driven development process for automotive software [KMS+18, DGH+19]. We leveraged SysML to enable the integrated flow from requirements to implementation to integration.

To facilitate the modeling of products, resources, and processes in the context of Industry 4.0, we also conceived a multi-level framework for production engineering based on these concepts [BKL+18] and addressed to bridge the gap between functions and the physical product architecture by modeling mechanical functional architectures in SysML [DRW+20]. For that purpose, we also did a detailed examination of the upcoming SysML 2.0 standard [JPR+22] and examined how to extend the SPES/CrEST methodology for a systems engineering approach [BBR20].

Research within the excellence cluster Internet of Production considers fast decision making at production time with low latencies using contextual data traces of production systems, also known as Digital Shadows (DS) [SHH+20]. We have investigated how to derive Digital Twins (DTs) for injection molding [BDH+20], how to generate interfaces between a cyber-physical system and its DT [KMR+20], and have proposed model-driven architectures for DT cockpit engineering [DMR+20].

State Based Modeling (Automata)

Today, many computer science theories are based on statemachines in various forms including Petri nets or temporal logics. Software engineering is particularly interested in using statemachines for modeling systems. Our contributions to state based modeling can currently be split into three parts: (1) understanding how to model object-oriented and distributed software using statemachines resp. Statecharts [GKR96, BCR07b, BCGR09a, BCGR09], (2) understanding the refinement [PR94, RK96, Rum96, RW18] and composition [GR95, GKR96, RW18] of statemachines, and (3) applying statemachines for modeling systems.

In [Rum96, RW18] constructive transformation rules for refining automata behavior are given and proven correct. This theory is applied to features in [KPR97]. Statemachines are embedded in the composition and behavioral specification concepts of Focus [GKR96, BR07].

We apply these techniques, e.g., in MontiArcAutomaton [RRW13, RRW14a, RRW13, RW18], in a robot task modeling language [THR+13], and in building management systems [FLP+11b].

Model-Based Assistance and Information Services (MBAIS)

Assistive systems are a special type of information system: they (1) provide situational support for human behavior (2) based on information from previously stored and real-time monitored structural context and behavior data (3) at the time the person needs or asks for it [HMR+19]. To create them, we follow a model centered architecture approach [MMR+17] which defines systems as a compound of various connected models. Used languages for their definition include DSLs for behavior and structure such as the human cognitive modeling language [MM13], goal modeling languages [MRV20, MRZ21] or UML/P based languages [MNRV19]. [MM15] describes a process of how languages for assistive systems can be created. MontiGem [AMN+20a] is used as the underlying generator technology.

We have designed a system included in a sensor floor able to monitor elderlies and analyze impact patterns for emergency events [LMK+11]. We have investigated the modeling of human contexts for the active assisted living and smart home domain [MS17] and user-centered privacy-driven systems in the IoT domain in combination with process mining systems [MKM+19], differential privacy on event logs of handling and treatment of patients at a hospital [MKB+19], the mark-up of online manuals for devices [SM18a] and websites [SM20], and solutions for privacy-aware environments for cloud services [ELR+17] and in IoT manufacturing [MNRV19]. The user-centered view of the system design allows to track who does what, when, why, where, and how with personal data, makes information about it available via information services and provides support using assistive services.

Modeling Robotics Architectures and Tasks

Robotics can be considered a special field within Cyber-Physical Systems which is defined by an inherent heterogeneity of involved domains, relevant platforms, and challenges. The engineering of robotics applications requires the composition and the interaction of diverse distributed software modules. This usually leads to complex monolithic software solutions hardly reusable, maintainable, and comprehensible, which hampers the broad propagation of robotics applications.

The MontiArcAutomaton language [RRW12, RRW14a] extends the ADL MontiArc and integrates various implemented behavior modeling languages using MontiCore [RRW13c, RRRW15b, HKR21] that perfectly fit robotic architectural modeling.

The iserveU modeling framework describes domains, actors, goals, and tasks of service robotics applications [ABH+16, ABH+17] with declarative models. Goals and tasks are translated into models of the planning domain definition language (PDDL) and then solved [ABK+17]. Thus, domain experts focus on describing the domain and its properties only.

The LightRocks [THR+13, BRS+15] framework allows robotics experts and laymen to model robotic assembly tasks. In [AHRW17, AHRW17b], we define a modular architecture modeling method for translating architecture models into modules compatible with different robotics

middleware platforms.

Many of the concepts in robotics were derived from automotive software [BBR07, BR12b].

Automotive, Autonomic Driving & Intelligent Driver Assistance

Introducing and connecting sophisticated driver assistance, infotainment, and communication systems as well as advanced active and passive safety-systems result in complex embedded systems. As these feature-driven subsystems may be arbitrarily combined by the customer, a huge amount of distinct variants needs to be managed, developed, and tested. A consistent requirement management connecting requirements with features in all development phases for the automotive domain is described in [GRJA12].

The conceptual gap between requirements and the logical architecture of a car is closed in [GHK+07, GHK+08]. A methodical embedding of the resulting function nets and their quality assurance using automated testing is given in the SMaRDT method [DGH+19, KMS+18].

[HKM+13] describes a tool for delta modeling for Simulink [HKM+13]. [HRRW12] discusses the means to extract a well-defined Software Product Line from a set of copy and paste variants.

Potential variants of components in product lines can be identified using similarity analysis of interfaces [KRR+16], or execute tests to identify similar behavior [RRS+16]. [RSW+15] describes an approach to using model checking techniques to identify behavioral differences of Simulink models. In [KKR19], we model dynamic reconfiguration of architectures applied to cooperating vehicles.

Quality assurance, especially of safety-related functions, is a highly important task. In the Carolo project [BR12b, BR12], we developed a rigorous test infrastructure for intelligent, sensor-based functions through fully-automatic simulation [BBR07]. This technique allows a dramatic speedup in the development and the evolution of autonomous car functionality, and thus enables us to develop software in an agile way [BR12b].

[MMR10] gives an overview of the state-of-the-art in development and evolution on a more general level by considering any kind of critical system that relies on architectural descriptions.

MontiSim simulates autonomous and cooperative driving behavior [GKR+17] for testing various forms of errors as well as spatial distance [FIK+18, KKRZ19]. As tooling infrastructure, the SSELab storage, versioning, and management services [HKR12] are essential for many projects.

Internet of Things, Industry 4.0 & the MontiThings Tool

The Internet of Things (IoT) requires the development of increasingly complex distributed systems. The MontiThings ecosystem [KRS+22] provides an end-to-end solution to modeling, deploying [KKR+22], and analyzing [KMR21] failure-tolerant [KRS+22] IoT systems and connecting them to synthesized digital twins [KMR+20]. We have investigated how model-driven methods can support the development of privacy-aware [ELR+17, HHK+14] cloud systems [PR13], distributed systems security [HHR+15], privacy-aware process mining [MKM+19], and distributed robotics applications [RRRW15b].

In the course of Industry 4.0, we have also turned our attention to mechanical or electrical applications [DRW+20]. We identified the digital representation, integration, and (re-)configuration of automation systems as primary Industry 4.0 concerns [WCB17]. Using a multi-level modeling framework, we support machine as a service approaches [BKL+18].

Smart Energy Management

In the past years, it became more and more evident that saving energy and reducing CO_2 emissions are important challenges. Thus, energy management in buildings as well as in neighborhoods becomes equally important to efficiently use the generated energy. Within several research projects, we developed methodologies and solutions for integrating heterogeneous systems at different scales.

During the design phase, the Energy Navigators Active Functional Specification (AFS) [FPPR12, KPR12] is used for the technical specification of building services already.

We adapted the well-known concept of statemachines to be able to describe different states of a facility and validate it against the monitored values [FLP+11b]. We show how our data model, the constraint rules, and the evaluation approach to compare sensor data can be applied [KLPR12].

Cloud Computing and Services

The paradigm of Cloud Computing is arising out of a convergence of existing technologies for web-based application and service architectures with high complexity, criticality, and new application domains. It promises to enable new business models, facilitate web-based innovations, and increase the efficiency and cost-effectiveness of web development [KRR14].

Application classes like Cyber-Physical Systems and their privacy [HHK+14, HHK+15a], Big Data, Apps, and Service Ecosystems bring attention to aspects like responsiveness, privacy, and open platforms. Regardless of the application domain, developers of such systems need robust methods and efficient, easy-to-use languages and tools [KRS12].

We tackle these challenges by perusing a model-based, generative approach [PR13]. At the core of this approach are different modeling languages that describe different aspects of a cloud-based system in a concise and technology-agnostic way. Software architecture and infrastructure models describe the system and its physical distribution on a large scale.

We apply cloud technology for the services we develop, e.g., the SSELab [HKR12] and the Energy Navigator [FPPR12, KPR12] but also for our tool demonstrators and our development platforms. New services, e.g., for collecting data from temperature sensors, cars, etc. are now easily developed and deployed, e.g., in production or Internet-of-Things environments.

Security aspects and architectures of cloud services for the digital me in a privacy-aware environment are addressed in [ELR+17].

Model-Driven Engineering of Information Systems & the MontiGem Tool

Information Systems provide information to different user groups as the main system goal. Using our experiences in the model-based generation of code with MontiCore [KRV10, HKR21], we developed several generators for such data-centric information systems.

MontiGem [AMN+20a] is a specific generator framework for data-centric business applications that uses standard models from UML/P optionally extended by GUI description models as sources [GMN+20]. While the standard semantics of these modeling languages remains untouched, the generator produces a lot of additional functionality around these models. The generator is designed flexible, modular, and incremental, handwritten and generated code pieces are well integrated [GHK+15a, NRR15a], tagging of existing models is possible [GLRR15], e.g., for the definition of roles and rights or for testing [DGH+18].

We are using MontiGem for financial management [GHK+20, ANV+18], for creating digital twin cockpits [DMR+20], and various industrial projects. MontiGem makes it easier to create low-code development platforms for digital twins [DHM+22]. When using additional DSLs, we can develop assistive systems providing user support based on goal models [MRV20], privacy-preserving information systems using privacy models and purpose trees [MNRV19], and process-aware digital twin cockpits using BPMN models [BMR+22].

We have also developed an architecture of cloud services for the digital me in a privacy-aware environment [ELR+17] and a method for retrofitting generative aspects into existing applications [DGM+21].

[ABH+16] Kai Adam, Arvid Butting, Robert Heim, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Model-Driven Separation of Concerns for Service Robotics. In *International Workshop on Domain-Specific Modeling (DSM'16)*, pages 22–27. ACM, October 2016.

[ABH+17] Kai Adam, Arvid Butting, Robert Heim, Oliver Kautz, Jérôme Pfeiffer, Bernhard Rumpe, and Andreas Wortmann. *Modeling Robotics Tasks for Better Separation of Concerns, Platform-Independence, and Reuse.* Aachener Informatik-Berichte, Software Engineering, Band 28. Shaker Verlag, December 2017.

[ABK+17] Kai Adam, Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Executing Robot Task Models in Dynamic Environments. In *Proceedings of MODELS 2017. Workshop EXE*, CEUR 2019, September 2017.

[AHRW17] Kai Adam, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. Engineering Robotics Software Architectures with Exchangeable Model Transformations. In *International Conference on Robotic Computing (IRC'17)*, pages 172–179. IEEE, April 2017.

[AHRW17b] Kai Adam, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. Modeling Robotics Software Architectures with Modular Model Transformations. *Journal of Software Engineering for Robotics (JOSER)*, 8(1):3–16, 2017.

[AKK+21] Abdallah Atouani, Jörg Christian Kirchhof, Evgeny Kusmenko, and Bernhard Rumpe. Artifact and Reference Models for Generative Machine Learning Frameworks and Build Systems. In Eli Tilevich and Coen De Roover, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 21)*, pages 55–68. ACM, October 2021.

[AMN+20a] Kai Adam, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In *40 Years EMISA: Digital Ecosystems of the Future: Methodology, Techniques and Applications (EMISA'19)*, LNI P-304, pages 59–66. Gesellschaft für Informatik e.V., May 2020.

[ANV+18] Kai Adam, Lukas Netz, Simon Varga, Judith Michael, Bernhard Rumpe, Patricia Heuser, and Peter Letmathe. Model-Based Generation of Enterprise Information Systems. In Michael Fellmann and Kurt Sandkuhl, editors, *Enterprise Modeling and Information Systems Architectures (EMISA'18)*, CEUR Workshop Proceedings 2097, pages 75–79. CEUR-WS.org, May 2018.

[BBC+18] Inga Blundell, Romain Brette, Thomas A. Cleland, Thomas G. Close, Daniel Coca, Andrew P. Davison, Sandra Diaz-Pier, Carlos Fernandez Musoles, Padraig Gleeson, Dan F. M. Goodman, Michael Hines, Michael W. Hopkins, Pramod Kumbhar, David R. Lester, Bóris Marin, Abigail Morrison, Eric Müller, Thomas Nowotny, Alexander Peyser, Dimitri Plotnikov, Paul Richmond, Andrew Rowley, Bernhard Rumpe, Marcel Stimberg, Alan B. Stokes, Adam Tomkins, Guido Trensch, Marmaduke Woodman, and Jochen Martin Eppler. Code Generation in Computational Neuroscience: A Review of Tools and Techniques. *Journal Frontiers in Neuroinformatics*, 12, 2018.

[BBD+21b] Fabian Becker, Pascal Bibow, Manuela Dalibor, Aymen Gannouni, Viviane Hahn, Christian Hopmann, Matthias Jarke, Istvan Koren, Moritz Kröger, Johannes Lipp, Judith Maibaum, Judith Michael, Bernhard Rumpe, Patrick Sapel, Niklas Schäfer, Georg J. Schmitz, Günther Schuh, and Andreas Wortmann. A Conceptual Model for Digital Shadows in Industry and its Application. In Aditya Ghose, Jennifer Horkoff, Vitor E. Silva Souza, Jeffrey Parsons, and Joerg Evermann, editors, *Conceptual Modeling, ER 2021*, pages 271–281. Springer, October 2021.

[BBD+21a] Tim Bolender, Gereon Bürvenich, Manuela Dalibor, Bernhard Rumpe, and Andreas Wortmann. Self-Adaptive Manufacturing with Digital Twins. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 156–166. IEEE Computer Society, May 2021.

[BBR07] Christian Basarke, Christian Berger, and Bernhard Rumpe. Software & Systems Engineering Process and Tools for the Development of Autonomous Driving Intelligence. *Journal of Aerospace Computing, Information, and Communication (JACIC)*, 4(12):1158–1174, 2007.

[BBR20] Manfred Broy, Wolfgang Böhm, and Bernhard Rumpe. Advanced Systems Engineering - Die Systeme der Zukunft. White paper, fortiss. Forschungsinstitut für softwareintensive Systeme, Munich, July 2020.

[BCGR09] Manfred Broy, María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Considerations and Rationale for a UML System Model. In K. Lano, editor, *UML 2 Semantics and Applications*, pages 43–61. John Wiley & Sons, November 2009.

[BCGR09a] Manfred Broy, María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Definition of the UML System Model. In K. Lano, editor, *UML 2 Semantics and Applications*, pages 63–93. John Wiley & Sons, November 2009.

[BCR07a] Manfred Broy, María Victoria Cengarle, and Bernhard Rumpe. Towards a System Model for UML. Part 2: The Control Model. Technical Report TUM-I0710, TU Munich, Germany, February 2007.

[BCR07b] Manfred Broy, María Victoria Cengarle, and Bernhard Rumpe. Towards a System Model for UML. Part 3: The State Machine Model. Technical Report TUM-I0711, TU Munich, Germany, February 2007.

[BDH+20] Pascal Bibow, Manuela Dalibor, Christian Hopmann, Ben Mainz, Bernhard Rumpe, David Schmalzing, Mauritius Schmitz, and Andreas Wortmann. Model-Driven Development of a Digital Twin for Injection Molding. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *International Conference on Advanced Information Systems Engineering (CAiSE'20)*, Lecture Notes in Computer Science 12127, pages 85–100. Springer International Publishing, June 2020.

[BDJ+22] Philipp Brauner, Manuela Dalibor, Matthias Jarke, Ike Kunze, István Koren, Gerhard Lakemeyer, Martin Liebenberg, Judith Michael, Jan Pennekamp, Christoph Quix, Bernhard Rumpe, Wil van der Aalst, Klaus Wehrle, Andreas

Wortmann, and Martina Ziefle. A Computer Science Perspective on Digital Transformation in Production. *Journal ACM Transactions on Internet of Things*, 3:1–32, February 2022.

[BDL+18] Arvid Butting, Manuela Dalibor, Gerrit Leonhardt, Bernhard Rumpe, and Andreas Wortmann. Deriving Fluent Internal Domain-specific Languages from Grammars. In *International Conference on Software Language Engineering (SLE'18)*, pages 187–199. ACM, 2018.

[BDR+21] Christian Brecher, Manuela Dalibor, Bernhard Rumpe, Katrin Schilling, and Andreas Wortmann. An Ecosystem for Digital Shadows in Manufacturing. In *54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0*. Elsevier, September 2021.

[BEH+20] Arvid Butting, Robert Eikermann, Katrin Hölldobler, Nico Jansen, Bernhard Rumpe, and Andreas Wortmann. A Library of Literals, Expressions, Types, and Statements for Compositional Language Design. *Journal of Object Technology (JOT)*, 19(3):3:1–16, October 2020.

[BEK+18b] Arvid Butting, Robert Eikermann, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Modeling Language Variability with Reusable Language Components. In *International Conference on Systems and Software Product Line (SPLC'18)*. ACM, September 2018.

[BEK+19] Arvid Butting, Robert Eikermann, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Systematic Composition of Independent Language Features. *Journal of Systems and Software (JSS)*, 152:50–69, June 2019.

[Ber10] Christian Berger. *Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles*. Aachener Informatik-Berichte, Software Engineering, Band 6. Shaker Verlag, 2010.

[BGH+97] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Towards a Precise Semantics for Object-Oriented Modeling Techniques. In Jan Bosch and Stuart Mitchell, editors, *Object-Oriented Technology, ECOOP'97 Workshop Reader*, LNCS 1357. Springer Verlag, 1997.

[BGH+98a] Ruth Breu, Radu Grosu, Christoph Hofmann, Franz Huber, Ingolf Krüger, Bernhard Rumpe, Monika Schmidt, and Wolfgang Schwerin. Exemplary and Complete Object Interaction Descriptions. *Journal Computer Standards & Interfaces*, 19(7):335–345, November 1998.

[BGH+98b] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Systems, Views and Models of UML. In *Proceedings of the Unified Modeling Language, Technical Aspects and Applications*, pages 93–109. Physica Verlag, Heidelberg, Germany, 1998.

[BGRW17] Arvid Butting, Timo Greifenberg, Bernhard Rumpe, and Andreas Wortmann. Taming the Complexity of Model-Driven Systems Engineering Projects. In *Part of the Grand Challenges in Modeling (GRAND'17) Workshop*, July 2017.

[BGRW18] Arvid Butting, Timo Greifenberg, Bernhard Rumpe, and Andreas Wortmann. On the Need for Artifact Models in Model-Driven Systems Engineering Projects. In Martina Seidl and Steffen Zschaler, editors, *Software Technologies: Applications and Foundations*, LNCS 10748, pages 146–153. Springer, January 2018.

[BHH+17] Arvid Butting, Arne Haber, Lars Hermerschmidt, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. In *European Conference on Modelling Foundations and Applications (ECMFA'17)*, LNCS 10376, pages 53–70. Springer, July 2017.

[BHK+17] Arvid Butting, Robert Heim, Oliver Kautz, Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. A Classification of Dynamic Reconfiguration in Component and Connector Architecture Description Languages. In *Proceedings of MODELS 2017. Workshop ModComp*, CEUR 2019, September 2017.

[BHK+21] Tobias Brockhoff, Malte Heithoff, István Koren, Judith Michael, Jérôme Pfeiffer, Bernhard Rumpe, Merih Seran Uysal, Wil M. P. van der Aalst, and Andreas Wortmann. Process Prediction with Digital Twins. In *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 182–187. ACM/IEEE, October 2021.

[BHP+98] Manfred Broy, Franz Huber, Barbara Paech, Bernhard Rumpe, and Katharina Spies. Software and System Modeling Based on a Unified Formal Semantics. In *Workshop on Requirements Targeting Software and Systems Engineering (RTSE'97)*, LNCS 1526, pages 43–68. Springer, 1998.

[BHR+18] Arvid Butting, Steffen Hillemacher, Bernhard Rumpe, David Schmalzing, and Andreas Wortmann. Shepherding Model Evolution in Model-Driven Development. In *Joint Proceedings of the Workshops at Modellierung 2018 (MOD-WS 2018)*, CEUR Workshop Proceedings 2060, pages 67–77. CEUR-WS.org, February 2018.

[BHR+21] Arvid Butting, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. Compositional Modelling Languages with Analytics and Construction Infrastructures Based on Object-Oriented Techniques - The MontiCore Approach. In Heinrich, Robert and Duran, Francisco and Talcott, Carolyn and Zschaler, Steffen, editor, *Composing Model-Based Analysis Tools*, pages 217–234. Springer, July 2021.

[BJRW18] Arvid Butting, Nico Jansen, Bernhard Rumpe, and Andreas Wortmann. Translating Grammars to Accurate Metamodels. In *International Conference on Software Language Engineering (SLE'18)*, pages 174–186. ACM, 2018.

[BKL+18] Christian Brecher, Evgeny Kusmenko, Achim Lindt, Bernhard Rumpe, Simon Storms, Stephan Wein, Michael von Wenckstern, and Andreas Wortmann. Multi-Level Modeling Framework for Machine as a Service Applications Based on Product Process Resource Models. In *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control (ISCSIC'18)*. ACM, September 2018.

[BKR+20] Jens Christoph Bürger, Hendrik Kausch, Deni Raco, Jan Oliver Ringert, Bernhard Rumpe, Sebastian Stüber, and Marc Wiartalla. *Towards an Isabelle Theory for distributed, interactive systems - the untimed case.* Aachener Informatik Berichte, Software Engineering, Band 45. Shaker Verlag, March 2020.

[BKRW17a] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Architectural Programming with MontiArcAutomaton. In *In 12th International Conference on Software Engineering Advances (ICSEA 2017)*, pages 213–218. IARIA XPS Press, May 2017.

[BKRW17] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Semantic Differencing for Message-Driven Component & Connector Architectures. In *International Conference on Software Architecture (ICSA'17)*, pages 145–154. IEEE, April 2017.

[BKRW19] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Continuously Analyzing Finite, Message-Driven, Time-Synchronous Component & Connector Systems During Architecture Evolution. *Journal of Systems and Software (JSS)*, 149:437–461, March 2019.

[BMR+22] Dorina Bano, Judith Michael, Bernhard Rumpe, Simon Varga, and Matthias Weske. Process-Aware Digital Twin Cockpit Synthesis from Event Logs. *Journal of Computer Languages (COLA)*, 70, June 2022.

[BPR+20] Arvid Butting, Jerome Pfeiffer, Bernhard Rumpe, and Andreas Wortmann. A Compositional Framework for Systematic Modeling Language Reuse. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 35–46. ACM, October 2020.

[BR07] Manfred Broy and Bernhard Rumpe. Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. *Informatik-Spektrum*, 30(1):3–18, Februar 2007.

[BR12b] Christian Berger and Bernhard Rumpe. Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System. In *Automotive Software Engineering Workshop (ASE'12)*, pages 789–798, 2012.

[BR12] Christian Berger and Bernhard Rumpe. Engineering Autonomous Driving Software. In C. Rouff and M. Hinchey, editors, *Experience from the DARPA Urban Challenge*, pages 243–271. Springer, Germany, 2012.

[BRS+15] Arvid Butting, Bernhard Rumpe, Christoph Schulze, Ulrike Thomas, and Andreas Wortmann. Modeling Reusable, Platform-Independent Robot Assembly Processes. In *International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2015)*, 2015.

[CBCR15] Tony Clark, Mark van den Brand, Benoit Combemale, and Bernhard Rumpe. Conceptual Model of the Globalization for Domain-Specific Languages. In *Globalizing Domain-Specific Languages*, LNCS 9400, pages 7–20. Springer, 2015.

[CCF+15a] Betty H. C. Cheng, Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, and Bernhard Rumpe, editors. *Globalizing Domain-Specific Languages*, LNCS 9400. Springer, 2015.

[CEG+14] Betty H.C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pelliccione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, and Norha M. Villegas. Using Models at Runtime to Address Assurance for Self-Adaptive Systems. In Nelly Bencomo, Robert France, Betty H.C. Cheng, and Uwe Aßmann, editors, *Models@run.time*, LNCS 8378, pages 101–136. Springer International Publishing, Switzerland, 2014.

[CFJ+16] Benoit Combemale, Robert France, Jean-Marc Jézéquel, Bernhard Rumpe, James Steel, and Didier Vojtisek. *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, November 2016.

[CGR08] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. System Model Semantics of Class Diagrams. Informatik-Bericht 2008-05, TU Braunschweig, Germany, 2008.

[CGR09] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Variability within Modeling Language Definitions. In *Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, LNCS 5795, pages 670–684. Springer, 2009.

[DEKR19] Imke Drave, Robert Eikermann, Oliver Kautz, and Bernhard Rumpe. Semantic Differencing of Statecharts for Object-oriented Systems. In Slimane Hammoudi, Luis Ferreira Pires, and Bran Selić, editors, *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD'19)*, pages 274–282. SciTePress, February 2019.

[DGH+18] Imke Drave, Timo Greifenberg, Steffen Hillemacher, Stefan Kriebel, Matthias Markthaler, Bernhard Rumpe, and Andreas Wortmann. Model-Based Testing of Software-Based System Functions. In *Conference on Software Engineering and Advanced Applications (SEAA'18)*, pages 146–153, August 2018.

[DGH+19] Imke Drave, Timo Greifenberg, Steffen Hillemacher, Stefan Kriebel, Evgeny Kusmenko, Matthias Markthaler, Philipp Orth, Karin Samira Salman, Johannes Richenhagen, Bernhard Rumpe, Christoph Schulze, Michael Wenckstern, and Andreas Wortmann. SMArDT modeling for automotive software testing. *Journal on Software: Practice and Experience*, 49(2):301–328, February 2019.

[DGM+21] Imke Drave, Akradii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. A Methodology for Retrofitting Generative Aspects in Existing Applications. *Journal of Object Technology (JOT)*, 20:1–24, November 2021.

[DHH+20] Imke Drave, Timo Henrich, Katrin Hölldobler, Oliver Kautz, Judith Michael, and Bernhard Rumpe. Modellierung, Verifikation und Synthese von validen Planungszuständen für Fernsehausstrahlungen. In Dominik Bork, Dimitris Karagiannis, and Heinrich C. Mayr, editors, *Modellierung 2020*, pages 173–188. Gesellschaft für Informatik e.V., February 2020.

[DHM+22] Manuela Dalibor, Malte Heithoff, Judith Michael, Lukas Netz, Jérôme Pfeiffer, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. Generating Customized Low-Code Development Platforms for Digital Twins. *Journal of Computer Languages (COLA)*, 70, June 2022.

[DKMR19] Imke Drave, Oliver Kautz, Judith Michael, and Bernhard Rumpe. Semantic Evolution Analysis of Feature Models. In Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi, editors, *International Systems and Software Product Line Conference (SPLC'19)*, pages 245–255. ACM, September 2019.

[DMR+20] Manuela Dalibor, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits. In Gillian Dobbie, Ulrich Frank, Gerti Kappel, Stephen W. Liddle, and Heinrich C. Mayr, editors, *Conceptual Modeling*, pages 377–387. Springer International Publishing, October 2020.

[DRW+20] Imke Drave, Bernhard Rumpe, Andreas Wortmann, Joerg Berroth, Gregor Hoepfner, Georg Jacobs, Kathrin Spuetz, Thilo Zerwas, Christian Guist, and Jens Kohl. Modeling Mechanical Functional Architectures in SysML. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 79–89. ACM, October 2020.

[EFLR99] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Meta-Modelling Semantics of UML. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, pages 45–60. Kluver Academic Publisher, 1999.

[EFLR99a] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. The UML as a Formal Modeling Notation. In J. Bézivin and P.-A. Muller, editors, *The Unified Modeling Language. «UML»'98: Beyond the Notation*, LNCS 1618, pages 336–348. Springer, Germany, 1999.

[EJK+19] Rolf Ebert, Jahir Jolianis, Stefan Kriebel, Matthias Markthaler, Benjamin Pruenster, Bernhard Rumpe, and Karin Samira Salman. Applying Product Line Testing for the Electric Drive System. In Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi, editors, *International Systems and Software Product Line Conference (SPLC'19)*, pages 14–24. ACM, September 2019.

[ELR+17] Robert Eikermann, Markus Look, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Architecting Cloud Services for the Digital me in a Privacy-Aware Environment. In Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim, editors, *Software Architecture for Big Data and the Cloud*, chapter 12, pages 207–226. Elsevier Science & Technology, June 2017.

[FEL+98] Robert France, Andy Evans, Kevin Lano, and Bernhard Rumpe. The UML as a formal modeling notation. *Journal Computer Standards & Interfaces*, 19(7):325–334, November 1998.

[FHR08] Florian Fieber, Michaela Huhn, and Bernhard Rumpe. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. *Informatik-Spektrum*, 31(5):408–424, Oktober 2008.

[FIK+18] Christian Frohn, Petyo Ilov, Stefan Kriebel, Evgeny Kusmenko, Bernhard Rumpe, and Alexander Ryndin. Distributed Simulation of Cooperatively Interacting Vehicles. In *International Conference on Intelligent Transportation Systems (ITSC'18)*, pages 596–601. IEEE, 2018.

[FLP+11] M. Norbert Fisch, Markus Look, Claas Pinkernell, Stefan Plesser, and Bernhard Rumpe. Der Energie-Navigator - Performance-Controlling für Gebäude und Anlagen. *Technik am Bau (TAB) - Fachzeitschrift für Technische Gebäudeausrüstung*, Seiten 36-41, März 2011.

[FLP+11b] M. Norbert Fisch, Markus Look, Claas Pinkernell, Stefan Plesser, and Bernhard Rumpe. State-based Modeling of Buildings and Facilities. In *Enhanced Building Operations Conference (ICEBO'11)*, 2011.

[FND+98] Max Fuchs, Dieter Nazareth, Dirk Daniel, and Bernhard Rumpe. BMW-ROOM An Object-Oriented Method for ASCET. In *SAE'98, Cobo Center (Detroit, Michigan, USA)*, Society of Automotive Engineers, 1998.

[FPPR12] M. Norbert Fisch, Claas Pinkernell, Stefan Plesser, and Bernhard Rumpe. The Energy Navigator - A Web-Platform for Performance Design and Management. In *Energy Efficiency in Commercial Buildings Conference (IEECB'12)*, 2012.

[GHK+07] Hans Grönniger, Jochen Hartmann, Holger Krahn, Stefan Kriebel, and Bernhard Rumpe. View-based Modeling of Function Nets. In *Object-oriented Modelling of Embedded Real-Time Systems Workshop (OMER4'07)*, 2007.

[GHK+08] Hans Grönniger, Jochen Hartmann, Holger Krahn, Stefan Kriebel, Lutz Rothhardt, and Bernhard Rumpe. Modelling Automotive Function Nets with Views for Features, Variants, and Modes. In *Proceedings of 4th European Congress ERTS - Embedded Real Time Software*, 2008.

[GHK+08a] Hans Grönniger, Jochen Hartmann, Holger Krahn, Stefan Kriebel, Lutz Rothhardt, and Bernhard Rumpe. View-Centric Modeling of Automotive Logical Architectures. In *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IV*, Informatik Bericht 2008-02. TU Braunschweig, 2008.

[GHK+15] Timo Greifenberg, Katrin Hölldobler, Carsten Kolassa, Markus Look, Pedram Mir Seyed Nazari, Klaus Müller, Antonio Navarro Perez, Dimitri Plotnikov, Dirk Reiß, Alexander Roth, Bernhard Rumpe, Martin Schindler, and Andreas Wortmann. A Comparison of Mechanisms for Integrating Handwritten and Generated Code for Object-Oriented Programming Languages. In *Model-Driven Engineering and Software Development Conference (MODELSWARD'15)*, pages 74–85. SciTePress, 2015.

[GHK+15a] Timo Greifenberg, Katrin Hölldobler, Carsten Kolassa, Markus Look, Pedram Mir Seyed Nazari, Klaus Müller, Antonio Navarro Perez, Dimitri Plotnikov, Dirk

Reiß, Alexander Roth, Bernhard Rumpe, Martin Schindler, and Andreas Wortmann. Integration of Handwritten and Generated Object-Oriented Code. In *Model-Driven Engineering and Software Development*, Communications in Computer and Information Science 580, pages 112–132. Springer, 2015.

[GHK+20] Arkadii Gerasimov, Patricia Heuser, Holger Ketteniß, Peter Letmathe, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend. In Judith Michael and Dominik Bork, editors, *Companion Proceedings of Modellierung 2020 Short, Workshop and Tools & Demo Papers*, pages 22–30. CEUR Workshop Proceedings, February 2020.

[GHR17] Timo Greifenberg, Steffen Hillemacher, and Bernhard Rumpe. *Towards a Sustainable Artifact Model: Artifacts in Generator-Based Model-Driven Projects*. Aachener Informatik-Berichte, Software Engineering, Band 30. Shaker Verlag, December 2017.

[GKPR08] Hans Grönniger, Holger Krahn, Claas Pinkernell, and Bernhard Rumpe. Modeling Variants of Automotive Systems using Views. In *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen*, Informatik Bericht 2008-01, pages 76–89. TU Braunschweig, 2008.

[GKR96] Radu Grosu, Cornel Klein, and Bernhard Rumpe. Enhancing the SysLab System Model with State. Technical Report TUM-I9631, TU Munich, Germany, July 1996.

[GKR+06] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. MontiCore 1.0: Ein Framework zur Erstellung und Verarbeitung domänspezifischer Sprachen. Informatik-Bericht 2006-04, CFG-Fakultät, TU Braunschweig, August 2006.

[GKR+07] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Textbased Modeling. In *4th International Workshop on Software Language Engineering, Nashville*, Informatik-Bericht 4/2007. Johannes-Gutenberg-Universität Mainz, 2007.

[GKR+08] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. MontiCore: A Framework for the Development of Textual Domain Specific Languages. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume*, pages 925–926, 2008.

[GKR+17] Filippo Grazioli, Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, and Michael von Wenckstern. Simulation Framework for Executing Component and Connector Models of Self-Driving Vehicles. In *Proceedings of MODELS 2017. Workshop EXE*, CEUR 2019, September 2017.

[GLPR15] Timo Greifenberg, Markus Look, Claas Pinkernell, and Bernhard Rumpe. Energieeffiziente Städte - Herausforderungen und Lösungen aus Sicht des Software Engineerings. In Linnhoff-Popien, Claudia and Zaddach, Michael and Grahl, Andreas, Editor, *Marktplätze im Umbruch: Digitale Strategien für Services im*

Mobilen Internet, Xpert.press, Kapitel 56, Seiten 511-520. Springer Berlin Heidelberg, April 2015.

[GLRR15] Timo Greifenberg, Markus Look, Sebastian Roidl, and Bernhard Rumpe. Engineering Tagging Languages for DSLs. In *Conference on Model Driven Engineering Languages and Systems (MODELS'15)*, pages 34–43. ACM/IEEE, 2015.

[GMN+20] Arkadii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. Continuous Transition from Model-Driven Prototype to Full-Size Real-World Enterprise Information Systems. In Bonnie Anderson, Jason Thatcher, and Rayman Meservy, editors, *25th Americas Conference on Information Systems (AMCIS 2020)*, AIS Electronic Library (AISeL), pages 1–10. Association for Information Systems (AIS), August 2020.

[GMR+16] Timo Greifenberg, Klaus Müller, Alexander Roth, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. Modeling Variability in Template-based Code Generators for Product Line Engineering. In *Modellierung 2016 Conference*, LNI 254, pages 141–156. Bonner Köllen Verlag, March 2016.

[GR95] Radu Grosu and Bernhard Rumpe. Concurrent Timed Port Automata. Technical Report TUM-I9533, TU Munich, Germany, October 1995.

[GR11] Hans Grönniger and Bernhard Rumpe. Modeling Language Variability. In *Workshop on Modeling, Development and Verification of Adaptive Systems*, LNCS 6662, pages 17–32. Springer, 2011.

[Gre19] Timo Greifenberg. *Artefaktbasierte Analyse modellgetriebener Softwareentwicklungsprojekte.* Aachener Informatik-Berichte, Software Engineering, Band 42. Shaker Verlag, August 2019.

[GRJA12] Tim Gülke, Bernhard Rumpe, Martin Jansen, and Joachim Axmann. High-Level Requirements Management and Complexity Costs in Automotive Development Projects: A Problem Statement. In *Requirements Engineering: Foundation for Software Quality (REFSQ'12)*, 2012.

[GRR09] Hans Grönniger, Jan Oliver Ringert, and Bernhard Rumpe. System Model-based Definition of Modeling Language Semantics. In *Proc. of FMOODS/FORTE 2009, LNCS 5522*, Lisbon, Portugal, 2009.

[GRR10] Hans Grönniger, Dirk Reiß, and Bernhard Rumpe. Towards a Semantics of Activity Diagrams with Semantic Variation Points. In *Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, LNCS 6394, pages 331–345. Springer, 2010.

[Hab16] Arne Haber. *MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems.* Aachener Informatik-Berichte, Software Engineering, Band 24. Shaker Verlag, September 2016.

[Her19] Lars Hermerschmidt. *Agile Modellgetriebene Entwicklung von Software Security & Privacy.* Aachener Informatik-Berichte, Software Engineering, Band 41. Shaker Verlag, June 2019.

[HHK+13] Arne Haber, Katrin Hölldobler, Carsten Kolassa, Markus Look, Klaus Müller, Bernhard Rumpe, and Ina Schaefer. Engineering Delta Modeling Languages. In *Software Product Line Conference (SPLC'13)*, pages 22–31. ACM, 2013.

[HHK+14] Martin Henze, Lars Hermerschmidt, Daniel Kerpen, Roger Häußling, Bernhard Rumpe, and Klaus Wehrle. User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things. In *Conference on Future Internet of Things and Cloud (FiCloud'14)*. IEEE, 2014.

[HHK+15] Arne Haber, Katrin Hölldobler, Carsten Kolassa, Markus Look, Klaus Müller, Bernhard Rumpe, Ina Schaefer, and Christoph Schulze. Systematic Synthesis of Delta Modeling Languages. *Journal on Software Tools for Technology Transfer (STTT)*, 17(5):601–626, October 2015.

[HHK+15a] Martin Henze, Lars Hermerschmidt, Daniel Kerpen, Roger Häußling, Bernhard Rumpe, and Klaus Wehrle. A comprehensive approach to privacy in the cloud-based Internet of Things. *Journal Future Generation Computer Systems*, 56:701–718, 2015.

[HHR+15] Lars Hermerschmidt, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. Generating Domain-Specific Transformation Languages for Component & Connector Architecture Descriptions. In *Workshop on Model-Driven Engineering for Component-Based Software Systems (ModComp'15)*, CEUR Workshop Proceedings 1463, pages 18–23, 2015.

[HJK+21] Steffen Hillemacher, Nicolas Jäckel, Christopher Kugler, Philipp Orth, David Schmalzing, and Louis Wachtmeister. Artifact-Based Analysis for the Development of Collaborative Embedded Systems. In *Model-Based Engineering of Collaborative Embedded Systems*, pages 315–331. Springer, January 2021.

[HJRW20] Katrin Hölldobler, Nico Jansen, Bernhard Rumpe, and Andreas Wortmann. Komposition Domänenspezifischer Sprachen unter Nutzung der MontiCore Language Workbench, am Beispiel SysML 2. In Dominik Bork, Dimitris Karagiannis, and Heinrich C. Mayr, editors, *Modellierung 2020*, pages 189–190. Gesellschaft für Informatik e.V., February 2020.

[HKM+13] Arne Haber, Carsten Kolassa, Peter Manhart, Pedram Mir Seyed Nazari, Bernhard Rumpe, and Ina Schaefer. First-Class Variability Modeling in Matlab/Simulink. In *Variability Modelling of Software-intensive Systems Workshop (VaMoS'13)*, pages 11–18. ACM, 2013.

[HKR+07] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. An Algebraic View on the Semantics of Model Composition. In *Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'07)*, LNCS 4530, pages 99–113. Springer, Germany, 2007.

[HKR+09] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Scaling-Up Model-Based-Development for Large Heterogeneous Systems with Compositional Modeling. In *Conference on Software Engineeering in Research and Practice (SERP'09)*, pages 172–176, July 2009.

[HKR+11] Arne Haber, Thomas Kutz, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. Delta-oriented Architectural Variability Using MontiCore. In *Software Architecture Conference (ECSA'11)*, pages 6:1–6:10. ACM, 2011.

[HKR12] Christoph Herrmann, Thomas Kurpick, and Bernhard Rumpe. SSELab: A Plug-In-Based Framework for Web-Based Project Portals. In *Developing Tools as Plug-Ins Workshop (TOPI'12)*, pages 61–66. IEEE, 2012.

[HKR+16] Robert Heim, Oliver Kautz, Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. Retrofitting Controlled Dynamic Reconfiguration into the Architecture Description Language MontiArcAutomaton. In *Software Architecture - 10th European Conference (ECSA'16)*, LNCS 9839, pages 175–182. Springer, December 2016.

[HKR21] Katrin Hölldobler, Oliver Kautz, and Bernhard Rumpe. *MontiCore Language Workbench and Library Handbook: Edition 2021.* Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, May 2021.

[HLN+15a] Arne Haber, Markus Look, Pedram Mir Seyed Nazari, Antonio Navarro Perez, Bernhard Rumpe, Steven Völkel, and Andreas Wortmann. Composition of Heterogeneous Modeling Languages. In *Model-Driven Engineering and Software Development*, Communications in Computer and Information Science 580, pages 45–66. Springer, 2015.

[HLN+15] Arne Haber, Markus Look, Pedram Mir Seyed Nazari, Antonio Navarro Perez, Bernhard Rumpe, Steven Völkel, and Andreas Wortmann. Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components. In *Model-Driven Engineering and Software Development Conference (MODELSWARD'15)*, pages 19–31. SciTePress, 2015.

[HMR+19] Katrin Hölldobler, Judith Michael, Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. Innovations in Model-based Software and Systems Engineering. *Journal of Object Technology (JOT)*, 18(1):1–60, July 2019.

[HNRW16] Robert Heim, Pedram Mir Seyed Nazari, Bernhard Rumpe, and Andreas Wortmann. Compositional Language Engineering using Generated, Extensible, Static Type Safe Visitors. In *Conference on Modelling Foundations and Applications (ECMFA)*, LNCS 9764, pages 67–82. Springer, July 2016.

[Hoe18] Katrin Hölldobler. *MontiTrans: Agile, modellgetriebene Entwicklung von und mit domänenspezifischen, kompositionalen Transformationssprachen.* Aachener Informatik-Berichte, Software Engineering, Band 36. Shaker Verlag, December 2018.

[HR04] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer Journal*, 37(10):64–72, October 2004.

[HR17] Katrin Hölldobler and Bernhard Rumpe. *MontiCore 5 Language Workbench Edition 2017.* Aachener Informatik-Berichte, Software Engineering, Band 32. Shaker Verlag, December 2017.

[HRR98] Franz Huber, Andreas Rausch, and Bernhard Rumpe. Modeling Dynamic Component Interfaces. In *Technology of Object-Oriented Languages and Systems (TOOLS 26)*, pages 58–70. IEEE, 1998.

[HRR10] Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. Towards Architectural Programming of Embedded Systems. In *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteterSysteme VI*, Informatik-Bericht 2010-01, pages 13 – 22. fortiss GmbH, Germany, 2010.

[HRR+11] Arne Haber, Holger Rendel, Bernhard Rumpe, Ina Schaefer, and Frank van der Linden. Hierarchical Variability Modeling for Software Architectures. In *Software Product Lines Conference (SPLC'11)*, pages 150–159. IEEE, 2011.

[HRR12] Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen University, February 2012.

[HRRS11] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. Delta Modeling for Software Architectures. In *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteterSysteme VII*, pages 1 – 10. fortiss GmbH, 2011.

[HRRS12] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. Evolving Delta-oriented Software Product Line Architectures. In *Large-Scale Complex IT Systems. Development, Operation and Management, 17th Monterey Workshop 2012*, LNCS 7539, pages 183–208. Springer, 2012.

[HRRW12] Christian Hopp, Holger Rendel, Bernhard Rumpe, and Fabian Wolf. Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware. In *Software Engineering Conference (SE'12)*, LNI 198, Seiten 181-192, 2012.

[HRW15] Katrin Hölldobler, Bernhard Rumpe, and Ingo Weisemöller. Systematically Deriving Domain-Specific Transformation Languages. In *Conference on Model Driven Engineering Languages and Systems (MODELS'15)*, pages 136–145. ACM/IEEE, 2015.

[HRW18] Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. Software Language Engineering in the Large: Towards Composing and Deriving Languages. *Journal Computer Languages, Systems & Structures*, 54:386–405, 2018.

[JPR+22] Nico Jansen, Jerome Pfeiffer, Bernhard Rumpe, David Schmalzing, and Andreas Wortmann. The Language of SysML v2 under the Magnifying Glass. *Journal of Object Technology (JOT)*, 21:1–15, July 2022.

[JWCR18] Rodi Jolak, Andreas Wortmann, Michel Chaudron, and Bernhard Rumpe. Does Distance Still Matter? Revisiting Collaborative Distributed Software Design. *IEEE Software Journal*, 35(6):40–47, 2018.

[KER99] Stuart Kent, Andy Evans, and Bernhard Rumpe. UML Semantics FAQ. In A. Moreira and S. Demeyer, editors, *Object-Oriented Technology, ECOOP'99 Workshop Reader*, LNCS 1743, Berlin, 1999. Springer Verlag.

[KKP+09] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain Specific Languages. In *Domain-Specific Modeling Workshop (DSM'09)*, Techreport B-108, pages 7–13. Helsinki School of Economics, October 2009.

[KKR19] Nils Kaminski, Evgeny Kusmenko, and Bernhard Rumpe. Modeling Dynamic Architectures of Self-Adaptive Cooperative Systems. *Journal of Object Technology (JOT)*, 18(2):1–20, July 2019.

[KKR+22] Jörg Christian Kirchhof, Anno Kleiss, Bernhard Rumpe, David Schmalzing, Philipp Schneider, and Andreas Wortmann. Model-driven Self-adaptive Deployment of Internet of Things Applications with Automated Modification Proposals. *Journal ACM Transactions on Internet of Things*, 3:1–30, November 2022.

[KKRZ19] Jörg Christian Kirchhof, Evgeny Kusmenko, Bernhard Rumpe, and Hengwen Zhang. Simulation as a Service for Cooperative Vehicles. In Loli Burgueño, Alexander Pretschner, Sebastian Voss, Michel Chaudron, Jörg Kienzle, Markus Völter, Sébastien Gérard, Mansooreh Zahedi, Erwan Bousse, Arend Rensink, Fiona Polack, Gregor Engels, and Gerti Kappel, editors, *Proceedings of MODELS 2019. Workshop MASE*, pages 28–37. IEEE, September 2019.

[KLPR12] Thomas Kurpick, Markus Look, Claas Pinkernell, and Bernhard Rumpe. Modeling Cyber-Physical Systems: Model-Driven Specification of Energy Efficient Buildings. In *Modelling of the Physical World Workshop (MOTPW'12)*, pages 2:1–2:6. ACM, October 2012.

[KMA+16] Jörg Kienzle, Gunter Mussbacher, Omar Alam, Matthias Schöttle, Nicolas Belloir, Philippe Collet, Benoit Combemale, Julien Deantoni, Jacques Klein, and Bernhard Rumpe. VCU: The Three Dimensions of Reuse. In *Conference on Software Reuse (ICSR'16)*, LNCS 9679, pages 122–137. Springer, June 2016.

[KMP+21] Hendrik Kausch, Judith Michael, Mathias Pfeiffer, Deni Raco, Bernhard Rumpe, and Andreas Schweiger. Model-Based Development and Logical AI for Secure and Safe Avionics Systems: A Verification Framework for SysML Behavior Specifications. In *Aerospace Europe Conference 2021 (AEC 2021)*. Council of European Aerospace Societies (CEAS), November 2021.

[KMR+20] Jörg Christian Kirchhof, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. Model-driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 90–101. ACM, October 2020.

[KMR21] Jörg Christian Kirchhof, Lukas Malcher, and Bernhard Rumpe. Understanding and Improving Model-Driven IoT Systems through Accompanying Digital Twins. In Eli Tilevich and Coen De Roover, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 21)*, pages 197–209. ACM, October 2021.

[KMS+18] Stefan Kriebel, Matthias Markthaler, Karin Samira Salman, Timo Greifenberg, Steffen Hillemacher, Bernhard Rumpe, Christoph Schulze, Andreas Wortmann,

Philipp Orth, and Johannes Richenhagen. Improving Model-based Testing in Automotive Software Engineering. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE'18)*, pages 172–180. ACM, June 2018.

[KNP+19] Evgeny Kusmenko, Sebastian Nickels, Svetlana Pavlitskaya, Bernhard Rumpe, and Thomas Timmermanns. Modeling and Training of Neural Processing Systems. In Marouane Kessentini, Tao Yue, Alexander Pretschner, Sebastian Voss, and Loli Burgueño, editors, *Conference on Model Driven Engineering Languages and Systems (MODELS'19)*, pages 283–293. IEEE, September 2019.

[KPR97] Cornel Klein, Christian Prehofer, and Bernhard Rumpe. Feature Specification and Refinement with State Transition Diagrams. In *Workshop on Feature Interactions in Telecommunications Networks and Distributed Systems*, pages 284–297. IOS-Press, 1997.

[KPR12] Thomas Kurpick, Claas Pinkernell, and Bernhard Rumpe. Der Energie Navigator. In H. Lichter and B. Rumpe, Editoren, *Entwicklung und Evolution von Forschungssoftware. Tagungsband, Rolduc, 10.-11.11.2011*, Aachener Informatik-Berichte, Software Engineering, Band 14. Shaker Verlag, Aachen, Deutschland, 2012.

[KPRS19] Evgeny Kusmenko, Svetlana Pavlitskaya, Bernhard Rumpe, and Sebastian Stüber. On the Engineering of AI-Powered Systems. In Lisa O'Conner, editor, *ASE19. Software Engineering Intelligence Workshop (SEI19)*, pages 126–133. IEEE, November 2019.

[KR18a] Oliver Kautz and Bernhard Rumpe. On Computing Instructions to Repair Failed Model Refinements. In *Conference on Model Driven Engineering Languages and Systems (MODELS'18)*, pages 289–299. ACM, October 2018.

[Kra10] Holger Krahn. *MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering.* Aachener Informatik-Berichte, Software Engineering, Band 1. Shaker Verlag, März 2010.

[KRB96] Cornel Klein, Bernhard Rumpe, and Manfred Broy. A stream-based mathematical model for distributed information processing systems - SysLab system model. In *Workshop on Formal Methods for Open Object-based Distributed Systems*, IFIP Advances in Information and Communication Technology, pages 323–338. Chapmann & Hall, 1996.

[KRR14] Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe. *Trusted Cloud Computing.* Springer, Schweiz, December 2014.

[KRR+16] Philipp Kehrbusch, Johannes Richenhagen, Bernhard Rumpe, Axel Schloßer, and Christoph Schulze. Interface-based Similarity Analysis of Software Components for the Automotive Industry. In *International Systems and Software Product Line Conference (SPLC '16)*, pages 99–108. ACM, September 2016.

[KRRS19] Stefan Kriebel, Deni Raco, Bernhard Rumpe, and Sebastian Stüber. Model-Based Engineering for Avionics: Will Specification and Formal Verification e.g.

Based on Broy's Streams Become Feasible? In Stephan Krusche, Kurt Schneider, Marco Kuhrmann, Robert Heinrich, Reiner Jung, Marco Konersmann, Eric Schmieders, Steffen Helke, Ina Schaefer, Andreas Vogelsang, Björn Annighöfer, Andreas Schweiger, Marina Reich, and André van Hoorn, editors, *Proceedings of the Workshops of the Software Engineering Conference. Workshop on Avionics Systems and Software Engineering (AvioSE'19)*, CEUR Workshop Proceedings 2308, pages 87–94. CEUR Workshop Proceedings, February 2019.

[KRRW17] Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, and Michael von Wenckstern. Modeling Architectures of Cyber-Physical Systems. In *European Conference on Modelling Foundations and Applications (ECMFA'17)*, LNCS 10376, pages 34–50. Springer, July 2017.

[KRS12] Stefan Kowalewski, Bernhard Rumpe, and Andre Stollenwerk. Cyber-Physical Systems - eine Herausforderung für die Automatisierungstechnik? In *Proceedings of Automation 2012, VDI Berichte 2012*, Seiten 113-116. VDI Verlag, 2012.

[KRS+18a] Evgeny Kusmenko, Bernhard Rumpe, Sascha Schneiders, and Michael von Wenckstern. Highly-Optimizing and Multi-Target Compiler for Embedded System Models: C++ Compiler Toolchain for the Component and Connector Language EmbeddedMontiArc. In *Conference on Model Driven Engineering Languages and Systems (MODELS'18)*, pages 447 – 457. ACM, October 2018.

[KRS+22] Jörg Christian Kirchhof, Bernhard Rumpe, David Schmalzing, and Andreas Wortmann. MontiThings: Model-driven Development and Deployment of Reliable IoT Applications. *Journal of Systems and Software (JSS)*, 183:1–21, January 2022.

[KRV06] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Roles in Software Development using Domain Specific Modelling Languages. In *Domain-Specific Modeling Workshop (DSM'06)*, Technical Report TR-37, pages 150–158. Jyväskylä University, Finland, 2006.

[KRV07a] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Efficient Editor Generation for Compositional DSLs in Eclipse. In *Domain-Specific Modeling Workshop (DSM'07)*, Technical Reports TR-38. Jyväskylä University, Finland, 2007.

[KRV07b] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Integrated Definition of Abstract and Concrete Syntax for Textual Languages. In *Conference on Model Driven Engineering Languages and Systems (MODELS'07)*, LNCS 4735, pages 286–300. Springer, 2007.

[KRV08] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Monticore: Modular Development of Textual Domain Specific Languages. In *Conference on Objects, Models, Components, Patterns (TOOLS-Europe'08)*, LNBIP 11, pages 297–315. Springer, 2008.

[KRV10] Holger Krahn, Bernhard Rumpe, and Stefen Völkel. MontiCore: a Framework for Compositional Development of Domain Specific Languages. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(5):353–372, September 2010.

[KRW20] Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Automated semantics-preserving parallel decomposition of finite component and connector architectures. *Automated Software Engineering Journal*, 27:119–151, April 2020.

[Kus21] Evgeny Kusmenko. *Model-Driven Development Methodology and Domain-Specific Languages for the Design of Artificial Intelligence in Cyber-Physical Systems*. Aachener Informatik-Berichte, Software Engineering, Band 49. Shaker Verlag, November 2021.

[LMK+11] Philipp Leusmann, Christian Möllering, Lars Klack, Kai Kasugai, Bernhard Rumpe, and Martina Ziefle. Your Floor Knows Where You Are: Sensing and Acquisition of Movement Data. In Arkady Zaslavsky, Panos K. Chrysanthis, Dik Lun Lee, Dipanjan Chakraborty, Vana Kalogeraki, Mohamed F. Mokbel, and Chi-Yin Chow, editors, *12th IEEE International Conference on Mobile Data Management (Volume 2)*, pages 61–66. IEEE, June 2011.

[Loo17] Markus Look. *Modellgetriebene, agile Entwicklung und Evolution mehrbenutzerfähiger Enterprise Applikationen mit MontiEE*. Aachener Informatik-Berichte, Software Engineering, Band 27. Shaker Verlag, March 2017.

[LRSS10] Tihamer Levendovszky, Bernhard Rumpe, Bernhard Schätz, and Jonathan Sprinkle. Model Evolution and Management. In *Model-Based Engineering of Embedded Real-Time Systems Workshop (MBEERTS'10)*, LNCS 6100, pages 241–270. Springer, 2010.

[MKB+19] Felix Mannhardt, Agnes Koschmider, Nathalie Baracaldo, Matthias Weidlich, and Judith Michael. Privacy-Preserving Process Mining: Differential Privacy for Event Logs. *Business & Information Systems Engineering*, 61(5):1–20, October 2019.

[MKM+19] Judith Michael, Agnes Koschmider, Felix Mannhardt, Nathalie Baracaldo, and Bernhard Rumpe. User-Centered and Privacy-Driven Process Mining System Design for IoT. In Cinzia Cappiello and Marcela Ruiz, editors, *Proceedings of CAiSE Forum 2019: Information Systems Engineering in Responsible Information Systems*, pages 194–206. Springer, June 2019.

[MM13] Judith Michael and Heinrich C. Mayr. Conceptual modeling for ambient assistance. In *Conceptual Modeling - ER 2013*, LNCS 8217, pages 403–413. Springer, 2013.

[MM15] Judith Michael and Heinrich C. Mayr. Creating a domain specific modelling method for ambient assistance. In *International Conference on Advances in ICT for Emerging Regions (ICTer2015)*, pages 119–124. IEEE, 2015.

[MMR10] Tom Mens, Jeff Magee, and Bernhard Rumpe. Evolving Software Architecture Descriptions of Critical Systems. *IEEE Computer Journal*, 43(5):42–48, May 2010.

[MMR+17] Heinrich C. Mayr, Judith Michael, Suneth Ranasinghe, Vladimir A. Shekhovtsov, and Claudia Steinberger. *Model Centered Architecture*, pages 85–104. Springer International Publishing, 2017.

[MNRV19] Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. Towards Privacy-Preserving IoT Systems Using Model Driven Engineering. In Nicolas Ferry, Antonio Cicchetti, Federico Ciccozzi, Arnor Solberg, Manuel Wimmer, and Andreas Wortmann, editors, *Proceedings of MODELS 2019. Workshop MDE4IoT*, pages 595–614. CEUR Workshop Proceedings, September 2019.

[MPRW22] Judith Michael, Jérôme Pfeiffer, Bernhard Rumpe, and Andreas Wortmann. Integration Challenges for Digital Twin Systems-of-Systems. In *10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, pages 9–12. ACM, May 2022.

[MRR10] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. A Manifesto for Semantic Model Differencing. In *Proceedings Int. Workshop on Models and Evolution (ME'10)*, LNCS 6627, pages 194–203. Springer, 2010.

[MRR11d] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic Differencing for Activity Diagrams. In *Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pages 179–189. ACM, 2011.

[MRR11a] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. An Operational Semantics for Activity Diagrams using SMV. Technical Report AIB-2011-07, RWTH Aachen University, Aachen, Germany, July 2011.

[MRR11e] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In *Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, LNCS 6981, pages 592–607. Springer, 2011.

[MRR11b] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CDDiff: Semantic Differencing for Class Diagrams. In Mira Mezini, editor, *ECOOP 2011 - Object-Oriented Programming*, pages 230–254. Springer Berlin Heidelberg, 2011.

[MRR11c] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Modal Object Diagrams. In *Object-Oriented Programming Conference (ECOOP'11)*, LNCS 6813, pages 281–305. Springer, 2011.

[MRR11f] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically Configurable Consistency Analysis for Class and Object Diagrams. In *Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, LNCS 6981, pages 153–167. Springer, 2011.

[MRR11g] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Summarizing Semantic Model Differences. In Bernhard Schätz, Dirk Deridder, Alfonso Pierantonio, Jonathan Sprinkle, and Dalila Tamzalit, editors, *ME 2011 - Models and Evolution*, October 2011.

[MRR13] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Synthesis of Component and Connector Models from Crosscutting Structural Views. In Meyer, B. and Baresi, L. and Mezini, M., editor, *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, pages 444–454. ACM New York, 2013.

[MRR14a] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Synthesis of Component and Connector Models from Crosscutting Structural Views (extended abstract). In Wilhelm Hasselbring and Nils Christian Ehmke, editors, *Software Engineering 2014*, LNI 227, pages 63–64. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH, 2014.

[MRR14b] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Verifying Component and Connector Models against Crosscutting Structural Views. In *International Conference on Software Engineering (ICSE'14)*, pages 95–105. ACM, 2014.

[MRRW16] Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe, and Michael von Wenckstern. Consistent Extra-Functional Properties Tagging for Component and Connector Models. In *Workshop on Model-Driven Engineering for Component-Based Software Systems (ModComp'16)*, CEUR Workshop Proceedings 1723, pages 19–24, October 2016.

[MRV20] Judith Michael, Bernhard Rumpe, and Simon Varga. Human behavior, goals and model-driven software engineering for assistive systems. In Agnes Koschmider, Judith Michael, and Bernhard Thalheim, editors, *Enterprise Modeling and Information Systems Architectures (EMSIA 2020)*, pages 11–18. CEUR Workshop Proceedings, June 2020.

[MRZ21] Judith Michael, Bernhard Rumpe, and Lukas Tim Zimmermann. Goal Modeling and MDSE for Behavior Assistance. In *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 370–379. ACM/IEEE, October 2021.

[MS17] Judith Michael and Claudia Steinberger. Context modeling for active assistance. In Cristina Cabanillas, Sergio España, and Siamak Farshidi, editors, *Proc. of the ER Forum 2017 and the ER 2017 Demo Track co-located with the 36th Int. Conference on Conceptual Modelling (ER 2017)*, pages 221–234, 2017.

[Naz17] Pedram Mir Seyed Nazari. *MontiCore: Efficient Development of Composed Modeling Language Essentials*. Aachener Informatik-Berichte, Software Engineering, Band 29. Shaker Verlag, June 2017.

[NRR15a] Pedram Mir Seyed Nazari, Alexander Roth, and Bernhard Rumpe. Mixed Generative and Handcoded Development of Adaptable Data-centric Business Applications. In *Domain-Specific Modeling Workshop (DSM'15)*, pages 43–44. ACM, 2015.

[NRR16] Pedram Mir Seyed Nazari, Alexander Roth, and Bernhard Rumpe. An Extended Symbol Table Infrastructure to Manage the Composition of Output-Specific Generator Information. In *Modellierung 2016 Conference*, LNI 254, pages 133–140. Bonner Köllen Verlag, March 2016.

[PR13] Antonio Navarro Pérez and Bernhard Rumpe. Modeling Cloud Architectures as Interactive Systems. In *Model-Driven Engineering for High Performance and Cloud Computing Workshop*, CEUR Workshop Proceedings 1118, pages 15–24, 2013.

[PBI+16] Dimitri Plotnikov, Inga Blundell, Tammo Ippen, Jochen Martin Eppler, Abigail Morrison, and Bernhard Rumpe. NESTML: a modeling language for spiking neurons. In *Modellierung 2016 Conference*, LNI 254, pages 93–108. Bonner Köllen Verlag, March 2016.

[PFR02] Wolfgang Pree, Marcus Fontoura, and Bernhard Rumpe. Product Line Annotations with UML-F. In *Software Product Lines Conference (SPLC'02)*, LNCS 2379, pages 188–197. Springer, 2002.

[Pin14] Claas Pinkernell. *Energie Navigator: Software-gestützte Optimierung der Energieeffizienz von Gebäuden und technischen Anlagen.* Aachener Informatik-Berichte, Software Engineering, Band 17. Shaker Verlag, 2014.

[Plo18] Dimitri Plotnikov. *NESTML - die domänenspezifische Sprache für den NEST-Simulator neuronaler Netzwerke im Human Brain Project.* Aachener Informatik-Berichte, Software Engineering, Band 33. Shaker Verlag, February 2018.

[PR94] Barbara Paech and Bernhard Rumpe. A new Concept of Refinement used for Behaviour Modelling with Automata. In *Proceedings of the Industrial Benefit of Formal Methods (FME'94)*, LNCS 873, pages 154–174. Springer, 1994.

[PR99] Jan Philipps and Bernhard Rumpe. Refinement of Pipe-and-Filter Architectures. In *Congress on Formal Methods in the Development of Computing System (FM'99)*, LNCS 1708, pages 96–115. Springer, 1999.

[PR01] Jan Philipps and Bernhard Rumpe. Roots of Refactoring. In Kilov, H. and Baclavski, K., editor, *Tenth OOPSLA Workshop on Behavioral Semantics. Tampa Bay, Florida, USA, October 15.* Northeastern University, 2001.

[PR03] Jan Philipps and Bernhard Rumpe. Refactoring of Programs and Specifications. In Kilov, H. and Baclavski, K., editor, *Practical Foundations of Business and System Specifications*, pages 281–297. Kluwer Academic Publishers, 2003.

[Rei16] Dirk Reiß. *Modellgetriebene generative Entwicklung von Web-Informationssystemen.* Aachener Informatik-Berichte, Software Engineering, Band 22. Shaker Verlag, May 2016.

[Rin14] Jan Oliver Ringert. *Analysis and Synthesis of Interactive Component and Connector Systems.* Aachener Informatik-Berichte, Software Engineering, Band 19. Shaker Verlag, Aachen, Germany, December 2014.

[RK96] Bernhard Rumpe and Cornel Klein. Automata Describing Object Behavior. In B. Harvey and H. Kilov, editors, *Object-Oriented Behavioral Specifications*, pages 265–286. Kluwer Academic Publishers, 1996.

[RKB95] Bernhard Rumpe, Cornel Klein, and Manfred Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - Syslab-Systemmodell. Technischer Bericht TUM-I9510, TU München, Deutschland, März 1995.

[Rot17] Alexander Roth. *Adaptable Code Generation of Consistent and Customizable Data Centric Applications with MontiDex.* Aachener Informatik-Berichte, Software Engineering, Band 31. Shaker Verlag, December 2017.

[RR11] Jan Oliver Ringert and Bernhard Rumpe. A Little Synopsis on Streams, Stream Processing Functions, and State-Based Stream Processing. *International Journal of Software and Informatics*, 2011.

[RRRW15b] Jan Oliver Ringert, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Language and Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. *Journal of Software Engineering for Robotics (JOSER)*, 6(1):33–57, 2015.

[RRS+16] Johannes Richenhagen, Bernhard Rumpe, Axel Schloßer, Christoph Schulze, Kevin Thissen, and Michael von Wenckstern. Test-driven Semantical Similarity Analysis for Software Product Line Extraction. In *International Systems and Software Product Line Conference (SPLC '16)*, pages 174–183. ACM, September 2016.

[RRSW17] Jan Oliver Ringert, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. Teaching Agile Model-Driven Engineering for Cyber-Physical Systems. In *International Conference on Software Engineering: Software Engineering and Education Track (ICSE'17)*, pages 127–136. IEEE, May 2017.

[RRW12] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. A Requirements Modeling Language for the Component Behavior of Cyber Physical Robotics Systems. In Seyff, N. and Koziolek, A., editor, *Modelling and Quality in Requirements Engineering: Essays Dedicated to Martin Glinz on the Occasion of His 60th Birthday*, pages 133–146. Monsenstein und Vannerdat, Münster, 2012.

[RRW13] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. From Software Architecture Structure and Behavior Modeling to Implementations of Cyber-Physical Systems. In *Software Engineering Workshopband (SE'13)*, LNI 215, pages 155–170, 2013.

[RRW13c] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. MontiArcAutomaton: Modeling Architecture and Behavior of Robotic Systems. In *Conference on Robotics and Automation (ICRA'13)*, pages 10–12. IEEE, 2013.

[RRW14a] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. *Architecture and Behavior Modeling of Cyber-Physical Systems with MontiArcAutomaton.* Aachener Informatik-Berichte, Software Engineering, Band 20. Shaker Verlag, December 2014.

[RRW15] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. Tailoring the MontiArcAutomaton Component & Connector ADL for Generative Development. In *MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, pages 41–47. ACM, 2015.

[RSW+15] Bernhard Rumpe, Christoph Schulze, Michael von Wenckstern, Jan Oliver Ringert, and Peter Manhart. Behavioral Compatibility of Simulink Models for Product Line Maintenance and Evolution. In *Software Product Line Conference (SPLC'15)*, pages 141–150. ACM, 2015.

[Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme.* Herbert Utz Verlag Wissenschaft, München, Deutschland, 1996.

[Rum02] Bernhard Rumpe. Executable Modeling with UML - A Vision or a Nightmare? In T. Clark and J. Warmer, editors, *Issues & Trends of Information Technology Management in Contemporary Associations, Seattle*, pages 697–701. Idea Group Publishing, London, 2002.

[Rum03] Bernhard Rumpe. Model-Based Testing of Object-Oriented Systems. In *Symposium on Formal Methods for Components and Objects (FMCO'02)*, LNCS 2852, pages 380–402. Springer, November 2003.

[Rum04c] Bernhard Rumpe. Agile Modeling with the UML. In *Workshop on Radical Innovations of Software and Systems Engineering in the Future (RISSEF'02)*, LNCS 2941, pages 297–309. Springer, October 2004.

[Rum11] Bernhard Rumpe. *Modellierung mit UML, 2te Auflage*. Springer Berlin, September 2011.

[Rum12] Bernhard Rumpe. *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring, 2te Auflage*. Springer Berlin, Juni 2012.

[Rum16] Bernhard Rumpe. *Modeling with UML: Language, Concepts, Methods*. Springer International, July 2016.

[Rum17] Bernhard Rumpe. *Agile Modeling with UML: Code Generation, Testing, Refactoring*. Springer International, May 2017.

[RW18] Bernhard Rumpe and Andreas Wortmann. Abstraction and Refinement in Hierarchically Decomposable and Underspecified CPS-Architectures. In Lohstroh, Marten and Derler, Patricia Sirjani, Marjan, editor, *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, LNCS 10760, pages 383–406. Springer, 2018.

[Sch12] Martin Schindler. *Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P*. Aachener Informatik-Berichte, Software Engineering, Band 11. Shaker Verlag, 2012.

[SHH+20] Günther Schuh, Constantin Häfner, Christian Hopmann, Bernhard Rumpe, Matthias Brockmann, Andreas Wortmann, Judith Maibaum, Manuela Dalibor, Pascal Bibow, Patrick Sapel, and Moritz Kröger. Effizientere Produktion mit Digitalen Schatten. *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb*, 115(special):105–107, April 2020.

[SM18a] Claudia Steinberger and Judith Michael. Towards Cognitive Assisted Living 3.0. In *International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops 2018)*, pages 687–692. IEEE, march 2018.

[SM20] Claudia Steinberger and Judith Michael. Using Semantic Markup to Boost Context Awareness for Assistive Systems. In *Smart Assisted Living: Toward An Open Smart-Home Infrastructure*, Computer Communications and Networks, pages 227–246. Springer International Publishing, 2020.

[SRVK10] Jonathan Sprinkle, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. Metamodelling: State of the Art and Research Challenges. In *Model-Based Engineering of Embedded Real-Time Systems Workshop (MBEERTS'10)*, LNCS 6100, pages 57–76. Springer, 2010.

[TAB+21] Carolyn Talcott, Sofia Ananieva, Kyungmin Bae, Benoit Combemale, Robert Heinrich, Mark Hills, Narges Khakpour, Ralf Reussner, Bernhard Rumpe, Patrizia Scandurra, and Hans Vangheluwe. Composition of Languages, Models, and Analyses. In Heinrich, Robert and Duran, Francisco and Talcott, Carolyn and Zschaler, Steffen, editor, *Composing Model-Based Analysis Tools*, pages 45–70. Springer, July 2021.

[THR+13] Ulrike Thomas, Gerd Hirzinger, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. A New Skill Based Robot Programming Language Using UML/P Statecharts. In *Conference on Robotics and Automation (ICRA'13)*, pages 461–466. IEEE, 2013.

[Voe11] Steven Völkel. *Kompositionale Entwicklung domänenspezifischer Sprachen.* Aachener Informatik-Berichte, Software Engineering, Band 9. Shaker Verlag, 2011.

[WCB17] Andreas Wortmann, Benoit Combemale, and Olivier Barais. A Systematic Mapping Study on Modeling for Industry 4.0. In *Conference on Model Driven Engineering Languages and Systems (MODELS'17)*, pages 281–291. IEEE, September 2017.

[Wei12] Ingo Weisemöller. *Generierung domänenspezifischer Transformationssprachen.* Aachener Informatik-Berichte, Software Engineering, Band 12. Shaker Verlag, 2012.

[Wor16] Andreas Wortmann. *An Extensible Component & Connector Architecture Description Infrastructure for Multi-Platform Modeling.* Aachener Informatik-Berichte, Software Engineering, Band 25. Shaker Verlag, November 2016.

[Wor21] Andreas Wortmann. *Model-Driven Architecture and Behavior of Cyber-Physical Systems.* Aachener Informatik-Berichte, Software Engineering, Band 50. Shaker Verlag, October 2021.

[ZPK+11] Massimiliano Zanin, David Perez, Dimitrios S Kolovos, Richard F Paige, Kumardev Chatterjee, Andreas Horst, and Bernhard Rumpe. On Demand Data Analysis and Filtering for Inaccurate Flight Trajectories. In *Proceedings of the SESAR Innovation Days.* EUROCONTROL, 2011.