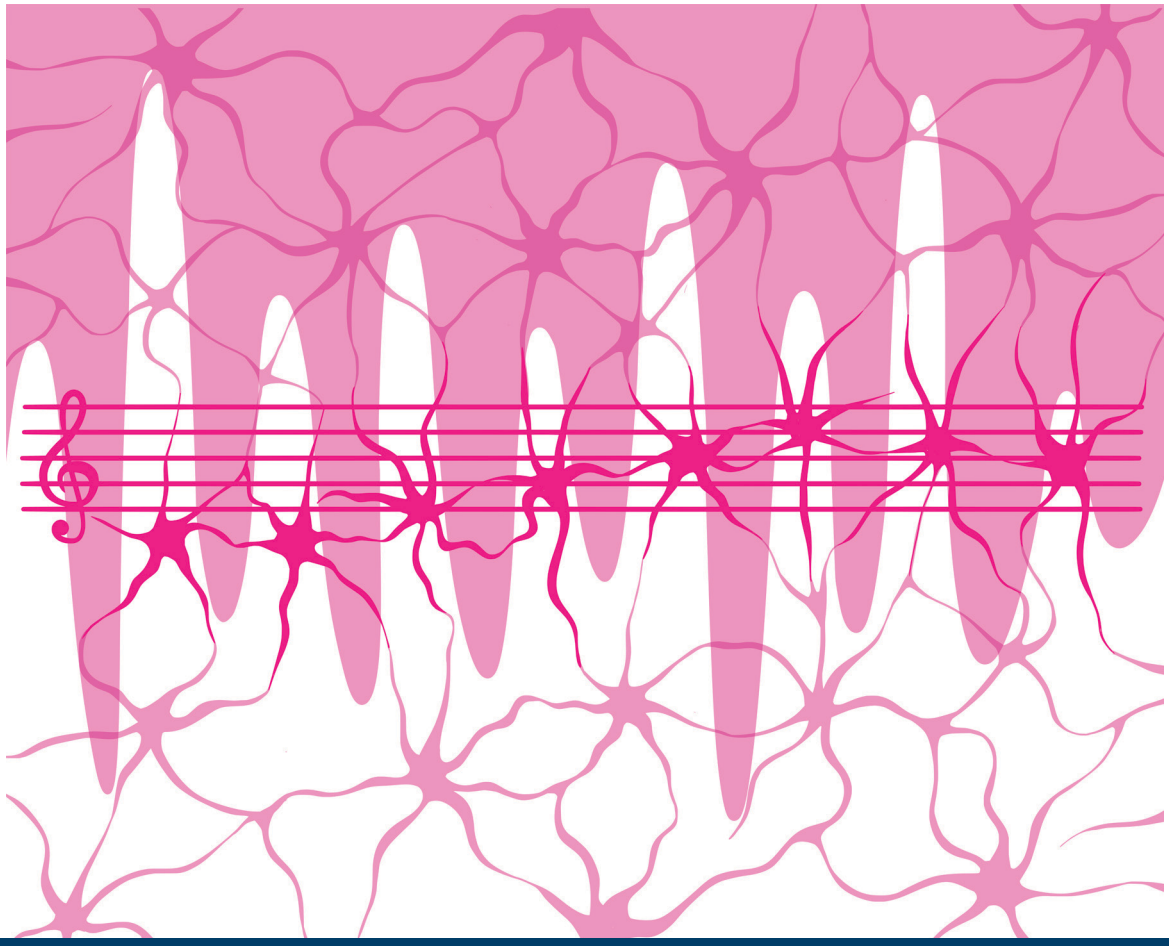

Sequence learning and computational efficiency in large-scale spiking neuronal networks

Melissa Lober

Information
Band / Volume 134
ISBN 978-3-69367-015-6

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Schriften des Forschungszentrums Jülich
Reihe Information / Information Band / Volume 134

Forschungszentrum Jülich GmbH
Institute for Advanced Simulation (IAS)
Computational and Systems Neuroscience (IAS-6)

Sequence learning and computational efficiency in large-scale spiking neuronal networks

Melissa Lober

Schriften des Forschungszentrums Jülich
Reihe Information / Information Band / Volume 134

ISSN 1866-1777 ISBN 978-3-69367-015-6

Bibliografische Information der Deutschen Nationalbibliothek.
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte Bibliografische Daten
sind im Internet über http://dnb.d-nb.de abrufbar.

Herausgeber und Vertrieb:	Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag 52425 Jülich Tel.: +49 2461 61-5368 Fax: +49 2461 61-6103 zb-publikation@fz-juelich.de www.fz-juelich.de/zb
Umschlaggestaltung:	Grafische Medien, Forschungszentrum Jülich GmbH
Druck:	Grafische Medien, Forschungszentrum Jülich GmbH
Copyright:	Forschungszentrum Jülich 2026

Schriften des Forschungszentrums Jülich
Reihe Information / Information, Band / Volume 134

D 82 (Diss. RWTH Aachen University, 2026)

ISSN 1866-1777
ISBN 978-3-69367-014-9 (Print)
ISBN 978-3-69367-015-6 (E-Book)

Vollständig frei verfügbar über das Publikationsportal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess.



This is an Open Access publication distributed under the terms of the Creative Commons Attribution License 4.0, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

SUMMARY

The brain learns and processes sequential information with remarkable efficiency, performing a wide range of cognitive functions from predicting upcoming events and detecting anomalies to replaying memories at variable speeds. Understanding how neuronal circuits achieve this, and how such processes can be studied computationally at biologically relevant scales, are two central challenges of modern computational neuroscience. This thesis addresses both through biological network models for sequence learning on the one hand, and through advances in simulation and benchmarking infrastructure on the other.

The first line of investigation addresses two fundamental questions about sequence learning in cortical networks. The first concerns temporal structure: natural sequences are not only defined by the order of their elements but also by the timing between them, and learned sequences can be recalled at different speeds depending on behavioral context. This thesis extends a biologically motivated network model to encode both the order and the temporal structure of sequences through the self-organized formation of sparse, context-specific neuronal groups. The model learns in an unsupervised manner via local synaptic plasticity mechanisms, and demonstrates that the speed of sequence replay can be flexibly modulated by oscillatory network activity. The second question concerns capacity: what are the limits of biologically plausible sequence learning, and how do they scale with sequence complexity? Through systematic simulation experiments, the thesis demonstrates that the primary bottleneck is not the size of the network but the learning dynamics itself, with learning time increasing supralinearly with sequence length.

The methodological contributions of this thesis concern the computational challenges that arise when neuronal networks are simulated at large scales. Large-scale cortical models require distributing computation across many processors, and as network size grows, synchronization overhead between processors becomes the dominant bottleneck. By exploiting the structural organization of the mammalian cortex, in particular the separation between short local and long inter-area synaptic conduction delays, this thesis develops a structure-aware simulation strategy that substantially reduces the synchronization overhead and thus increases simulation efficiency of large-scale multi-area networks. Ensuring that such efficiency gains are reliably detected and preserved as simulation software evolves requires systematic performance validation. To this end, a continuous benchmarking framework is developed that integrates automated performance assessment into the software development pipeline, guaranteeing reproducibility of benchmarking results across researchers and hardware systems.

Together, these contributions advance the understanding of how biological networks learn and replay structured sequences, while developing the computational tools needed to study such processes at biologically relevant scales.

ZUSAMMENFASSUNG

Das Gehirn lernt und verarbeitet sequentielle Informationen mit bemerkenswerter Effizienz und erfüllt dabei ein breites Spektrum kognitiver Funktionen, von der Vorhersage bevorstehender Ereignisse und der Erkennung von Anomalien bis hin zur Wiedergabe von Erinnerungen mit variabler Geschwindigkeit. Zu verstehen, wie neuronale Netzwerke dies leisten, und wie solche Prozesse auf biologisch relevanten Skalen durch Simulation untersucht werden können, sind zwei zentrale Herausforderungen der modernen Computergestützten Neurowissenschaft. Diese Arbeit adressiert beide Aspekte: einerseits durch biologisch motivierte Netzwerkmodelle für das Sequenzlernen, andererseits durch Fortschritte in der Simulations- und Benchmarking-Infrastruktur.

Im Bereich des Sequenzlernens werden zwei grundlegende Fragen untersucht. Die erste betrifft die zeitliche Struktur von Sequenzen: Natürliche Sequenzen sind nicht nur durch die Reihenfolge ihrer Elemente, sondern auch durch die zeitlichen Abstände zwischen ihnen charakterisiert. Zudem können gelernte Sequenzen je nach Verhaltenskontext mit unterschiedlicher Geschwindigkeit abgerufen werden. Diese Arbeit erweitert ein biologisch motiviertes Netzwerkmodell dahingehend, dass es sowohl die Reihenfolge als auch die zeitliche Struktur von Sequenzen durch die selbstorganisierte Bildung spärlicher, kontextspezifischer neuronaler Gruppen kodiert. Das Modell lernt auf unüberwachte Weise durch lokale synaptische Plastizitätsmechanismen, und es wird gezeigt, dass die Wiedergabegeschwindigkeit gelernter Sequenzen durch oszillatorische Netzwerkaktivität flexibel gesteuert werden kann. Die zweite Frage betrifft die Kapazität des Modells: Welche Grenzen hat biologisch plausibles Sequenzlernen, und wie skalieren diese mit der Komplexität der zu lernenden Sequenzen? Systematische Simulationsexperimente zeigen, dass nicht die Netzwerkgröße, sondern die Lerndynamik selbst den entscheidenden Engpass darstellt: Die Lernzeit wächst überlinear mit der Sequenzlänge.

Die methodischen Beiträge dieser Arbeit betreffen die rechnerischen Herausforderungen bei der Simulation neuronaler Netzwerke auf großen Skalen. Großskalige kortikale Modelle erfordern die Verteilung der Berechnung auf viele Prozessoren, wobei mit wachsender Netzwerkgröße der Synchronisationsaufwand zwischen den Prozessoren zum dominierenden Engpass wird. Durch die gezielte Ausnutzung der strukturellen Organisation des Säugetierkortex, insbesondere der Trennung zwischen kurzen lokalen und langen interarealen synaptischen Leitungsverzögerungen, wird eine strukturbewusste Simulationsstrategie entwickelt, die diesen Aufwand erheblich reduziert und die Simulationseffizienz großskaliger Multi-Areal-Netzwerke deutlich steigert. Damit solche Effizienzgewinne beim Weiterentwickeln der Simulationssoftware zuverlässig nachgewiesen und erhalten werden können, ist eine systematische Leistungsvalidierung unerlässlich. Zu diesem Zweck wird ein kontinuierliches Benchmarking-Framework entwickelt, das automatisierte Leistungsmessungen in den Softwareentwicklungsprozess integriert und die Reproduzierbarkeit von Benchmarking-Ergebnissen über verschiedene Forscher und Hardwaresysteme hinweg sicherstellt.

Insgesamt vertieft diese Arbeit das Verständnis davon, wie biologische Netzwerke strukturierte Sequenzen erlernen und abrufen, und liefern gleichzeitig die rechnerischen Werkzeuge, die für die Untersuchung solcher Prozesse auf biologisch relevanten Skalen benötigt werden.

PUBLICATIONS

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

Unsupervised online learning of complex sequences in spiking neuronal networks
Younes Bouhadjar*, Melissa Lober*, Emre Neftci, Markus Diesmann, and Tom Tetzlaff
Published in IEEE Press: Bouhadjar et al. (2025a)
Results of this publication enter Chapter 2

Learning sequence timing and control of replay speed in networks of spiking neurons
Melissa Lober, Younes Bouhadjar, Markus Diesmann, and Tom Tetzlaff
Published on arxiv: Lober et al. (2026a)
Currently under review
Results of this publication enter Chapter 3

Exploiting network topology in brain-scale simulations of spiking neural networks
Melissa Lober, Markus Diesmann, and Susanne Kunkel
Published in Neuromorphic Computing and Engineering: Lober et al. (2026)
Results of this publication enter Chapter 4

Continuous benchmarking: Keeping pace with an evolving ecosystem of models and technologies
Jan Vogelsang, Melissa Lober, Catherine Mia Schöfmann, José Villamar, Dennis Terhorst, Johanna Senk, Hans Ekkehard Plesser, Markus Diesmann, Susanne Kunkel, and Anno C. Kurth
Published on arxiv: Vogelsang et al. (2026)
Results of this publication enter Chapter 5

Author contributions are stated at the beginning of each chapter.

* Shared first authorship.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Dr. Markus Diesmann, who gave me the opportunity to pursue this doctoral research and who has been an inspiring presence throughout. His genuine enthusiasm for science and his ability to find meaning and excitement in every result, whether expected or surprising, has shaped the way I think about research. I am grateful for his encouragement, his reliability, and for teaching me what it means to do science with rigor and persistence.

I am equally grateful to Dr. Susanne Kunkel, who supervised one of the main projects of this thesis. Our meetings were something I genuinely looked forward to, and I have rarely experienced such a natural and productive working relationship. Her technical expertise, attentiveness, and ability to understand my thinking made the work better in every respect.

My sincere thanks also go to Dr. Tom Tetzlaff, who supervised the other main project of this thesis. His calm and thoughtful presence made every meeting a learning experience that extended well beyond the immediate scope of the work. He took the time to teach me not only about the science but about how to approach problems with care and intellectual honesty, and his patience and dedication left a lasting impression on me.

I would like to thank Prof. Dr. Björn Kampa for serving as my second supervisor and for bringing a valuable experimental neuroscience perspective to the PhD-plan meetings that helped steer the projects in the right direction.

I am also grateful to Prof. Dr. Sonja Grün, who together with Prof. Dr. Markus Diesmann leads the Institute for Advanced Simulation (IAS-6) at Forschungszentrum Jülich. It is their personal mission to cultivate a warm, collaborative, and intellectually stimulating atmosphere there, and that is not something to take for granted.

The administrative and technical staff of IAS-6 deserve special recognition. Their dedication keeps the institute running smoothly and allows everyone else to focus on their research. Their helpfulness and friendliness contributed greatly to the warm atmosphere of the institute.

More broadly, I am grateful to all members of IAS-6, past and present. Over the years I have had the privilege of meeting many like-minded people, some of whom have become close friends. The willingness of those who came before me to share their knowledge and take time out of their days to help me find my footing was invaluable, and I hope to pass some of that spirit on in turn. The discussions, scientific and otherwise, that filled the corridors, offices, and lunch tables of the institute have broadened my perspective in ways I could not have anticipated.

Finally, I owe a great deal to my close friends, to Jan, and to my family. Their unwavering belief in me, their encouragement during the difficult moments, and their companionship along the way have been an essential source of strength throughout this journey. Thank you.

CONTENTS

I Introduction

1 Introduction 3
1.1 Computational Neuroscience . . . 3
1.1.1 Neurons . . . 3
1.1.2 Synaptic Plasticity . . . 4
1.1.3 Neuronal Networks . . . 5
1.1.4 Neuronal Coding . . . 6
1.1.5 Bottom-up Meets Top-down Modelling . . . 6
1.2 Sequences in the Brain . . . 7
1.2.1 Universality of Sequences . . . 7
1.2.2 The Spiking Temporal Memory Model . . . 8
1.3 Large-scale Simulation of Spiking Networks . . . 9
1.3.1 Simulating at Scale . . . 9
1.3.2 Simulation of Structured Networks . . . 9
1.3.3 Conventional vs. Neuromorphic Computing . . . 10
1.3.4 Performance Benchmarking of Neuronal Simulators . . . 11
1.4 Scope and Structure of this Thesis . . . 12

II Sequence Processing in Cortical Networks

2 Learning Capacity of the Spiking Temporal Memory Model 17
2.1 Introduction . . . 18
2.2 Methods . . . 18
2.2.1 Network Model . . . 18
2.2.2 Training Data and Procedure . . . 20
2.2.3 Network Performance Estimation . . . 21
2.2.4 Parameter Optimization . . . 22
2.2.5 Simulation Details . . . 22
2.3 Results . . . 23
2.3.1 Sequence Learning Dynamics . . . 23
2.3.2 Sequence Learning Capacity . . . 23
2.4 Discussion . . . 24
3 Sequence Timing and Replay Speed 27
3.1 Introduction . . . 28
3.2 Methods . . . 29
3.2.1 The spiking Temporal Memory (sTM) Model . . . 29
3.2.2 Readout Neurons . . . 30
3.2.3 Hyperparameter Tuning . . . 31
3.2.4 Simulation Details . . . 31
3.3 Results . . . 32
3.3.1 Learning Sequence Timing . . . 32
3.3.2 Modulating Replay Speed . . . 33

3.3.3 Driving Replay with Constant or Oscillatory Background Inputs 33
3.3.4 Range of Accessible Replay Speeds . . . 37
3.3.5 Testable Predictions . . . 41
3.4 Discussion . . . 43

III Efficient Simulation of Large-Scale Networks

4 Exploiting Network Topology in Brain-Scale Network Simulations 49
4.1 Introduction . . . 50
4.2 Results . . . 52
4.2.1 Structure-aware Simulation Strategy . . . 53
4.2.2 Theoretical Analysis of Synchronization . . . 56
4.2.3 Benchmark Results . . . 62
4.3 Methods . . . 70
4.3.1 Reference Implementation . . . 70
4.3.2 Benchmark Model . . . 74
4.3.3 Benchmarking Systems . . . 76
4.4 Discussion . . . 77
5 Continuous Benchmarking 81
5.1 Introduction . . . 82
5.2 Principles of CI-beNNch . . . 82
5.2.1 Standardized and Centralized Hierarchical Configurations . . 84
5.2.2 Layered Abstraction . . . 84
5.2.3 CI-based Execution . . . 85
5.3 CI-beNNch in Action . . . 85
5.3.1 Running a Benchmark for the First Time . . . 85
5.3.2 Reproducing a Benchmark . . . 86
5.3.3 Rerunning a Benchmark with . . . 87
5.4 CI-beNNch in Practice . . . 87
5.4.1 Barrier-free Spike Delivery . . . 88
5.5 Discussion . . . 91
5.5.1 Practical Utility . . . 92
5.5.2 Limitations . . . 92
5.5.3 Beyond Continuous Benchmarking . . . 93

IV Discussion

6 Discussion and Outlook 97
6.1 Summary . . . 97
6.2 Synthesis . . . 99
6.2.1 The Role of Biological Constraints in Model Design . . . 99
6.2.2 Scale as a Unifying Challenge . . . 100
6.2.3 The Interplay between Model and Simulation Infrastructure . 101
6.2.4 Structure-aware Simulation and Neuromorphic Computing . 102
6.3 Limitations . . . 102
6.4 Outlook . . . 103
6.5 Concluding Remarks . . . 104

v Appendix
7 Supplementary Material: Sequence Timing and Replay Speed 107
7.1 Detailed Description of the Network Model 107
7.2 Model and Simulation Parameters 117
8 Supplementary Material: Exploiting Network Topology 122

Bibliography 123

LIST OF FIGURES

Figure 2.1 Sketch of the network architecture . . . 19
Figure 2.2 Sequence learning progress . . . 23
Figure 2.3 Prediction performance and learning speed . . . 25
Figure 3.1 Sketch of the sTM network architecture . . . 30
Figure 3.2 Encoding long intervals and rhythm through repetitive elements . . . 33
Figure 3.3 Effect of background activity on membrane potential . . . 34
Figure 3.4 Modulation of replay speed by constant background input . . . 35
Figure 3.5 Modulation of replay speed by oscillatory background input 36
Figure 3.6 Range of achievable replay duration with oscillatory background input . . . 38
Figure 3.7 Phase invariance of replay speed . . . 39
Figure 3.8 Stabilization of replay in the presence of noise by oscillatory input . . . 40
Figure 4.1 Strong-scaling of the multi-area model of macaque visual cortex in the ground state . . . 52
Figure 4.2 Conventional and structure-aware simulation scheme illustrated for an example multi-area model . . . 54
Figure 4.3 Conventional and structure-aware simulation flow . . . 55
Figure 4.4 MPI collective performance for increasing message sizes . . . 56
Figure 4.5 Graphical intuition of the predicted reduction in overall synchronization time . . . 57
Figure 4.6 Theoretical analysis of the advantage of structure-aware strategy over the conventional . . . 58
Figure 4.7 Performance comparison of conventional and structure-aware simulation strategy . . . 63
Figure 4.8 Effect of inter-area variability and delay on performance . . . 65
Figure 4.9 Performance comparison for a real-world network model on JURECA-DC and SuperMUC-NG . . . 68
Figure 4.10 Per-process cycle time analysis for simulations of the MAM 68
Figure 4.11 Fundamental data structures of the reference code NEST . . . 73
Figure 4.12 Comparison of strong scaling performance of MAM and MAM-benchmark . . . 75
Figure 5.1 Overview of the continuous benchmarking process . . . 83
Figure 5.2 Weak-scaling performance of the HPC-Benchmark model . 89
Figure 5.3 Strong-scaling performance of the microcircuit model . . . 90
Figure 5.4 Strong-scaling performance of the multi-area model . . . 91
Figure 8.1 Temporal evolution of the cycle time . . . 122

LIST OF TABLES

Table 5.1 Execution of a benchmark using beNNch vs. CI-beNNch . . 86
Table 7.1 Summary of the network model 107
Table 7.2 Description of the populations 108
Table 7.3 Description of the connectivity 108
Table 7.4 Description of the neuron model 109
Table 7.5 Description of the synapse model 110
Table 7.6 Description of the plasticity model 111
Table 7.7 Description of the input 113
Table 7.8 Description of the performance measurement 116
Table 7.9 Description of the initial conditions of the network 116
Table 7.10 Model and simulation parameters 117

Part I

INTRODUCTION

1

INTRODUCTION

The brain is, fundamentally, shaped by and for the processing of temporally structured information. Nearly every cognitive ability we possess, from perceiving speech and planning movements to recalling memories and interpreting music, depends on the capacity to represent, learn, and reproduce structured sequences of events (Lashley, 1951; Clegg et al., 1998; Hawkins & Blakeslee, 2007; Dehaene et al., 2015). Understanding how the brain achieves this, and how computational tools can be used to study it at scale, is a central challenge of modern computational neuroscience. This thesis addresses four interconnected problems at the intersection of sequence learning, temporal coding, network capacity, and large-scale simulation: how neuronal networks can learn the timing of events in sequences; how the capacity of such networks scales with sequence complexity; how brain-scale simulations can be made computationally tractable; and how the software tools enabling these simulations can be continuously validated. The introduction provides the conceptual background necessary to situate these questions and gives the reader an overview of the topics involved.

1.1 COMPUTATIONAL NEUROSCIENCE

1.1.1 *Neurons*

The mammalian central nervous system is a highly complex biological system whose full description encompasses many cell types, signaling mechanisms, and organizational principles. The following provides a simplified account focusing on the aspects most relevant to the models developed in this thesis. The nervous system is composed of a vast number of discrete, highly interconnected functional units called neurons (Ramon y Cajal, 1899). These electrically excitable cells have a characteristic morphology: a cell body or soma, an axon that carries output signals away from the cell, and a branching tree of dendrites to collect input from other neurons. They communicate by generating brief, stereotyped voltage deflections known as action potentials or spikes (Hodgkin & Huxley, 1952), which propagate along the axon to synaptic contacts with other neurons. At each synapse, the arriving spike triggers the release of neurotransmitters into the synaptic cleft, which either depolarize the recipient neuron towards the firing threshold, in the case of excitatory neurotransmitters such as glutamate, or hyperpolarize it away from threshold, in the case of inhibitory neurotransmitters such as GABA (Kandel et al., 1991). The strength of this interaction, determined by the amount of neurotransmitter released and the density of postsynaptic receptors, is referred to as the synaptic weight. Each neuron releases the same cocktail of neurotransmitters at all of its synapses, a constraint known as Dale's law (Eccles et al., 1954). In the cortex, approximately 80 % of neurons are excitatory, with the remainder forming a diverse population of

inhibitory interneurons (Abeles, 1991). Since action potentials are highly stereotyped in shape and amplitude, the information they carry must lie in their timing and pattern rather than in their waveform. Neurons are therefore frequently abstracted as threshold units in which the membrane acts as a resistor-capacitor (RC) circuit: input currents charge the membrane capacitance, causing the membrane potential to rise until it crosses a critical threshold, at which point a spike is emitted and the potential is reset to its resting value (Lapicque, 1907; Dayan & Abbott, 2001). This framework is known as the integrate-and-fire approximation.

In computational neuroscience, there is a fundamental trade-off between the biophysical detail of the neuron model and the size of the network that can feasibly be simulated. For this reason, neurons are often modelled as point neurons, which collapse the entire dendritic tree into a single compartment and disregard its spatial structure, retaining only the timing of synaptic inputs through explicit transmission delays. While this simplification sacrifices some biological realism, it enables the simulation of networks containing millions of neurons and has proven sufficient to reproduce many key features of cortical dynamics (Brunel, 2000).

1.1.2 *Synaptic Plasticity*

The brain's ability to learn and adapt arises from its capacity to modify the strength and structure of synaptic connections between neurons, a property known as synaptic plasticity (Buonomano & Merzenich, 1998). Computational neuroscience typically distinguishes between two broad forms. Functional plasticity refers to changes in the efficacy of existing synapses, realized through the regulation of neurotransmitter release or postsynaptic receptor density (Citri & Malenka, 2007). Structural plasticity, on the other hand, involves the physical creation and elimination of synaptic contacts, a process that continues throughout adulthood and is believed to underlie long-term reorganization of neuronal circuits (Holtmaat & Svoboda, 2009).

The conceptual foundation for synaptic plasticity as a learning mechanism was laid by Hebb (1949), who proposed that the connection between two neurons should be strengthened when they are repeatedly active in close temporal proximity. This principle, often summarized as neurons that fire together wire together, is suggested to provide a basis for associative learning and the formation of cell assemblies that encode distributed representations. A biologically observed refinement of this idea is spike-timing-dependent plasticity (STDP; Markram et al. (1997); Bi & Poo (1998)), in which the direction and magnitude of synaptic change depend on the precise relative timing of pre- and postsynaptic spikes. In its classical formulation, a synapse is potentiated if the presynaptic neuron fires before the postsynaptic neuron, and depressed if the order is reversed. However, the relationship between spike timing and synaptic change exhibits considerable heterogeneity across brain regions, cell types, and synaptic compartments: the sign and magnitude of plasticity can depend on factors beyond somatic spike timing, including dendritic processing at the postsynaptic site (Kampa et al., 2007). The classical formulation described here nevertheless provides a useful framework for understanding how sequential structure can be learned, as it naturally reinforces connections along the causal chain

of a sequence. A structural form of STDP, in which the timing of spikes governs the creation and removal of synapses rather than the modulation of existing ones, has also been identified experimentally (Liao et al., 1995) and is employed in the models used in this thesis. STDP has the potential to destabilize the network by creating positive feedback loops, causing runaway excitation or the silencing of neurons. Biological circuits counteract this through homeostatic plasticity, a set of slower regulatory mechanisms that adjust synaptic strengths or neuronal excitability to maintain firing rates within a functional range (Abbott & Nelson, 2000; Tetzlaff et al., 2011).

1.1.3 *Neuronal Networks*

The synaptic weights modified by plasticity mechanisms define the connectivity structure of the network as a whole, and to a large extent, it is this structure that determines the collective computational properties of neuronal circuits. To appreciate the scale of this connectivity, consider that a square millimeter of cortical sheet contains on the order of 10,000 to 100,000 neurons and more than 100 million synapses (Shapson-Coe et al., 2021), yielding thousands of synaptic inputs per neuron. In humans, the neocortex as a whole unfolds to an approximately isotropic sheet covering a quarter of a square meter at a thickness of only a few millimeters (Toro et al., 2008; Fischl & Dale, 2000), containing a total of around 16 billion neurons with up to 10,000 incoming connections each (Abeles, 1991).

Cortical networks are neither sparsely nor randomly connected in a trivial sense, but rather highly structured across multiple spatial scales. At the local scale, the vertical dimension of the cortex exhibits a layered organization constituting the cortical microcircuit, in which excitatory and inhibitory populations across layers interact according to specific connectivity motifs (Binzegger et al., 2004). In the horizontal dimensions, connection probability decays with distance on a scale of a few hundred micrometers (Stepanyants et al., 2007; Boucsein et al., 2011; Kurth et al., 2025). At larger scales, the cortex decomposes into roughly a hundred areas connected by long-range cortico-cortical projections that organize functional specificity (Felleman & Van Essen, 1991). These long-range connections span substantially larger anatomical distances than local intra-area connections and therefore exhibit larger conduction delays. Experimental measurements report a mean axonal conduction velocity of approximately 3.5 m/s for projections between visual areas in macaque monkey (Girard et al., 2001), and given that inter-area distances can reach several tens of millimeters (Schmidt et al., 2018a), the resulting transmission delays are on the order of several milliseconds, substantially larger than the shortest local delays.

In practice, the exact connectivity graph of a biological cortical neuronal network is not known and, even where data exist, would be computationally prohibitive to instantiate explicitly in models containing millions of neurons. Computational models therefore typically instantiate networks probabilistically: synaptic connections are drawn according to rules such as a fixed connection probability between neuron pairs, a fixed in- or out-degree per neuron, or distance-dependent connection probabilities. This approach allows large-scale networks to be generated efficiently while

respecting the statistical regularities observed in anatomy. The resulting random connectivity then serves as a substrate on which plasticity mechanisms can act, gradually carving out functional subnetworks from the initial random scaffold.

1.1.4 *Neuronal Coding*

A particularly important consideration for modelling is the neuronal coding scheme. A long-standing debate in neuroscience concerns whether the brain primarily encodes information through firing rates, that is, the number of spikes a neuron produces per unit time (Adrian, 1926), or through the precise temporal relationships between spikes (Segundo et al., 1963). The observation that single neuron responses vary from trial to trial even under identical stimulation (Shadlen & Newsome, 1998; Festa et al., 2021) initially motivated a statistical, rate-based view of neuronal coding. Rate-based models are grounded in this view and offer significant computational advantages. However, they cannot capture phenomena that depend on the precise timing of individual action potentials, such as spike-timing-dependent plasticity (STDP, Bi & Poo (1998)), the correlation structure of population activity (Riehle et al., 1997), or temporal coding schemes in which the pattern of spike times itself carries information. Since these aspects are central to the questions addressed in this thesis, spiking network models provide the appropriate level of description.

A further property of cortical activity that is central to the models developed in this thesis is sparsity. In the cortex, neuronal activity is sparse in multiple senses: at any given moment, only a small fraction of neurons are active simultaneously (Barth & Poulet, 2012), and those that do fire do so at remarkably low rates. While cortical firing rates can in principle reach up to 30 spikes per second, the bulk of the distribution lies at 1 spike per second or below (Swadlow, 1988; de Kock & Sakmann, 2009), reflecting a non-continuous, irregular firing pattern rather than sustained activity. This sparse activity is most likely not merely a consequence of energy efficiency constraints, though metabolic considerations certainly play a role (Attwell & Laughlin, 2001); it also has important computational consequences. When only a small subset of neurons is active at any one time, different activity patterns are highly dissimilar from one another, making them easy to distinguish and reducing interference between stored representations. This property is known as pattern separation and it directly increases the storage capacity of a network, since more distinct representations can coexist without mutual interference. It also enables multiplexing, since the same neuron can participate in multiple different representations without ambiguity, as long as it is never active in more than one representation at the same time. This is particularly advantageous for sequence learning, where the same neuron may participate in multiple sequences and disambiguation between similar contexts is essential.

1.1.5 *Bottom-up Meets Top-down Modelling*

Computational models offer a complementary approach to purely experimental investigation of the brain, but their success depends critically on the choice of abstraction. Depending on the scientific question under investigation, computational

neuroscience models can broadly be characterized as following either a bottom-up or a top-down approach, and the most powerful frameworks often combine elements of both.

Bottom-up models start from the biophysical properties of neurons and the structural organization of their connections, and ask what dynamics emerge from these constituents. Rather than asking directly how complex cognitive functions arise, bottom-up modelling focuses on the dynamics of neuronal circuits and how they are shaped by anatomy and physiology. The accumulation of anatomical and electrophysiological knowledge over more than a century of neuroscience provides an expanding basis for constraining such models: connectivity data from predictive connectomics and electron microscopy can inform the wiring of large-scale network models (Binzegger et al., 2004; Schmidt et al., 2018a; Shapson-Coe et al., 2024), electrophysiological recordings can constrain neuron model parameters (Billeh et al., 2020), and hypotheses about neuromodulation can be systematically explored in simulation. Experimental data can then be used not only to calibrate these models but also to validate them, closing the loop between theory and experiment.

Top-down models, by contrast, start from the functional level and ask what neuronal mechanisms could give rise to observed behavior or cognitive capabilities. Rather than being constrained primarily by anatomical data, these models are evaluated by their ability to reproduce system-level phenomena such as pattern completion, sequence prediction, memory recall, or decision-making (Hopfield, 1982). Critics of the purely bottom-up approach have argued that without a clear notion of what the system is supposed to compute, biologically realistic models risk emphasizing unimportant details while missing the functional principles that evolution has optimized (Eliasmith & Trujillo, 2014). This critique motivates the incorporation of top-down functional constraints alongside bottom-up biological ones and vice versa.

1.2 SEQUENCES IN THE BRAIN

1.2.1 *Universality of Sequences*

The importance of sequential processing pervades neuroscience and cognitive science. When we speak or listen, we process a continuous stream of phonemes whose interpretation depends critically on their order and timing. When we play an instrument, we execute motor sequences with millisecond precision. When we recall an episode, we mentally re-traverse a temporal trajectory of events. Learning these sequences is not simply a matter of memorizing which element follows which; it also requires encoding the temporal intervals between elements (Levitin & Cook, 1996; Boltz, 2010), and being able to replay learned sequences flexibly — faster or slower, in response to partial cues, and across varying behavioral contexts. Behavioral and electrophysiological observations reinforce this: a melody practiced slowly remains recognizable when played rapidly, and neuronal replay of hippocampal sequences during sleep occurs at highly compressed timescales compared to the original experience (Ji & Wilson, 2007; Euston et al., 2007; Liu et al., 2019; Buch et al., 2021). These observations suggest that the brain encodes both the ordinal

structure of sequences and their temporal structure, and that dedicated mechanisms allow for flexible modulation of replay speed. A key challenge for computational models is to simultaneously account for three tightly coupled operations: predicting upcoming elements based on context, detecting unexpected deviations from learned sequences, and autonomously replaying sequences from a partial cue.

1.2.2 *The Spiking Temporal Memory Model*

The striking uniformity of the neocortex across areas and species has long suggested that all cortical regions may implement a common algorithm (Mountcastle, 1978). Inspired by this observation, Hawkins & Blakeslee (2007) proposed that this canonical computation is the unsupervised learning and prediction of sequences of sensory patterns, and formalized this idea in the Hierarchical Temporal Memory (HTM) model (Hawkins et al., 2011; Hawkins & Ahmad, 2016). The HTM framework is built around several key concepts: neurons are organized into minicolumns that represent distinct tokens; dendritic segments act as context detectors that place neurons into predictive states before an expected input arrives; sparse activity patterns provide robust and separable representations of sequence elements; and a structural plasticity rule drives the unsupervised formation of sequence-specific connectivity. Together, these mechanisms enable a single unified architecture to perform context-dependent prediction, anomaly detection, and sequence replay, entirely without supervision. In its original formulation, the HTM model operates at a relatively abstract level, where neurons are described as discrete, binary units that update their states in discrete time steps and the plasticity rule is defined algorithmically rather than derived from biophysical principles.

The spiking Temporal Memory (sTM) model (Bouhadjar et al., 2022) provides a biologically plausible instantiation of this algorithm using continuous-time spiking neurons, and forms the basis of the first two papers of this thesis. The network is organized into minicolumns, each containing a population of excitatory neurons and a shared inhibitory interneuron that implements a winner-take-all mechanism (Maass, 2000), enforcing sparse activity across the network. Lateral excitatory connections between minicolumns exclusively target distal dendritic branches. The key biophysical mechanism underlying predictive states in the sTM is nonlinear dendritic integration (Antic et al., 2010). When lateral input to the dendrite exceeds a local threshold, a dendritic action potential is generated, producing a sustained subthreshold depolarization of the soma. This places the neuron in a predictive state: when feedforward input subsequently arrives, predictively active neurons fire earlier than others, triggering inhibition that suppresses the remainder of the minicolumn. The result is a sparse, context-specific neuronal assembly whose identity depends on the full preceding sequence history. Synaptic connections between excitatory neurons evolve through a structural form of STDP (Liao et al., 1995; Lüscher et al., 2000; Deger et al., 2012), through which repeated sequence presentations progressively form and consolidate the lateral connections underlying these context-specific representations. Once learned, a sequence can be autonomously replayed from a partial cue, reinstating the full sequence through a chain of predictions propagating through the learned subnetwork (Bouhadjar et al., 2022, 2023).

The sTM model sits precisely at the intersection of the bottom-up and the top-down approach. It is top-down in the sense that it is designed around a specific functional objective, namely the learning and replay of sequential patterns, and is evaluated by its ability to perform context-dependent prediction, anomaly detection, and autonomous replay. At the same time, it is bottom-up in that its mechanisms are grounded in biologically observed properties of cortical neurons: nonlinear dendritic integration, sparse population coding, lateral inhibition, and structural STDP.

1.3 LARGE-SCALE SIMULATION OF SPIKING NETWORKS

1.3.1 *Simulating at Scale*

Simulating the dynamics of cortical circuits at biologically meaningful scales requires not only well-designed spiking network models, but also the computational tools to run them efficiently. The multi-level organization of the cortex, spanning the local microcircuit, the cortical area, and the inter-area network, imposes fundamental constraints on meaningful model downscaling. Theoretical work has shown that reducing network size inevitably perturbs dynamics, particularly at the level of pairwise correlations that directly interact with plasticity mechanisms such as STDP (Bi & Poo, 1998), even when first-order statistics such as firing rates are preserved (van Albada et al., 2015). These considerations motivate the drive toward ever larger simulations that faithfully represent the structural organization of the brain. Simulating such systems requires distributing the computation across many processors via high-performance computing (HPC) infrastructure. Frameworks such as NEST (Gewaltig & Diesmann, 2007) implement distributed spiking network simulation using the Message Passing Interface (MPI) standard (Message Passing Interface Forum, 2021), in which multiple independent processes advance the simulation in discrete time steps and periodically exchange spike information to preserve causality. The minimum synaptic transmission delay determines how far the simulation can progress independently on each process before a global communication step is required (Morrison et al., 2005). Over the past decade, successive optimizations have systematically reduced the cost of communication and spike delivery (Jordan et al., 2018; Pronold et al., 2022a), shifting the dominant bottleneck toward synchronization overhead, that is, the time processors spend waiting for one another at the global communication barrier (Fernandez-Musoles et al., 2019).

1.3.2 *Simulation of Structured Networks*

The challenge of efficiently mapping network structure onto computing architectures has a long history. Early work on parallel discrete-event simulation established graph partitioning as a central tool for distributing interacting entities across processors while minimizing communication overhead (Kernighan & Lin, 1970; Fujimoto, 1990), and these approaches were subsequently adapted for neuronal simulations (Fernandez-Musoles et al., 2019). More recent frameworks extend this idea: COR-TEX employs indegree subgraph decomposition of instantiated network graphs to

enable highly parallel execution (Lyu et al., 2024), and STACS uses graph-based partitioning to optimize spike communication for spatially structured networks (Wang et al., 2024). Approaches directly exploiting network structure have also been explored, including tiling-based methods that decompose neuronal tissue into spatial blocks to exploit the locality of cortical connectivity (Kozloski & Wagner, 2011; Yamaura et al., 2020), and the SPLIT simulator, which demonstrated that exploiting population structure and synaptic delays yields efficient hardware mappings (Hammarlund & Ekeberg, 1998). The modular event-driven framework NEOSIM similarly highlighted the role of axonal delays in organizing parallel execution (Goddard et al., 2001).

The systematic separation of temporal scales between local and long-range interactions is a structural feature of large-scale cortical models that has, until recently, not been exploited in simulation algorithms. It is only now that network models have become large enough to expose the non-local hierarchical architecture of the brain, and that compute nodes have become powerful enough to shift the bottleneck decisively toward communication, that revisiting the mapping of network structure onto computing architecture becomes both necessary and promising.

1.3.3 *Conventional vs. Neuromorphic Computing*

Conventional computing architectures follow the von Neumann design, in which a processing unit and memory are physically separated and connected by a shared data bus (von Neumann, 1993). This separation creates a fundamental throughput limitation known as the von Neumann bottleneck. The processor can only operate as fast as data can be transferred between memory and the processing unit, and this bandwidth constraint becomes increasingly severe as the computational demands of large-scale neuronal simulations grow. Furthermore, conventional processors, specifically central processing units (CPUs), execute instructions sequentially, whereas neuronal circuits operate through massively parallel, event-driven communication between billions of neurons. Simulating this parallelism on sequential hardware requires explicit management of spike communication across processors, giving rise to the synchronization overhead discussed in the previous section. Neuromorphic computing addresses these limitations by designing hardware that more closely mirrors the architecture of the brain itself (Mead, 1990). Rather than separating memory and computation, neuromorphic systems co-locate processing and storage, in its final form even at the level of individual synapses, eliminating the von Neumann bottleneck for spike-based computation. Communication is event-driven and asynchronous, meaning that processors only exchange information when a spike is emitted, rather than at every time step. Systems such as BrainScaleS (Pehle et al., 2022), SpiNNaker (Rhodes et al., 2019), and Intel's Loihi (Davies et al., 2018) exploit these principles to offer highly parallel, low-power platforms for spiking network simulation, where efficiently mapping network structure onto hardware resources is crucial for performance. Nonetheless, conventional HPC platforms continue to serve as an essential reference, offering greater flexibility at potentially lower cost. Kurth et al. (2022) demonstrates that for simulation of a biologically realistic cortical microcircuit model, conventional HPC systems keep pace with

specialized neuromorphic systems such as SpiNNaker in terms of both real-time performance and energy efficiency (Rhodes et al., 2019), highlighting the continued importance of advancing simulation methods on conventional hardware.

1.3.4 *Performance Benchmarking of Neuronal Simulators*

Advancing simulation efficiency is only meaningful if performance can be measured reliably and reproducibly. As simulation software grows in complexity and HPC hardware evolves rapidly, ensuring that a simulator continues to deliver adequate performance across changing configurations requires systematic benchmarking, meaning the evaluation of metrics such as time-to-solution, memory footprint, and energy consumption across a range of hardware resources, software builds, and model parameters (Albers et al., 2022). This need is recognized within the broader field of Research Software Engineering (Baxter et al., 2012; Speck & Wyatt, 2024), which addresses the specific challenges of writing, extending, and maintaining scientific software. A useful distinction is drawn between verification, concerned with the correctness of the software, and validation, concerned with meeting the needs of users (Oberkampf & Roy, 2010). Correctness is routinely enforced through automated testing in continuous integration (CI) pipelines (Shahin et al., 2017), a software engineering practice in which a suite of tests is automatically executed whenever a change is made to the codebase, ensuring that newly introduced errors are detected immediately rather than accumulating unnoticed. However, performance validation has historically received far less systematic attention, despite the fact that certain scientific questions are only accessible at sufficient scale (Anderson, 1972; van Albada et al., 2015). The complexity of HPC benchmarking spans multiple dimensions — hardware configuration, software configuration, model parameters, and researcher communication — all of which must be controlled to produce reproducible results (Albers et al., 2022). A functionally correct change to a simulation codebase may nonetheless degrade performance in ways that are not immediately obvious, and detecting such regressions requires benchmarks to be run regularly and under controlled conditions.

1.4 SCOPE AND STRUCTURE OF THIS THESIS

This thesis is organized around four research questions that together span the problem of how the brain learns and replays sequences, and how spiking neuronal networks can be studied computationally at scale.

What are the limits of biologically plausible sequence learning, and how do they scale with the complexity of the sequences to be learned? A key dimension of sequence complexity is recurrence: natural sequences frequently contain elements that repeat in distinct temporal contexts, and a capable model must assign each occurrence a unique internal representation conditioned on its history (Hawkins & Ahmad, 2016; Ahmad & Hawkins, 2016). Previous capacity estimates were based on combinatorial arguments that do not account for whether the plasticity dynamics can actually realize all theoretically possible connectivity patterns. Chapter 2 addresses this question by systematically varying sequence length and complexity and studying the conditions under which reliable learning is achieved.

How can a biologically plausible spiking network learn not only the order of sequence elements, but also their temporal structure, and how can the speed of sequence replay be flexibly modulated? Natural sequences are not defined by order alone; their rhythm and timing are equally essential (Levitin & Cook, 1996; Boltz, 2010). The original sTM model provides no explicit mechanism for encoding element durations or inter-element intervals, and replay was demonstrated only in a narrow parameter regime close to the upper speed limit imposed by neuronal and synaptic time constants (Bouhadjar et al., 2022). Chapter 3 addresses this question by extending the sTM framework to incorporate temporal structure and by investigating mechanisms for flexible replay speed modulation.

How can the structural organization of the brain be leveraged to make large-scale spiking network simulations more efficient? As network size and processor count grow, the synchronization overhead incurred when processors wait for one another becomes the dominant bottleneck in distributed simulation (Fernandez-Musoles et al., 2019), a problem progressively exposed by successive optimizations of other simulation phases (Jordan et al., 2018; Pronold et al., 2022a). Chapter 4 addresses this question by investigating how the structural organization of large-scale cortical models (Schmidt et al., 2018b,a) can be leveraged to improve the efficiency of distributed simulation in NEST.

How can the performance of scientific simulation software be systematically validated as models and hardware continue to evolve? A functionally correct change to a simulation codebase may nonetheless degrade performance in ways that are not immediately obvious, yet systematic performance validation has received far less attention than correctness testing (Albers et al., 2022; Baxter et al., 2012; Speck & Wyatt, 2024). Chapter 5 addresses this question by developing a continuous benchmarking framework that integrates performance assessment into

the continuous integration pipeline and provides the methodological infrastructure used for validation in Chapter 4.

Part II

SEQUENCE PROCESSING IN CORTICAL NETWORKS

2

LEARNING CAPACITY OF THE SPIKING TEMPORAL MEMORY MODEL

The chapter is based on the following publication:

Unsupervised online learning of complex sequences in spiking neuronal networks. Y. Bouhadjar*, M. Lober*, E. Neftci, M. Diesmann, and T. Tetzlaff (2026) IEEE Press, DOI: 10.1109/ICONS69015.2025.00038.

* Shared first authorship.

Author contributions:
Under the supervision of Dr. Tom Tetzlaff (TT), Prof. Dr. Markus Diesmann (MD), and Prof. Dr. Emre Neftci (EN), the author and Younes Bouhadjar (YB) jointly conceptualized the work. The author and YB jointly developed the computational model and the code for running the simulations. YB conducted the numerical experiments. The results were discussed and refined in ongoing exchange with YB and TT.

2.1 INTRODUCTION

For sequences drawn from naturalistic domains — language, music, or episodic experience — a critical challenge is the presence of repeated elements: the same word, note, or landmark can appear in many different temporal contexts, and a capable network must assign each occurrence a distinct internal state that reflects its history (Dehaene et al., 2015; Hawkins & Blakeslee, 2007). As sequence length grows, such repetitions become increasingly frequent, placing demands on the plasticity mechanisms that form and maintain the sequence-specific subnetworks (Bouhadjar et al., 2022). The capacity of the HTM Temporal Memory has previously been characterized theoretically in terms of the number of distinct transitions a network can support, as a function of sparsity and dendritic segment count (Hawkins & Ahmad, 2016; Ahmad & Hawkins, 2016). However, these estimates are based on combinatorial arguments and do not account for whether the plasticity dynamics can actually realize all theoretically possible connectivity patterns. This chapter addresses this gap empirically, systematically increasing sequence length and studying how reliably and efficiently the sTM network (Bouhadjar et al., 2022) achieves learning. To enable successful learning of long, complex sequences, the plasticity rules of the original model are revised, building on earlier extensions of the framework (Bouhadjar et al., 2023, 2025b). The analysis reveals clear trade-offs between plasticity parameters, network architecture, and learning speed, and provides quantitative estimates of the sequences the spiking Temporal Memory architecture can support. These findings complement the mechanistic extensions of Chapter 3 by establishing the operational boundaries of the sTM framework, and inform the design choices for future extensions of the model.

2.2 METHODS

2.2.1 *Network Model*

In the following, we provide an overview of the network model. A comprehensive mathematical description of the individual model components, their dynamics, and parameters can be found at `https://github.com/YounesBouhadjar/SpikingTemporalMemory`.

The network is organized into $M = 26$ subpopulations (minicolumns), each consisting of $n_{\mathrm{E}} = 240$ excitatory neurons ($\mathcal{E}_k$, $k = 1, \ldots, M$) and a single inhibitory neuron ($\mathcal{I}_k$; Figure 2.1). Within each subpopulation, all excitatory neurons share the same stimulus preference and reliably spike in response to the presentation of the corresponding (suprathreshold) stimulus ($\mathcal{X}_k$). Excitatory neurons across the network are sparsely and randomly connected, forming recurrent excitatory circuits. Each inhibitory neuron receives inputs from its associated excitatory neurons and projects back to them, thereby implementing a local winner-take-all (WTA) competition that promotes sparse activation of subpopulations (see below).

All neurons are modeled as point neurons with leaky integrate-and-fire (LIF) dynamics. Connections between excitatory neurons target distal dendritic branches, where synaptic input currents are integrated in a nonlinear manner and generate

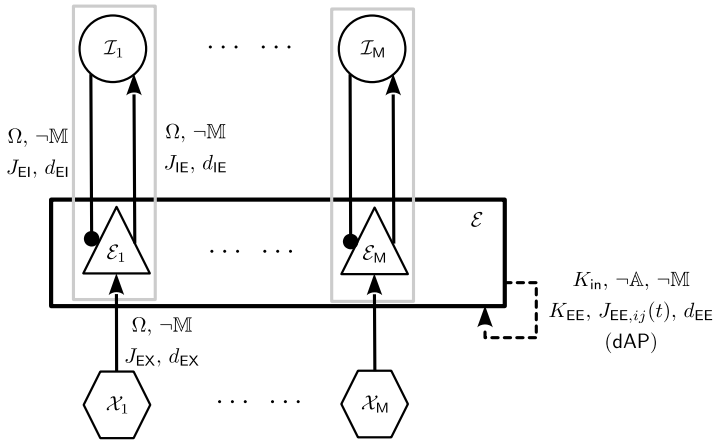

Figure 2.1: **Sketch of the network architecture.** *Populations:* $\mathcal{E}_k$: excitatory populations. $\mathcal{I}_k$: inhibitory populations. $\mathcal{X}_k$: external inputs ($k = 1, \ldots, M$). *Connectivity rules:* Triangular arrow heads: excitatory connections. Circular arrow heads: inhibitory connections. Solid lines: static connections. Dashed lines: plastic connections. K_{in}: random connectivity with fixed indegree. Ω: all-to-all connectivity. $\neg\mathbb{M}$: multiple connections between two neurons not permitted. $\neg\mathbb{A}$: self-connections not permitted. Recurrent connections between excitatory neurons (dashed) target distal dendritic branches and may trigger dendritic action potentials (dAPs) signaling predictions. *Connectivity parameters:* synaptic weights J_{xy}, spike transmission delays d_{xy} ($x, y \in \{E, I, X\}$). For details on the graphical notation, see Senk et al. (2022). A detailed mathematical description of the model is provided at `https://github.com/YounesBouhadjar/SpikingTemporalMemory`.

dendritic action potentials (dAPs) if the total synaptic input current on the dendritic branch exceeds a critical value. These dAPs lead to a subthreshold plateau depolarization of the soma lasting for several tens of milliseconds, and thereby bring neurons into a more excitable state. In the TM model, these plateau depolarizations serve as "predictions". Upon arrival of the corresponding stimulus, predictive neurons elicit a response spike with a shorter latency, as compared to non-predictive neurons. If the number of predictive neurons within a population exceeds the predefined pattern size ρ, the WTA mechanism is triggered, resulting in the selective activation of only the predicted neurons at the presentation of the respective stimulus. The resulting sparse subset of active neurons represents a specific sequence element in its current temporal context. Note that the "distal dendritic branches" are modeled as distinct nonlinear input channels in a point neuron model; we do not employ classical multicompartments models here.

Synapses between excitatory neurons are plastic, undergoing structural spike-timing-dependent changes alongside a continuous synaptic leak. This form of structural plasticity modulates the maturity of a synapse in a Hebbian manner. The synaptic weight is zero, unless the maturity exceeds a critical value. Through repeated presentation of a sequence, the network effectively sculpts a sparse sequence-specific pathway across the excitatory subnetwork. In this study, we introduce several modifications to the plasticity mechanisms of the original model. First, we remove the homeostatic mechanism that controls the extent of synaptic potentiation based on the dendritic activity of the postsynaptic neuron. This homeostatic control was originally introduced to implement a competition between sequences with overlapping elements. The success of the mechanism, however, relied heavily on a tuning of the homeostatic set point (the dendritic target activity) to the specific task, reducing generality and biological plausibility. Second, we introduce a slow, activity independent decay of the synaptic maturity (Akaho, 1993; Clark & Abbott, 2024). This mechanism allows the network to prune connections that were formed during early learning but are no longer used, thereby freeing up synaptic resources for acquiring new sequences. In addition to supporting continual learning, this synaptic decay is consistent with experimental observations of synaptic turnover in biological neurons (Holtmaat et al., 2006; Malakasis et al., 2023). Finally, we introduce a depression that depends on the relative timing of pre- and post-synaptic firing. In the original spiking TM model, the synaptic maturity was reduced by a fixed amount at each presynaptic spike. In the original model, outgoing connections of frequently active neurons, such as those representing common elements in a sequence, experienced disproportionately strong depression, thereby weakening their ability to predict subsequent sequence elements. The revised mechanism removes this imbalance by making synaptic depression context sensitive.

2.2.2 *Training Data and Procedure*

An input sequence of length C is constructed by randomly, independently and uniformly drawing C elements from a vocabulary of $A = 26$ unique tokens. As a result, the average token multiplicity C/A, i.e., the average number of repetitions of some sequence element, increases with sequence length C and decreases with

vocabulary size A. Recurring tokens in complex sequences must be disambiguated based on the temporal context alone. The network must learn to represent identical input elements with distinct, sparse activation patterns within the same subpopulation, depending on the history of previously shown sequence elements. For a given vocabulary, longer sequences therefore correspond to a greater sequence complexity. Once generated, the sequence is presented to the network one element at a time with a fixed inter-stimulus interval ΔT. Each training epoch consists of a single sequence presentation, with a pause of ΔT_{seq} between successive epochs. Throughout training, the spiking activity of all neurons in the network is continuously recorded. To assess learning progress, the training process is periodically paused for performance evaluation. During this phase, the network is stimulated with the target sequence, and both spiking activity and dendritic currents are recorded. The latter serve as indicators of prediction within the network and are used for performance measurement (see Section 2.2.3). Details on the sequence model, the stimulation procedure and related parameter values are provided at `https://github.com/YounesBouhadjar/SpikingTemporalMemory`.

2.2.3 *Network Performance Estimation*

To assess how well the network learns and predicts sequences, we follow a performance evaluation approach similar to that described in Bouhadjar et al. (2022). The prediction performance is evaluated based on dendritic action potentials (dAPs) recorded during sequence presentation. For each element in the input sequence, we monitor the dAP activity across all excitatory subpopulations $\mathcal{E}_k$ within the time interval between the presentation of the current and the previous sequence element. A subpopulation is considered predictive if at least $\rho/2$ of its neurons elicit a dAP, with ρ denoting the pattern size. The predictive state of the network for each presented element of the sequence is encoded as a binary vector $\boldsymbol{o} \in \{0,1\}^M$, where $o_k = 1$ if subpopulation k is predictive, and $o_k = 0$ otherwise. This vector is compared to a binary target vector $\boldsymbol{v}$, which encodes the identity of the subpopulation receiving the current input. The prediction error

$$\epsilon = \frac{1}{C}\sqrt{\sum_{k=1}^{M}(o_k - v_k)^2} \tag{2.1}$$

is defined as the Euclidean distance between $\boldsymbol{o}$ and $\boldsymbol{v}$, normalized by the sequence length C. To further analyze performance, we also monitor the false positive rate,

$$\text{fp} = \frac{1}{C}\sum_{k=1}^{M}\Theta\left(o_k - v_k\right), \tag{2.2}$$

and the false negative rate,

$$\text{fn} = \frac{1}{C}\sum_{k=1}^{M}\Theta\left(v_k - o_k\right). \tag{2.3}$$

Here, $\Theta(\cdot)$ denotes the Heaviside function. Predictive activity in subpopulations not targeted by the current input contributes to fp, while failure to predict the target subpopulation contributes to fn.

In addition to the prediction performance, we also evaluate the time to solution, which quantifies how quickly the network is able to learn a given sequence. A model is considered to have successfully learned the sequence once the prediction error rate ϵ drops below a predefined success threshold $\epsilon^* = 0.1$. The time to solution is reported as the number of required learning episodes n^*, as well as in form of the corresponding biological time τ^*, which accounts for the actual simulated duration of the stimulus presentation and is given by

$$\tau^* = n^*(C\Delta T + \Delta T_{\text{seq}}), \tag{2.4}$$

with inter-stimulus interval ΔT and inter-sequence interval ΔT_{seq} defining the pause between successive epochs.

In the predictive mode, the dAP triggered plateau potentials are subthreshold events and not directly visible to other neurons. In nature, they can therefore hardly serve as a readout of the network's predictions. However, correct sequence element predictions can be decoded based on the context-specific sparse patterns of somatic action potentials. In this study, we extract the prediction performance directly from the dAPs to simplify the performance evaluation and thereby speed up the hyperparameter tuning.

2.2.4 *Parameter Optimization*

To optimize the learning performance of the model for sequences of a given length C_{opt}, we adjust the plasticity parameters of the model, specifically the potentiation rate λ_+, the depression rate λ_-, and the synaptic decay time constant τ_{P}. To this end, we perform a grid search across the space spanned by these parameters and identify the parameter combination that minimizes the trial averaged prediction error ϵ. The rates λ_+ and λ_- are varied between 0.01 and 0.8, while τ_{P} is sampled from a range spanning 0.5 s to 80 s. This results in a total of 1352 unique parameter combinations for each optimization run. For each parameter combination, simulations are repeated with three different random-number-generator seeds to account for the variability in network initialization and connectivity. Performances for the parameter optimization are then assessed by averaging the error over these three trials.

2.2.5 *Simulation Details*

The network simulations are performed in NEST (Gewaltig & Diesmann, 2007) version 3.6 Villamar et al. (2023). The neuron and synapse models are formalized with the help of NESTML (Plotnikov et al., 2016; Linssen et al., 2025) version 8.0.0 (Linssen et al., 2024). Network states are synchronously updated using exact integration of the system dynamics on a discrete time grid with step size $\Delta t = 0.1$ ms (Rotter & Diesmann, 1999).

2.3 RESULTS

2.3.1 *Sequence Learning Dynamics*

Figure 2.2 illustrates the evolution of the prediction error, the false negative rate (fn), and the false positive rate (fp) over the course of training for two sequences of different length. At the beginning of the training, synapses between excitatory neurons of sequentially activated subpopulations start to form due to the structural Hebbian potentiation. This leads to a rapid decrease in the fn rate, indicating that the network begins to correctly predict upcoming elements. However, this improvement comes at the cost of an increased fp rate, as subpopulations not targeted by the current input also begin to exhibit predictive activity. This effect arises because the network has not yet developed the capacity to disambiguate recurring elements appearing in different temporal contexts. Consequently, fn and fp rates often display alternating behavior, reflecting an ongoing competition. The learning curves, particularly the fp rate, are therefore not strictly monotonic. This non-trivial evolution highlights the complexity of the learning process, where forming accurate, context-sensitive representations requires multiple phases of refinement.

The examples shown in Figure 2.2 already indicate that the time to solution increases more than proportionally with sequence length. While the fn rate typically drops rapidly, the fp rate decays more slowly. A substantial portion of the training, especially for longer sequences, is spent to remove inappropriate dendritic activity across the network. As the training progresses, the model gradually carves out sparse and context-specific pathways, eventually minimizing both fn and fp.

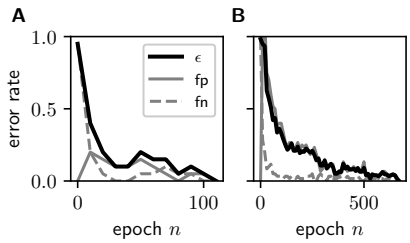

Figure 2.2: **Sequence learning progress.** Dependence of the prediction error ϵ (solid black), the false positive rate fp (solid gray), and the false negative rate fn (dashed gray) on the number of training epochs n for two different sequence lengths $C = 20$ (A) and $C = 60$ (B). Plasticity parameters $\lambda_+ = 0.6$, $\lambda_- = 0.1$, $\tau_P = 80\,\mathrm{s}$ are optimized for a sequence length $C_{\mathrm{opt}} = 40$.

2.3.2 *Sequence Learning Capacity*

The results presented in Figure 2.3 demonstrate that spiking TM networks comprising few thousand neurons can learn random sequences composed of several tens of elements. The network successfully learns sequences whose lengths exceed

the vocabulary size, and can therefore cope with recurring tokens within the input sequences.

After tuning the plasticity parameters for a given sequence length C_{opt} ($C_{\text{opt}} = 20$ in Figure 2.3A,C and $C_{\text{opt}} = 40$ in Figure 2.3B,D), the network exhibits high prediction performance for a broad range of sequence lengths near this reference length. This finding aligns with the concept of "hyperparameter transfer" (Yang et al., 2022). Such transferability alleviates the need for continuous fine-tuning when dealing with slightly varying sequence lengths. Sequences that are shorter than the reference sequence length C_{opt} are typically learned more efficiently and accurately, compared to sequences that are longer than C_{opt}. For instance, when the model parameters are optimized for a sequence length C_{opt}=20 (Figure 2.3A), the network performs well for sequence lengths ranging approximately between 10 and 40. Beyond this range, particularly for significantly longer sequences, performance deteriorates. Moreover, the results highlight that increasing the reference sequence length C_{opt} shifts the network's optimal learning range towards larger sequence lengths (Figure 2.3B).

The time to solution, i.e., the time required to reduce the prediction error below the success threshold $\epsilon^* = 0.1$, increases with the sequence length C in a supralinear manner (Figure 2.3C,D). For our choice of network parameters, learning a sequence of length $C = C_{\text{opt}} = 40$, for example, requires $n^* \approx 130$ presentations of the sequence, corresponding to a duration of $\tau^* \approx 5\,\text{min}$. Increasing the sequence length to $C = 100$ results in a time to solution of $n^* \approx 940$, i.e., $\tau^* \approx 80\,\text{min}$ (Figure 2.3D). From a biological perspective, these numbers are certainly not unrealistically large. However, they illustrate the practical challenge of systematically investigating biological learning by means of simulation, and emphasize the need for accelerated computing architectures (Kauth et al., 2020, 2023; Pehle et al., 2022; Dietrich et al., 2025).

2.4 DISCUSSION

The present chapter shows that spiking neuronal networks representing local cortical microcircuits at the submillimeter scale can successfully learn complex sequences in an unsupervised, online manner based on local synaptic plasticity mechanisms. The maximum lengths of the sequences that are successfully learned by the proposed model reach behaviorally relevant numbers, such as the number of notes in a melody or the number of words in a poem. These numbers need to be set in relation to network parameters such as the total network size and the number of neurons representing a sequence element, the pattern size ρ. With larger networks or smaller patterns, the network capacity certainly increases. A systematic investigation of the network capacity and its dependence on network parameters is dedicated to future studies.

Previous work suggests that the capacity of the TM model should be substantially larger than what is observed here (Ahmad & Hawkins, 2016; Hawkins & Ahmad, 2016). In these studies, however, the theoretical capacity limit is derived purely based on combinatorial constraints, that is, on the number of appropriate connectivity patterns that could be realized given a set of network parameters without generating

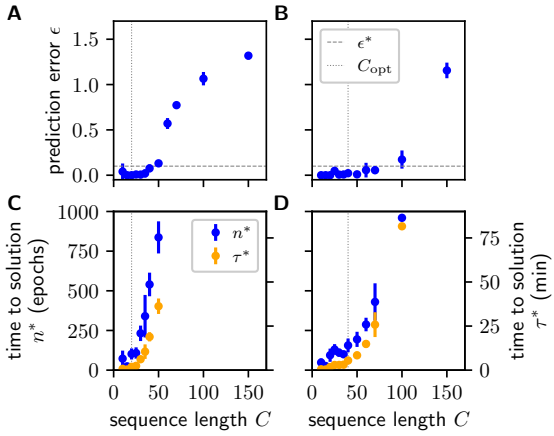

Figure 2.3: **Prediction performance and learning speed.** Dependence of the prediction error ϵ (A, B) and the time to solution n^* (number of epochs; blue) and τ^* (total learning duration in seconds; yellow; C, D) on the sequence length C. Plasticity parameters are optimized for sequence lengths $C_{\text{opt}} = 20$ (A, C; $\lambda_+ = 0.6$, $\lambda_- = 0.8$, $\tau_P = 50\,\text{s}$) and $C_{\text{opt}} = 40$ (B, D; $\lambda_+ = 0.6$, $\lambda_- = 0.1$, $\tau_P = 80\,\text{s}$), respectively (vertical dotted lines). Symbols and error bars represent mean and standard deviation across an ensemble of 5 different network realizations. In C and D, only trials with prediction error smaller than $\epsilon^* = 0.1$ (dashed horizontal lines in A, B) are used.

errors. These considerations do not take into account whether any learning scheme or plasticity mechanism can actually form such connectivity patterns. Given the results of the present study, we conclude that the plasticity dynamics constitutes a much more severe constraint on the sequence learning capacity than the network or pattern size. This is highlighted by the observation that the time to solution increases with sequence length in a strong, supralinear manner. With the given model and its specific plasticity rule, learning longer sequences may potentially be possible, but becomes very slow. It remains an open question to what extent different plasticity mechanisms or network architectures could speed up the learning process for longer sequences in spiking neuronal networks.

The plasticity mechanism employed in this study is composed of a spike-timing dependent contribution and a slow, activity independent decay of synaptic maturity. The decay prunes connections that are formed during early learning but are no longer needed after learning, thereby freeing synaptic resources and proving instrumental in achieving good learning performance. As an unwanted side effect, however, it also slowly prunes functional connections that have emerged after successful learning, unless the network continues to be exposed to the learned sequences. In future versions of the model, this forgetting could be avoided by equipping the plasticity dynamics with a consolidation process that prevents the removal of strong, learned connections (Zenke et al., 2015; Laborieux et al., 2021).

The present study is based on randomly generated sequences in which tokens are drawn independently with identical probability. With this setup, sequence complexity can be systematically varied by changing sequence length. Natural sequences exhibit more structure: tokens occur with different probabilities, and consecutive elements are often correlated. This structure in real-world sequences limits the direct comparability of the capacity estimates reported here to naturalistic settings, and a more detailed investigation of the sequence learning capacity of the spiking TM model for real-world data is dedicated to future studies.

3

SEQUENCE TIMING AND REPLAY SPEED

The chapter is based on the following publication:

Learning sequence timing and control of replay speed in networks of spiking neurons. M. Lober, Y. Bouhadjar, M. Diesmann, and T. Tetzlaff (2026) *arXiv preprint*. DOI: 10.48550/arXiv.2605.22523. Currently under review.

Author contributions:
Under the supervision of Dr. Tom Tetzlaff (TT) and Prof. Dr. Markus Diesmann (MD) the author conceptualized and led the work presented in this chapter. The author was responsible for the methodology, the implementation of the computational experiments, the analysis of the results, and the visualization. Software development was carried out jointly with Younes Bouhadjar (YB) and TT. The original manuscript was written by the author and refined through ongoing discussions with YB, MD, and TT.

3.1 INTRODUCTION

An essential aspect of sequence processing is time. Natural sequences are not only defined by the order of their elements, but also by their durations and temporal structure (Levitin & Cook, 1996; Boltz, 2010). Musical melodies, spoken language, and motor behaviors all rely on precisely controlled timing. Moreover, behavioral experience indicates that learned sequences can later be replayed at different speeds: motor sequences can be executed slowly during practice and rapidly during performance. Psychophysical studies further show that humans flexibly rescale temporal sequences while preserving relative timing (Hardy et al., 2018), and episodic memories can be mentally replayed in compressed or expanded form (Ji & Wilson, 2007; Euston et al., 2007; Liu et al., 2019; Buch et al., 2021). At the same time, this temporal flexibility is not unlimited, suggesting that neural mechanisms impose constraints on both the learning and replay of temporal structure. Despite extensive experimental and theoretical work on sequence learning, the mechanisms that allow neural circuits to encode element durations and flexibly modulate replay speed remain unclear.

While modern sequence models such as Transformers or deep recurrent networks can learn to predict and reproduce sequential inputs successfully, they rely on dense representations, global learning rules, and positional encodings with no identified counterpart in neural circuitry. Understanding how the brain solves sequence processing therefore requires a different approach, grounded in sparse population codes, local synaptic plasticity, and intrinsic temporal dynamics that characterize cortical computation.

As established in the previous chapters, the sTM model provides a biologically grounded framework for learning and replaying the order of sequence elements. However, it remains unclear whether it can learn temporal features such as the duration of sequence elements and inter-element intervals, and how the speed of replay can be modulated. The original formulation predicts bounds on replay speed that arise from synaptic and neuronal time constants (Bouhadjar et al., 2022), yet it cannot predict sequences whose elements are separated by intervals exceeding these intrinsic timescales, and the replay speed depends only on intrinsic time constants rather than on the temporal structure present during learning. The replay speed in the original sTM model can be influenced by parameters such as the neuronal firing threshold, which determine how readily dendritic events trigger somatic spikes. However, replay was demonstrated only for a specific parameter regime in which sequences were replayed at a high intrinsic speed, close to the upper limit set by neuronal and synaptic time constants. Thus, a mechanism that enables robust and flexible modulation of the replay speed over a broader range, without reconfiguring intrinsic cellular parameters, remains to be established.

The present chapter addresses both issues within a unified spiking TM framework. We propose a mechanism that enables the learning and replay of sequences with complex rhythmic structure, including long and heterogeneous temporal intervals, while preserving sparse, context-dependent coding. In addition, we propose that oscillatory background input can serve as a flexible clock signal for replay. Oscillations are ubiquitous in cortical and hippocampal circuits and have long been

hypothesized to provide temporal reference frames or clock-like signals for neural computation (Fries, 2005; Buzsáki, 2006). This mechanism suggests a functional role of network oscillations in controlling sequence dynamics.

3.2 METHODS

3.2.1 *The spiking Temporal Memory (sTM) Model*

This section provides an overview of the key components of the sTM model (Bouhadjar et al., 2022) and its adaptations used in this study. The model builds directly on the formulation introduced in Chapter 2. A detailed mathematical description of the model and the full set of parameters are given in the supplement to this chapter Chapter 7.

The sTM is a randomly and sparsely connected recurrent neural network (Figure 3.1) organized into M subpopulations (minicolumns; Mountcastle (1997); Buxhoeveden & Casanova (2002)), each consisting of n_{E} excitatory ($\mathcal{E}_k$; $k = 1, \ldots, M$) and n_{I} inhibitory neurons ($\mathcal{I}_k$; here: $M = 6$, $n_{\mathrm{E}} = 200$, $n_{\mathrm{I}} = 1$). Excitatory neurons within each minicolumn project to their associated inhibitory neuron, which in turn inhibits all excitatory neurons in the network. On the one hand, this architecture enforces a competition between minicolumns (Lundqvist et al., 2006; Johansson & Lansner, 2007). On the other hand, it implements a local winner-take-all (WTA) mechanism that promotes sparse activity (Maass, 2000). During sequence presentation, excitatory neurons within each minicolumn receive stimulus-specific, suprathreshold bottom-up inputs ($\mathcal{X}_k$). Lateral excitatory connections between excitatory neurons target distal dendritic branches. When the synaptic input to a dendritic branch exceeds a threshold, a dendritic action potential (dAP; "NMDA spike") is generated, leading to a long-lasting (100 ms) subthreshold plateau depolarization of the soma (Antic et al., 2010). These plateau potentials signal predictions: after successful learning, they precede the arrival of an expected bottom-up input. When the expected input arrives, predictive neurons fire earlier than others, confirming the prediction. If a sufficient number of ρ neurons within a minicolumn are in a predictive state and hence fire earlier, the local inhibitory feedback kicks in and prevents all other (late) neurons in the minicolumn from firing (WTA). This selection of a sparse subset of active neurons leads to a context-specific representation of the current sequence element. After learning, the same input can activate different neuronal groups (assemblies Palm (1982)) depending on the preceding sequence. These assemblies are defined by their consistent co-activation rather than by stronger recurrent connectivity within the group (Wagner-Carena et al., 2025). This produces a temporally evolving pattern of context-specific sparse activity ("bar code") that represents the progression of the sequence (see Figure 3.2A). In the absence of any additional information, the first element in a sequence can not be anticipated. A bottom-up input representing the first element in the sequence leads to a non-sparse response of all neurons in the corresponding minicolumn, and therefore to an ambiguous activation of all possible assemblies present in this minicolumn. Here, we avoid this ambiguity by assuming that the start of a given sequence is predicted by an additional top-down input ($\mathcal{T}$), which activates the dendrites of a sparse subset

of neurons in the corresponding minicolumn prior to the arrival of the bottom-up stimulus representing the first sequence element. Upon arrival of this stimulus, only this sparse subset of neurons fires. Synapses between excitatory neurons are plastic and evolve according to a structural spike-timing-dependent plasticity rule combined with a continuous synaptic weight decay ("forgetting") (Bouhadjar et al., 2025b), as introduced in Chapter 2.

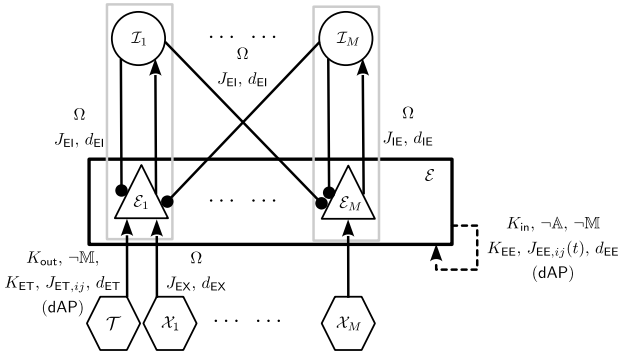

Figure 3.1: **Sketch of the sTM network architecture.** Graphical description of network according to Senk et al. (2022). *Populations:* $\mathcal{E}_k$: excitatory populations. $\mathcal{I}_k$: inhibitory populations. $\mathcal{X}_k$: bottom-up inputs ($k = 1, \ldots, M$). $\mathcal{T}$: top-down input ($\mathcal{E}_1$ hosts assembly representing the first element of the input sequence). *Connectivity rules:* Triangular arrow heads: excitatory connections. Circular arrow heads: inhibitory connections. Solid lines: static connections. Dashed lines: plastic connections. K_{in}: random connectivity with fixed indegree. K_{out}: random connectivity with fixed outdegree. Ω: all-to-all connectivity. $\neg\mathbb{M}$: multiple connections between two neurons not permitted. $\neg\mathbb{A}$: self-connections not permitted. Recurrent connections between excitatory neurons (dashed) target distal dendritic branches and may trigger dendritic action potentials (dAPs) signaling predictions. *Connectivity parameters:* synaptic weights J_{xy}, spike transmission delays d_{xy} ($x, y \in \{\mathrm{E}, \mathrm{I}, \mathrm{X}, \mathrm{T}\}$). A detailed mathematical description of the model is provided in Chapter 7.

3.2.2 *Readout Neurons*

To decode network activity after learning, we employ linear regression to learn a linear mapping from network states to target labels (bottom panels in Figure 3.2). To this end, we present all C elements of the sequence to the network once and record the resulting spiking activity of all N_{E} excitatory neurons. The network state at each sequence position $c \in \{1, \ldots, C\}$ is represented by a binary vector $\boldsymbol{r}_c \in \{0,1\}^{N_{\mathrm{E}}}$, where the component $i \in [1, N_{\mathrm{E}}]$ is set to one if neuron i fires a somatic spike within a temporal window of $[-4\,\mathrm{ms}, +14\,\mathrm{ms}]$ relative to the presentation time of the cth sequence element, and zero otherwise. Concatenating these vectors yields the network state matrix

$$\boldsymbol{R} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_C] \in \{0,1\}^{N_{\mathrm{E}} \times C}. \tag{3.1}$$

The target label for sequence position c is a binary vector $\boldsymbol{y}_c \in \{0,1\}^C$ with a one in the cth component and zeros elsewhere, encoding the identity of the presented sequence element. Concatenating these vectors yields the target label matrix

$$\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_C] = \boldsymbol{I}_C \in \{0,1\}^{C \times C}, \tag{3.2}$$

which is identical to the $C \times C$ identity matrix $\boldsymbol{I}_C$. Linear regression finds a weight matrix $\boldsymbol{W} \in \mathbb{R}^{C \times N_E}$ such that

$$\hat{\boldsymbol{Y}} = \boldsymbol{W}\boldsymbol{R} \approx \boldsymbol{Y}, \tag{3.3}$$

where $\hat{\boldsymbol{Y}} \in \mathbb{R}^{C \times C}$ is the decoder matrix. The components of $\hat{\boldsymbol{Y}}$ range from 0 to 1 and quantify the decoding accuracy: the component $\hat{\boldsymbol{Y}}_{cc'}$ reflects how confidently the decoder identifies position c' as belonging to sequence element c. Perfect decoding corresponds to $\hat{\boldsymbol{Y}} = \boldsymbol{I}_C$. Here, we obtain the weight matrix $\boldsymbol{W}$ using `sklearn.linear_model.LinearRegression()` (Pedregosa et al., 2011) via the Functional Neural Architectures (FNA) toolbox (Duarte et al., 2021).

3.2.3 *Hyperparameter Tuning*

To obtain optimal learning performance, we perform hyperparameter optimization (Goodfellow et al., 2016, Chapter 11) of the synaptic plasticity rule for the most challenging input sequence used throughout the study: the reference melody at one third of the baseline speed, corresponding to a sequence length of $C = 24$. We optimize the potentiation rate λ_+, depression rate λ_- and permanence leak time constant τ_P. The search ranges are $\lambda_+ \in [0.1, 0.5]$, $\lambda_- \in [0.15, 0.65]$ and $\tau_P \in [25\,\mathrm{s}, 50\,\mathrm{s}]$. The hyperparameter tuning uses the Bayesian optimization algorithm of the *Weights & Biases* framework (Shahriari et al., 2016; Biewald, 2020) evaluating learning performance according to the sequence prediction error η defined in Table 7.8. Subsequently, we use the resulting optimal parameter set to train the network on the reference melody at all three learning speeds: baseline, half of baseline, and one third of baseline speed. This approach is motivated by our previous findings (Bouhadjar et al., 2025b), which show that plasticity parameters optimized for longer sequences typically generalize well to shorter sequences, allowing hyperparameter transfer. While the sTM model exhibits high theoretical storage capacity (Ahmad & Hawkins, 2016; Hawkins & Ahmad, 2016), practical learning efficiency depends on the dynamics of synaptic plasticity. In particular, the number of training iterations required increases with sequence length (Bouhadjar et al., 2019), which can make learning long sequences computationally demanding.

3.2.4 *Simulation Details*

All network simulations in this study are performed using the neuronal simulator NEST (Gewaltig & Diesmann, 2007), version 3.8 (Graber et al., 2024a). The excitatory neuron model and the plastic synapse model are defined in the domain-specific language NESTML (Plotnikov et al., 2016; Linssen et al., 2025), version 8.0.0 (Linssen et al., 2024). Simulations are carried out with synchronous updates using exact

integration of the system dynamics on a discrete-time grid with a fixed step size Δt (Rotter & Diesmann, 1999). The source code and simulation data necessary to reproduce the findings of this study are available on `zenodo` (Lober et al., 2026b).

3.3 RESULTS

3.3.1 *Learning Sequence Timing*

A key temporal constraint of the sTM model arises from the finite duration of the plateau potentials, typically lasting few hundred milliseconds (Schiller et al., 2000; Milojkovic, 2005; Larkum et al., 2009; Antic et al., 2010). As a result, the maximum interval between two consecutive elements in a sequence that can be bridged by dendritic activity is limited. This poses a challenge for learning and replaying slow sequences or rhythms with extended temporal gaps. Moreover, the time to the prediction of an upcoming sequence element, i.e., the onset of the dAP, as well as the time to the activation of the corresponding subset of neurons during autonomous replay is dictated by physiological parameters such as synaptic or membrane time constants, and independent of the inter-element intervals presented during learning. It is therefore unclear how the sTM model can learn (relative) time intervals or the duration of individual sequence elements. To represent the temporal structure of a sequence, the network requires a mechanism to encode time intervals between successive elements. The solution proposed here is to discretize time into elementary intervals shorter than the duration of a dendritic plateau potential and to construct longer intervals from a concatenation of these elementary intervals. In this study, this is realized by a delay line of sequentially activated neuronal assemblies. In the simplest case, these delay lines can be learned through repeated presentations of the same input. The network thereby learns a sequence of distinct context-specific assemblies within the same minicolumn. Their sequential activation then encodes the duration of the corresponding element.

Throughout this study, we consider a set of sTM networks, each trained on a single sequence derived from a variation of a short melody (opening of "Oh, Pretty Woman" by Roy Orbison, 1964). The base sequence $s = \{$"C1","C1","E1","G1","B1","D2","C2","B1"$\}$ consists of $C = 8$ elements drawn from a set of six musical notes (Figure 3.2A). Slower variants of this reference melody are constructed by repeating each note once or twice, yielding sequences of length $C = 16$ and $C = 24$. During training, sequences are presented 500 times, with a fixed inter-element interval of $\Delta T = 40$ ms. We refer to the inter-element interval during training as the encoding interval.

Figure 3.2 illustrates our approach for two sTM networks trained on the reference melody at baseline speed ($C = 8$) and on a slowed down version with a dilation factor of three ($C = 24$), respectively. The temporal structure ("rhythm") is represented by small populations of neurons, similar to a "bar code" that evolves over time. These bar codes provide an implicit representation of time elapsed since sequence onset, and enable a reliable decoding of sequence progression, i.e., the time since sequence start (see Figure 3.2 bottom panels).

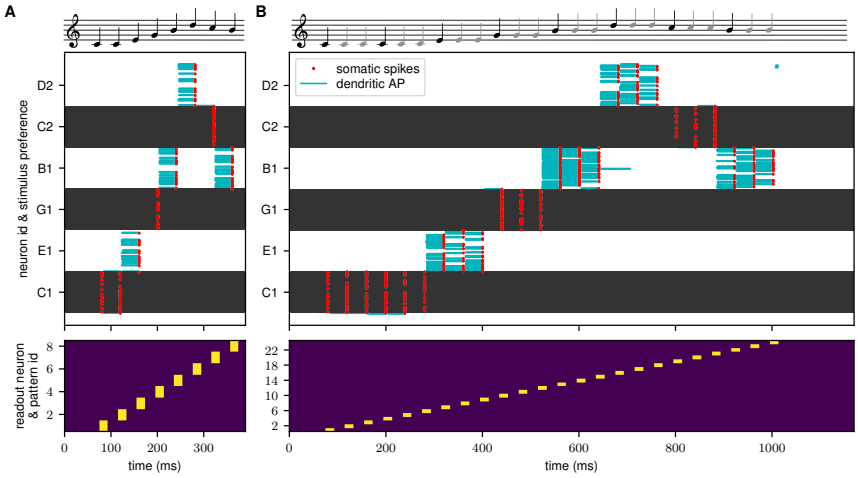

Figure 3.2: **Encoding long intervals and rhythm through repetitive elements.** Network activity (middle row) after learning a melody (top row) presented at baseline speed (**A**) and a third of the baseline speed (**B**). Gray musical notes in the top row of panel B indicate repeated presentations of the preceding note, each with identical duration. Red dots in the middle row mark spikes, blue lines indicate dAP plateaus. The bottom row shows the readout neuron activity (violet: inactive, yellow: active), obtained by decoding the assembly identity from the spiking activity.

3.3.2 *Modulating Replay Speed*

The sTM network can be configured into a "replay mode" by increasing the excitability of the excitatory neurons, such that the depolarization caused by a plateau potential alone is sufficient to elicit a spike. In the original formulation of the sTM model (Bouhadjar et al., 2022), this is achieved by reducing the spike threshold of excitatory neurons. From a dynamical perspective, this manipulation is equivalent to providing a constant background current to all excitatory neurons in the network. In principle, this procedure permits a modulation of the replay speed by changing the magnitude of the background current. In the following, we will however show that this mechanism offers only limited flexibility in controlling the speed at which activity propagates through the network. An oscillatory background input, instead, provides a clock signal that can be used to flexibly and robustly regulate the speed of sequence replay.

3.3.3 *Driving Replay with Constant or Oscillatory Background Inputs*

A plateau potential generated by a dAP can directly trigger a somatic spike if it occurs during a phase of increased excitability, which may be caused by a sufficiently strong unspecific background input. This background input could be a constant input (Figure 3.3A), or a transient input, such as an oscillatory input

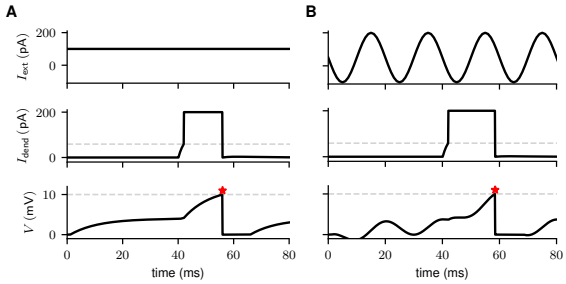

Figure 3.3: **Illustration of the effect of different types of background activity on the membrane potential of an excitatory neuron in the sTM model.** The top row shows the external background current I_{ext} applied to the neuron: a constant input of magnitude $\bar{I} = 100\,\text{pA}$ (**A**) or an oscillatory input with frequency $f = 50\,\text{Hz}$, amplitude $a = 150\,\text{pA}$, and offset $\bar{I} = 50\,\text{pA}$ (**B**). In both cases, the neuron additionally receives a predictive input to its dendrite at around 40 ms (middle row). The resulting dendritic current I_{dend} increases until it reaches a threshold I_{dAP} (gray dashed line), triggering a plateau potential I_{dAP}. The superposition of background and dendritic input leads to a strong depolarization of the somatic membrane potential V (bottom row) and a crossing of the spike threshold V_θ (gray dashed line). The neuron elicits a somatic spike (red star), which, in turn, triggers a reset of the dendritic current.

(Figure 3.3B). As the background input is applied to all excitatory neurons in the sTM network, stable replay requires a proper choice of parameters: on the one hand, the background input must be strong enough to trigger a somatic spike in neurons that are in a predicted state. On the other hand, it must be weak enough to leave unpredicted neurons below the firing threshold. Only the superposition of the plateau potential and the background input should elicit a spike. Under these conditions, stable replay corresponds to the sequential activation of all neuronal assemblies encoding the sequence in the correct order, with no spurious activation of assemblies that do not belong to the replayed sequence. The properties of the background input therefore determine whether stable replay of learned sequences is possible, and the speed at which learned sequences are replayed in response to a cue. For a constant background input, the only critical parameter is its magnitude $\bar{I}$. For a transient background input, not only the magnitude, but also the timing relative to the plateau potential matters. Moreover, the duration of the high-excitability phase must be long enough to allow the membrane potential to reach the spike threshold. For oscillatory inputs, sequence replay therefore depends on proper choices of the oscillation frequency f, the phase ϕ, the amplitude a, and the offset $\bar{I}$. In the following, we will compare sequence replay in sTM networks driven by constant or oscillatory background inputs, and investigate the range of parameters that permit a stable replay of learned sequences and how different parameter settings affect the replay speed.

For both constant and oscillatory background input, the network replays learned sequences over a range of speeds $f_{\text{out}} = 10 \ldots 70\,\text{Hz}$ (Figure 3.4 and Figure 3.5). Here, the replay speed f_{out} is defined as the inverse of the average inter-assembly

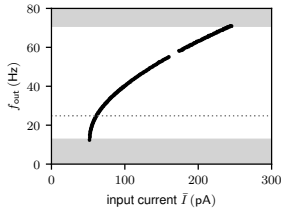

Figure 3.4: **Modulation of replay speed by a constant background input.** Dependence of the replay speed f_{out} on the magnitude $\bar{I}$ of a constant input current for the reference melody learned at baseline speed (Figure 3.2A). The gray dotted line marks the encoding speed $f_{enc} = 1/\Delta T = 25\,\text{Hz}$, i.e., the inverse inter-assembly interval during learning. Regions shaded in light gray correspond to parameter regimes where no replay is observed.

interval during replay, computed from the last four activations of neuronal assemblies in the sequence. The duration of the plateau potential $\tau_{dAP} = 100\,\text{ms}$, which determines the maximum interval between consecutive assemblies, sets the lower bound of this range. Spike transmission latencies for connections between excitatory neurons, rise times of dendritic synaptic currents, and membrane time constants of excitatory neurons determine the upper bound. The range of replay speeds is hence solely determined by neurophysiological parameters and the parameters of the background input. The encoding speed $f_{enc} = 1/\Delta T$, i.e., the speed at which assemblies are activated during learning (the inverse encoding interval) has no effect on the replay speed (provided it is not too small or too high for the sequence to be learned). Through modulation of the background input parameters, sequence replay may be slower or faster than the encoding speed (cf. gray dotted line in Figure 3.4 and gray arrow in Figure 3.5A). This mechanism therefore permits a modulation of the replay speed within certain bounds, without the need to relearn the sequence at the desired replay speed, and without the need to change intrinsic neuronal parameters.

Constant and oscillatory background inputs support similar replay speed ranges. However, the two types of inputs differ in how sensitively the replay speed depends on the input parameters. With a constant background current, slow replay requires fine tuning. For replay speeds between 10 Hz and 30 Hz, a tiny change in the input current has a strong effect (Figure 3.4). Oscillatory background input, in contrast, provides a more robust and flexible control of the replay speed (Figure 3.5). Over wide parameter ranges, the replay speed scales linearly with the oscillation frequency. In the green band in Figure 3.5B, the replay speed matches the input frequency, $f_{out} = f$, such that successive oscillation cycles activate successive neuronal assemblies. Within this regime, the external oscillation acts as a "clock signal" determining the replay speed in a 1:1 manner. A broad range of replay speeds can be achieved by adjusting the oscillation frequency while keeping the amplitude fixed (vertical paths in Figure 3.5A,B and Figure 3.5C–E). Unless the oscillation amplitude is adjusted accordingly, the low-pass filtering properties of the neuronal membrane lead to an attenuation of the membrane potential response as the oscillation frequency increases. At some critical frequency, the superposition

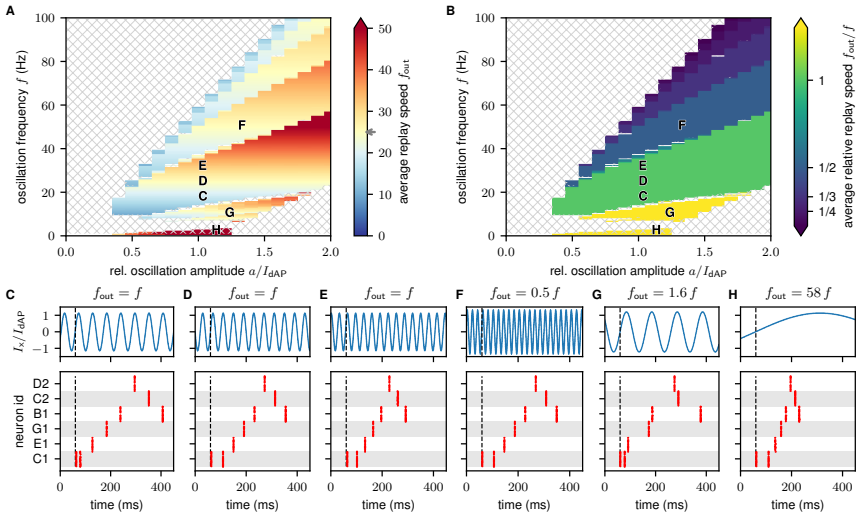

Figure 3.5: **Modulation of replay speed by oscillatory background input. A**: Dependence of the replay speed f_{out} on the relative oscillation amplitude a/I_{dAP} and the oscillation frequency f during replay of the reference melody (Figure 3.2A). The oscillation offset $\bar{I}$ is zero. The replay speed is averaged across eight different oscillation phases. The gray arrow in the colorbar marks the encoding speed $f_{\text{enc}} = 1/\Delta T = 25\,\text{Hz}$, i.e., the inverse inter-assembly interval during learning. Hatched areas represent parameter regions in which stable replay occurs for fewer than four of the eight phases. **B**: Same as A, but for relative replay speed f_{out}/f. Green and blue bands correspond to regions where the replay speed is an integer fraction of the oscillation frequency, i.e., $f_{\text{out}} = f, f/2, f/3$, etc. Yellow regions indicate replay speeds exceeding the oscillation frequency, $f_{\text{out}} > f$. Bold letters in A and B mark the parameter settings illustrated in panels C–H. **C–H**: background oscillation (top) and corresponding replay activity (bottom) for different oscillation frequencies f and amplitudes a: $f = 18\,\text{Hz}$, $a/I_{\text{dAP}} = 1.1$ (C), $f = 25\,\text{Hz}$, $a/I_{\text{dAP}} = 1.1$ (D), $f = 32\,\text{Hz}$, $a/I_{\text{dAP}} = 1.1$ (E), $f = 50\,\text{Hz}$, $a/I_{\text{dAP}} = 1.3$ (F), $f = 10\,\text{Hz}$, $a/I_{\text{dAP}} = 1.2$ (G), $f = 1\,\text{Hz}$, $a/I_{\text{dAP}} = 1.1$ (H). For all examples, the initial input triggers the replay at oscillation phase $\phi = 0$ (vertical dashed lines).

of the oscillatory input and the plateau potential may therefore no longer reach the spike threshold within a single oscillation cycle. Beyond this point, the membrane may still reach the spike threshold by integrating the input over multiple oscillation cycles. In this regime, successive assemblies are activated only at every second, third, or higher-order oscillation cycle (blue bands in Figure 3.5B). This transition to a regime where the replay speed is an integer fraction of the oscillation frequency is also observed when the oscillation amplitude is decreased while keeping the frequency fixed (horizontal paths in Figure 3.5A,B). This effect allows the same input frequency to generate multiple replay speeds depending on the oscillation amplitude. If the oscillation frequency is too high or the amplitude too low, replay is no longer possible (hatched regions in Figure 3.5A,B). At low frequencies below 10 Hz, the replay speed exceeds the frequency f of the oscillatory input. Here, multiple assembly activations occur within a single oscillation peak, resulting in irregular replay (Figure 3.5G). At very low frequencies < 5 Hz, the entire sequence may replay within a single oscillation peak (Figure 3.5H). In this regime, sequence replay is fast and compressed, reminiscent of what is observed during slow-wave sleep in the hippocampus and in the neocortex (Ji & Wilson, 2007; Euston et al., 2007; Buzsáki, 2015; Klinzing et al., 2019). At these low frequencies, the oscillation period is longer than the duration of the plateau potential. The predictive signal therefore decays before the next oscillation peak arrives. As a result, replay is effectively limited to a single oscillation peak, restricting the maximum sequence length (Euston et al., 2007). Moreover, replay success becomes sensitive to the oscillation phase at the time of replay initiation (see Section 3.3.4.1).

3.3.4 *Range of Accessible Replay Speeds*

As shown in the previous section, global background input allows the sTM network to modulate replay speed over a range of approximately 10 Hz to 70 Hz. For our example sequence, the reference melody learned at baseline speed consisting of eight elements, this corresponds to a total replay duration, i.e., the time between the first and the last assembly activation, between about 90 ms and 700 ms. Within this range, the intrinsic neuronal and synaptic time constants limit the replay speed on the fast end. On the slow end, it is limited by the duration of the plateau potentials. Extending the replay duration beyond this temporal window requires modifying how the network represents the sequence. As discussed in Section 3.3.1, slower sequences can be learned by repeating elements during training such that each token is encoded by several successive neuronal assemblies within the same minicolumn. During replay, the network must activate all of these assemblies sequentially, which elongates the path through the network and thereby increases total replay duration. By varying the parameters of the oscillatory background input, a range of replay speeds can be achieved for each learned sequence representation (Figure 3.6). For the reference melody learned at baseline speed, the distribution of replay durations peaks at approximately 180 ms, corresponding to a replay speed of $f_{\text{out}} = 40$ Hz. When the system learns the same sequence at half or one third of baseline speed, longer replay durations become possible, reaching up to approximately 1.5 s and 2.3 s, respectively. This increase follows from the extended

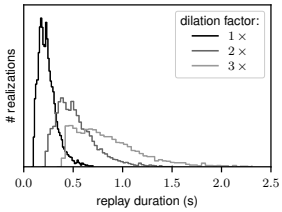

Figure 3.6: **Range of replay durations achievable with oscillatory background input for sequences with different temporal dilation factors.** Distribution of replay duration across the ensemble of tested parameter combinations with $a \in \{0, 0.1, \ldots, 3\}\, I_{\mathrm{dAP}}$, $f \in \{1, 1.5, \ldots, 100\}\,\mathrm{Hz}$, $\phi \in \{0, 1/4, \ldots, 2\}\,\pi$, and $\bar{I} = 0\,\mathrm{pA}$ for a network trained on the reference melody with three different temporal dilation factors (see legend). The cases "1×" and "3×" are shown in Figure 3.2A and B, respectively.

representational pathway in the network. Lower learning speeds recruit more neuronal assemblies per sequence element, all of which must be traversed during replay. If we assume that the inter-assembly interval during replay is constant across the sequence and given by $1/f_{\mathrm{out}}$, the total replay duration can be approximated as $(C-1)/f_{\mathrm{out}}$, where C denotes the total number of assemblies used to encode the sequence. Increasing C hence leads to a shift of the entire distribution of replay durations to larger values (Figure 3.6). Together, these examples illustrate that the total replay duration depends both on the background input driving replay and on the sequence speed during learning. A sequence learned at some given speed can be replayed at a range of different speeds without relearning. Extending this range, however, requires learning a new representation of the sequence.

3.3.4.1 *Phase Invariance*

In natural settings, the timing of the external cue stimulus that triggers replay is unlikely to be locked to an intrinsic oscillation. The phase of the oscillation at replay onset is therefore effectively random and can hardly be controlled Bouhadjar et al. (2023). To be biologically plausible, the mechanism proposed in this study for controlling the replay speed must hence be insensitive to a variability in the replay onset phase. For many combinations of the oscillation frequency and amplitude, this is indeed the case (Figure 3.5A). At low frequencies, however, replay driven by oscillatory background input becomes unreliable. In this regime, successful replay depends strongly on the phase of the oscillation at the time of the cue. If the cue occurs shortly before or during a trough in the background input, the predictive dendritic signal may decay before the next peak arrives, preventing the subsequent neuronal assembly from becoming active.

Even if the predictive signal reliably leads to a response spike, the timing of this spike may differ for different onset phases. Parameter regions that support stable replay may therefore exhibit variability in the replay speed. Here, we investigate this phase dependence of the replay speed by analyzing the variability of the individual inter-assembly intervals during replay across different oscillation phases (Figure 3.7). For frequencies $f > 20\,\mathrm{Hz}$, the replay speed is largely invariant to

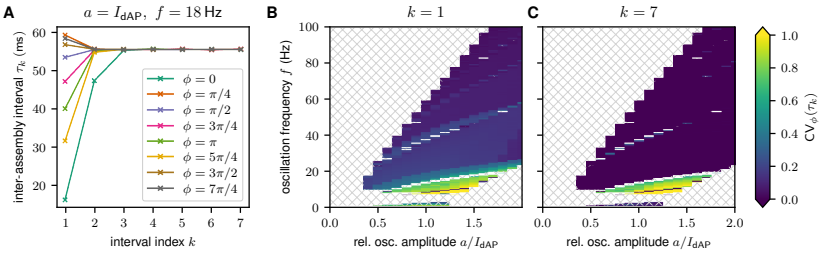

Figure 3.7: **Phase invariance of replay speed. A**: Dependence of inter-assembly interval τ_k on interval index k for eight different oscillation phases at replay onset (legend) of the reference melody at baseline speed (Figure 3.2A). **B,C**: Variability of replay timing, quantified by the coefficient of variation $\mathrm{CV}_\phi(\tau_k)$, of the first ($k = 1$; B) and the last inter-assembly interval ($k = 7$; C) as a function of oscillation frequency f and relative amplitude a/I_{dAP} with offset $\bar{I} = 0\,\mathrm{pA}$. Hatched areas represent parameter regions where stable replay occurs for fewer than four of the eight phases.

the oscillation phase. The first inter-assembly intervals after replay onset exhibit some small amount of variability, but this initial variability quickly disappears as replay progresses. Only at low frequencies, the across-phase variability of the inter-assembly intervals is large and persists across the entire sequence.

3.3.4.2 *Robustness of Replay in the Presence of Noise*

In the present study, we model the background input as a constant or an oscillatory current injected into the excitatory neurons of the sTM network. In the biological system, this background input is the result of synaptic inputs from presynaptic sources firing in a tonic or oscillatory manner. The net background currents are therefore subject to fluctuations due to the variability in the presynaptic spike times. Here, we investigate the robustness of sequence replay in the presence of such noise for varying replay speeds and for both constant and oscillatory background input (Figure 3.8).

To account for the close spatial proximity and shared input statistics of neurons within a minicolumn, we model background noise as an additive shot-noise process (low-pass filtered Poissonian spike trains) with a high correlation coefficient $c = 0.9$ (see Equation 7.15 in Section 3.2). Thus, neurons within a minicolumn receive 90% of their noise from a shared source and the remaining 10% from independent sources. Neurons in different minicolumns receive independent noise. We parameterize the noise amplitude by the standard deviation σ of the resulting fluctuations in the neuronal membrane potential.

In the presence of noise, some neurons in an activated assembly reach the spike threshold earlier than others. Hence, increasing the noise amplitude σ leads to a dispersion of spike times within an assembly. This desynchronization eventually causes failures in triggering dAPs in the next assembly, and ultimately leads to a breakdown of replay. The noise induced desynchronization can be counteracted to some extent by increasing the level of noise correlation within each minicolumn,

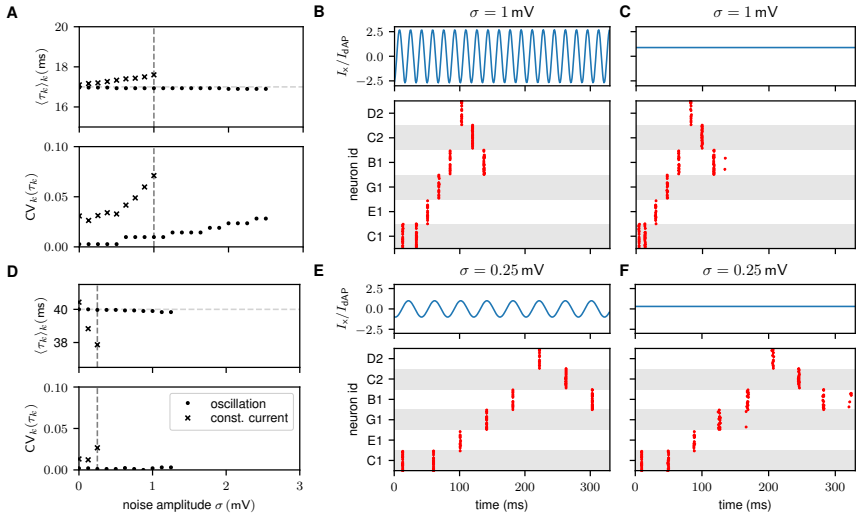

Figure 3.8: **Stabilization of sequence replay by oscillatory input in the presence of noise. A,D**: Dependence of the mean inter-assembly interval $\langle \tau_k \rangle_k$ (top panels) and the coefficient of variation $\mathrm{CV}_k(\tau_k)$ of the inter-assembly intervals τ_k (bottom panels) on the amplitude σ of a correlated background noise ($c = 0.9$). Mean inter-assembly intervals and variation coefficients are obtained from the last four elements ($k \in [4, 7]$) of the sequence. Crosses and circles represent data for constant and oscillatory background input, respectively (legend). **B–F**: Background current (top) and spiking activity (bottom) for different configurations of background input: $f = 59\,\mathrm{Hz}$, $a/I_{\mathrm{dAP}} = 2.7$ (B), $\bar{I}/I_{\mathrm{dAP}} = 0.885$ (C), $f = 25\,\mathrm{Hz}$, $a/I_{\mathrm{dAP}} = 1$ (E), $\bar{I}/I_{\mathrm{dAP}} = 0.305$ (F) with $\phi = 0$ and $\bar{I} = 0\,\mathrm{pA}$. Examples are shown at noise amplitudes σ corresponding to the highest noise levels at which replay remains stable for both constant and oscillatory background input (see vertical dashed lines in A,D). Top and bottom rows correspond to parameter combinations resulting in a faster and a slower replay, respectively ($\langle \tau_k \rangle_k = 17\,\mathrm{ms}$ and $40\,\mathrm{ms}$ for $\sigma = 0\,\mathrm{mV}$). Data shown here are obtained from a single noise realization.

which keeps the spike times of neurons within the same assembly aligned even at moderate noise amplitudes (see Bouhadjar et al. (2023)). However, some fraction of synaptic input noise is likely to be uncorrelated across neurons (here, 10%), such that the desynchronizing effect of noise cannot fully be avoided. For transient (e.g., oscillatory) background input, the desynchronizing effect of noise is reduced by the mechanisms described in Mainen & Sejnowski (1995); Mohns et al. (1999); Goedeke & Diesmann (2008): transient inputs lead to a faster threshold crossing, such that there is less time for noise to accumulate. As a result, the spike times of neurons in the same assembly are less dispersed. Overall, we observe that sequence replay is more robust to noise when driven by oscillatory background input, as compared to constant background input. Oscillatory background input permits larger (and biologically realistic) noise amplitudes (Figure 3.8A,D, top panels). The range of tolerable noise amplitudes is larger for faster replay, because the spike threshold is

approached more quickly, which, in turn, synchronizes assembly spikes. This effect is observed for both constant and oscillatory background input. Within the range of tolerable noise amplitudes, the variability of the inter-assembly intervals across sequence elements increases more rapidly with σ if the background is constant, as compared to oscillatory background input (Figure 3.8A,D, bottom panels). In other words, with oscillatory background input, the replay speed remains more stable across sequence elements. Similarly, in the presence of oscillatory background input, the replay speed remains constant when increasing the noise level, up to the point where replay breaks down (Figure 3.8A,D, top panels). With constant background input, in contrast, the replay speed systematically changes when increasing the noise level.

3.3.5 *Testable Predictions*

The mechanisms proposed in this work give rise to several experimentally testable predictions.

3.3.5.1 *Working memory with finite duration through stereotyped sequential activation of sparse assemblies*

The proposed mechanism represents extended temporal intervals by repeated activation of stimulus-specific, sparse neuronal subsets within the same minicolumn (Figure 3.2). This suggests that working memory for temporally extended stimuli relies on a progression through a finite sequence of distinct assemblies. Experimentally, this predicts that during maintenance of rhythmic or temporally extended stimuli, neural recordings should reveal sequential activation of stimulus-specific subpopulations even when the external input is constant. The duration of a remembered item should correlate with the number of assembly activations rather than with the persistence of a single active ensemble. High-resolution recordings should therefore reveal a "bar code"-like temporal evolution of sparse population activity.

This representation further implies a cost for encoding slow sequences. Longer intervals require more assemblies and increased spiking activity, making them more demanding to learn. Accordingly, learning slower sequences should require more training trials than faster ones. Consistent with this idea, human psychophysics shows improved temporal precision at faster speeds, while slower sequences exhibit increased variability (Hardy et al., 2018).

3.3.5.2 *Replay speed is controlled by global oscillatory dynamics*

According to the mechanism proposed in this study, the speed of sequence replay in sTM networks scales with the frequency of a global oscillatory input (Figure 3.5). This predicts that the speed of replay in the hippocampus and in the neocortex correlates with the dominant frequency of ongoing oscillations measurable in EEG or LFP signals. Moreover, during low-frequency oscillations (1–4 Hz), the number of sequence elements that can be replayed within a single cycle should be limited. Consistent with this, replay in bats during sleep is temporally compressed but fragmented, suggesting constraints on replay duration (Eliav et al., 2025).

In addition, replay onset during slow-wave sleep should be phase-locked to the underlying oscillation, as successful initiation requires a favorable phase.

3.3.5.3 *Oscillation frequency determines the perceived duration of time intervals*

The proposed mechanism predicts a direct link between the frequency of global background oscillations and the subjective estimation of time. In our framework, elapsed time is encoded by the sequential progression of activity through an assembly chain, and the replay speed scales with oscillation frequency. When replay proceeds faster than the speed at which the sequence was learned, activity traverses the assembly chain more rapidly, causing the network to reach assemblies associated with later sequence elements sooner. This corresponds to an overestimation of elapsed time: the internal representation signals that more time has passed than has actually elapsed. Conversely, slower replay driven by lower oscillation frequencies should lead to an underestimation of elapsed time. These predictions hold within each of the distinct bands identified in Figure 3.5B, where replay speed scales linearly with the oscillation frequency. These predictions are consistent with experimental findings showing that pharmacological and physiological manipulations affecting the speed of neural dynamics systematically bias time estimation. Increased dopamine levels, which have been associated with accelerated neural dynamics, lead to a prolongation of estimated time intervals (see Hass & Durstewitz (2016) and references therein). Cooling of the brain during duration estimation tasks in rats causes a slowing of neuronal dynamics and a corresponding underestimation of elapsed time (Monteiro et al., 2023).

3.3.5.4 *Replay speed correlates with sequence speed during learning*

The temporal range over which a sequence can be replayed depends on how it was encoded during learning (Figure 3.6). Sequences learned at faster speeds have shorter representational chains and therefore more restricted replay ranges. This leads to two predictions: (i) replay duration should correlate with the speed at which the sequence was learned, and (ii) replay at speeds far outside the learning regime may require relearning at a different speed. These predictions can be tested in psychophysics experiments.

3.3.5.5 *Rapid fading of across-phase interval variability*

In the model presented here, the time intervals between the first elements of a replayed sequence often exhibit some degree of variability across different oscillation phases. Typically, this initial interval variability quickly disappears as the replay progresses, such that the intervals between later sequence elements are largely invariant to the oscillation phase at replay onset (Figure 3.7). This fading of across-trial interval variability during replay should be observable in neural recordings of replay events, and could be tested in psychophysics experiments. Indeed, psychophysics experiments investigating the precision of time estimation are in line with our prediction: in general, the across-trial variability (standard deviation) of estimated time grows linearly with the length of the time interval to be

estimated. This finding is known as "Weber's law" (Gibbon, 1977; Hass & Durstewitz, 2016), and can be explained by an accumulation of noise across successive assembly transitions during replay. At short and at long time intervals, however, the variability of time estimation is often disproportionately large, leading to deviations from Weber's law (Bizo et al., 2006). The larger variability at short intervals could be explained by the phase-dependent variability of the initial replay intervals, which dominates when the assembly chain encoding the estimated time interval is short.

3.3.5.6 *Oscillations enhance robustness against noise*

Oscillatory input stabilizes replay by periodically aligning neuronal excitability (Figure 3.8). Replay during strong coherent oscillations should therefore be more temporally precise and less susceptible to noise, whereas low-amplitude oscillatory states should exhibit higher variability and failure rates. This prediction could be tested using analyses similar to those in (Denker et al., 2011). The prediction is consistent with experimental findings showing that temporally structured inputs increase spike-time reliability (Mainen & Sejnowski, 1995) and that slow cortical rhythms provide a temporal reference frame that stabilizes neural representations (Kayser et al., 2009). The noise-reducing effect of oscillations should moreover have an influence on the variability of time estimation: the slope of Weber's law (see above) should depend on the characteristics of background oscillations.

3.3.5.7 *Timing sensitive replay relies on stable oscillations*

Oscillations in cortical activity are often transient and exhibit high levels of variability in frequency, amplitude, and phase (Burns et al., 2011; Xing et al., 2012). The mechanism proposed here suggests that during tasks where timing matters, the global oscillations serving as a clock signal must be stable over the timescale of the sequence to be replayed (in the relevant cortex area).

3.4 DISCUSSION

This chapter extends the spiking Temporal Memory model to address two key aspects of sequence processing: the learning and representation of sequence element durations and inter-element time intervals, and the control of replay speed after learning. The results show that the temporal structure of sequences can be encoded in the sTM model by a sequential activation of context-specific neuronal assemblies, such that elapsed time corresponds to a progression along an assembly chain. This permits a discretized representation of time that preserves sparse, context-dependent coding, and supports unsupervised learning of sequences with complex temporal structure via local synaptic plasticity mechanisms. In addition, oscillatory background input provides a robust clock signal by which the replay speed can be flexibly modulated, while constant background input requires fine tuning and is less robust. This establishes a functional separation between the representation of the temporal structure of a sequence, encoded in the network connectivity, and the temporal scale of autonomous replay, controlled by global background input. The results further reveal constraints on the replay dynamics imposed

by neurophysiological parameters: spike transmission delays, membrane time constants, and synaptic time constants limit the maximum replay speed, whereas the duration of plateau potentials caused by dendritic action potentials sets a lower bound.

The timing mechanisms employed in this study are not entirely new. A large number of models of time encoding and perception exist in computational neuroscience, and an overview is provided in (Hass & Durstewitz, 2016), which classifies existing models into four categories: ramping activity models, delay-line models, state-dependent timing models, and neural oscillator models. The model presented in this chapter falls into the second category: a delay-line model in which time intervals are encoded by the position of activity along a sequence of neuronal groups. One of the key advantages of delay-line models is their ability to store temporal structure within the connectivity of the network, making the representation of time very robust and independent of fine tuning of parameters such as synaptic weights or time constants (Roxin et al., 2004; Vogels & Abbott, 2005; Lundqvist et al., 2006; Gavornik & Shouval, 2011; Kriener et al., 2014). A drawback is that longer time intervals require more neurons. As shown in (Hass et al., 2008), delay-line models can explain psychophysical results obtained from humans such as the Weber law and deviations from it at short and long time intervals. Delay-line models are therefore plausible candidates for the storage of temporal information in biological neuronal circuits. The model proposed in this study extends the basic delay-line model by combining it with a neural oscillator model, which provides a mechanism for controlling the replay speed after learning without the need to relearn the sequence representation. Moreover, the model permits the storage of timing and content information within the same neuronal circuit, and does not require a separate timing network as in some other approaches (Maes et al., 2020; Calderon et al., 2022; Liang et al., 2020).

A number of earlier studies propose that sequences can be represented by transitions between attractor states in recurrent neuronal networks, where individual sequence elements are stored within clusters of neurons in an auto-associative manner and hetero-associative connections implement transitions between attractors (Palm, 1982; Horn & Usher, 1989; Wennekers & Palm, 2009; Lansner et al., 2013; Martinez et al., 2019; Pereira & Brunel, 2020; Cone & Shouval, 2021). One of the challenges in these models is to explain how transitions between attractor states can be initiated and how their timing can be controlled. Some studies suggest that transitions could be triggered by well-timed, unspecific external signals (Wennekers & Palm, 2009), leaving open the question of how the timing of these signals is itself controlled. In the present model, an unspecific oscillatory background input is similarly used to trigger transitions between sequence elements during replay, but the relative time intervals between elements are stored within the network structure rather than being determined by the external signal. Other studies employ intrinsic dynamical mechanisms such as short-term synaptic plasticity, adaptation, or transient network dynamics to trigger attractor transitions (Lansner et al., 2013; Martinez et al., 2019; Pereira & Brunel, 2020; Cone & Shouval, 2021). With these approaches, however, precise control of timing typically requires fine tuning of parameters such as synaptic weights or time constants (Roxin et al., 2004; Vogels

& Abbott, 2005; Lundqvist et al., 2006; Gavornik & Shouval, 2011; Kriener et al., 2014). Storing temporal structure in the network connectivity instead, as in the sTM model, allows for a much more robust representation of time.

The model proposed in this study suggests a trade-off between the number of neurons required to represent a given time interval and the flexibility in modulating the replay speed. The total time interval represented by an assembly chain is determined by the number of assemblies and the encoding interval, that is, the time interval between successive assemblies during training. A given time interval could be realized by a chain composed of many assemblies with short encoding intervals, or by a short chain with longer encoding intervals. Shorter chains require fewer neurons, but if the encoding interval is large and close to the maximum possible interval, the sequence cannot be slowed down during replay. If the encoding interval is close to its minimum, determined by synaptic and membrane time constants, the sequence cannot be sped up. Maximum flexibility in replay speed modulation is therefore achieved when the encoding interval is in the intermediate range, at approximately half the duration of the dAP plateau potentials. A related trade-off has been discussed in the context of other delay-line models (Hass et al., 2008).

For the variant of the model presented in this study, long time intervals between successive sequence elements are learned by repeated activation of the minicolumn corresponding to the previous sequence element, recruiting additional assemblies within the same minicolumn. As an alternative, the temporal gap between consecutive sequence elements could be filled by recruiting assemblies in other minicolumns encoding silence. A combination of these two encoding strategies may allow the model to capture more complex temporal structures, such as sequences containing both sustained elements and pauses.

Representing time through sequential state transitions naturally relates to time cells observed in hippocampus, striatum, and cortex (Eichenbaum, 2014; Tiganj et al., 2017), where elapsed time is similarly encoded by a progression of neuronal activity patterns. In the present framework, however, longer intervals require more transitions, increasing resource usage and allowing noise to accumulate (Hass et al., 2008). This may contribute to a reduced temporal precision for longer durations, consistent with experimental observations (Tiganj et al., 2017; Cao et al., 2022). This accumulation of variability across successive transitions has been shown to give rise to Weber's law, that is, a proportional increase of timing variability with interval duration (Hass et al., 2008). The present framework suggests that this scaling may depend on the properties of the background input, and investigating how different background input regimes modulate this scaling is a promising direction for future work.

The model presented in this study provides a potential link to replay phenomena observed during sleep. Replay events in hippocampus and cortex are often temporally compressed and occur within transient windows of synchronized network activity (Ji & Wilson, 2007; Euston et al., 2007; Carr et al., 2012; Buzsáki, 2015; Klinzing et al., 2019). In the present framework, similar fast and phase-dependent replay emerges under low-frequency oscillatory input, where successful propagation depends on the timing of initiation relative to the oscillation cycle. This regime limits the number of sequence elements that can be traversed and may lead to

fragmented replay, consistent with experimental observations (Eliav et al., 2025). At the same time, temporal compression during replay may facilitate synaptic plasticity by bringing successive assemblies closer together in time, potentially enabling the formation of shortcut connections between non-adjacent elements (Gupta et al., 2010; Pfeiffer & Foster, 2013).

Part III

EFFICIENT SIMULATION OF LARGE-SCALE NETWORKS

4

EXPLOITING NETWORK TOPOLOGY IN BRAIN-SCALE NETWORK SIMULATIONS

The chapter is based on the following publication:

Exploiting network topology in brain-scale simulations of spiking neural networks. M. Lober, M. Diesmann, and S. Kunkel (2026) Neuromorphic Computing and Engineering, DOI: 10.1088/2634-4386/ae762e.

Author contributions:
Under the supervision of Prof. Dr. Markus Diesmann (MD) and Dr. Susanne Kunkel (SK), the author, MD, and SK jointly conceptualized the work presented in this chapter. The author implemented the proposed approach in the NEST simulation engine, developed the benchmark model, carried out all numerical experiments, and performed the analysis and visualization of the results. Throughout the entire process, the work was refined in ongoing discussions with MD and SK.

4.1 INTRODUCTION

As neuroscience uncovers the structure of the mammalian brain, the field progressively demands the construction and simulation of larger neuronal network models. The local cortical circuit exhibits a high degree of conservation across mammalian species, yet from mice to humans the volume of the brain increased by three orders of magnitude, suggesting that larger brains support increased cognitive abilities and more flexible behaviors (Benson-Amram et al., 2016; Herculano-Houzel, 2017). Since the rise of large-scale artificial neural networks, which provide independent evidence that large-scale networks have qualitatively different capabilities (Berti et al., 2025), the criticism that large brain models are too technically demanding has further faded. Beyond the functional and downscaling considerations already discussed in the introduction of this thesis, the mammalian cortex decomposes into around a hundred areas connected by long-range cortico-cortical projections, and certain species additionally exhibit a patchy superstructure establishing connections between more distant neurons (Voges et al., 2010). This multi-level architecture, combined with the spatio-temporal sparsity of cortical activity (Barth & Poulet, 2012), imposes further constraints on meaningful downscaling and motivates the simulation of cortical networks at their full biological scale. A key structural feature of large-scale cortical models that is directly relevant to their efficient simulation is the separation of temporal scales between local and long-range connections. Experimental measurements report a mean axonal conduction velocity of approximately 3.5 m/s for projections between visual areas in macaque monkey (Girard et al., 2001), and given that inter-area distances can reach several tens of millimeters (Schmidt et al., 2018a), inter-area transmission delays are on the order of several milliseconds — substantially larger than the shortest local delays. This systematic separation of temporal scales provides the key motivation for the partitioning and communication scheme introduced in this chapter.

The dominant performance bottleneck of distributed spiking network simulations has shifted repeatedly as simulation technology evolved. Early implementations caused spike communication to dominate runtime. The introduction of directed spike communication substantially reduced this cost and shifted the bottleneck to spike delivery (Jordan et al., 2018), and subsequent optimizations targeting spike delivery (Pronold et al., 2022a) again shifted the balance, rendering communication and in particular synchronization overhead the limiting factor (Fernandez-Musoles et al., 2019). This history of shifting bottlenecks reflects the steady progress of the field, but also reveals that each resolved bottleneck exposes the next, and that communication efficiency will remain a central challenge as network models continue to grow in size and structural complexity.

The present chapter addresses this challenge by revisiting an idea with a long history in computational neuroscience: mapping the structural organization of the neural network onto the structure of the computing system. Earlier work on structure-aware simulation has demonstrated that exploiting the modular organization of brain models and the temporal structure imposed by transmission delays can yield substantial improvements in simulation efficiency (Hammarlund & Ekeberg, 1998; Goddard et al., 2001; Kozloski & Wagner, 2011). Unlike graph partitioning ap-

proaches, which aim to uncover the hidden structure of an instantiated graph, these works exploit the fact that the modular organization of brain models is known a priori from the underlying biology and is directly reflected in the instantiation rules used during network construction. The present work adopts the same perspective, and extends it by exploiting not only the modular organization of the network but also the temporal structure imposed by transmission delays to reduce the frequency of communication.

It is only now that network models have become large enough to expose the non-local hierarchical architecture of the brain. The combination of these two factors, powerful compute nodes and large structured networks, entices us to revisit the idea of effectively mapping the structure of the neuronal network onto the structure of the computing system. The idea is not to derive an algorithm for the most general connectivity graph, but a scheme that provides advantage for the brain models presently under study.

Distributed large-scale spiking network simulations on HPC systems use the MPI standard (Message Passing Interface Forum, 2021), which enables multiple independent processes to communicate across compute nodes in a distributed-memory environment. Each MPI process (independent parallel worker with separate memory space) is responsible for a subset of neurons, allowing the simulation to scale beyond the resources of a single compute node. Each MPI process employs multithreading to harness all cores of a compute node efficiently. Such simulations typically advance in discrete time steps and require periodic global exchange of spikes to preserve causality. The global minimum transmission delay d_{min} determines how far the simulation can progress independently on each MPI process before communication is required (Morrison et al., 2005). We refer to this interval with fixed length d_{min} as a *simulation cycle* (Figure 4.3). Each cycle consists of three phases: *deliver*, which applies incoming spikes to local neurons; *update*, which advances neuronal states and generates spikes; and *collocate*, which collects emitted spikes for communication. At the end of each cycle, all processes exchange spikes via a collective MPI operation in the *communicate* phase.

Figure 4.1a reports the strong-scaling performance of all mentioned simulation phases on the example of a multi-area model of macaque visual cortex (MAM; Schmidt et al. (2018b,a)). While simulation phases such as neuronal update and spike delivery scale reasonably well with the number of compute nodes, as expected, the graph reveals spike communication as the primary bottleneck. A common assumption is that the communication cost arises mainly from the volume of data exchanged between nodes or by network latency, i.e. the time until the first byte of a message arrives. However, standard MPI benchmarks (Figure 4.1b) show that both, the pure data transfer time and the intrinsic latency of the collective communication, are substantially lower than the communication time observed in the actual large-scale network simulations. This discrepancy reveals that neither bandwidth limitations nor MPI latency are the primary bottleneck in our case. Instead, we find that the dominant contribution to spike-communication time is synchronization overhead. This originates from the variability across compute nodes in completing the simulation cycle. Already Fernandez-Musoles et al. (2019) point out synchronization as one of the key performance bottlenecks of distributed

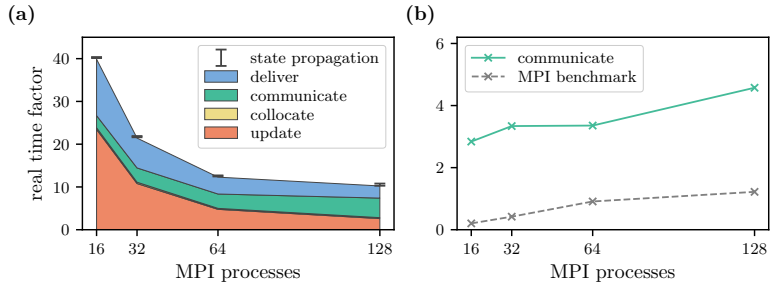

Figure 4.1: **Strong-scaling of the multi-area model of macaque visual cortex (MAM) in the ground state**. (a) Real time factors, defined as wall-clock time normalized by simulated model time, stacked for each phase of the simulation (legend). All simulations cover $T_{\text{model}} = 10\,\text{s}$ of biological time. Error bars (at line resolution) indicate variability across three simulations using different random seeds. (b) Real time factor of communication phase extracted from panel (a) including synchronization. Dashed curve marks time attributed to pure MPI communication, as estimated from MPI benchmarks (see Figure 4.4) for a $T_{\text{model}} = 10\,\text{s}$ simulation and average buffer sizes per target rank of 1408, 837, 514, 317 bytes reported by the simulations in (a) using 16, 32, 64, 128 MPI processes, respectively. All data obtained on SuperMUC-NG using NEST version 3.6.

large-scale spiking neuronal networks. These findings suggest that considerable performance headroom remains untapped, and that optimizing communication schemes in distributed neuronal simulations is both necessary and feasible.

This Chapter presents a novel approach that rethinks neuron distribution and communication between compute nodes. By reducing synchronization overhead, we achieve a substantial reduction in communication time. Furthermore, we show that these improvements extend beyond communication itself, also accelerating subsequent phases such as spike delivery.

4.2 RESULTS

On conventional hardware, simulations of large-scale recurrent spiking networks with bio-plausible synaptic densities and transmission delays are bottlenecked by spike routing (Figure 4.1). The routing in such distributed simulations requires, first, the exchange of spikes among all compute nodes and, second, the local delivery of spikes to their target synapses and neurons. In the case that spikes are exchanged using blocking collective MPI communication, each MPI call introduces a synchronization barrier. This entails waiting times due to variations in processing times between MPI processes. To address this bottleneck, we introduce a structure-aware communication scheme that exploits the separation of delay scales between intra- and inter-area connections. By decoupling fast local interactions from slower long-range communication, the proposed approach reduces synchronization frequency without affecting the temporal resolution of spikes. Spike delivery is inherently irregular: it is unpredictable which neurons emit spikes in any given simulation cycle, and the local target synapses of any two neurons typically reside in different

memory locations. The local target synapses of a single neuron, however, are laid out consecutively in memory, allowing a spike to reach several targets in a row. Previous work established that cache efficiency in the spike-delivery phase depends critically on the number of target synapses each neuron has on the local process and thread: the more local targets, the better the cache reuse (Pronold et al., 2022a,b). When scaling up simulations, targets are increasingly spread out, and each spike finds ever fewer local targets per process and thread. As a consequence target synapses are increasingly likely to lie in uncached memory, resulting in irregular memory access. To recover cache-efficiency, the new structure-aware scheme aggregates intra-area synapses on a single compute node, thereby increasing target locality.

This chapter investigates to what extent a structure-aware approach alleviates these bottlenecks in simulations of multi-area models. Section 4.2.1 introduces the proposed simulation strategy. Section 4.2.2 and Section 4.2.2.1 develop theoretical models of why the structure-aware approach reduces synchronization times and spike-delivery times, respectively. These models rest on simplifying assumptions that are not fully satisfied in practice and should therefore be understood as providing qualitative mechanistic insight into the expected performance gains rather than exact quantitative predictions. Finally, Section 4.2.3 presents benchmarking results obtained with our reference simulation code NEST comparing the conventional and the structure-aware simulation strategy.

4.2.1 *Structure-aware Simulation Strategy*

Conventionally, simulation codes balance computational workload by distributing neurons and their incoming synapses across MPI processes and threads in a round-robin fashion disregarding neuronal populations (Morrison et al., 2005; Migliore et al., 2006). For the specific case of a multi-area simulation, this structure-agnostic approach entails that the short transmission delays within areas enforce frequent global MPI communication in order to maintain causality (Figure 4.2). The structure-aware distribution strategy, in contrast, maps areas to MPI processes. This allows for a reduction in global MPI communication taking advantage of the minimum inter-area transmission delay $d_{\text{min}}^{\text{inter}}$ being significantly longer than the overall minimum delay d_{min}. Henceforth, we constrain inter-area delays in such a way that $d_{\text{min}}^{\text{inter}}$ is a multiple of d_{min} and let

$$D = \frac{d_{\text{min}}^{\text{inter}}}{d_{\text{min}}} \tag{4.1}$$

define the integer ratio between these two minimum delays.

4.2.1.1 *Simulation Flow*

To evolve a model network for biological time T_{model}, the simulation needs to go through $S = T_{\text{model}}/d_{\text{min}}$ simulation cycles (Figure 4.3), where we assume T_{model} to be a multiple of d_{min}. Each simulation cycle consists of three consecutive phases:

DELIVER Each MPI process delivers spikes from its receive buffer to the corresponding local target neurons.

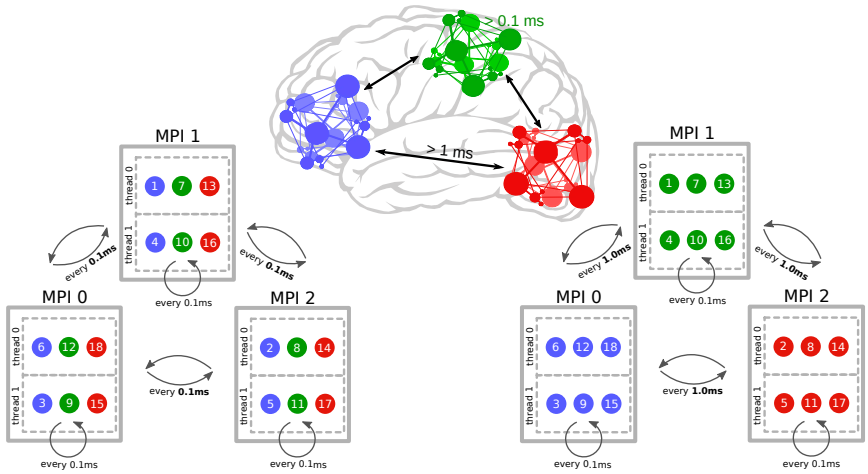

Figure 4.2: **Conventional and structure-aware simulation scheme illustrated for an example multi-area model.** Neurons of three areas (middle) are color coded in blue, red, and green. Transmission delays are significantly shorter within areas than between areas; here, the minimum delays are 0.1 ms and 1.0 ms, respectively. The minimum delay between any pair of neurons represented on two different MPI processes dictates the communication interval for the exchange of spikes between the processes. In conventional simulation technology, neurons are distributed across MPI processes and threads according to a round-robin scheme (left) to balance workload. Therefore, network structure can not be exploited, and global MPI communication is required every 0.1 ms. The structure-aware distribution scheme (right) maps areas to MPI processes, increasing the required interval for global communication to 1.0 ms. Brain outline in background by Gordon Johnson via Pixabay.

UPDATE Each MPI process propagates the state of its local neurons, which includes the detection of threshold crossings and where applicable, the buffering of emitted spikes in the process-local spike register (Plesser et al., 2007). The temporary buffering enables the operation of threads free of barriers (see Figure SI1 in Vogelsang et al. (2026)).

COLLOCATE Each MPI process transfers newly emitted spikes from the local spike register to the MPI send buffer and prepares them for communication with other processes.

In conventional simulation technology, all MPI processes globally communicate spikes after every simulation cycle s. In case of a blocking collective MPI communication, this requires all MPI processes to synchronize. The structure-aware simulation strategy, however, only requires global communication after every D-th cycle, and a local exchange of spikes suffices for all other cycles. Batching long-range spikes over D cycles does not affect the temporal resolution of spike transmission or the resulting network dynamics. Each spike is placed into the neuron-local ring buffer (Section 4.3.1) of its target neuron with the correct remaining transmission delay upon arrival, ensuring that it contributes to state propagation at precisely the

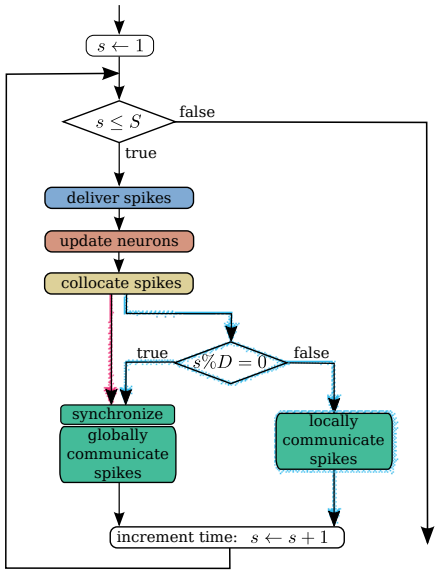

Figure 4.3: **Conventional and structure-aware simulation flow.** Flow chart of the iteration over S simulation cycles highlighting differences between the conventional simulation strategy (pink shaded arrow) and the structure-aware strategy (cyan shaded arrows). For every simulation cycle, each MPI process first delivers incoming spikes from the MPI receive buffer to their local target neurons (blue box). Second, each process updates all process-local neurons (red box). And third, it collocates new spikes in the MPI send buffer (yellow box). In the conventional simulation scheme, every simulation cycle terminates with a global MPI communication of spikes (left green box). In the structure-aware scheme, most cycles terminate with a process-local exchange of spikes (right green box), and only every D-th cycle terminates with a global MPI communication.

correct point in simulated time regardless of when it was communicated between processes. The local communication is a technicality left to the MPI library; it needs to efficiently swap local send and receive buffer. In the following, communication comprises both data exchange and synchronization overhead unless stated otherwise. For both simulation strategies, after every spike exchange, a new simulation cycle starts, and each process delivers incoming spikes from the MPI receive buffer to their local target neurons.

4.2.1.2 *Communication Cost and Scaling*

Communicating spikes only every D-th cycle in the structure-aware strategy implies communicating D-times more data at once. However, collective MPI communication benchmarks on SuperMUC-NG (Figure 4.4) demonstrate that the time required for a single `MPI_Alltoall()` operation scales sublinearly with the message size in the relevant range. As a result, aggregating spikes over multiple simulation cycles and sending them in a single communication step is expected to lead to a net reduction in communication time despite the increased data volume per call. For example, for

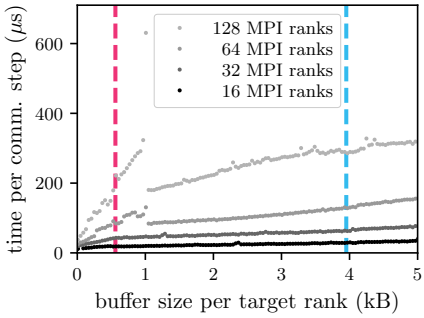

Figure 4.4: **MPI collective performance for increasing message sizes.** Time required for a single `MPI_Alltoall()` call (average over 1000 calls) as a function of buffer size when using the OpenMPI library on SuperMUC-NG, shown for increasing numbers of MPI processes as indicated in legend. Dashed vertical lines indicate the typical buffer sizes per target rank for the MAM-benchmark (see Section 4.3.2), with $\sim 130,000$ neurons per MPI rank, simulated with the conventional (pink) or structure-aware (cyan) strategy.

128 MPI processes and assuming a delay ratio of $D = 10$, the benchmarks predict a reduction in data-exchange time by 86 %. The distinct jumps occurring for 64 and 128 MPI processes likely reflect switches within the OpenMPI library (Gabriel et al., 2004) between different collective communication algorithms.

Besides reducing global MPI communication, the structure-aware distribution scheme aggregates all synapses between neurons of the same area on a single MPI process and thereby effectively increases the average number of local target synapses per neuron, which enables a more cache-efficient delivery of spikes.

Section 4.3.1 describes implementation details of our reference code NEST and the necessary changes to the code enabling the structure-aware simulation strategy. Section 4.2.3 provides benchmarking results of multi-area model simulations with NEST for both the conventional and the structure-aware strategy, in particular also the contributions of deliver, update, collocate, communicate, and synchronize to the overall runtime. Let us now explore the stochastic nature of the synchronization problem.

4.2.2 *Theoretical Analysis of Synchronization*

The structure-aware simulation strategy reduces the number of required global MPI communications in distributed simulations of multi-area models by a factor of D (see Equation 4.1 for definition). Correspondingly, in case the collective communication is blocking, MPI processes also synchronize D times less often. This reduction in the number of required synchronizations potentially decreases overall synchronization time and thus runtime (Figure 4.5), which in the following we explain with the help of a theoretical model. For all $m \in M$ MPI processes and $s \in S$ simulation cycles,

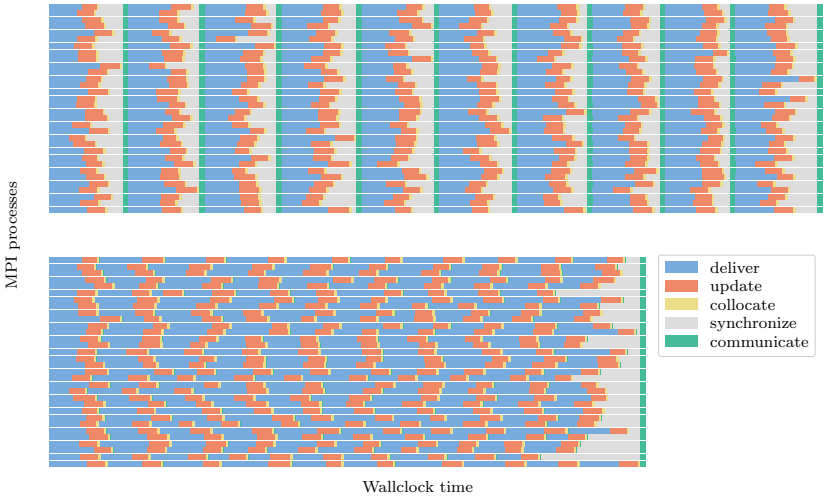

Figure 4.5: **Graphical intuition of the predicted reduction in overall synchronization time and thus overall runtime in a multi-area model simulation.** Illustration of $S = 10$ simulation cycles on $M = 32$ MPI processes when using the structure-aware simulation strategy (bottom) instead of the conventional strategy (top). All timing data is artificial and generated for illustration purposes. Wallclock time spent per simulation cycle is color coded: deliver (blue bars), update (red bars), and collocate (yellow bars). The exchange of spike data between MPI processes (communicate) is assumed to be taking up minimal time (green bars). The conventional strategy requires global communication after every simulation cycle. In case of collective blocking MPI communication, this entails synchronization (gray bars), which means that the process taking longest for the cycle requires all other processes to wait. The structure-aware strategy requires the same wallclock times per simulation cycle but reduces the number of global communications by a factor of D (Equation 4.1); here we assume $D = 10$. This allows for the 10 cycles to be simulated without intermittent synchronization and thus levels out variations such that the overall synchronization times are lower than the sums of the corresponding per-cycle synchronization times in the conventional case.

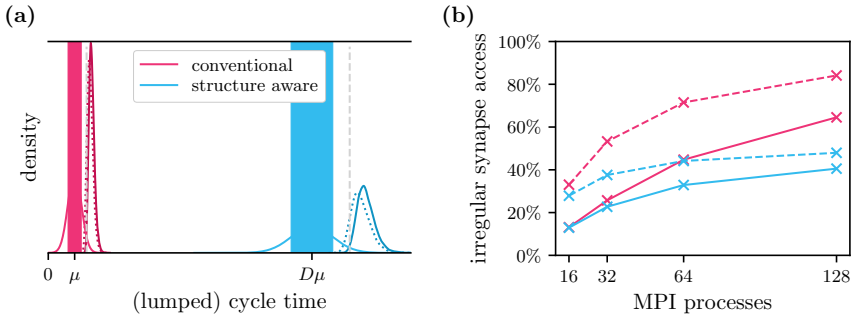

Figure 4.6: **Theoretical analysis explaining the advantage of the structure-aware strategy over the conventional strategy in simulations of multi-area models.** (a) Probability distributions underlying the theoretical model of synchronization times. Distributions of cycle times for the conventional strategy (pink curve) and lumped cycle times for the structure-aware strategy (cyan curve) with standard deviations indicated by light pink/cyan bands, as well as respective distributions of maxima per (lumped) cycle assuming $M = 128$ (solid dark pink/cyan) and $M = 64$ (dotted) MPI processes. Dashed vertical lines indicate the upper 3.5 % quantile of the cycle time distributions for $M = 128$ processes, respectively. (b) Predicted fraction of irregular memory access to synapses in the spike-delivery phase in simulations of multi-area models as a function of number of MPI processes for the conventional (pink) and the structure-aware strategy (cyan) and for $T_\mathrm{M} = 48$ (solid lines) and $T_\mathrm{M} = 128$ (dashed lines) threads per MPI process. Weak-scaling scenario assuming $N_M \approx 130,000$ neurons per process and $K_N \approx 6000$ synapses per neuron. Equally sized areas of N_M neurons and equal amounts of $K_N^\mathrm{intra} = K_N^\mathrm{inter} \approx 3000$ intra-area and inter-area synapses are assumed for the structure-aware case.

we assume that cycle times are governed by a normal distribution (Figure 4.6a) with mean μ and standard deviation σ:

$$t_{m,s} \sim \mathcal{N}\left(\mu, \sigma^2\right). \tag{4.2}$$

The conventional simulation strategy requires synchronization after every cycle. Disregarding data exchange, the total runtime therefore amounts to the sum of maximum cycle times over all S cycles:

$$T_\mathrm{wall}^\mathrm{conv} = \sum_{s=1}^{S} \max\left\{t_{m,s}\right\}_{m=1}^{M}. \tag{4.3}$$

Approximations of order statistics and more specifically expected maxima for different sample sizes are available in the literature, for example, Blom (1958) introduces an approximation for small sample sizes. With increasing sample size drawing values from the tails of the normal distribution becomes more likely, and hence the expected maximum increases. Applied to the runtime estimate of Equation 4.3, this means that with increasing number of MPI processes M, we expect larger maximum cycle times (Figure 4.6a) and thus an increase in overall runtime.

When using a structure-aware simulation strategy, each MPI process goes through D cycles and only after every D-th cycle all processes synchronize. The overall runtime amounts to

$$T_{\text{wall}}^{\text{struc}} = \sum_{l=1}^{S/D} \max \left\{ t_{m,l} \right\}_{m=1}^{M}, \tag{4.4}$$

where

$$t_{m,l} = \sum_{s=(l-1)D+1}^{lD} t_{m,s}. \tag{4.5}$$

denotes the lumped cycle times. Throughout this study, we use the terms *cycle times* and *lumped cycle times* interchangeably in the context of the structure-aware strategy to refer to $t_{m,l}$. According to the central limit theorem (CLT), mean and variance of the initial normal distribution (Equation 4.2) are scaled by a factor of D such that

$$t_{m,l} \sim \mathcal{N}\left(D\mu, D\sigma^2\right). \tag{4.6}$$

The CLT argument applies only under the independence assumption which Section 4.2.3.1 shows to be violated due to persistent serial correlations in per-process cycle times. Under this independence assumption, the lumped cycle times are expected to be D times larger than the original cycle times but only with a $\sqrt{D}$ times larger standard deviation (Figure 4.6a). The ratio of the coefficients of variation of the two distributions

$$\frac{CV^{\text{struc}}}{CV^{\text{conv}}} = \frac{1}{\sqrt{D}} \tag{4.7}$$

quantifies that the structure-aware strategy reduces relative dispersion from the mean by $1/\sqrt{D}$. This directly translates into an expected decrease in overall synchronization time by this factor in the following way. The expected overall runtimes can be expressed as

$$\begin{aligned} \mathbb{E}[T_{\text{wall}}^{\text{conv}}] &= \sum_{s=1}^{S} \mathbb{E}[\max \left\{ t_{m,s} \right\}_{m=1}^{M}] \\ &= \sum_{s=1}^{S} \mu + \xi_M \sigma \\ &= S\mu + S\xi_M \sigma, \end{aligned} \tag{4.8}$$

in the conventional case and

$$\begin{aligned} \mathbb{E}[T_{\text{wall}}^{\text{struc}}] &= \sum_{l=1}^{S/D} \mathbb{E}[\max \left\{ t_{m,l} \right\}_{m=1}^{M}] \\ &= \sum_{l=1}^{S/D} D\mu + \xi_M \sqrt{D}\sigma \\ &= S\mu + S\xi_M \frac{1}{\sqrt{D}}\sigma \end{aligned} \tag{4.9}$$

in the structure-aware case. In both cases, the expected maximum is defined by the factor ξ_M, which indicates how many standard deviations away from the mean it is located. The factor ξ_M depends on the number of MPI processes M and can be approximated (e.g., Blom (1958)). The left summand $S\mu$ in Equation 4.8 and Equation 4.9 corresponds to the expected contributions of deliver, update, and collocate to the overall runtime. These contributions cancel out in the difference of the expected runtimes leaving the right summands:

$$\mathbb{E}[T^{\text{conv}}_{\text{wall}}] - \mathbb{E}[T^{\text{struc}}_{\text{wall}}] = S\xi_M\sigma - S\xi_M\frac{1}{\sqrt{D}}\sigma, \tag{4.10}$$

which correspond to the contribution of synchronize to the overall runtimes. The ratio of the expected overall synchronization times is just

$$\begin{aligned}\frac{\mathbb{E}[T^{\text{struc}}_{\text{synch}}]}{\mathbb{E}[T^{\text{conv}}_{\text{synch}}]} &= \frac{S\xi_M\frac{1}{\sqrt{D}}\sigma}{S\xi_M\sigma} \\ &= \frac{1}{\sqrt{D}}.\end{aligned} \tag{4.11}$$

This theoretical prediction provides a first idea of the savings in synchronization time achievable with a structure-aware simulation strategy. In particular, since $\frac{1}{\sqrt{D}}$ declines rapidly, we conclude that the structure-aware approach is already effective for small ratios D of minimum inter-area transmission delay to overall minimum delay (Equation 4.1), while little more can be gained by further increasing D. This derivation assumes fully independent consecutive cycle times. In the opposite limiting case of full correlation between all D consecutive cycle times of a given MPI process, the lumped cycle time Dt has standard deviation $D\sigma$ (law of functions of a random variable, Papoulis & Pillai (2002)) and thus the same coefficient of variation as the original distribution, yielding a ratio of $\mathbb{E}[T^{\text{struc}}_{\text{synch}}]/\mathbb{E}[T^{\text{conv}}_{\text{synch}}] = 1$.

For concrete cycle-time measurements obtained in benchmarking simulations (Section 4.2.3), it is possible to estimate the expected interval of the per-cycle maxima. To this end, we let $p_{[q,+\infty)}$ define the probability for a cycle time to fall within the interval $[q, +\infty)$ of the cycle-time distribution. For each simulation cycle, the probability of drawing the maximum cycle time from this interval is

$$p^{\max}_{[q,+\infty)} = 1 - \left(1 - p_{[q,+\infty)}\right)^M \tag{4.12}$$

given a specific number of M MPI processes. For a simulation with $M = 128$ MPI processes, this means, for example, that the upper 3.5 % of the cycle-time distribution contribute approximately to the upper 99 % of the resulting distribution of per-cycle maxima. Or vice versa, the upper 99 % of the per-cycle maxima are expected to fall within the interval containing the upper 3.5 % of measured cycle times (Figure 4.6a).

4.2.2.1 *Analysis of Spike Delivery*

Delivering a spike to its first target synapse on a specific MPI process and thread typically entails access to a memory location that is not yet cached, whereas subsequent target synapses are then sequentially accessed. The following theoretical

analysis allows us to predict the fraction of first-synapse accesses in multi-area model simulations for the conventional and the structure-aware strategy.

We consider multi-area networks of N neurons, each neuron with an average number of K_N incoming synapses. We assume that the simulation is carried out using M MPI processes, each running T_M threads, resulting in $T = M\,T_M$ threads in total. $N_M = N/M$ and $N_T = N/T$ denote the process-local and thread-local number of neurons, respectively.

Conventionally, neurons are distributed across processes and threads in a round-robin fashion. The probability that a neuron has at least one target on a specific thread is given by

$$p_{\text{target}} = 1 - (1 - 1/N)^{N_T K_N}\,, \tag{4.13}$$

where all neurons are considered potential source neurons, and hence, $1/N$ is the probability to draw a specific neuron as source. When a neuron emits a spike, it needs to reach on average K_N synapses distributed across $p_{\text{target}}\,T$ threads. Hence, the fraction of irregular memory access to synapses in the spike-delivery phase is just

$$f_{\text{irr}}^{\text{conv}} = \frac{p_{\text{target}}\,T}{K_N} \tag{4.14}$$

for the conventional round-robin distribution scheme.

For the structure-aware strategy, we distinguish between the intra-area synapses, which are all hosted by the same MPI process as the source neuron, and the inter-area synapses, which are hosted by the remaining $M-1$ MPI processes. For simplicity, we assume equally sized areas of N_M neurons and equal amounts of $K_N^{\text{intra}} = K_N^{\text{inter}}$ intra-area and inter-area targets per neuron. The probability that a neuron has at least one intra-area target on a specific thread on the MPI process hosting the area is given by

$$p_{\text{target}}^{\text{intra}} = 1 - (1 - 1/N_M)^{N_T K_N^{\text{intra}}}\,, \tag{4.15}$$

with the probability $1/N_M$ of drawing a specific neuron of the area as source. For inter-area targets, the probability

$$p_{\text{target}}^{\text{inter}} = 1 - (1 - 1/(N - N_M))^{N_T K_N^{\text{inter}}}\,, \tag{4.16}$$

considers the remaining $N - N_M$ neurons as potential sources. The fraction of irregular memory access to the intra-area synapses distributed across $p_{\text{target}}^{\text{intra}}\,T_M$ threads of the MPI process hosting the area amounts to $p_{\text{target}}^{\text{intra}}\,T_M/K_N^{\text{intra}}$. For the inter-area synapses distributed across the $p_{\text{target}}^{\text{inter}}\,T_M(M-1)$ threads, the fraction of irregular memory access is $p_{\text{target}}^{\text{inter}}T_M(M-1)/K_N^{\text{inter}}$. Combining the two yields the fraction of irregular memory access to synapses in the spike-delivery phase

$$f_{\text{irr}}^{\text{struc}} = \frac{p_{\text{target}}^{\text{intra}}\,T_M + p_{\text{target}}^{\text{inter}}\,T_M(M-1)}{K_N} \tag{4.17}$$

for the structure-aware distribution scheme.

A comparison of the fractions of irregular access predicted for the conventional round-robin distribution scheme and the structure-aware scheme in a weak-scaling scenario reveals an advantage of the structure-aware strategy, which grows with the number of MPI processes and threads (Figure 4.6b). At $M = 16$ MPI processes, the fraction of irregular access to synapses in the spike-delivery phase is still similar for both strategies. However, at $M = 32$ MPI processes the structure-aware strategy shows a reduction in irregular access by 12 % for $T_M = 48$ threads and 29 % for $T_M = 128$ threads compared to the conventional strategy. At $M = 128$ processes, the theory predicts an even larger reduction by 37 % for $T_M = 48$ and 43 % for $T_M = 128$.

4.2.3 *Benchmark Results*

The above theoretical analyses predict a gain in performance for the spike routing in distributed simulations of multi-area models. This applies to both the synchronization required for the regular spike exchange between compute nodes (Section 4.2.2) and the subsequent node-local spike delivery (Section 4.2.2.1). To evaluate these predictions, we use two network models throughout the benchmarking experiments: the multi-area model of macaque visual cortex (MAM; Schmidt et al. (2018b,a)) and the MAM-benchmark, a simplified variant of the MAM with homogeneous area sizes and constant firing rates designed for controlled scaling experiments (see Section 4.3.2 for details). To this end, we have incorporated the structure-aware simulation strategy (Section 4.2.1) in our reference simulation code (Section 4.3.1), which allows us to compare conventional and structure-aware scheme in a consistent setting.

4.2.3.1 *Weak-scaling*

First, we analyze the behavior of the two simulation strategies under weak scaling (Figure 4.7a) using the MAM-benchmark, where the minimum inter-area delay of $d_{\text{min}}^{\text{inter}} = 1\text{ms}$ is $D = 10$ (Equation 4.1) times larger than the overall minimum delay of $d_{\text{min}} = 0.1\text{ms}$. The real time factor increases from 9.4 at $M = 16$ MPI processes to 22.7 at $M = 128$ (slope 0.12) in the conventional case but only from 8.5 to 15.7 (slope 0.06) in the structure-aware case. Hence, the structure-aware approach not only achieves an overall reduction in runtime by up to 30 % but also shows a more favorable scaling behavior than the conventional strategy. Consistent with the theoretical analysis, the performance improvement is due to speed ups in the spike-delivery and communication phase, where the latter includes both data exchange and synchronization. We observe the largest gain at $M = 128$ MPI processes, where spike-delivery time reduces by 25 % when using the structure-aware instead of the conventional scheme, data-exchange time reduces by 76 %, and synchronization time reduces by 48 %.

The theoretical analysis of spike delivery predicts 37 % less irregular memory access for a MAM-benchmark simulation with the structure-aware strategy with $M = 128$ MPI processes and $T_M = 48$ threads per process compared to the conventional approach (Section 4.2.2.1). The weak-scaling experiments demonstrate that this does not translate one-to-one into a reduction in spike-delivery time. However,

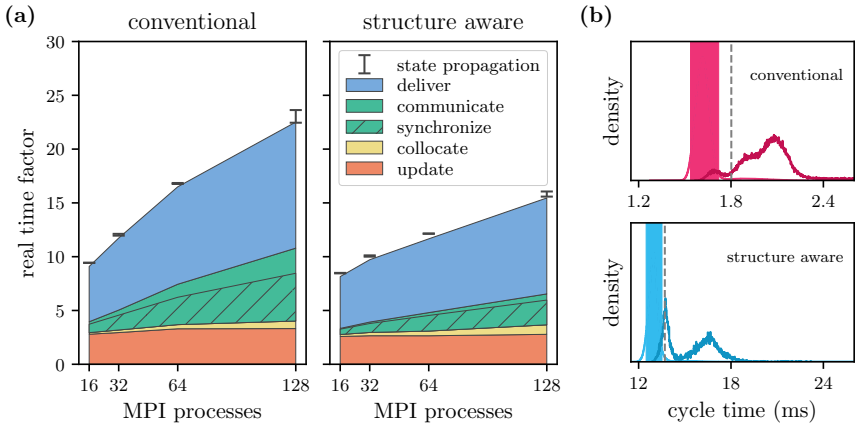

Figure 4.7: **Performance comparison of conventional and structure-aware simulation strategy.** Simulations of the MAM-benchmark with overall minimum delay $d_{\text{min}} = 0.1\text{ms}$ and minimum inter-area delay $d_{\text{min}}^{\text{inter}} = 1\text{ms}$ on SuperMUC-NG using one MPI process and $T_M = 48$ threads per compute node. $N_M = 130,000$ neurons per process with $K_N = 6000$ synapses per neuron are simulated for $T_{\text{model}} = 10\,\text{s}$ biological time. (a) Weak-scaling experiment increasing the number of areas with the number of MPI processes M, such that the total number of areas corresponds to M. Real time factor as a function of number of MPI processes for the conventional (left) and structure-aware (right) case with contributions of different simulation phases (legend). Black error bars indicate variability across three runs. (b) Distribution of cycle times for the conventional strategy (top, pink) and lumped cycle times for the structure-aware strategy (bottom, cyan) for a simulation with 128 MPI processes (seed 654). Arithmetic means are 1.6 ms (conventional) and 13.0 ms (structure-aware); shaded bands indicate the standard deviations. Dashed vertical lines mark the upper 3.5 % quantile each distribution. Lighter curves show the kernel density estimates of all cycle times, darker curves the per-cycle maxima across MPI processes ($D = 10$ times increased horizontal scale for the structure-aware case).

as theoretically predicted (Figure 4.6b), spike delivery increasingly benefits from the structure-aware approach under weak scaling. In the conventional case, the target synapses of each neuron are ever more distributed across MPI processes and threads, until eventually all targets are fully dispersed. Cache efficiency of spike delivery decreases accordingly. In the structure-aware case, on the other hand, only the inter-area target synapses increasingly spread out under weak scaling, while the intra-area targets remain on the same MPI process. This retains cache efficiency for the intra-area spike delivery, which explains the observed performance advantage over the conventional simulation strategy.

The observed reduction in data-exchange time is not immediately intuitive. Although the structure-aware strategy performs global communication $D = 10$ times less often than the conventional approach, each communication operation transmits ten times more data. The MAM-benchmarks are, however, in line with the above `MPI_Alltoall()` benchmarks (Figure 4.4) demonstrating that the combined

latency and data-exchange time increases sublinearly with the message size. Consequently, sending fewer but larger messages as in the structure-aware strategy is more efficient than sending many smaller ones. The achieved 76 % reduction in data-exchange time for the MAM benchmark at $M = 128$ processes is broadly consistent with the 86 % reduction predicted from the MPI benchmark.

The theoretical prediction of a 68 % reduction in synchronization time (Section 4.2.2) also exceeds the measured improvement of 48 %. To explain the deviation from the theory, we inspect the measured cycle times in the simulations of the MAM-benchmark (Figure 4.7b). The cycle time distributions are shown for a single seed, as Figure 4.7a exhibits negligible variability across seeds and each run already yields $\sim 10^5$ samples per process, producing stable statistics. The distribution of measured cycle times in the conventional case is bimodal showing a major-mode peak around 1.62 ms next to a minor-mode peak around 1.90 ms. The longest recorded cycle time is 18.35 ms. In the structure-aware case, we also observe a bimodal distribution with a major-mode peak at 13.0 ms and a minor-mode peak at 16.62 ms; the longest cycle time is 29.66 ms. The main body of the structure-aware distribution is shifted by a factor of approximately 8.1 with respect to the conventional distribution, and the ratio of the respective arithmetic means is 8.0. This value is smaller than the theoretically predicted shift of $D = 10$ (Equation 4.6), which we attribute to the 24 % shorter delivery phase in the structure-aware scheme. Moreover, as predicted, the relative dispersion from the mean is significantly smaller in the structure-aware case ($CV_{\text{struc}} = 0.040$) than in the conventional case ($CV_{\text{conv}} = 0.056$), which accounts for the observed reduction in synchronization time. However, the ratio of the coefficients of variation of the two distributions (0.71) falls between the theoretical bound of $1/\sqrt{D} = 0.32$ derived under the assumption of fully independent consecutive cycle times (Equation 4.7) and the bound of 1.0 derived under the assumption of fully correlated consecutive cycle times. This reflects the presence of partial serial correlations in the cycle times of individual MPI processes. The temporal evolution of per-process cycle times (Figure 8.1 in appendix) reveals that processes are not only systematically faster or slower on average but also exhibit extended phases of elevated or reduced cycle times that may persist over several thousand cycles.

Based on the measured cycle-time distributions, the theoretical analysis predicts that the upper 99 % of the per-cycle maxima are located between 1.80 ms and 18.35 ms in the conventional case and between 13.70 ms and 29.66 ms in the structure-aware case (Equation 4.12). In fact, only the upper 91 % (conventional) and 84 % (structure aware) fall within the predicted interval. The theory assumes the cycle times of individual processes to be fully uncorrelated. However, this is not the case for the measured cycle times (see Figure 8.1 in appendix). MPI processes often concertedly take longer or shorter time to go through specific simulation cycles.

On the other hand, the measured 91 % (conventional) and 84 % (structure aware) of the per-cycle maxima do not stray too far from the theoretical 99 %. This means that the corresponding upper 3.5 % of the measured cycle times do not typically coincide in the same cycles but dominate individual cycles – otherwise, they would not prevail the per-cycle maxima distributions. The upper 3.5 % mainly comprise the minor-modes of the recorded cycle times. These large cycle times are unrelated to changes in spiking activity as the MAM-benchmark does not exhibit different

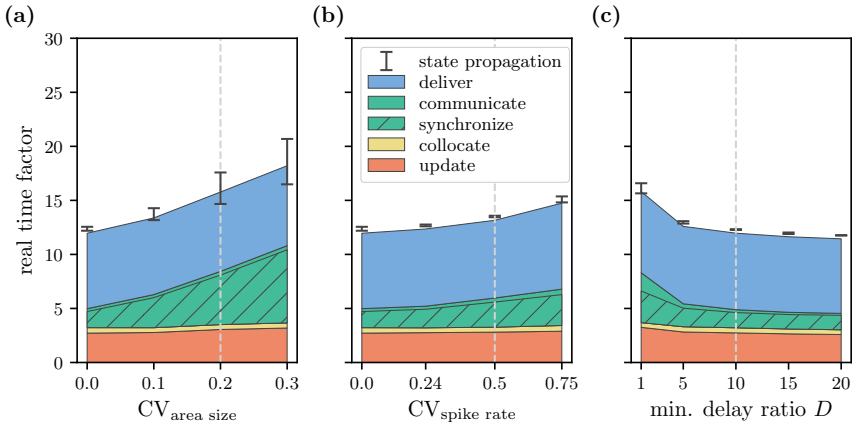

Figure 4.8: **Effect of inter-area variability and delay on the performance of the structure-aware simulation strategy**. Simulation of the MAM-benchmark on SuperMUC-NG using $M = 64$ MPI processes and compute nodes with one area per process. Color scheme, and benchmarking configurations as in Figure 4.7 unless otherwise stated. Real time factor as a function of (a) variability in area size with a fixed mean area size of 130,000 neurons, (b) spike-rate variability across areas with a fixed mean rate of 2.5 spikes per second, and (c) ratio D of minimum inter-area delay to fixed overall minimum delay $d_{\text{min}} = 0.1$ ms. Area sizes and spike rates are drawn from normal distributions with coefficients of variation $\text{CV}_{\text{area size}}$ and $\text{CV}_{\text{spike rate}}$, respectively. Each data point represents the average over three independent sampling seeds. Gray dashed lines indicate the parameter values corresponding to the MAM in Figure 4.9.

activity modes. Unraveling the source of the occasional large cycle times is however not within the scope of this study. Whenever these extremes influence isolated simulation cycles of individual processes, the structure-aware strategy can mitigate their impact on synchronization time.

4.2.3.2 *Effect of Heterogeneity and Delays*

In all of the above simulations, MPI processes handle the same workload in terms of area size and spike rate, not only in the conventional case but also when using the structure-aware strategy. This is due to the homogeneous network configuration. Besides, the minimum delay ratio of $D = 10$ imposes a lower bound of $d_{\text{min}}^{\text{inter}} = 1$ms on the delays between areas given the overall minimum delay of $d_{\text{min}} = 0.1$ms. We now assess the robustness of the structure-aware scheme to heterogeneous conditions. To this end, we analyze the performance of simulations of the MAM-benchmark with a fixed size of 64 areas using $M = 64$ MPI processes and compute nodes for a set of controlled parameter variations (Figure 4.8).

In two separate experiments, we investigate the effect of, firstly, different area sizes, and secondly, different per-area spike rates. Area sizes and spike rates are drawn from normal distributions, respectively, where the standard deviations increase but the means are fixed and correspond to the $M = 64$ case of the weak-

scaling experiments (Figure 4.7a). To interpret the effect on the different simulation phases, we need to consider that in the reference implementation, synchronization time accounts for the variability across MPI processes (Section 4.3.1). If one MPI process is systematically slower than the others in the update, collocate, or deliver phase, for example, due to hosting the largest area, it will throughout the simulation make the other processes wait in the synchronization phase.

With increasing variability in area size, the overall runtime also increases (Figure 4.8a). The individual update times of the MPI processes correlate with the more and more uneven amounts of local neurons. However, as expected, this imbalance is not reflected by an increase in the averaged update time but in the synchronization time. The measured real time factors also increasingly vary across runs. This is because, for larger standard deviations of the underlying normal distribution, the 64 sampled area sizes of a specific simulation run are more likely to misrepresent the distribution.

For the investigated regime of low spike rates with less than 10 spikes per second, a higher spike-rate variability only moderately affects synchronization time and thus overall runtime (Figure 4.8b). We infer from this that the MPI processes hosting the more active areas do not spend significantly more time going through a simulation cycle than the processes hosting the less active areas. This means that detecting the spikes of the process-local neurons and buffering them in the spike register are not the most time-consuming steps of the update phase. The collocate phase contributes little to the overall runtime to begin with, such that an elevated local spike rate on some MPI processes can not introduce a major workload imbalance between processes. For the deliver phase, we need to distinguish between the delivery of intra-area spikes and inter-area spikes to their process-local targets. The MPI processes hosting the more active areas have an increased workload compared with the less active ones in the intra-area spike delivery. Yet, the effect on synchronization time is moderate. Intra-area spike delivery is less critical for performance than inter-area spike delivery as the latter is considerably more demanding in terms of memory access (Section 4.2.2.1). However, as the mean spike rate is fixed in this experiment, the increase in spike-rate variability does not substantially change the total number of spikes that each area receives from the other 63 areas. This limits the impact on synchronization time.

Finally, in a third experiment, we systematically vary the minimum delay ratio D (Equation 4.1) to assess how the performance of the structure-aware scheme depends on the degree of separation between intra- and inter-area delay distributions (Figure 4.8c). As the minimum delay is fixed at $d_{\text{min}} = 0.1\,\text{ms}$, increasing D raises the lower cutoff for inter-area delays to $d_{\text{min}}^{\text{inter}} = D \cdot d_{\text{min}}$. This constrains the heterogeneity of inter-area delays, but it allows us to decrease the frequency of global spike communication (Section 4.2.1). For $D = 1$, global spike exchange occurs after every simulation cycle, matching the communication frequency of the conventional scheme. Even in the absence of any minimum delay separation, the structure-aware scheme still outperforms the conventional one (see Figure 4.7a at 64 MPI processes for reference), as the improved cache efficiency of the structure-aware neuron distribution reduces spike-delivery time regardless of the communication scheme. Communication time decreases rapidly when increasing the ratio to $D = 5$.

This is due to a decrease in both synchronization time and data-exchange time. The gain in performance is less pronounced when further increasing the ratio to $D = 10$, and it becomes negligible for larger values of $D > 10$. The above weak-scaling experiments show that the theoretical analysis (Section 4.2.2) does not allow for exact quantitative predictions of the observed reduction in synchronization time when using the structure-aware simulation strategy instead of the conventional scheme (Section 4.2.3.1). However, the qualitative prediction that the structure-aware scheme is most effective for small minimum delay ratios (Equation 4.11) is confirmed by the rapid decrease in synchronization time for small values of $D < 10$ observed in this experiment. Taken together, these results provide insight into the applicability of the structure-aware approach across different network configurations. The communication benefit requires a meaningful separation of delay scales, whereas the delivery benefit of the structure-aware neuron distribution persists even when no such separation exists.

4.2.3.3 *Real-world Network Model*

The MAM-benchmark allows us to analyze the performance of multi-area model simulations with the conventional strategy and the structure-aware strategy under controlled conditions. It generates stable workload under weak scaling (Section 4.2.3.1) and enables selective changes to the network configuration to systematically study the effects on different simulation phases (Section 4.2.3.2). To evaluate the potential benefits of the structure-aware approach for day-to-day research, we now compare the two simulation strategies for a real-world network model (Figure 4.9) – the multi-area model of macaque visual cortex (MAM). Section 4.3.2 summarizes the properties of both MAM and MAM-benchmark.

We analyze the performance of simulations of the MAM in its ground state on two HPC systems, SuperMUC-NG as well as JURECA-DC (Section 4.3.3), where besides differences in processor type, available memory, and interconnect, JURECA-DC offers more cores per node than SuperMUC-NG. Next to the conventional and the structure-aware simulation strategy, we test a third strategy, which represents an intermediate step between the two. This intermediate strategy distributes neurons across MPI processes in a structure-aware way (Section 4.3.1.1) but carries out conventional global communication every $d_{\text{min}} = 0.1\,\text{ms}$. This exposes whether the observed effects of the structure-aware simulation strategy on different simulation phases are due to the neuron-distribution scheme or the reduction in global communication.

On both systems, the structure-aware neuron distribution alone leads to a reduction in spike-delivery time that even exceeds the improvement predicted by the theoretical model for irregular synapse access (Section 4.2.2.1). This is because the theoretical model assumes a homogeneous connectivity, with equal probabilities for intra- and inter-area connections. In contrast, the MAM is based on experimental connectivity data and exhibits inhomogeneities including a higher proportion of intra-area connections, which amplifies the reduction in irregular synapse access for local connections in structure-aware simulations. The gain in spike-delivery performance, however, is accompanied by an increased synchronization time. As expected from the heterogeneity of the network, there is an unbalanced computa-

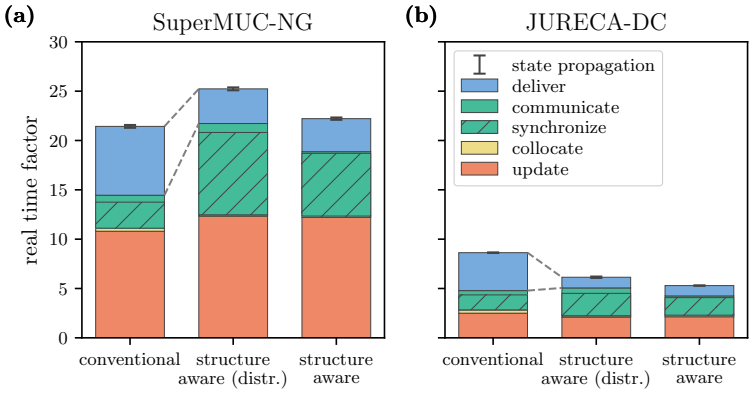

Figure 4.9: **Performance comparison of conventional and structure-aware simulation strategy for a real-world network model on two HPC systems.** Simulations of the multi-area model of macaque visual cortex (MAM) in the ground state using $M = 32$ MPI processes and one process per compute node on (a) SuperMUC-NG with $T_M = 48$ threads per process and (b) JURECA-DC with $T_M = 128$ per process. Color scheme and black error bars for three runs as in Figure 4.7. Left and right bars in each panel show the real time factor for the conventional and the structure-aware strategy, respectively. Middle bars show the real time factors when using a structure-aware distribution scheme for neurons and synapses but conventional global communication every $d_{\text{min}} = 0.1$ ms. Dashed lines highlight the change in spike delivery time from the conventional to the structure-aware neuron distribution scheme.

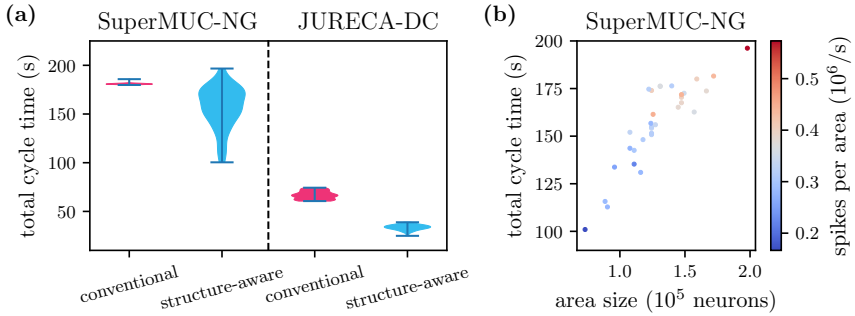

Figure 4.10: **Per-process cycle time analysis for simulations of the MAM underlying the results shown in Figure 4.9.** (a) Violin plots of the distribution of total cycle times across MPI processes for simulations using the conventional (pink) and structure-aware (cyan) strategy on SuperMUC-NG and JURECA-DC. (b) Total cycle time per MPI process as a function of area size for the structure-aware strategy on SuperMUC-NG, with total spike count per area encoded by marker color.

tional load across the MPI processes. This imbalance is not visible in phases such as delivery, update and collocation, where values are averaged over MPI processes, but manifests directly in the longer synchronization phase. The synchronization phase is reduced again when using the fully structure-aware strategy that separates short and long range communication, yet remains larger than that of the conventional approach.

The high synchronization time is consistent with our observations for the structure-aware scheme on the MAM-benchmark (Figure 4.8a), where increasing heterogeneity in area size and spike rate leads to an increase in synchronization. The impact of spike rate heterogeneity is expected to be stronger in the MAM than in the MAM-benchmark (Figure 4.8b), as the latter employs a simplified neuron model that produces spikes at a constant rate without propagating membrane potentials (see Section 4.3.2 for more details). In contrast, the state update of the integrate-and-fire neurons in the MAM is more sensitive to variations in firing activity across areas and over time. As a result, both structural heterogeneity (area size) and dynamical heterogeneity (neuronal activity) contribute to the elevated synchronization time observed for the structure-aware scheme in the biologically more realistic model.

On both machines, differences in neuron count and neuronal activity directly translate into varying cycle times of the MPI processes for the structure-aware simulation strategy. Figure 4.10 provides a direct visualisation of this load imbalance at the per-process level. On SuperMUC-NG the distribution of cycle times across MPI processes is substantially broader for the structure-aware scheme than for the conventional scheme, consistent with the larger synchronization times observed on the system (Figure 4.9a). On JURECA-DC, the distributions of the two schemes are of comparable width, consistent with the similarly small difference in synchronization times observed (Figure 4.9b). The cycle time of each MPI process on SuperMUC-NG correlates with both the size and the total spike count of the area it hosts (Figure 4.10b). Since these two quantities are themselves correlated in the MAM — larger areas naturally produce more spikes in total — the individual contributions of structural and dynamical heterogeneity to the load imbalance cannot be fully disentangled from this data alone. The process hosting the largest area with the highest spike count, which is area V1 in the MAM, exhibits the largest average cycle time, whereas the process hosting the smallest area with the fewest spikes shows the shortest. In the MAM, V1 hosts 53 % more neurons and generates approximately 68 % more spikes than the network-wide averages. On SuperMUC-NG, this translates into a 24 % longer average cycle time for the corresponding MPI process compared to the mean across processes, and a 94 % larger total cycle time for the process hosting the largest area compared to the one hosting the smallest. On JURECA-DC, the same differences in area size and spike count result in only a 7 % longer cycle time and a 54 % larger total cycle time difference between the most and least loaded processes. This indicates that JURECA-DC, with its higher core counts and greater per-node computational capacity, is less sensitive to workload imbalance, leading to smaller synchronization penalties.

Overall, the results obtained on JURECA-DC follow the same qualitative trends as on SuperMUC-NG, but simulations are faster due to the larger number of hardware threads per node and the more recent processors. For all simulation

strategies, the update, delivery, and communication phases are shorter than on SuperMUC-NG. Spike collocation time remains comparable across both systems, as this phase is executed by the master thread and does not benefit from additional threading. In contrast to SuperMUC-NG, the update phase for the simulations using a structure-aware neuron distribution is faster than for the conventional scheme, suggesting that additional architectural differences, such as cache size or memory bandwidth, influence computation. As predicted by the theoretical analysis in Figure 4.6b, the benefit of the structure-aware neuron distribution increases with the number of available hardware threads. Consequently, the reduction in delivery time achieved by the structure-aware strategies is more pronounced on JURECA-DC than on SuperMUC-NG. Together, these effects result in a net speed-up of the fully structure-aware strategy compared to the conventional approach by 42 %.

Overall, the results demonstrate that for a model as unbalanced as the MAM, the fully structure-aware strategy outperforms the conventional approach on high-capacity hardware, while performing comparably when the individual nodes struggle with the computational load of an area.

4.3 METHODS

4.3.1 *Reference Implementation*

The neuronal simulation tool NEST version 3.6 (Villamar et al., 2023) serves as the reference simulation framework for all implementations presented in this chapter. NEST is an open-source software tool for large-scale networks of spiking neurons, designed to efficiently model neuronal dynamics on computing platforms ranging from desktop systems to high-performance supercomputers. Users control network specification and simulation procedure through a high-level Python module PyNEST (Eppler et al., 2009; Zaytsev & Morrison, 2014), while all computationally intensive tasks are executed in a C++ simulation kernel. To exploit multi-core hardware and enable users to run large-scale networks, NEST employs hybrid parallelization, distributing neurons across MPI processes and using multi-threading within each process via OpenMP. The code assigns neurons to threads and MPI processes in a round-robin fashion based on their unique global id (GID) given by order of creation. This distribution scheme ensures that neurons belonging to the same population are evenly distributed across compute nodes. Therefore, the round-robin distribution strategy helps to reduce workload imbalances arising from correlated activity within the populations.

Simulations advance in discrete cycles that consist of three phases: event delivery to target neurons, neuronal state updates, spike collocation. Following these phases, a global spike communication step exchanges spikes between MPI processes to preserve causality (see Figure 4.3). NEST records the wall-clock time of the individual simulation phases using internal high-resolution timers (see NEST documentation[1]

1 https://nest-simulator.readthedocs.io/en/stable/index.html

for further details). For a given simulation cycle s and MPI process i, we define cycle time as

$$T_{s,i} = T_{s,i}^{\text{deliver}} + T_{s,i}^{\text{update}} + T_{s,i}^{\text{collocate}} . \tag{4.18}$$

The measure excludes the time required at the end of each cycle for communication between the nodes. As soon as each MPI process completes its local collocate phase, it enters the communication phase. However, spike exchange can only begin after all MPI processes have reached this point. Consequently, faster processes must wait for the slowest process $i_{\max}$ with the largest cycle time $T_{s,i_{\max}}$, before the actual spike communication can proceed. We refer to this waiting period as the *synchronization* phase and denote its duration for MPI process i by T_i^{synch}. Following synchronization, spikes are exchanged via a collective `MPI_Alltoall()` operation. To distinguish synchronization overhead from the time spent on the actual data exchange, we insert an explicit `MPI_Barrier` immediately in front of the `MPI_Alltoall()` call. The synchronization time is then measured as the average time spent by the MPI processes waiting at this barrier. The combined time spent on the `MPI_Barrier` and the subsequent `MPI_Alltoall()` call is referred to the communication phase with duration T_i^{comm}. Timers for all phases are advanced in each simulation cycle. For analysis, we average the resulting cumulative phase durations across all MPI processes to obtain the total wall-clock time of the simulation. We quantify simulation performance using the real time factor, defined as wall-clock time normalized by simulated model time.

The communication algorithm sends spikes only to those processes that host at least one of their postsynaptic targets. Target tables on the presynaptic side (see Figure 4.11), revealing the MPI processes associated with each neuron's outgoing connections, facilitate this procedure. With the optional *spike compression* feature (NEST 3.0; Albers et al. (2022)) the algorithm sends spikes to target MPI processes only once as opposed to one per target thread. Spikes carry the information of the source neuron identifier and the time step at which the spike occurred within the cycle. Upon arrival at the target process, the spike uses the identifier to look up the corresponding target synapses and neurons via connection and source tables that are presorted by source neuron. If a process generates more spikes than fit into the current spike buffer, all MPI processes resize their buffers accordingly, followed by a secondary communication round. Spikes delivered to their target neurons are placed into a neuron-local ring buffer (a circular data structure used to store time-delayed inputs). Each spike remains in the ring buffer for a duration corresponding to its transmission delay minus the time already elapsed since the source neuron fired. Once this remaining buffering time has expired, the spike is retrieved from the ring buffer and contributes to the state propagation of the target neuron in the update phase of the corresponding simulation cycle.

NEST provides a collection of neuron and synapse models at different levels of biological and computational complexity. Beyond the built-in models, users can extend the simulator by defining custom neuron and synapse models using the domain-specific model description language NESTML (Plotnikov et al., 2016; Linssen et al., 2025) or by implementing new models directly in C++.

4.3.1.1 *Structure-aware Distribution of Workload*

Our reference implementation of the structure-aware distribution of neurons does not modify the round-robin distribution scheme of the reference simulation platform NEST. Instead, we realize all changes required to override the effective neuron placement at the level of the Python user interface. This is possible due to a feature introduced in NEST version 3.0 (Hahne et al., 2021), which represents neuron populations as node collection objects with array-like semantics, including indexing, slicing, and element-wise operations.

For multi-area models with homogeneous neuron parameterization, all neurons can be created en bloc in a single `Create()` call, as opposed to instantiating neurons for each area separately. This yields a single node collection containing all neurons in the model. With detailed knowledge of both the model's area sizes and the way the NEST kernel assigns neurons to MPI processes under the round-robin scheme, areas can then be retrospectively defined such that all neurons belonging to a given area are hosted on one compute node. Figure 4.2 illustrates the effect of this procedure using, as an example, a three-area network instantiated on three MPI processes, each running two threads. A conventional executable model specification expressed in the Python interface of NEST, creates neurons area by area, thereby distributing areas evenly across the available MPI processes by the round-robin scheme. As a result, each process hosts neurons belonging to multiple areas. The structure-aware distribution scheme creates all neurons at once and distributes them evenly across the MPI processes in exactly the same way as in the conventional scheme, resulting in an identical placement of neuron identifiers. However, in a subsequent step, the model specification assigns the neurons on the same MPI process to the same area by constructing a respective node collection. Thus, the present study does not require changes to the Python interface of NEST, but exploits high-level commands for the manipulation of neuron populations and knowledge of how NEST assigns neurons to compute nodes.

For models with heterogeneous area sizes, this procedure may require creating more neurons than in the standard round-robin scheme. Since the underlying distribution mechanism remains unchanged, NEST still assigns the same number of neurons to each MPI process. To ensure that the largest area can be placed entirely on a single process, the total number of neurons must therefore be determined by the maximum area size. As a consequence, MPI processes hosting smaller areas contain additional "ghost neurons", which are explicitly tagged as "frozen" and do not take part in the simulation. The number of synapses is not affected by this procedure.

The structure-aware distribution scheme can be applied using standard NEST without modifications to internal data structures and communication mechanisms. Using NEST version 3.6, we evaluated the performance impact of the resulting load imbalance for simulations of heterogeneous multi-area models on simulation phases other than communication, such as spike delivery. As expected, the tests revealed degradation in the communication phase, but also showed promising results for the other simulation phases. Based on these findings, we concluded that changes to the kernel code targeting spike communication (described in the following section) could further improve overall performance.

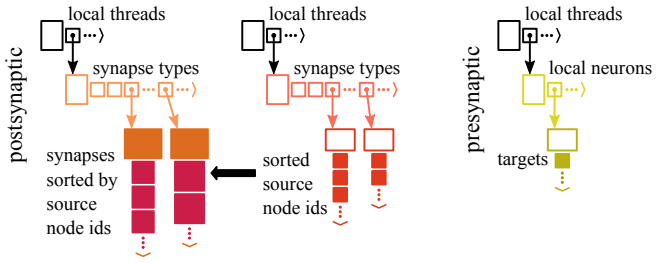

Figure 4.11: **Fundamental data structures of the reference code NEST.** Data structures required per MPI process to store the local connections and enable delivery of spikes to local target connections: connection table (left), source table (middle), and target table (right). The outermost dimension of each table is a resizable array of pointers (black) to thread-specific inner resizable arrays. On the receiver side (postsynaptic), the inner resizable arrays of the connection table and the source table store the thread-local connections (pink filled squares) and the corresponding node ids of the presynaptic neurons (red filled squares) sorted by synapse type. On the sender side (presynaptic), the target table stores the locations of the target connections (green filled squares) for every thread-local neuron. Locations are defined by the MPI ranks and the positions in the connection table. Adapted from Jordan et al. (2018), Figure 4A.

4.3.1.2 *Two Separate Communication Pathways*

To exploit the structure-aware neuron distribution described in the previous section, short- and long-range connections must be explicitly distinguished during the first two main phases of the simulation, network construction and simulation preparation (described below), as they are handled differently in the third main phase: state propagation.

Network construction primarily involves the creation of neurons followed by the creation of synaptic connections between them. Since structure-aware neuron creation itself does not require infrastructural changes, the first kernel-level modifications introduced by the structure-aware NEST implementation arise during synapse creation. At this stage, synaptic data structures such as connection and source tables (see Figure 4.11) are duplicated so that connections and their source neuron identifiers are stored separately for short- and long-range connections. The `Connect()` call at the Python user interface level accepts a Boolean parameter "long_range" which is used to assign the newly created connection to the respective data structure.

While duplicating data structures may raise concerns about increased memory consumption, the additional overhead is small. The outer dimensions of the connection and source tables are defined by thread identifiers and synapse-type identifiers, and the actual connection objects and source-neuron identifiers are not duplicated. Instead, they are distributed across two separate data structures. This results in two shorter data lists as opposed to a single longer one, which may additionally improve cache efficiency during spike delivery.

Following network construction, the simulation enters a preparation phase. This phase first jointly sorts the connection and source tables by source neuron identifiers. Subsequently, it creates a target table that defines the MPI processes hosting the postsynaptic targets of each local neuron and thread (see Figure 4.11). The structure-aware strategy again duplicates this target table to mirror the separation of postsynaptic connection and source tables into short- and long-range components. Target table construction thus requires two construction steps, each involving an `MPI_Alltoall()` communication of the relevant connectivity data, potentially performed in multiple rounds. As with the duplicated connection and source tables, the additional memory overhead due to duplicating the target tables is small, as the algorithm does not replicate the target locations but rather distributes them across the two target tables.

Finally, the third main phase begins: the state propagation phase advances the dynamical state of the neuronal network. This main phase of the simulation subdivides into the minor phases update, collocate, communicate, and deliver. The structure-aware scheme, excludes neurons tagged as "frozen" on a given MPI process from the update phase (see Figure 4.3). When a neuron crosses the firing threshold, the algorithm temporarily stores outgoing spikes in a spike register, separately for short- and long-range connections. The structure-aware implementation thus duplicates the data structure. The scheme distinguishes between a local exchange of spikes within the same MPI process and a global exchange of spikes across all MPI processes. Spikes destined for short-range targets are communicated locally and cleared from the corresponding spike register after each exchange. Spikes going to targets of long-range connection are accumulated on the presynaptic side over multiple simulation cycles before being exchanged in the global communication step. The number of cycles over which these spikes are buffered is determined by the minimum delay of long-range connections in the network. Both communication pathways have separate MPI buffers. Since neurons typically exhibit both, short- and long-range outgoing connections, the algorithm inserts their spikes with high probability into both spike registers and MPI buffers. This leads to an increase in memory usage in the structure-aware scheme. Further, buffering long-range spikes over several simulation cycles temporarily raises memory demand on the presynaptic side and results in larger message sizes in the global MPI communication. Once a spike arrives at the target compute node, the separate connection and source tables for short- and long-range connections ensure correct delivery to the postsynaptic neurons, without the need for further duplication of data structures on the postsynaptic side.

4.3.2 *Benchmark Model*

In this study, we evaluate performance using two spiking neuronal network models. The first is the multi-area model of macaque visual cortex (MAM; Schmidt et al. (2018b,a)). The second is a simplified variant of this model, referred to as the MAM-benchmark, which is designed for scaling experiments and controlled parameter variations.

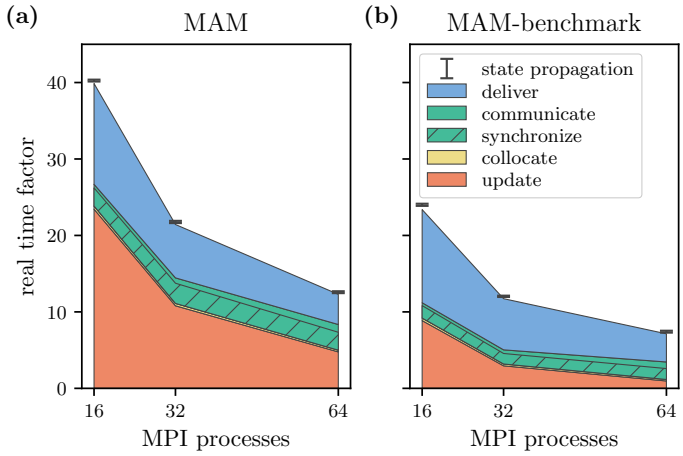

Figure 4.12: Comparison of strong scaling performance of MAM (a) and MAM-benchmark (b) on SuperMUC-NG, modeling 32 areas. Color scheme, black error bars and benchmarking configurations as in Figure 4.7.

The MAM comprises one square millimeter of each of the 32 visual cortical areas of macaque monkey. A layered cortical microcircuit (Potjans & Diesmann, 2014) represents the individual areas. Each layer consists of an excitatory and an inhibitory population of integrate-and-fire model neurons all with identical intrinsic parameters. Neuron densities and connection probabilities vary across areas and are derived from experimental data, resulting in across-area heterogeneity in both size and spiking activity. The coefficient of variation of area size is ~ 0.2 with an average area size of $130,000$. The areas are highly interconnected, with approximately a third of all synaptic connections targeting neurons in other areas (~ 1800 long-range connections per neuron). Transmission delays are drawn from Gaussian distributions. In contrast to the original model we impose a lower cutoff d_{min}^{inter} on the delay distribution of the inter-area connections. For the range of d_{min}^{inter} values considered in this study, we observe no noticeable effect on network dynamics (Grundler, 2025) although approximately 16 % of all inter-area connections have a delay below 2 ms. There exist two dynamical regimes in the MAM depending on its parameterization: a ground state characterized by low and stable firing rates; and a metastable state with episodes of high synchrony and elevated firing rates. The present work considers the ground state with a mean firing rate of 2.5 spikes per second and a coefficient of variation per time step of ~ 0.5 spikes per second.

The MAM-benchmark is inspired by the MAM but deliberately more homogeneous. All areas contain the same number of neurons, and each neuron has the same number of intra- and inter-area connections. These numbers are chosen to reflect the average neuron count and connectivity of the MAM across areas, resulting in approximately $130,000$ neurons per area and $6,000$ outgoing connections per neuron, with half of the connections targeting neurons within the same area and half targeting neurons in other areas. Intra-area and inter-area delays are drawn from Gaussian distributions with a mean of 1.25 ms and 5 ms, and a standard

deviation of 0.625 ms and 2.5 ms, respectively. As in the MAM, we impose a lower cutoff $d_{\min}^{inter}$ on inter-area delays. A key difference between the two models lies in the neuron dynamics. The MAM-benchmark employs a single neuron type, referred to as the *ignore-and-fire* neuron. While this neuron receives and emits spikes in the same way as a standard integrate-and-fire neuron, it does not propagate a membrane potential. Instead, it generates spikes at a predefined interval and phase, independent of synaptic input. This simplified neuron model enables us to scale the benchmark-MAM to smaller or larger numbers of areas while maintaining constant activity levels, without the need for retuning network parameters.

Figure 4.12 compares the strong-scaling performance of the MAM and the MAM-benchmark. The average delivery, communication and collocation times are very similar for both models, indicating that they exhibit comparable performance-relevant characteristics. As expected, the update phase is noticeably faster for the MAM-benchmark due to simpler neuron dynamics.

We perform each simulation throughout the study with three different random seeds $\{12, 654, 91856\}$ for the NEST random number generator. Because connectivity generation in NEST is stochastic and relies on this generator, each seed produces a distinct network instantiation with different connectivity realizations.

4.3.3 *Benchmarking Systems*

We perform benchmarks on two HPC systems: JURECA-DC at the Jülich Research Centre, Germany, and SuperMUC-NG at the Leibniz Supercomputing Centre (LRZ) in Garching, Germany. Automatic NUMA balancing by the Linux kernel is set to off in both cases.

The JURECA system (Thörnig & von St. Vieth, 2021) comprises a CPU-only partition (JURECA-DC), as well as a CPU-GPU partition (JURECA-DC Booster). In this study, we use the standard compute nodes of JURECA-DC Phase 2 (480 standard nodes in total). Each node is equipped with two AMD EPYC 7742 processors, with 64 cores each, clocked at 2.25 GHz. In total the nodes have 128 cores and a main memory of 512 GiB. The CPU partition has a peak performance of 3.54 Petaflops per second. Nodes are interconnected via Mellanox HDR100 InfiniBand. Applications are compiled using the GNU Compiler Collection (GCC), and we link against the OpenMPI library for distributed-memory communication and OpenMP for shared-memory parallelism within a node. We use the general-purpose memory allocator jemalloc version 5.3.0.

SuperMUC-NG Phase 1 (Bastian et al., 2024) consists of 6,480 compute nodes, each equipped with two Intel Skylake Xeon Platinum 8174 processors running at 3.1 GHz, providing 48 cores per node. The majority of compute nodes have 96 GiB of main memory. The peak performance is 26.3 Petaflops per second. Compute nodes are connected via an Intel OmniPath interconnect and are bundled into eight domains, each employing a fat-tree topology. As on JURECA-DC, we compile applications via GCC and use hybrid parallelism via OpenMP and OpenMPI.

On both systems, we employ one MPI process per node using all available physical cores per node. Hyperthreading is turned off.

Benchmarks on JURECA-DC use the automated benchmarking pipeline *CI-beNNch* (Vogelsang et al., 2026) presented in Chapter 5.

4.4 DISCUSSION

The results presented in this chapter show that MPI synchronization dominates spike communication as the number of MPI processes increases, while pure MPI data transfer accounts for only a minor fraction of the overall cost. The structure-aware simulation strategy introduced here combines a neuron distribution aligned with the modular organization of multi-area networks with a communication scheme that exploits the separation of temporal scales between local and long-range transmission delays. Consistent with the proposed theoretical analysis of simulation performance, the new strategy substantially reduces spike delivery and synchronization overhead, with the magnitude of the improvement depending on network heterogeneity and hardware characteristics.

Confining brain areas to individual compute nodes is the precondition for exploiting the delays of long-range connections in the communication between nodes. The disadvantage is that this destroys the load balancing of the conventional round-robin distribution over compute nodes. As a consequence, compute nodes hosting brain areas with a larger number of neurons or higher activity carry a higher computational load than others, and in each communication cycle all nodes must wait until the most heavily loaded node has completed its work. When the nodes of an HPC system are stressed by the memory or cache bandwidth demands of the neuronal network, the additional compute time inflicted by this imbalance may outweigh the gain from fewer synchronizations. A simple scaling argument explains this: if the times for the deliver and update phases are large, their variability across nodes is also large in absolute terms, and even a small relative increase in imbalance can shadow the gain from reduced synchronization frequency. The comparison of results across different HPC systems illustrates this effect directly. The multi-area model (MAM) of macaque visual cortex (Schmidt et al., 2018b,a) is an established and challenging benchmark, and its modular organization makes it particularly suitable for evaluating structure-aware communication strategies. However, the model exhibits known limitations. The meta-stable state most compatible with experimental observations is reached only at a specific setting of a coupling parameter that cannot be explained within the scope of the model, and this state is accompanied by an excess of spike synchrony on a fine timescale in some cortical areas. The occasional need to communicate and process large bursts of synchronous spikes places additional stress on the computing system: in the structure-aware distribution this increases load imbalance, while in the round-robin scheme synchronous neurons are spread across all nodes. The present work therefore focuses on the ground state of the network, at the cost of underestimating the natural variability of spiking activity over time. An improved version of the MAM that addresses these issues is presently under investigation (Pronold et al., 2024), and once established, such a model may replace the original MAM as the reference benchmark. The MAM-benchmark model used for systematic scalability studies maintains constant activity levels across network sizes and enables controlled parameter variations,

with strong-scaling characteristics comparable to those of the full MAM under the conventional simulation scheme. However, its simplified neuron model has an update cost that is largely independent of spiking activity, and as a consequence it does not capture the workload imbalances arising from heterogeneous firing rates that significantly influence synchronization and performance in the full MAM. The MAM-benchmark therefore does not fully expose the performance implications of the structure-aware scheme for more biologically realistic models, although it remains a valuable tool for controlled scalability studies.

While the use of the MAM constrains the range of network architectures considered, the results obtained with the MAM-benchmark indicate that the proposed approach is not limited to strongly modular systems. Performance gains persist even when the separation between inter- and intra-area delays is reduced, suggesting that improved locality alone is beneficial and pointing toward a broader applicability to networks more homogeneous delay distributions. Whether partitioning algorithms could automatically discover delay-based modularity in networks where structure is not known a priori is an interesting direction for future work, but lies outside the scope of the present study, which focuses on the biologically relevant case where network structure is defined explicitly during model construction.

Even in the idealized case of equal-sized cortical areas and homogeneous stationary spike rates, synchronization between compute nodes remains the major contribution to total communication time. The dominant contribution to state propagation is spike delivery, and its variability is the origin of synchronization overhead. Considerable efforts have already been undertaken to streamline the routing of spikes to their target neurons (Pronold et al., 2022a), though not all of these ideas have yet entered production code. More detailed measurements and advanced performance modeling are needed to identify remaining bottlenecks and develop strategies to circumvent them. The parallel compute power of compute nodes is still increasing. Current high-end compute nodes feature in the range of 256 computational cores. For a network of $100,000$ neurons distributed across all cores of such a node, each core would only have to take care of around 400 neurons and consequently oversee only a limited amount of memory. In principle this should remove the von Neumann bottleneck for the intra-node computation. But to what extent can we exploit the microscopic parallelism of spiking neuronal networks with conventional computer architectures? Initial discussions over the past few years have led to the formation of a new consortium to develop respective performance models Deutsche Forschungsgemeinschaft (DFG) (2024).

The present work assigns the network representation of each cortical area to just one compute node. With the help of `MPI_Group` future work can relax this constraint to a more flexible scheme where a cortical area extends over multiple compute nodes forming an `MPI_Group`. In this way the number of compute nodes per area adapts to the size of the area such that number of neurons per compute node remains approximately constant. Within an `MPI_Group`, communication occurs in intervals of the minimal delay of the connections of the area until the minimal delay of the long-range connections is covered and a communication between the areas needs to occur. The scheme regains load balancing with respect to network structure.

A fundamental limitation remains the need to first synchronize threads and then synchronize compute nodes in certain intervals due to the collective communication. Previous studies have shown that the choice of communication mechanism (blocking, non-blocking, or collective) significantly affects scalability in distributed spiking simulations Thibeault et al. (2013). However, even non-blocking approaches do not fully eliminate synchronization constraints at scale. Recent work has improved the interplay between MPI and OpenMP, facilitating asynchronous communication Protze et al. (2022). It remains to be investigated whether spiking network simulation technology can benefit from these improvements, or whether this would require novel hardware support.

5

CONTINUOUS BENCHMARKING

The chapter is based on the following publication:

Continuous benchmarking: Keeping pace with an evolving ecosystem of models and technologies. J. Vogelsang, M. Lober, C. M. Schöfmann, J. Villamar, D. Terhorst, J. Senk, H. E. Plesser, M. Diesmann, S. Kunkel, and A. C. Kurth (2026) *arXiv preprint*. DOI: 10.48550/arXiv.2604.15919. Submitted to Patterns.

Author contributions:
The work presented in this chapter was carried out by a team of authors under the supervision of Dr. Anno Kurth (AK), Dr. Susanne Kunkel (SK), Prof. Dr. Markus Diesmann (MD), and Dr. Johanna Senk (JS). The author contributed to the conceptualization of the framework together with Jan Vogelsang (JVO), AK, Dennis Terhorst (DT), Catherine Schöfmann (CS), and José Villamar (JVI), jointly developing the overall design and architecture of CI-beNNch. Together with JVO, DT, the author was involved in the software development and maintenance and used the framework to run benchmarks. The author was also responsible for the usability analysis of the framework, including its comparison to the predecessor tool beNNch and the demonstration of its practical utility for common benchmarking scenarios.

5.1 INTRODUCTION

Systematic performance benchmarking is an essential but historically neglected component of scientific software development. As established in the introduction of this thesis, a useful distinction exists between the verification and validation of scientific software (Oberkampf & Roy, 2010): while correctness is routinely enforced through automated testing in continuous integration pipelines (Shahin et al., 2017), performance validation has historically received far less systematic attention. Since certain scientific questions can only be investigated at sufficient scale (Anderson, 1972; van Albada et al., 2015), speed of code execution and memory requirements are part of the validation of scientific software, as they directly limit the range of addressable scientific questions.

The beNNch framework (Albers et al., 2022) was developed to address this gap for spiking neural network simulators, providing a standardized workflow for configuring and executing benchmarks and aiming to establish benchmarking as a collaborative, routine effort. In practice, however, beNNch fell short of its goals: the underlying environment configuration remained the responsibility of individual researchers, and the sensitive interplay between hardware and software configurations on HPC systems made reproducing results a challenging task requiring extensive expert knowledge (Villamar et al., 2025). These limitations were brought into sharp relief during the COVID-19 pandemic, when researchers working remotely across four laboratories attempted to use beNNch to reproduce existing benchmarks. In several cases, supposedly identical specifications produced significantly different results, and resolving the discrepancies through remote communication alone took months (Albers et al., 2022; Viererbl et al., 2022). This experience revealed that the communication challenges in complex benchmark experiments had been underestimated, and pointed to a concrete need for robust digital support for configuration management in benchmarking workflows.

This chapter presents CI-beNNch (Figure 5.1), a continuous benchmarking framework that addresses these shortcomings by integrating performance assessment directly into the continuous integration pipeline of the NEST simulator (Gewaltig & Diesmann, 2007). A unified, machine-agnostic entry point removes the need for individual hardware and software configuration, ensures identical execution across HPC systems, and enables the regular automated assessment of simulator performance as the codebase evolves. In contrast to earlier CI-based benchmarking approaches (Anzt et al., 2019; Pearce et al., 2023), CI-beNNch is designed for greater generality and a higher level of abstraction, reducing the domain knowledge required from individual contributors. The chapter describes the design of CI-beNNch only briefly, evaluates its usability relative to its predecessor beNNch, and demonstrates its practical utility across a range of benchmarking scenarios relevant to the computational neuroscience community.

5.2 PRINCIPLES OF CI-BENNCH

The goal of any benchmarking framework must be to achieve repeatability, replicability, and reproducibility independent of the individual who composed the

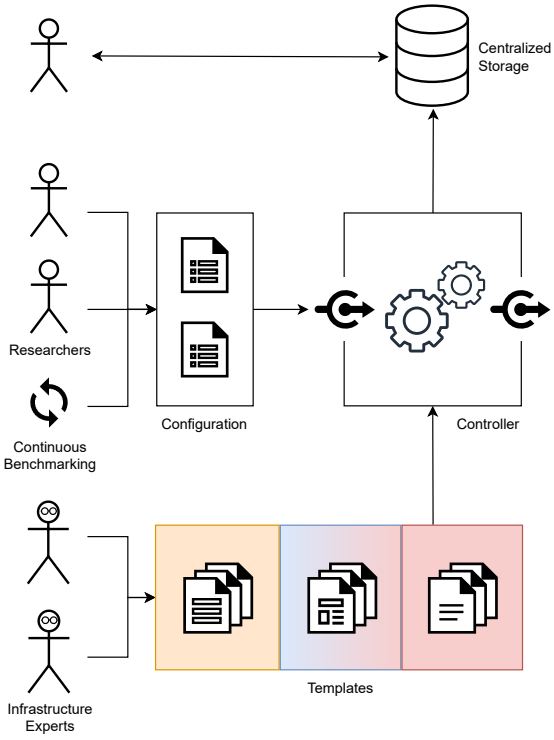

Figure 5.1: Overview of the continuous benchmarking process. Researchers specify their experiments via configurations and use them to trigger benchmarking runs through a unified entry point (middle row). A researcher can also query data and metadata from previous runs through a centralized storage (top row). Infrastructure experts (for example system administrators or senior researchers; bottom row) define the hierarchical templates consisting of workflow (orange), infrastructure (blue: platform, red: machine), and implementation (red) templates, which are then used by a controller to construct a benchmarking pipeline given the researcher's configuration and execute it to produce benchmark results.

benchmark and the infrastructure used for its execution. In beNNch, this goal was not fully met: the underlying environment configuration remained the responsibility of the individual researcher, base configurations were continuously overridden and changed over the course of multiple benchmarks, and porting a benchmark to a different machine required researchers to recreate the environment from scratch and adapt the tool itself. Expert knowledge required at each step was the main hindrance during routine use, turning out to be a major challenge not only for junior researchers with limited HPC knowledge, but also for senior staff who had to first learn the intricacies of beNNch before being able to run benchmarks.

5.2.1 *Standardized and Centralized Hierarchical Configurations*

To solve these issues, CI-beNNch proposes the integration of benchmarking suites into CI setups with a unified entry point from which all benchmarks are triggered. Thereby, the execution of a benchmark becomes independent of the researcher, unwanted side effects caused by different environments can be largely avoided, and a central location for benchmark results and automatically generated analyses can be established. Central to this approach is the use of hierarchical benchmark configurations. Adapting another researcher's specification to personal hardware and software constraints can be achieved by overwriting individual parts of a given benchmark specification while inheriting all unmodified values. This allows researchers to focus on their specific scientific questions while relying on experts for remaining configuration to be properly set up. Configurations are also easily exchangeable between different research groups. The configuration defines everything required to execute a benchmark, including software dependencies and the precise way of fulfilling them. Treating the benchmarking pipeline as a self-contained service effectively removes the dependence on the person executing the benchmark, maximizing reproducibility and reducing the expert knowledge needed. All benchmark configurations are stored in a shared location accessible to all researchers, following the idea of standard CI workflows where tests and builds are executed by a cloud runner without requiring human intervention (Zhao et al., 2017).

5.2.2 *Layered Abstraction*

CI-beNNch identifies three layers of abstraction for each benchmarking pipeline. The first layer, the *workflow layer*, defines the overarching goal as a workflow — such as benchmarking, profiling, or testing — and remains research group- and platform-agnostic. The second layer, the *architecture layer*, details the steps required to execute each stage of the workflow on the designated platform and machine, accounting for research group specific expertise and standards. It is logically split into platform-specific templates, which serve as a skeleton of the CI pipeline, and machine-specific templates, which function as high-level descriptions for groups of commands. The third layer, the *implementation layer*, specifies the precise commands to be executed within each step, and is both platform- and machine-agnostic. Each layer defines a template which is successively specialized by each subsequent layer, and the result is a complete template for a single benchmark run that is fully specialized by a

unique parameter combination when triggered. By fully separating benchmark configuration and execution, the implementation details of the benchmarking process are no longer left to the individual user but can be addressed by the appropriate expert. This facilitates a natural division of responsibilities: a machine expert handles the interaction with the HPC scheduler and machine-specific interfaces; a platform expert integrates machine-specific code into the general framework; a software expert and model expert define the software and model configurations respectively; and the researcher specifies user-level configuration and adapts the base configuration for their specific experiment.

5.2.3 *CI-based Execution*

A controller manages the execution of the benchmarking pipeline by taking a configuration from the researcher at the unified entry point, constructing the benchmarking pipeline, and monitoring its execution. Pipeline construction proceeds in three steps: first, the controller generates a platform instance from the workflow and platform templates; second, it produces a benchmark run script in which certain commands are explicitly defined while others retain template variables to be filled with values from the benchmark configuration; third, the script is instantiated for each distinct parameter combination in the experiment's parameter space. The controller itself may be executed anywhere it can be accessed by the research group and is able to reach the target machines. This design ensures that a benchmark is fully specified by the hierarchical configurations together with the templates defined for each layer, making the entire benchmarking process transparent, portable, and reproducible across diverse HPC environments.

5.3 CI-BENNCH IN ACTION

In this section, we explore specific, generally applicable implications of the design choices of the proposed continuous-benchmarking approach. We highlight advantages for commonly encountered situations in HPC benchmarking.

5.3.1 *Running a Benchmark for the First Time*

As noted above, the adoption of beNNch was primarily hindered by requiring an individual setup and configuration. Setting up and configuring beNNch is a laborious, error-prone, manual process and presents a high barrier of entry. In our experience, a successful use of beNNch usually requires the help of researchers already familiar with the tool as well as the peculiarities of the specific target machine. The difficulty for a new researcher running an existing benchmark is now significantly reduced, assuming that other experts already set up a CI-beNNch instance including a base configuration. The CI-based approach introduces a higher degree of automation and eliminates the need for direct machine access altogether. Running a first benchmark with default parameter settings can now be reduced to two simple steps: First, the researcher executes a pipeline with default settings via the CI entry point. CI-beNNch then automatically builds the software environment,

Table 5.1: Initial execution of a benchmark using the beNNch framework versus a CI-beNNch pipeline. Red dots: manual user intervention. Blue dots: researcher must be logged onto the target machine. Black circles: steps automatically triggered with default settings but can be customized.

	beNNch	CI-beNNch
1. environment	•• manually install software dependencies	
2. setup	•• deploy beNNch (e.g. fetch, configure, and adapt machine-local instance, setup results location) •• fetch models	• get access to the CI entry point
3. configuration	•• copy default benchmark configuration	○ research group provides base configuration which is used automatically
4. execution	•• start benchmarking script	• trigger CI pipeline
5. analysis	•• repeatedly check until all benchmarking jobs have finished •• navigate to results folder on machine and execute analysis script	○ analysis pipeline triggered automatically

executes the benchmark jobs and gathers the results on the target machine. In the second step, once the benchmark jobs have been completed, the researcher is provided with the acquired data and visualizations of the benchmarking results. Table 5.1 provides a comparison of the setup steps required for beNNch versus CI-beNNch, highlighting the reduced complexity and improved accessibility for first-time users.

5.3.2 *Reproducing a Benchmark*

To overcome the limitations of beNNch, the CI-beNNch approach decouples benchmark configurations from individual researchers. All benchmark configurations are stored centrally and can be accessed by any researcher within the same research group. The automatic setup based on a centrally stored configuration ensures that the environment and procedures of reproduced benchmarks align as closely as possible with those of the original. The automatic generation of a standardized

pipeline guarantees consistency across repeated benchmark executions, thereby facilitating repeatability and reproducibility and long-term reliability of research.

5.3.3 *Rerunning a Benchmark with . . .*

. . . a different benchmarking model

To comprehensively evaluate the performance of an HPC application, it must be tested across a range of different parameters. In the example of codes for the simulation of biological neural network, varying parameters may include biophysical mechanisms, connectivity patterns, activity dynamics, and network size. New benchmarking models are integrated into the benchmarking pool by updating a shared configuration, which specifies repository locations, model parameters, model version, and execution instructions. This ensures that once a model is added, it becomes immediately accessible to all researchers within the group.

. . . a different machine

Comparative benchmarking across different machines allows researchers to identify whether performance is bound by the software itself or the underlying hardware. In the absence of an automated setup, following centrally defined machine-specific instructions, each researcher needs to individually set up the benchmarking environment on the target machine. As noted above, creating these setups is time intensive, error-prone, and easily introduces differences hindering reproducibility. By contrast, in the proposed solution, the integration of a new target machine is done centrally for all users by adding additional machine-specific platform templates. After this initial setup, all researchers with access rights to the new machine can run benchmarks on it without further configuration.

. . . a different HPC application

Adapting to a new HPC application involves specifying its deployment steps, required software dependencies, run instructions, as well as which performance metrics can be measured. Once this information is provided in the central configuration of the CI-beNNch instance, all researchers of the group are able to utilize the new simulation tool without any further adaptations. In beNNch build environments and personalized modifications of beNNch are maintained individually by each user, often undocumented and inaccessible to others. This limits the reliability of shared build instructions and hinders reproducibility. In contrast, CI pipelines implicitly enable user-agnostic deployment and instructions which for the chosen simulation tool need to be verified only once.

5.4 CI-BENNCH IN PRACTICE

We developed a CI-beNNch instance based on GitLab CI/CD pipelines and Jacamar (Hayot-Sasson et al., 2025), an HPC focused CI/CD driver for GitLab which enables submitting jobs on HPC clusters. CI-beNNch has been implemented for

the HPC systems of the Jülich Supercomputing Centre (JSC), namely JUSUF (Von St. Vieth, 2021), JURECA (Thörnig & von St. Vieth, 2021), JUWELS (Krause, 2019), and JUPITER (Herten et al., 2024) as well as a local compute cluster.

By defining a centralized entry point according to the principles outlined above, the entire complex process of correctly executing a benchmark is hidden from the researcher. We chose GitLab as a web-based entry point as it provides a straightforward user interface to trigger new benchmarking pipelines. The entry point pipeline is executed on a cloud runner, which downloads the configuration file provided by the researcher and constructs the combined benchmark specification by recursively loading and overwriting the hierarchy of configuration files up to the base level. Based on the final specification, a sub-pipeline is constructed for each target machine by instantiating and combining a set of pipeline templates.

For the task of benchmarking neural simulation codes, we define seven workflow stages which run as separate jobs in the sub-pipeline without any user intervention: Preparation, Build, Execution, Transfer, Annotation, Analyze, Plot. First, the environment is prepared by installing dependencies and fetching model and simulator code. This step requires internet access, restricting the choice of nodes of the system on which the job can be run (for example a login node or a developer partition). Second, the simulator is compiled on a compute node. This ensures an identical architecture for the subsequent benchmark runs, and the inclusion of micro-architecture specific optimizations without requiring a cross-compilation environment. The benchmark execution job itself runs on a login node, and submits compute jobs, one per parameter combination. Additionally, the raw data output from each compute job is enriched by a set of hardware and environment metadata from the compute node (Villamar et al., 2025). The benchmark execution job waits for completion of all parallel compute jobs and aggregates the results of all benchmark runs. In the next stage, both the raw data and metadata are transferred to the final storage location. The collected data is subsequently analyzed (possible on any machine with access to the final storage location) by executing a default analysis routine which annotates the benchmarking data based on the recorded metadata. In this way, a standard for search and select mechanisms is provided. The final step produces a standardized graphical representation of the performance data and uploads it as an artifact to the pipeline. The user can view the generated plot through the same web interface as the entry point.

Beyond the use-case of CB, CI-beNNch is also suited for aiding feature development for HPC applications. There, similar problems arise when performance critical parts of the simulation code are touched: benchmarks should be performed across different scales on different machines. To highlight this use, we focus on the spiking neural network simulation engine `NEST` as an example (Gewaltig & Diesmann, 2007).

5.4.1 *Barrier-free Spike Delivery*

In `NEST` version 2.16 the *5g* simulation kernel has been introduced (Jordan et al., 2018) which significantly improved the performance for large-scale simulations. Only a couple of years later, systematic benchmarking across scales revealed a drop

in performance for specific cases, mainly due to newly introduced OpenMP barriers, thread-synchronization locations in the algorithm inevitably causing idle times of some threads. In NEST version 3.6 the *5g* simulation kernel changes were combined with an optimized communication concept, where all spikes are communicated in a single buffer again to eliminate extensive thread synchronization. The impact on performance is assessed across scales, with both weak and strong scaling, using the HPC-Benchmark model (Kunkel et al., 2014) (a scalable balanced random network), a microcircuit model (Potjans & Diesmann, 2014) (representing all layers of cortex with asynchronous irregular activity), and a multi-area model (Schmidt et al., 2018b) (a large-scale model of the macaque vision-related cortex) on the supercomputer JURECA-DC by comparing the last NEST version using the original *5g* simulation kernel—NEST 3.5 (Haug et al., 2023)—and the NEST release 3.8 (Graber et al., 2024b).

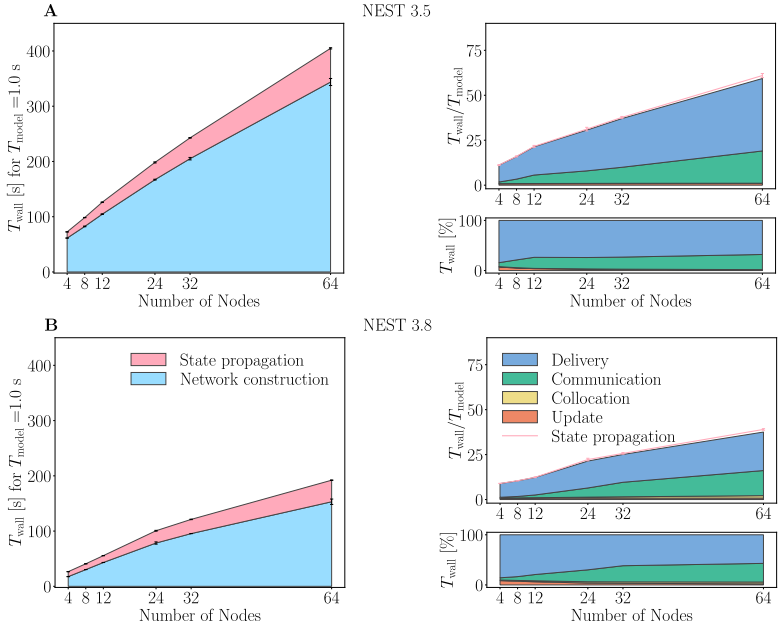

Figure 5.2: Weak-scaling performance of the HPC-Benchmark model on JURECA-DC using 2 MPI processes per node and 64 threads per process. (A) NEST 3.5. Left: absolute wall-clock time T_{wall} measured for both network construction and state propagation; simulating 90,000 neurons and 10^9 synapses per node for model time $T_{\text{model}} = 1$ s. Error bars show standard error across three random seeds for each phase separately. Right, upper panel: real-time factor defined as wall-clock time divided by model time for each phase of state propagation. Error bars show the variability of total state propagation times across three random seeds. Right, lower panel: relative contribution of phases to the state-propagation time. See the legend of panel B for an explanation of the colors used in the figure. (B) NEST 3.8. Same display as (A).

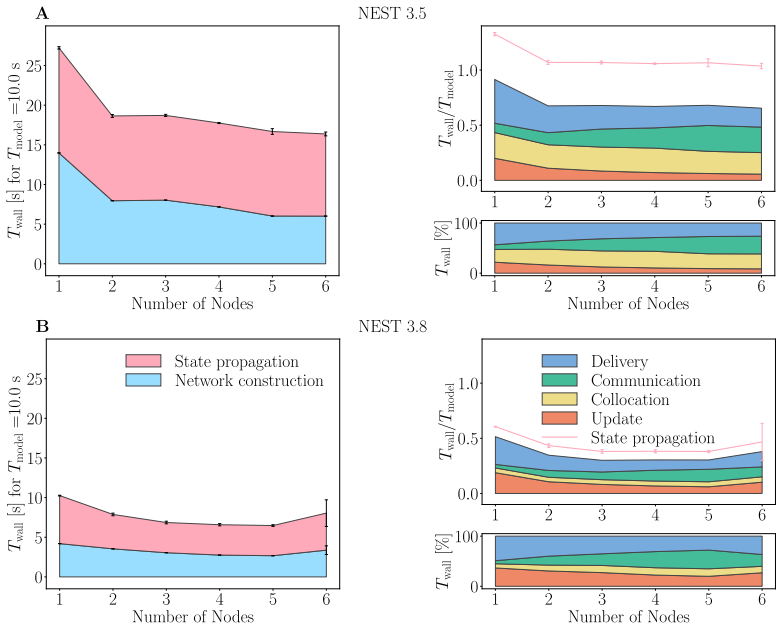

Figure 5.3: Strong-scaling performance of the microcircuit model on JURECA-DC using the same setup and display as in Figure 5.2. Model time $T_{\text{model}} = 10\,\text{s}$. (A) NEST 3.5. (B) NEST 3.8.

We distinguish four phases of state propagation: *update*, *collocation*, *communication*, and *delivery*. These concepts have proven useful in a sequence of earlier studies (for example (van Albada et al., 2021; Pronold et al., 2022b,a; Kurth et al., 2022; Albers et al., 2022)): *Update* measures the time required for updating the neuron dynamics. *Collocation* means the time to collocate MPI send buffers from the spike register. *Communication* refers to the time required to communicate spikes between computed nodes. And *delivery* is the time needed to deliver spikes from the MPI receive buffer to the node local synaptic targets.

When comparing NEST 3.5 and 3.8, the benchmarks expose a reduction in spike delivery time across all models, leading to a consistent reduction of total simulation times across all scales. For the HPC-Benchmark, delivery times decrease by 45% with a simulation time reduction of 26% (Figure 5.2). Note that in comparison with (Albers et al., 2022), Fig. 4, the network construction time is increased due to altering the definition for the HPC-Benchmark model, making it consistent with the other benchmark models. The microcircuit model further benefits from simplified spike collocation reducing spike collocation by 76% and spike delivery by 45% for a total reduction of 60% (Figure 5.3). We observe substantial improvements for simulations of the large-scale multi-area model in the meta-stable state which complete 37% faster (Figure 5.4). While in NEST 3.5 simulations at larger scale are predominantly slowed down by spike delivery, with the changes introduced

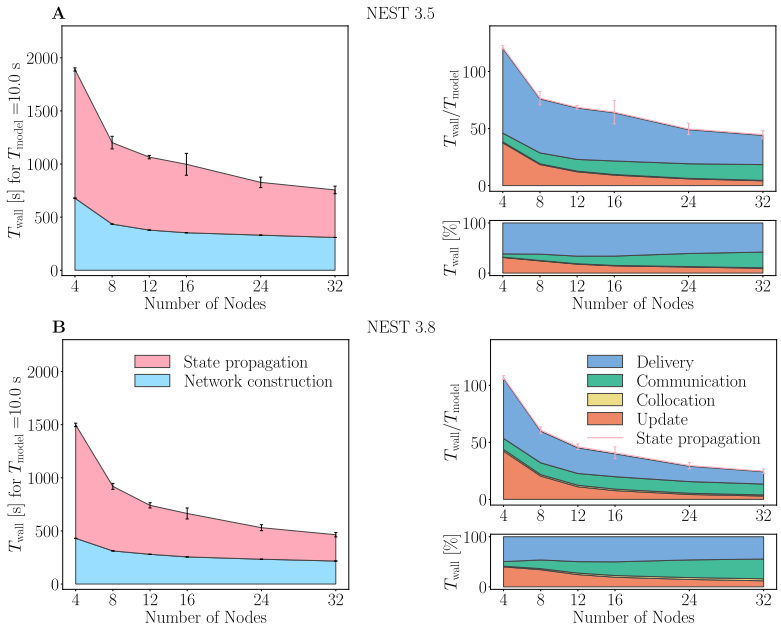

Figure 5.4: Strong-scaling performance of the multi-area model on JURECA-DC using the same setup and display as in Figure 5.2. The multi-area model is simulated in its meta-stable state for $T_{\text{model}} = 10\,\text{s}$. (A) NEST 3.5. (B) NEST 3.8.

between NEST 3.5 and NEST 3.8, spike delivery becomes less of a bottleneck compared to spike communication when scaling up the number of compute nodes.

Network construction time is also markedly lower in NEST 3.8 than in NEST 3.5. The difference is particularly prominent for the weak scaling experiments with the HPC-Benchmark model (Figure 5.2), showing that network construction time has been reduced by an average of 60%. This was achieved by a new approach to constructing the pre-synaptic connection infrastructure.

Changes to the simulation code were made by several researchers. A subgroup of them also was involved in assessing the performance of the suggested changes. CI-beNNch facilitated consistent benchmarking results between the individual researchers and thus contributed to rapid iterations in the development process.

5.5 DISCUSSION

This chapter presents CI-beNNch, an extension of the benchmarking concepts developed in Albers et al. (2022) that unifies and systematizes the performance assessment of HPC applications. The practical experience with its predecessor beNNch showed that the original approach severely underestimated the complexities in configuration, deployment, and execution of HPC application and model code, and that reproducibility of performance measurements could not in practice be ensured

even for researchers working in the same laboratory. CI-beNNch overcomes these limitations by abstracting configuration and deployment details through a unified, machine-agnostic entry point, hierarchical benchmark configurations, and centralized storage of results and metadata. The advantages are threefold: the approach guarantees repeatability by avoiding laborious and error-prone individual setups; ensures reproducibility by providing global and inter-individual configurations specifying environments, software deployment, and execution details; and enables replicability by allowing easy transfer of workflow details between systems.

5.5.1 *Practical Utility*

The practical utility of CI-beNNch is illustrated by its use in diagnosing an unexpected performance issue during benchmarking of a new feature of the NEST simulator. An extended warm-up phase at the start of each simulation was observed on one HPC system but not on another with nearly identical hardware. Identifying the root cause — different NUMA-balancing settings in the Linux kernel on the two systems — required a collaborative effort spanning several months and multiple researchers. CI-beNNch proved effective in this context: it guaranteed identical benchmarking setups on both systems, allowed configurations to be reused across machines with only minor adaptations, and enabled different researchers to contribute to simulator instrumentation, benchmark execution, and data analysis with minimal synchronous communication between them. The centralized storage of configurations and results ensured that all participants had access to the same experimental setup regardless of when they joined the effort. This experience illustrates how CI-beNNch lowers the coordination overhead that would otherwise be required for collaborative benchmarking across multiple systems and research groups.

5.5.2 *Limitations*

A fully executor-agnostic approach to benchmarking is in practice limited by the security and accounting requirements of HPC systems, which typically do not permit fully user-independent access. User environments of individual researchers are often automatically loaded when accessing the system, and while some systems allow the creation of shared user accounts, handling the associated compute budgets and access authorizations introduces practical difficulties. Containerization could in principle provide identical environments across systems, but for performance-critical applications it is essential to utilize machine-specific modules, and building high-performance containers requires privileged access and system-specific expertise that is often unavailable to non-privileged users. The approach taken in CI-beNNch therefore provides a weaker but practically achievable form of user independence: researcher-dependence is removed from benchmark configurations while accountability is maintained, placing the responsibility on the researcher to be aware of differences in default user environments across systems. A further limitation identified during practical use concerns data management: each benchmark archive is identified by an auto-generated unique identifier, which avoids name

collisions but makes it difficult to locate benchmarks executed in the past. These findings underscore the importance of robust data and metadata handling (Villamar et al., 2025), and point to data management as an area for future improvement.

5.5.3 *Beyond Continuous Benchmarking*

While the concepts developed in this chapter are targeted toward performance benchmarking, they also prove useful for application development more broadly. Mature scientific software projects often involve multiple senior developers alongside less experienced contributors. CI-beNNch lowers the barriers for rigorous benchmark execution that would otherwise require time-intensive onboarding of new contributors, and opens the possibility of supporting researchers with limited HPC expertise in transitioning models from local development environments to HPC systems — a step that typically poses a significant barrier requiring large time investments. Integrating successful optimizations, confirmed through systematic and automated benchmarking, into production code requires substantial additional effort that falls outside the typical academic scope of a PhD project and involves discussions within a larger group of experts. These challenges have only recently received widespread attention (Hocquet et al., 2024), and the field of Research Software Engineering is forming to address them (Speck & Wyatt, 2024).

Part IV

DISCUSSION

6

DISCUSSION AND OUTLOOK

This thesis explores sequence learning and computational efficiency in large-scale spiking neuronal networks, at the intersection of computational neuroscience and high-performance computing. Progress in this field increasingly depends not only on the quality of the models being developed, but also on the availability of tools capable of simulating and validating them at biologically meaningful scales. The thesis treats both perspectives: the identification and investigation of neuronal mechanisms underlying sequence learning and temporal coding through biologically grounded spiking network models, and the development of simulation and benchmarking infrastructure that enables the efficient instantiation of such models at large scales. Four research questions, introduced in Section 1.4, guided this work and are addressed in turn below.

The following sections discuss the main findings of each chapter in the context of the existing literature, identify themes that cut across the individual contributions, reflect on general limitations, and outline directions for future work.

6.1 SUMMARY

What are the limits of biologically plausible sequence learning, and how do they scale with the complexity of the sequences to be learned?

The work presented in Chapter 2 shows that spiking neuronal networks representing local cortical microcircuits at the submillimeter scale can successfully learn complex sequences on the order of tens of elements, using only local, unsupervised plasticity mechanisms. However, the limits of this learning are not primarily set by the size of the network or the number of neurons representing each sequence element, but by the plasticity dynamics itself. As sequence complexity is characterized by the probability of recurring sequence elements, it increases with sequence length. Learning complex sequences with recurring elements places mounting demands on the plasticity rule to form and maintain distinct context-specific representations. The time required to achieve reliable learning scales in a strongly supralinear manner with sequence length, meaning that while longer sequences can in principle be represented, learning them becomes progressively slower to the point of being impractical. This scaling behavior constitutes the fundamental limit of the model as studied here, and stands in direct contrast to theoretical capacity estimates derived from combinatorial arguments (Ahmad & Hawkins, 2016; Hawkins & Ahmad, 2016), which predict substantially larger capacities by counting the number of connectivity patterns that could in principle be realized without accounting for whether any plasticity mechanism can actually form them. The present results therefore provide an important empirical qualification of such theoretical estimates, and highlight the value of simulation-based capacity analysis as a complement to purely theoretical

estimates.

How can a biologically plausible spiking network learn not only the order of sequence elements, but also their temporal structure, and how can the speed of sequence replay be flexibly modulated?

Chapter 3 shows that the temporal structure of sequences can be encoded in the sTM model through a sequential activation of context-specific neuronal assemblies, such that elapsed time corresponds to a progression along an assembly chain. This permits a discretized representation of time that preserves sparse, context-dependent coding and supports unsupervised learning of sequences with complex temporal structure via local synaptic plasticity mechanisms. Synaptic background input can control the speed of autonomous sequence replay in response to a cue signal. While constant background input requires fine tuning, oscillatory background input provides a robust clock signal by which replay speed can be flexibly modulated. A sequence learned at a given speed can be replayed across a range of different speeds without the need for relearning of the underlying representation. This establishes a functional separation between the representation of the temporal structure of a sequence, encoded in the network connectivity, and the speed of replay, controlled by global background input. The results further reveal constraints on replay dynamics imposed by neurophysiological parameters: spike transmission delays, membrane time constants, and synaptic time constants limit the maximum replay speed, whereas the duration of plateau potentials caused by dendritic action potentials sets a lower bound. Together, both the content and the temporal structure of a learned sequence are represented by the same chain of assemblies formed by small, context-specific groups of neurons, enabling well-timed prediction, anomaly detection, and autonomous replay of temporally structured sequences.

How can the structural organization of the brain be leveraged to make large-scale spiking network simulations more efficient?

The structural organization of the mammalian cortex, with its clear separation between densely connected local circuits and more sparsely connected long-range inter-area projections, can be directly leveraged to reduce the dominant bottleneck in distributed spiking network simulations. Chapter 4 demonstrates that this bottleneck is MPI synchronization overhead rather than the volume of data exchanged or network latency, arising from the variability in the time at which different compute nodes complete their local work. By aligning the distribution of neurons across compute nodes with the modular organization of the network, neurons that communicate frequently are co-located on the same node, effectively speeding up local communication. Furthermore, by exploiting the fact that long-range inter-area connections have a substantially larger minimum delay than local connections, the modular neuron distribution allows global synchronization to occur only at the inter-area timescale rather than at every local delay interval, reducing the total number of synchronization steps required. Together, these two measures translate the biological separation of timescales directly into a computational advantage, yielding substantial reductions in both synchronization and spike-delivery time. The magnitude of the improvement depends on the degree of load imbalance

across nodes and the specific hardware characteristics of the target system, but performance gains persist even when the separation between local and long-range delays is reduced, suggesting that the approach has broader applicability beyond the specific model used for validation.

How can the performance of scientific simulation software be systematically validated as models and hardware continue to evolve?

The answer proposed in Chapter 5 is continuous benchmarking: the regular, automated assessment of simulator performance integrated directly into the software development pipeline, analogous to the continuous integration practice already established for correctness testing. The chapter demonstrates that the key obstacles to achieving this in practice are not technical but organizational. The sensitivity of HPC performance to minute differences in hardware and software configuration means that informal, researcher-dependent benchmarking workflows cannot reliably ensure reproducibility, even within the same laboratory. CI-beNNch addresses this by introducing three mutually reinforcing elements: a unified, machine-agnostic entry point that removes researcher-dependence from benchmark execution; hierarchical configurations that allow specialists to define reusable base setups while giving individual researchers the flexibility to adapt them for specific experiments; and centralized storage of results and metadata that enables benchmarks to be reproduced, compared, and built upon over time. Together, these elements guarantee repeatability by avoiding error-prone individual setups, ensure reproducibility by providing shared and versioned configurations, and enable replicability by making it straightforward to transfer a benchmarking workflow from one HPC system to another. The practical experience reported in the chapter confirms that this approach substantially reduces the coordination overhead of collaborative benchmarking and lowers the barrier for less experienced contributors, while providing the systematic performance tracking needed to detect regressions as simulation code evolves.

6.2 SYNTHESIS

6.2.1 *The Role of Biological Constraints in Model Design*

The spiking neuronal network model employed in Chapter 2 and Chapter 3 is grounded in specific biological mechanisms (dendritic computation, sparse coding, structural STDP, lateral inhibition) and is evaluated against functional criteria derived from behavioral observations. This dual grounding reflects the bottom-up meets top-down modelling philosophy discussed in the Chapter 1. From the single, unified architecture of the sTM model emerges a range of functionalities. The model learns sequences in a continual, unsupervised manner using only local plasticity rules, accounts for context dependence with high context depth, exhibits high storage capacity and low energy demands due to its sparse coding, supports multiple operation modes including prediction, anomaly detection, and autonomous replay, and, as shown in Chapter 3, can also encode and replay complex temporal structure. These capabilities are not the result of separate dedicated modules but arise from the same set of biological mechanisms operating together.

By constraining the model to mechanisms that are known to exist in cortical circuits, the resulting architecture captures functional properties that purely top-down models optimized for a single objective would not naturally exhibit.

Another instructive example of the value of this approach comes from the capacity analysis presented in Chapter 2. Theoretical estimates of sTM model capacity based on combinatorial arguments suggest that the network should be able to represent far more transitions than what can be learned by the model in a reasonable amount of time. The simulation-based analysis reveals that this discrepancy arises because plasticity dynamics imposes constraints that combinatorial theory does not capture. This is a concrete illustration of the broader lesson that biologically grounded simulation can reveal limitations and trade-offs that purely theoretical analysis misses.

6.2.2 *Scale as a Unifying Challenge*

A thread connecting all work presented in this thesis is the challenge of scale. This quest is motivated by the finding in nature (Benson-Amram et al., 2016; Herculano-Houzel, 2017) and Artificial Intelligence (Berti et al., 2025) that larger networks come with increased cognitive capabilities, and that spiking neuronal networks may play out their advantage at the spatio-temporally ultra sparse representations of the brain. The sequence learning model operates at a scale of tens of minicolumns and few thousands of neurons, orders of magnitude smaller than the average cortical area. The models investigated in Chapter 4 and Chapter 5 operate at the scale of millions of neurons. Chapter 5 provides the infrastructure to validate performance at this scale. Together, these contributions form a trajectory from the mechanisms of sequence learning at small scale to the tools needed to study such mechanisms at biological scale. Bridging this gap and simulating the sTM model or its successors in a network large enough to be compared directly with cortical recordings, could be aided by both, the structure-aware simulation approach of Chapter 4 and the performance validation infrastructure of Chapter 5, and represents a concrete long-term goal to which this thesis contributes.

In principle, the sTM model is well suited for scaling, since its architecture is modular by design, and increasing the number of minicolumns expands the network's representational capacity without requiring fundamental changes to the underlying mechanisms. Each minicolumn operates according to the same local rules, and the sparse, context-dependent coding scheme ensures that adding more minicolumns does not introduce interference with existing representations. A further dimension of the scaling challenge is nevertheless revealed by the capacity analysis in Chapter 2: even if a larger network could in principle represent longer sequences, the supralinear increase in learning time with sequence length means that the plasticity dynamics itself may become the bottleneck before network size does, motivating the search for faster learning mechanisms.

Despite operating at a smaller scale, the sTM model already produces phenomena that are directly comparable to observations in biological circuits. In particular, Chapter 3 shows that under low-frequency oscillatory input, the model produces temporally compressed replay that is initiated in a phase-dependent manner relative

to the oscillation cycle. This regime bears a close resemblance to the compressed replay events observed in hippocampus and cortex during sleep (Ji & Wilson, 2007; Euston et al., 2007; Carr et al., 2012; Buzsáki, 2015; Klinzing et al., 2019).

6.2.3 *The Interplay between Model and Simulation Infrastructure*

The development of CI-beNNch and the structure-aware simulation approach are not purely engineering contributions. They shape what scientific questions can be asked. The two contributions are also mutually reinforcing: the identification of synchronization overhead as the dominant bottleneck was itself made possible by systematic benchmarking across scales and hardware configurations, which is a capability that CI-beNNch provides and that ad hoc benchmarking could not reliably deliver. Conversely, the structure-aware simulation strategy provides a concrete use case that validates the benchmarking framework and demonstrates its practical utility for supporting iterative development of performance-critical simulation code. Beyond individual scientific results, CI-beNNch lowers the barrier for rigorous benchmark execution for researchers with limited HPC expertise, and opens the possibility of supporting the transition of models from local development environments to HPC systems. This step typically poses a significant practical obstacle. In this sense, the infrastructure developed in this thesis shapes not only what questions can be asked, but also who can ask them. Better tools enable new science, and new scientific demands drive the development of better tools.

This dynamic is not unique to the present thesis but reflects a broader pattern in the history of computational neuroscience. Approaches based on spatial tiling have shown favorable scaling behavior for networks with strong local connectivity, but their assumptions are difficult to generalize to heterogeneous multi-area models with dense long-range connectivity (Kozloski & Wagner, 2011; Yamaura et al., 2020). Earlier structure-aware simulators demonstrated that making explicit use of population organization and transmission delays can yield efficient distributed execution, and some of the underlying concepts were carried forward into communication libraries used by later simulation frameworks (Hammarlund & Ekeberg, 1998; Djurfeldt et al., 2010). At the other end of the spectrum, recent frameworks have achieved remarkable performance at extreme scale by tightly coupling algorithmic design with specific hardware architectures (Lyu et al., 2024). A common thread running through this body of work is that structure-aware algorithms have emerged from concrete neuroscientific needs rather than from a drive toward generality, which has ensured practical relevance but has also limited the transfer of these approaches into widely used generic simulation codes. Integrating confirmed optimizations into production code requires substantial additional effort that typically falls outside the academic scope of a PhD project, involving coordination across larger groups of developers and discussions that extend well beyond the original scientific contribution (Hocquet et al., 2024). The broader challenges of sustaining scientific software infrastructure (financial, organizational, cultural) play an important role here that goes beyond algorithmic choices alone (Aimone et al., 2023; Hocquet et al., 2024; Plesser et al., 2025). By grounding the present implementations in an established generic simulation engine, and by developing CI-beNNch as a reusable

benchmarking infrastructure, this thesis contributes not only scientific results but also transferable tools that lower the barrier for the broader community fostering sustainability in computational neuroscience.

6.2.4 *Structure-aware Simulation and Neuromorphic Computing*

The structure-aware simulation strategy introduced in Chapter 4 advances the conventional HPC reference code in a direction that is conceptually aligned with neuromorphic computing, even though the implementation targets standard CPU-based systems. By co-locating neurons that communicate frequently and reducing the frequency of global synchronization, the approach moves toward the event-driven, communication-efficient paradigm that neuromorphic hardware is designed around. The finding that synchronization rather than data transfer is the primary bottleneck is itself an insight that benefits the broader community, by separating physical from software limits and encouraging a closer examination of synchronization and asynchronous communication as targets for future optimization.

This connection between conventional HPC simulation and neuromorphic computing is further reinforced by the results of the sequence learning capacity study of Chapter 2. The plasticity mechanisms that drive learning in the sTM model are local, event-driven, and compatible with neuromorphic hardware implementations, suggesting that the sequence learning framework developed in this thesis could ultimately be deployed on neuromorphic platforms to achieve the kind of energy-efficient, large-scale sequence learning that biological circuits perform (Dietrich et al., 2025; Siegel et al., 2023; Bouhadjar et al., 2023).

6.3 LIMITATIONS

There is a substantial gap in scale between the sequence learning models and the large-scale simulation work. The sTM model is studied in networks of tens of minicolumns, while the simulation infrastructure is validated on a model containing millions of neurons organized into dozens of cortical areas. The two lines of work are conceptually connected but have not yet been combined in a single simulation. The capacity analysis of Chapter 2 is conducted for a fixed network size and pattern size, and a systematic investigation of how capacity scales with these parameters remains to be carried out.

The two sequence learning chapters together paint a coherent picture of the resource demands of the sTM framework. From Chapter 2, the primary limiting factor on sequence learning capacity is not network size but plasticity dynamics, with learning time scaling supralinearly with sequence length. Chapter 3 reveals a complementary constraint on the temporal side, namely that encoding longer time intervals requires more intermediate assemblies and hence more neurons, and noise accumulates across successive transitions, leading to reduced temporal precision for longer durations. These two limitations are distinct but related, since both reflect the cost of representing more complex structure within the same sparse, assembly-based coding scheme. Both aspects will need to be addressed as the model is extended toward more naturalistic and large-scale settings.

A limitation that applies to any HPC benchmarking infrastructure, including CI-beNNch, is that a fully executor-agnostic approach cannot be achieved in practice. Most HPC systems do not permit fully user-independent access due to security and accounting requirements, meaning that user-specific environment settings are often loaded automatically when accessing the system. While CI-beNNch substantially reduces researcher-dependence by removing it from benchmark configurations and enforcing shared, versioned setups, it cannot fully eliminate the influence of the underlying user environment. This limitation is not specific to CI-beNNch but reflects a fundamental tension between the reproducibility demands of systematic benchmarking and the security and accountability constraints of shared HPC infrastructure. This tension is likely to persist until HPC facilities provide more standardized support for reproducible software environments, a direction that is increasingly recognized as important by the community (EuroHPC, 2024).

6.4 OUTLOOK

Several concrete directions for future work follow naturally from the combination of results across this thesis. For the investigation of sequence learning, an immediate priority is to understand whether modifications to the plasticity rule could overcome the supralinear increase in learning time with sequence length. Concerning the modulation of replay speed in the sTM model, future work should investigate how different background input regimes modulate the scaling of timing variability with interval duration, and in particular whether the model reproduces Weber's law and known deviations from it at short and long intervals (Hass et al., 2008), which would provide a concrete link between the proposed mechanism and psychophysical observations.

A longer-term direction is the extension of the model to a hierarchical architecture in which each level learns sequences of sequences represented at lower levels. This is particularly relevant for naturalistic sequences, which exhibit rich compositional structure. Words are composed of phonemes, melodies of motifs, and sentences of phrases. A hierarchical network in which lower levels represent fine-grained sequential structure and higher levels combine these into increasingly abstract representations is therefore a natural fit for such data, and would bring the model considerably closer to the complexity of real-world sequential experience. A hierarchical learning architecture may be one of the mechanisms chosen by nature to overcome the problem of the supralinear increase in learning time pointed out above. Such a hierarchical network would naturally span multiple cortical areas, with different areas processing different input modalities and higher areas integrating across them, enabling, for example, the joint learning of an auditory melody and its visual representation as musical notation. This vision of a multi-area sequence learning network connects directly to the large-scale simulation infrastructure developed in Chapter 4 and Chapter 5. Simulating such a system at biologically realistic scale would require precisely the kind of topology-aware, efficiently benchmarked distributed simulation that these works provide.

A flexible extension of the structure-aware simulation approach would allow individual cortical areas to span multiple compute nodes using `MPI_Group`, such

that the number of neurons per node remains approximately constant regardless of area size. This would recover load balancing while retaining the communication benefits of the structure-aware scheme. More broadly, the present work suggests that as per-node computational power continues to grow, communication will increasingly dominate simulation time, and that architectural innovations beyond standard HPC may ultimately be required to keep pace with the scale demands of the field.

6.5 CONCLUDING REMARKS

The four contributions of this thesis span a range of scales and methods, from the biophysics of dendritic computation to the engineering of distributed simulation infrastructure. What unifies them is a shared commitment to biological plausibility and computational tractability: the belief that understanding the brain requires models grounded in its mechanisms, and tools capable of simulating those models at meaningful scales. The results presented here advance both sides of this endeavor: extending the range of sequences that biologically plausible spiking networks can learn and replay, and developing the simulation and benchmarking infrastructure needed to study such networks at biologically meaningful scales. Together, these contributions take steps toward the long-term goal of understanding how the brain learns and replays the structured sequences that underlie so much of cognition.

Part V

APPENDIX

7

SUPPLEMENTARY MATERIAL: SEQUENCE TIMING AND REPLAY SPEED

7.1 DETAILED DESCRIPTION OF THE NETWORK MODEL

Summary	
Populations	excitatory ($\mathcal{E}$), inhibitory ($\mathcal{I}$), bottom-up ($\mathcal{X}$), and top-down ($\mathcal{T}$)
Connectivity	• sparse random connectivity between excitatory neurons (plastic) • local recurrent connectivity between excitatory and inhibitory neurons (static)
Neuron model	• excitatory neurons: leaky integrate-and-fire (LIF) with nonlinear input integration (dendritic action potentials) • inhibitory neurons: leaky integrate-and-fire (LIF)
Synapse model	exponential or alpha-shaped postsynaptic currents (PSCs)
Plasticity	spike-timing dependent structural plasticity and weight decay in excitatory to excitatory connections
Input	bottom-up and top-down spike sources and background currents, connected to excitatory neurons

Table 7.1: Summary of the network model. Parameter values are given in Table Tab. 7.10.

Populations		
Name	**Elements**	**Size**
$\mathcal{E}_k$	excitatory neurons in subpopulation k, $\mathcal{E}_k \cap \mathcal{E}_l = \emptyset$ $(\forall k \neq l \in [1, M])$	n_E
$\mathcal{E} = \bigcup_{k=1}^{M} \mathcal{E}_k$	excitatory (E) neurons	N_E
$\mathcal{I}_k$	inhibitory neurons in subpopulation k, $\mathcal{I}_k \cap \mathcal{I}_l = \emptyset$ $(\forall k \neq l \in [1, M])$	n_I
$\mathcal{I} = \bigcup_{k=1}^{M} \mathcal{I}_k$	inhibitory (I) neurons	N_I
$\mathcal{X}_k$	spike generator for bottom-up input to subpopulation k, $\mathcal{X}_k \cap \mathcal{X}_l = \emptyset$ $(\forall k \neq l \in [1, M])$	1
$\mathcal{X} = \bigcup_{k=1}^{M} \mathcal{X}_k$	bottom-up spike generators	M
$\mathcal{T}_i$	spike generator triggering top-down prediction of sequence i, $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ $(\forall i \neq j \in [1, S])$	1
$\mathcal{T} = \bigcup_{i=1}^{S} \mathcal{T}_i$	spike generators triggering top-down predictions	S
$\pi_{i,1}$	excitatory neurons representing the first element of sequence i ("assembly"; $\forall i \in [1, S]$)	ρ

Table 7.2: Description of the populations. Parameter values are given in Tab. 7.10.

Connectivity		
Source	**Target**	**Pattern**
$\mathcal{E}$	$\mathcal{E}$	random; fixed in-degrees $K_i = K_{EE}$, delays $d_{ij} = d_{EE}$, and synaptic time constants $\tau_{ij} = \tau_{EE}$, plastic synaptic weights J_{ij} $(\forall i \in \mathcal{E}, \forall j \in \mathcal{E}$; "EE connections"); no self-connections ("autapses"), no multiple connections ("multapses")
$\mathcal{E}_k$	$\mathcal{I}_k$	all-to-all, fixed delays $d_{ij} = d_{IE}$, synaptic time constants $\tau_{ij} = \tau_{IE}$, and weights $J_{ij} = J_{IE}$ $(\forall k \in [1, M], \forall i \in \mathcal{I}_k, \forall j \in \mathcal{E}_k$; "IE connections")
$\mathcal{I}$	$\mathcal{E}$	all-to-all; fixed delays $d_{ij} = d_{EI}$, synaptic time constants $\tau_{ij} = \tau_{EI}$, and weights $J_{ij} = J_{EI}$ $(\forall i \in \mathcal{E}, \forall j \in \mathcal{I}$; "EI connections")
$\mathcal{X}_k$	$\mathcal{E}_k$	all-to-all, fixed delays $d_{ij} = d_{EX}$, synaptic time constants $\tau_{ij} = \tau_{EX}$, and weights $J_{ij} = J_{EX}$ $(\forall k \in [1, M], \forall i \in \mathcal{E}_k, \forall j \in \mathcal{X}_k$; "EX connections")
$\mathcal{T}_h$	$\pi_{h,1}$	all-to-all, fixed delays $d_{ij} = d_{ET}$, synaptic time constants $\tau_{ij} = \tau_{ET}$, and weights $J_{ij} = J_{ET}$ $(\forall h \in [1, S], \forall i \in \approx_{h,1}, \forall j \in \mathcal{T}_h$; "ET connections"); no multiple connections ("multapses")
all unmentioned connections are absent		

Table 7.3: Description of the connectivity. Parameter values are given in Table Tab. 7.10.

Neuron	
Type	leaky integrate-and-fire (LIF) dynamics
Description	dynamics of membrane potential $V_i(t)$ and spiking activity $\xi_i(t)$ of neuron i: • emission of the kth spike of neuron i at time t_i^k if $V_i(t_i^k) \geq \theta_i$ (7.1) with somatic spike threshold θ_i • spike train: $\xi_i(t) = \sum_k \delta(t - t_i^k)$ • reset and refractoriness: $V_i(t) = V_\mathrm{r} \quad \forall k,\ \forall t \in \left(t_i^k,\ t_i^k + \tau_{\mathrm{ref},i}\right]$ with refractory time $\tau_{\mathrm{ref},i}$ and reset potential V_r • subthreshold dynamics: $\tau_{\mathrm{m},i}\dot{V}_i(t) = -V_i(t) + R_{\mathrm{m},i} I_i(t)$ (7.2) with membrane resistance $R_{\mathrm{m},i} = \frac{\tau_{\mathrm{m},i}}{C_{\mathrm{m},i}}$, membrane time constant $\tau_{\mathrm{m},i}$, and total synaptic input current $I_i(t)$ (see Tab. 7.5) • excitatory neurons: $\tau_{\mathrm{m},i} = \tau_{\mathrm{m,E}}$, $C_{\mathrm{m},i} = C_\mathrm{m}$, $\theta_i = \theta_\mathrm{E}$, $\tau_{\mathrm{ref},i} = \tau_{\mathrm{ref,E}}$ $(\forall i \in \mathcal{E})$ • inhibitory neurons: $\tau_{\mathrm{m},i} = \tau_{\mathrm{m},I}$, $C_{\mathrm{m},i} = C_\mathrm{m}$, $\theta_i = \theta_\mathrm{I}$, $\tau_{\mathrm{ref},i} = \tau_{\mathrm{ref,I}}$ $(\forall i \in \mathcal{I})$

Table 7.4: Description of the neuron model. Parameter values are given in Tab. 7.10.

Synapse	
Type	continuous, exponential, or alpha-shaped postsynaptic currents (PSCs)
Description	• total synaptic input current excitatory neurons: $$I_i(t) = I_{\mathrm{ED},i}(t) + I_{\mathrm{EX},i}(t) + I_{\mathrm{EI},i}(t),\ \forall i \in \mathcal{E} \tag{7.3}$$ inhibitory neurons: $$I_i(t) = I_{\mathrm{IE},i}(t),\ \forall i \in \mathcal{I} \tag{7.4}$$ with dendritic, inhibitory, excitatory, and external input currents $I_{\mathrm{ED},i}(t)$, $I_{\mathrm{EI},i}(t)$, $I_{\mathrm{IE},i}(t)$, $I_{\mathrm{EX},i}(t)$ evolving according to $$I_{\mathrm{ED},i}(t) = \sum_{j\in\mathcal{E}} (\alpha_{ij} * \xi_j)(t - d_{ij}) + \sum_{\{k\in\mathcal{T}\mid i\in\pi_{k,1}\}} (\alpha_{ik} * \xi_k)(t - d_{ik}), \tag{7.5}$$ $$\tau_{\mathrm{EI}} \dot{I}_{\mathrm{EI},i} = -I_{\mathrm{EI},i}(t) + \sum_{j\in\mathcal{I}} J_{ij}\xi_j(t - d_{ij}), \tag{7.6}$$ $$\tau_{\mathrm{IE}} \dot{I}_{\mathrm{IE},i} = -I_{\mathrm{IE},i}(t) + \sum_{j\in\mathcal{E}} J_{ij}\xi_j(t - d_{ij}), \tag{7.7}$$ $$I_{\mathrm{EX},i}(t) = I_{\mathrm{S},i}(t) + I_{\mathrm{B},i}(t) \tag{7.8}$$ with alpha-function kernel $\alpha_{il}(t) = J_{il}\frac{e}{\tau_{\mathrm{ED}}} t e^{-t/\tau_{\mathrm{ED}}}\Theta(t)$, Heaviside function $\Theta(t) = \begin{cases} 1 & t \geq 0 \\ 0 & \text{else} \end{cases}$, and $I_{\mathrm{S},i}(t)$ and $I_{\mathrm{B},i}(t)$ being the stimulus and the background input (see Tab. 7.7:Input). • suprathreshold dynamics of dendritic currents (dAP generation): ◦ emission of kth dAP of neuron i at time $t^k_{\mathrm{dAP},i}$ if $I_{\mathrm{ED},i}(t^k_{\mathrm{dAP},i}) \geq \theta_{\mathrm{dAP}}$ ◦ dAP current plateau: $$I_{\mathrm{ED},i}(t) = I_{\mathrm{dAP}} \quad \forall k,\ \forall t \in \left(t^k_{\mathrm{dAP},i}, t^k_{\mathrm{dAP},i} + \tau_{\mathrm{dAP}}\right) \tag{7.9}$$ with dAP current plateau amplitude I_{dAP}, dAP current duration τ_{dAP}, and dAP activation threshold θ_{dAP} ◦ reset: $I_{\mathrm{ED},i}(t^k_{\mathrm{dAP},i} + \tau_{\mathrm{dAP}}) = 0$ $(\forall k)$ ◦ reset and refractoriness in response to emission of lth somatic spike of neuron i at time t^l_i: $$I_{\mathrm{ED},i}(t) = 0 \quad \forall l,\ \forall t \in \left(t^l_i, t^l_i + \tau_{\mathrm{ref},i}\right) \tag{7.10}$$

Table 7.5: Description of the synapse model. Parameter values are given in Tab. 7.10.

Plasticity	
Type	spike-timing dependent structural plasticity and permanence decay
EE synapses	• dynamics of synaptic permanence $P_{ij}(t)$ (maturity) in EE connections during learning: $\forall P_{\min} < P_{ij} < P_{\max}:$ $\frac{dP_{ij}}{dt} =$ $P_{\max}\lambda_+ \sum_{\{t_i^*\}'} \delta(t-[t_i^*+d_{\mathrm{EE}}])I_+(x_j(t),t_i^*,\Delta t_{\min},\Delta t_{\max})$ $-P_{\max}\lambda_- \sum_{\{t_j^*\}'} \delta(t-[t_j^*+d_{\mathrm{EE}}])I_-(x_i(t),t_j^*,\Delta t_{\max})$ (7.11) $+(P_{\min}-P)\frac{1}{\tau_P}$ $\forall\{t\|P_{ij}(t)<P_{\min}\}: \quad P_{ij}(t)=P_{\min}$ $\forall\{t\|P_{ij}(t)>P_{\max}\}: \quad P_{ij}(t)=P_{\max}$ with ◦ list of presynaptic spike times $\{t_j^*\}$, ◦ list of postsynaptic spike times $\{t_i^*\}' = \{t_i^*\|\forall t_j^*: t_i^*-t_j^*+d_{\mathrm{EE}} \geq \Delta t_{\min}\}$ ◦ increment functions $I_+(x_j(t),t_i^*,\Delta t_{\min},\Delta t_{\max}) = R_+(t_i^*-t_j^++d_{\mathrm{EE}})$ with $R_+(\tau) = \begin{cases} x_j(t) & \Delta t_{\min}<\tau<\Delta t_{\max} \\ (x_j(t)-1) & \tau<\Delta t_{\min} \\ 0 & \text{else,} \end{cases}$ (7.12) $I_-(x_i(t),t_j^*,\Delta t_{\max}) = R_-(t_j^*-t_i^-+d_{\mathrm{EE}})$ with $R_-(\tau) = \begin{cases} x_i(t) & \tau<\Delta t_{\max} \\ 0 & \text{else,} \end{cases}$ ◦ maximum permanence $P_{\max}$, minimum permanence $P_{\max}$, potentiation and depression rates λ_+, λ_-, decay time constant τ_P, delay d_{EE}, minimum $\Delta t_{\min}$ and maximum $\Delta t_{\max}$ time lags between pairs of pre- and postsynaptic spikes at which synapses are potentiated or depressed, nearest presynaptic spike time t_j^+ preceding t_i^*, nearest postsynaptic spike time t_i^- preceding t_j^*,

Table 7.6: Description of the plasticity model. Continued on the next page.

Plasticity	
EE synapses	◦ spike trace of presynaptic neuron j, evolving according to $$\frac{dx_j}{dt} = -\tau_+^{-1} x_j(t) + \sum_{t_j^*} \delta(t - t_j^*)$$ with presynaptic spike times t_j^* and potentiation time constant τ_+, ◦ spike trace of postynaptic neuron i, evolving according to $$\frac{dx_i}{dt} = -\tau_-^{-1} x_i(t) + \sum_{t_i^*} \delta(t - t_i^*)$$ with postynaptic spike times t_i^* and depression time constant τ_-. • dynamics of synaptic weights $J_{\mathrm{EE},ij}$ according to $$J_{\mathrm{EE},ij} = \begin{cases} J_{\max} & P_{ij} > P_\theta \\ 0 & \text{else,} \end{cases}$$ with maximum synaptic weight $J_{\max}$ and synapse maturity threshold P_θ.
all other synapses	non-plastic

Table 7.6: Description of the plasticity model. Parameter values are given in Tab. 7.10.

Input

- prediction mode:
  - repetitive stimulation of the network using the same set $\mathcal{S} = \{s_1, \ldots, s_S\}$ of sequences $s_i = \{\zeta_{i,1}, \zeta_{i,2}, \ldots, \zeta_{i,C_i}\}$ of ordered discrete items $\zeta_{i,j}$ with number of sequences S, length C_i of ith sequence, and total number of presentations n_{epoch}
  - presentation of sequence element $\zeta_{i,j}$ at time $t_{i,j}$ modeled by a single spike $\xi_k(t) = \delta(t - t_{i,j})$ generated by the corresponding external, bottom-up source $\mathcal{X}_k$
  - generated current in response to the bottom-up presentation of the sequence elements:

$$\tau_S \dot{I}_{S,i} = -I_{S,i}(t) + \sum_{j \in \mathcal{X}} J_{i,j} \xi_j (t - d_{ij}) \tag{7.13}$$

  - inter-stimulus interval $\Delta T = t_{i,j+1} - t_{i,j}$ between subsequent sequence elements $\zeta_{i,j}$ and $\zeta_{i,j+1}$ within a sequence s_i
  - inter-sequence time interval $\Delta T_{\text{seq}} = t_{i+1,1} - t_{i,C_i}$ between subsequent sequences s_i and s_{i+1}
  - top-down inputs $\mathcal{T}_i$ at times $t_{i,1} - \Delta T_{\text{td}}$ corresponding to the first elements $\zeta_{i,1}$ of all sequences s_i ($\forall i \in [1, S]$) to the dendrites of neurons in the respective assembly $\pi_{i,1}$
  - example sequence sets:
    - sequence set I: $\mathcal{S}$={{A,F,B,D}, {A,F,C,E}}
    - sequence set II: $\mathcal{S}$={{A,F,B,D}, {A,F,C,E}, {A,F,G,H}, {A,F,I,J}, {A,F,K,L}}

Table 7.7: Description of the input. Continued on the next page.

Input

- replay mode:
  - initiation of replay by stimulation of the minicolumn, corresponding to the first token of the sequence, by a single spike $\xi_k(t) = \delta(t - t_{i,1})$ at time $t_{i,1}$, generated by the corresponding external bottom-up source $\mathcal{X}_k$
  - top-down inputs $\mathcal{T}_i$ at times $t_{i,1} - \Delta T_{\text{td}}$ corresponding to the first elements $\zeta_{i,1}$ of all sequences s_i ($\forall i \in [1, S]$) to the dendrites of neurons in the respective assembly $\pi_{i,1}$
  - background current $I_{\text{B}}(t) = I_{\text{ext}}(t) + I_{\text{noise}}(t)$ with external input $I_{\text{ext}}(t)$ and background noise $I_{\text{noise}}(t)$
    - external input:

$$I_{\text{ext}}(t) = \begin{cases} \bar{I} & \text{(constant input)} \\ \bar{I} + a\sin(2\pi f t + \phi) & \text{(oscillatory input)} \end{cases} \tag{7.14}$$

      with offset $\bar{I}$, oscillation amplitude a, frequency f, and phase ϕ
    - noise input:

$$\tau_{\text{noise}} \dot{I}_{\text{noise},i} = -I_{\text{noise},i}(t) + J_{\text{noise}}[x_{\text{E},i}(t) - x_{\text{I},i}(t)] \tag{7.15}$$

      with excitatory and inhibitory Poissonian spike sources $x_{\text{E},i}(t)$ and $x_{\text{I},i}(t)$, synaptic weight J_{noise}, and time constant τ_{noise}

      statistics of background noise:
      - noise amplitude

$$\sigma = \sqrt{\frac{J_{\text{noise}}^2 \nu R^2 \tau_{\text{noise}}^2}{\tau_{\text{noise}} - \tau_{\text{m}}}} \tag{7.16}$$

        with spike rate ν of $x_{\text{E},i}(t)$ and $x_{\text{I},i}(t)$, representing standard deviation of the membrane potential of target neurons in the absence other inputs and a spike threshold
      - noise correlation

$$c_{ij} = \begin{cases} c & \text{if } \exists k : i, j \in \mathcal{M}_k \\ 0 & \text{else} \end{cases} \tag{7.17}$$

        representing correlation coefficient between noise input currents $I_{\text{noise},i}(t)$ and $I_{\text{noise},j}(t)$

Table 7.7: Description of the input. Parameter values are given in Tab. 7.10.

Performance measures

- prediction error:

$$\epsilon = \frac{1}{C}\sqrt{\sum_{k=1}^{M}(o_k - v_k)^2}, \tag{7.18}$$

with binary vector $\boldsymbol{o} \in \{0,1\}^M$, where $o_k = 1$ if subpopulation k is predictive (a minimum of half of assembly size ρ neurons elicit a plateau potential), and $o_k = 0$ otherwise. Binary target vector $\boldsymbol{v}$ encodes the identity of the subpopulation receiving the current input.

  - false positive rate:

$$\mathrm{fp} = \frac{1}{C}\sum_{k=1}^{M}\Theta\left(o_k - v_k\right), \tag{7.19}$$

quantifies predictive activity in subpopulations not targeted by the current input.

  - false negative rate:

$$\mathrm{fn} = \frac{1}{C}\sum_{k=1}^{M}\Theta\left(v_k - o_k\right), \tag{7.20}$$

quantifies failure to predict the target subpopulation.

- replay speed:
  - inter-element interval τ_k: time between the kth and $(k+1)$th assembly activation during replay ($k = 1, \ldots, C-1$)
  - mean inter-element interval:

$$\langle\tau_k\rangle_k = \frac{1}{4}\sum_{k=C-4}^{C-1}\tau_k, \tag{7.21}$$

averaged over the last four elements to exclude transient dynamics at replay onset

  - replay speed:

$$f_{\mathrm{out}} = \frac{1}{\langle\tau_k\rangle_k}, \tag{7.22}$$

defined as the inverse of the mean inter-element interval, quantifying the rate at which neuronal assemblies are activated during replay

Table 7.8: Description of the performance measurement. Continued on the next page.

Performance measures

- variability of replay timing:
  - coefficient of variation of inter-element intervals across input oscillation phases:

$$\mathrm{CV}_\phi(\tau_k) = \frac{\mathrm{std}_\phi(\tau_k)}{\langle \tau_k \rangle_\phi}, \tag{7.23}$$

    where $\langle \tau_k \rangle_\phi$ and $\mathrm{std}_\phi(\tau_k)$ denote the mean and standard deviation of the kth inter-element interval across the set of tested oscillation phases ϕ, respectively; quantifies the sensitivity of the kth inter-element interval to the oscillation phase at replay onset
  - coefficient of variation of inter-element intervals across sequence elements:

$$\mathrm{CV}_k(\tau_k) = \frac{\mathrm{std}_k(\tau_k)}{\langle \tau_k \rangle_k}, \tag{7.24}$$

    where $\langle \tau_k \rangle_k$ and $\mathrm{std}_k(\tau_k)$ denote the mean and standard deviation of the inter-element interval across the last four elements $k = C-4, \ldots, C-1$, respectively; quantifies the temporal regularity of replay, i.e., how uniformly the sequence is traversed across successive assembly activations

Table 7.8: Description of the performance measurement.

Initial conditions and network realizations

- membrane potentials: $V_i(0) = V_\mathrm{r}$ ($\forall i \in \mathcal{E} \cup \mathcal{I}$)
- dendritic currents: $I_{\mathrm{ED},i}(0) = 0$ ($\forall i \in \mathcal{E}$)
- bottom-up currents: $I_\mathrm{EX}(0) = 0$ ($\forall i \in \mathcal{E}$)
- inhibitory currents: $I_{\mathrm{EI},i}(0) = 0$ ($\forall i \in \mathcal{E}$)
- excitatory currents: $I_{\mathrm{IE},i}(0) = 0$ ($\forall i \in \mathcal{I}$)
- synaptic permanences: $P_{ij}(0) \sim \mathcal{U}(P_{0,\min}, P_{0,\max})$ (uniform distribution; $\forall i, j \in \mathcal{E}$)
- synaptic weights: $J_{ij}(0) = 0$ ($\forall i, j \in \mathcal{E}$)
- spike traces: $x_i(0) = 0$ ($\forall i \in \mathcal{E}$)
- connectivity and initial weights are randomly and independently drawn for each network realization

Table 7.9: Description of the initial conditions of the network. Parameter values are given in Tab. 7.10.

7.2 MODEL AND SIMULATION PARAMETERS

Name	Value	Description
Network		
M	6	number of subpopulations
n_E,n_I	200, 1	number of excitatory and inhibitory neurons per subpopulation
N_E	1200	number of excitatory neurons; $N_E = M\,n_E$
N_I	6	number of inhibitory neurons; $N_I = M\,n_I$
ρ	20	(target) number of active neurons per subpopulation after learning = minimal number of coincident excitatory inputs required to trigger a spike in postsynaptic inhibitory neurons
N_X	6	number of bottom-up spike sources; determined by vocabulary size
N_T	1	number of top-down spike sources; determined by number of training sequences
(Potential) Connectivity		
K_{EE}	240	number of excitatory inputs per excitatory neuron (EE in-degree)
p	$K_{EE}/N_E = 0.2$	connection probability
Excitatory neurons		
$\tau_{m,E}$	10 ms	membrane time constant
$\tau_{ref,E}$	10 ms	absolute refractory period
C_m	250 pF	membrane capacity
V_r	0.0 mV	reset potential
θ_E	10 mV	somatic spike threshold
I_{dAP}	200 pA	dAP current plateau amplitude
τ_{dAP}	100 ms	dAP duration
θ_{dAP}	59 pA	dAP threshold

Table 7.10: Model and simulation parameters (continued on next page).

Name	Value	Description
Inhibitory neurons		
$\tau_{m,I}$	5 ms	membrane time constant
$\tau_{ref,I}$	2 ms	absolute refractory period
C_m	250 pF	membrane capacity
V_r	0.0 mV	reset potential
θ_I	15 mV	spike threshold
Synapse		
J_{IE}	$\sim$ 678.06 pA	weight of IE connections (EPSC amplitude)
J_{EI}	~ -12915.49 pA	weight of EI connections (IPSC amplitude)
J_{EX}	$\sim$ 4112.20 pA	weight of EX connections targeting the somas (EPSC amplitude)
J_{ET}	$\sim 1.4\,\theta_{dAP}$	weight of ET connections targeting dendrites (EPSC amplitude)
J_{EN}	$\sim [0, 53.04]$ pA	weight of EN connections (EPSC amplitude)
τ_{EE}	5 ms	synaptic time constant of EE connections
τ_{EI}	1 ms	synaptic time constant of EI connections
τ_{IE}	0.5 ms	synaptic time constant of IE connections
τ_{EX}	2 ms	synaptic time constant of EX connection
τ_{ET}	5 ms	synaptic time constant of ET connections
τ_{EN}	2 ms	synaptic time constant of EN connections
d_{EE}	2 ms	delay of EE connections (dendritic)
d_{IE}	0.1 ms	delay of IE connections
d_{EI}	0.1 ms	delay of EI connections
d_{EX}	0.1 ms	delay of EX connections
d_{ET}	2 ms	delay of ET connections (dendritic)

Table 7.10: Model and simulation parameters (continued on next page).

Name	Value	Description
	Plasticity	
λ_+	0.39	potentiation rate
λ_-	0.55	depression rate
P_{ij}	$[P_{\min}, P_0]$	synaptic permanence
P_θ	20	synapse maturity threshold
$P_{\max}$	20	permanence upper bound
$P_{\min}$	1	permanence lower bound
$P_{0,\max}$	8	permanence initialization upper bound
$P_{0,\min}$	0	permanence initialization upper bound
$J_{\max}$	12.98 pA	maximum weight
τ_+	20 ms	plasticity time constant (potentiation)
τ_-	20 ms	plasticity time constant (depression)
τ_P	27.5 s	permanence leak time constant
$\Delta t_{\min}$	4 ms	minimum time lag between pairs of pre- and post-synaptic spikes at which synapses are potentiated given the spike trace of the presynaptic neuron of the current time step
$\Delta t_{\max}$	50 ms	maximum time lag between pairs of pre- and post-synaptic spikes at which synapses are potentiated
	Input	
S	1	number of sequences per sequence set
C_i	$\{8, 16, 24\}$	sequence length
n_{epoch}	500	total number of sequence set presentations during learning
ΔT	40 ms	inter-stimulus interval (during training)
ΔT_{seq}	$\mathcal{U}(100, 105)$ ms (uniform distribution)	inter-sequence interval
ΔT_{td}	10 ms	time interval between top-down and bottom-up input
ν	$10000\,\text{s}^{-1}$	rate of Poissonian spike sources generating background noise
σ	$[0, 3]$ mV	noise amplitude
c	0.9	noise correlation
J_{noise}	$[0, 53.04]$ pA	synaptic weight of inputs generating background noise; $J_{\text{noise}} = \sqrt{\frac{\sigma^2 (\tau_{\text{noise}} - \tau_{\text{m}})}{\nu R^2 \tau_{\text{noise}}^2}}$
	Simulation	
Δt	0.1 ms	time resolution

Table 7.10: Model and simulation parameters.

SUPPLEMENTARY MATERIAL: EXPLOITING NETWORK TOPOLOGY IN BRAIN-SCALE NETWORK SIMULATIONS

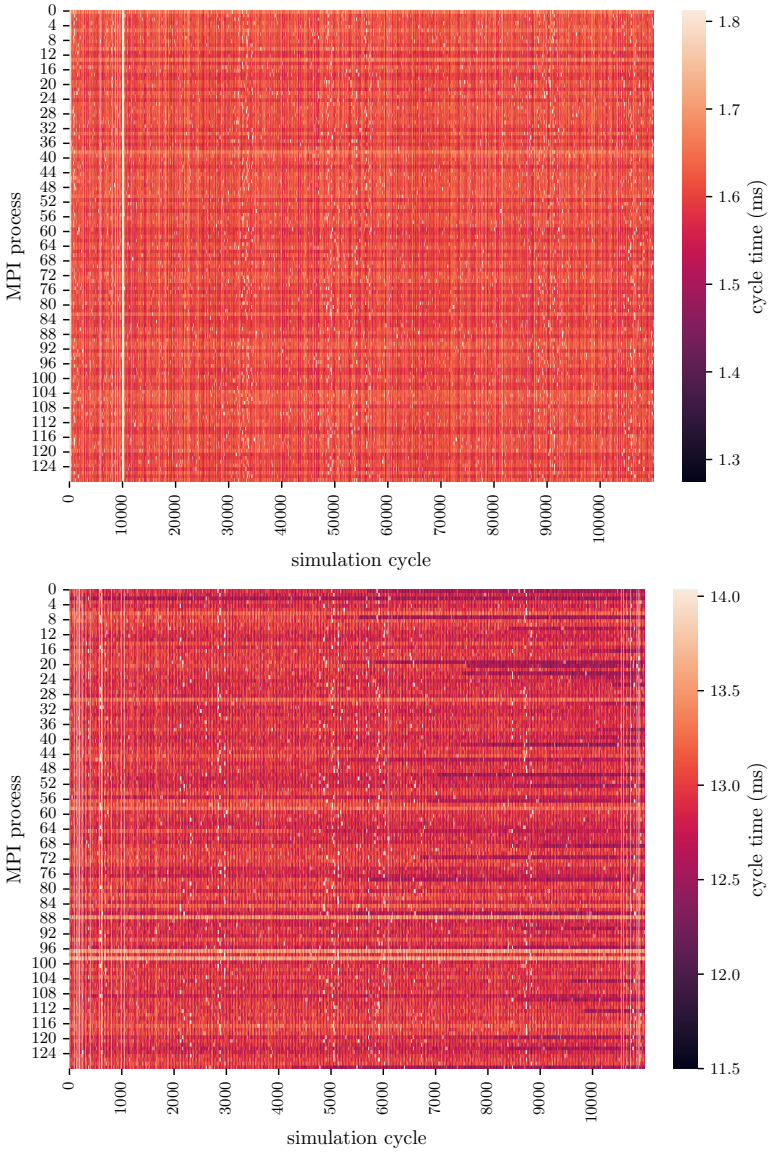

Figure 8.1: Temporal evolution of the cycle time during a MAM-benchmark simulation with 128 MPI processes (seed 654) on SuperMUC-NG, for the conventional (top) and the structure-aware strategy (bottom).

BIBLIOGRAPHY

Abbott, L. F., & Nelson, S. B. (2000). Synaptic plasticity: taming the beast. *Nat. Neurosci. 3*, 1178–1183. doi: 10.1038/81453. *(Cited on page 5)*

Abeles, M. (1991). *Corticonics: Neural Circuits of the Cerebral Cortex* (1st ed.). Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511574566. *(Cited on pages 4 and 5)*

Adrian, E. D. (1926). The impulses produced by sensory nerve endings: Part i. *J. Physiol. 61*(1), 49–72. doi: 10.1113/jphysiol.1926.sp002273. *(Cited on page 6)*

Ahmad, S., & Hawkins, J. (2016). How do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites. *ArXiv*, 1601.00720. doi: 10.48550/arxiv.1601.00720. *(Cited on pages 12, 18, 24, 31, and 97)*

Aimone, J. B., Awile, O., Diesmann, M., Knight, J. C., Nowotny, T., & Schürmann, F. (2023). Editorial: Neuroscience, computing, performance, and benchmarks: Why it matters to neuroscience how fast we can compute. *Front. Neuroinform. 17*. doi: 10.3389/fninf.2023.1157418. *(Cited on page 101)*

Akaho, S. (1993). Capacity and error correction ability of sparsely encoded associative memory with forgetting process. In *ICANN'93: Proceedings of the International Conference on Artificial Neural Networks Amsterdam, The Netherlands 13–16 September 1993 3*, pp. 707–710. Springer. *(Cited on page 20)*

Albers, J., Pronold, J., Kurth, A. C., Vennemo, S., Haghighi Mood, K., Patronis, A., Terhorst, D., Jordan, J., Kunkel, S., Tetzlaff, T., Diesmann, M., & Senk, J. (2022). A modular workflow for performance benchmarking of neuronal network simulations. *Front. Neuroinform. 16*, 837549. doi: 10.3389/fninf.2022.837549. *(Cited on pages 11, 12, 71, 82, 90, and 91)*

Anderson, P. W. (1972). More is different. *Science 177*(4047), 393–396. *(Cited on pages 11 and 82)*

Antic, S. D., Zhou, W.-L., Moore, A. R., Short, S. M., & Ikonomu, K. D. (2010). The decade of the dendritic nmda spike. *J. Neurosci. Res. 88*(14), 2991–3001. *(Cited on pages 8, 29, and 32)*

Anzt, H., Chen, Y.-C., Cojean, T., Dongarra, J., Flegar, G., Nayak, P., Quintana-Ortí, E. S., Tsai, Y. M., & Wang, W. (2019). Towards continuous benchmarking: An automated performance evaluation framework for high performance software. In *Proceedings of the platform for advanced scientific computing conference*, pp. 1–11. *(Cited on page 82)*

Attwell, D., & Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *J. Cereb. Blood Flow Metab. 21*, 1133–1145. doi: 10.1097/00004647-200110000-00001. *(Cited on page 6)*

Barth, A. L., & Poulet, J. F. (2012). Experimental evidence for sparse firing in the neocortex. *Trends in Neurosciences 35*(6), 345–355. doi: https://doi.org/10.1016/j.tins.2012.03.008. *(Cited on pages 6 and 50)*

Bastian, P., Kranzlmüller, D., Brüchle, H., & Mathias, G. (2024). High performance computing in science and engineering – garching/munich 2024 (results of projects on supermuc-ng). *Leibniz Supercomputing Centre*. doi: 10.25927/8qx9r-n2j13. *(Cited on page 76)*

Baxter, R., Hong, N. C., Gorissen, D., Hetherington, J., & Todorov, I. (2012). The research software engineer. In *Digital Research 2012*, pp. 1–3. Oxford, United Kingdom. https://www.pure.ed.ac.uk/ws/portalfiles/portal/65195747/DR2012_12_1_.pdf. *(Cited on pages 11 and 12)*

Benson-Amram, S., Dantzer, B., Stricker, G., Swanson, E. M., & Holekamp, K. E. (2016). Brain size predicts problem-solving ability in mammalian carnivores. *Proc. Natl. Acad. Sci. USA 113*(9), 2532–2537. doi: 10.1073/pnas.1505913113. *(Cited on pages 50 and 100)*

Berti, L., Giorgi, F., & Kasneci, G. (2025). Emergent abilities in large language models: A survey. https://arxiv.org/abs/2503.05788. *(Cited on pages 50 and 100)*

Bi, G., & Poo, M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci. 18*, 10464–10472. *(Cited on pages 4, 6, and 9)*

Biewald, L. (2020). Experiment tracking with weights and biases. https://www.wandb.com. Software available from wandb.com. *(Cited on page 31)*

Billeh, Y. N., Cai, B., Gratiy, S. L., Dai, K., Iyer, R., Gouwens, N. W., Abbasi-Asl, R., Jia, X., Siegle, J. H., Olsen, S. R., et al. (2020). Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex. *Neuron 106*(3), 388–403.e18. doi: https://doi.org/10.1016/j.neuron.2020.01.040. *(Cited on page 7)*

Binzegger, T., Douglas, R. J., & Martin, K. A. C. (2004). A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci. 39*(24), 8441–8453. doi: 10.1523/JNEUROSCI.1400-04.2004. *(Cited on pages 5 and 7)*

Bizo, L., Chu, J., Sanabria, F., & Killeen, P. (2006). The failure of weber's law in time perception and production. *Behavioural processes 71*, 201–10. doi: 10.1016/j.beproc.2005.11.006. *(Cited on page 43)*

Blom, G. (1958). *Statistical estimates and transformed beta-variables*. Phd thesis, University of Stockholm. *(Cited on pages 58 and 60)*

Boltz, M. G. (2010). Rate and duration memory of naturalistic sounds. *Acta Psychologica 135*(2), 168–181. doi: https://doi.org/10.1016/j.actpsy.2010.06.004. *(Cited on pages 7, 12, and 28)*

Boucsein, C., Nawrot, M., Schnepel, P., & Aertsen, A. (2011). Beyond the cortical column: abundance and physiology of horizontal connections imply a strong role for inputs from the surround. *Front. Neurosci. 5*, 32. doi: 10.3389/fnins.2011.00032. *(Cited on page 5)*

Bouhadjar, Y., Diesmann, M., Waser, R., Wouters, D. J., & Tetzlaff, T. (2019). Constraints on sequence processing speed in biological neuronal networks. In *Proceedings of the International Conference on Neuromorphic Systems*, pp. 1–9. *(Cited on page 31)*

Bouhadjar, Y., Siegel, S., Tetzlaff, T., Diesmann, M., Waser, R., & Wouters, D. J. (2023). Sequence learning in a spiking neuronal network with memristive synapses. *Neuromorphic Comput. Eng. 3*(3), 034014. doi: 10.1088/2634-4386/acf1c4. *(Cited on page 102)*

Bouhadjar, Y., Lober, M., Netfci, E., Diesmann, M., & Tetzlaff, T. (2025a). Unsupervised continual learning of complex sequences in spiking neuronal networks. In *Proceedings of the International Conference on Neuromorphic Systems (ICONS'25)*. *(Cited on page vii)*

Bouhadjar, Y., Lober, M., Neftci, E., Diesmann, M., & Tetzlaff, T. (2025b). Unsupervised online learning of complex sequences in spiking neuronal networks. In *Proceedings of the International Conference on Neuromorphic Systems (ICONS'25)*, pp. 202–207. doi: 10.1109/ICONS69015.2025.00038. *(Cited on pages 18, 30, and 31)*

Bouhadjar, Y., Wouters, D. J., Diesmann, M., & Tetzlaff, T. (2022). Sequence learning, prediction, and replay in networks of spiking neurons. *PLOS Comput. Biol. 18*(6), e1010233. doi: 10.1371/journal.pcbi.1010233. *(Cited on pages 8, 12, 18, 21, 28, 29, and 33)*

Bouhadjar, Y., Wouters, D. J., Diesmann, M., & Tetzlaff, T. (2023). Coherent noise enables probabilistic sequence replay in spiking neuronal networks. *PLOS Comput. Biol. 19*(5), e1010989. doi: 10.1371/journal.pcbi.1010989. *(Cited on pages 8, 18, 38, and 40)*

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci. 8*(3), 183–208. doi: 10.1023/a:1008925309027. *(Cited on page 4)*

Buch, E. R., Claudino, L., Quentin, R., Bönstrup, M., & Cohen, L. G. (2021). Consolidation of human skill linked to waking hippocampo-neocortical replay. *Cell Reports 35*(10), 109193. doi: https://doi.org/10.1016/j.celrep.2021.109193. *(Cited on pages 7 and 28)*

Buonomano, D. V., & Merzenich, M. M. (1998). Cortical plasticity: From synapses to maps. *Annu. Rev. Neurosci. 21*(1), 149–186. doi: 10.1146/annurev.neuro.21.1.149. *(Cited on page 4)*

Burns, S. P., Xing, D., & Shapley, R. M. (2011). Is Gamma-band activity in the local field potential of V1 cortex a "clock" or filtered noise? *J. Neurosci. 31*(26), 9658–9664. doi: 10.1523/JNEUROSCI.0660-11.2011. *(Cited on page 43)*

Buxhoeveden, D. P., & Casanova, M. F. (2002). The minicolumn hypothesis in neuroscience. *Brain 125*(5), 935–951. doi: 10.1093/brain/awf110. *(Cited on page 29)*

Buzsáki, G. (2006). *Rhythms of the Brain*. Oxford University Press. doi: 10.1093/acprof:oso/9780195301069.001.0001. *(Cited on page 29)*

Buzsáki, G. (2015). Hippocampal sharp wave-ripple: A cognitive biomarker for episodic memory and planning. *Hippocampus 25*(10), 1073–1188. doi: https://doi.org/10.1002/hipo.22488. *(Cited on pages 37, 45, and 101)*

Calderon, C. B., Verguts, T., & Frank, M. J. (2022). Thunderstruck: The acdc model of flexible sequences and rhythms in recurrent neural circuits. *PLoS Computational Biology 18*. https://api.semanticscholar.org/CorpusID:233245482. *(Cited on page 44)*

Cao, R., Bladon, J. H., Charczynski, S. J., Hasselmo, M. E., & Howard, M. W. (2022). Internally generated time in the rodent hippocampus is logarithmically compressed. *eLife 11*, e75353. doi: 10.7554/eLife.75353. *(Cited on page 45)*

Carr, M. F., Karlsson, M. P., & Frank, L. M. (2012). Transient slow gamma synchrony underlies hippocampal memory replay. *Neuron 75 4*, 700–13. https://api.semanticscholar.org/CorpusID:15574688. *(Cited on pages 45 and 101)*

Citri, A., & Malenka, R. C. (2007). Synaptic plasticity: Multiple forms, functions, and mechanisms. *Neuropharmacology 33*(1), 18–41. doi: 10.1038/sj.npp.1301559. *(Cited on page 4)*

Clark, D. G., & Abbott, L. (2024). Theory of coupled neuronal-synaptic dynamics. *Phys. Rev. X 14*(2), 021001. doi: 10.1103/PhysRevX.14.021001. *(Cited on page 20)*

Clegg, B. A., DiGirolamo, G. J., & Keele, S. W. (1998). Sequence learning. *Trends. Cogn. Sci. 2*(8), 275–281. *(Cited on page 3)*

Cone, I., & Shouval, H. Z. (2021). Learning precise spatiotemporal sequences via biophysically realistic learning rules in a modular, spiking network. *eLife 10*, e63751. doi: 10.7554/elife.63751. *(Cited on page 44)*

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. K., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro 38*(1), 82–99. *(Cited on page 10)*

Dayan, P., & Abbott, L. F. (2001). *Theoretical Neuroscience*. Cambridge, MA: MIT Press. *(Cited on page 4)*

de Kock, C. P. J., & Sakmann, B. (2009). Spiking in primary somatosensory cortex during natural whisking in awake head-restrained rats is cell-type specific. *Proc. Natl. Acad. Sci. USA 106*(38), 16446–16450. doi: 10.1073/pnas.0904143106. *(Cited on page 6)*

Deger, M., Helias, M., Rotter, S., & Diesmann, M. (2012). Spike-timing dependence of structural plasticity explains cooperative synapse formation in the neocortex. *PLOS Comput. Biol. 8*(9), e1002689. doi: 10.1371/journal.pcbi.1002689. *(Cited on page 8)*

Dehaene, S., Meyniel, F., Wacongne, C., Wang, L., & Pallier, C. (2015). The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees. *Neuron 88*(1), 2–19. *(Cited on pages 3 and 18)*

Denker, M., Roux, S., Lindén, H., Diesmann, M., Riehle, A., & Grün, S. (2011). The local field potential reflects surplus spike synchrony. *Cereb. Cortex 21*, 2681–2695. doi: 10.1093/cercor/bhr040. *(Cited on page 43)*

Deutsche Forschungsgemeinschaft (DFG) (2024). FOR 5880: Holistic Energy and Performance Modeling for Sustainable Computing (Mod4Comp). https://gepris.dfg.de/gepris/projekt/545776403. *(Cited on page 78)*

Dietrich, R., Spilger, P., Müller, E., Schemmel, J., & Knoll, A. C. (2025). *Sequence Learning with Analog Neuromorphic Multi-Compartment Neurons and On-Chip Structural STDP*, pp. 207–230. Springer Nature Switzerland. *(Cited on pages 24 and 102)*

Djurfeldt, M., Hjorth, J., Eppler, J. M., Dudani, N., Helias, M., Potjans, T. C., Bhalla, U. S., Diesmann, M., Hellgren Kotaleski, J., & Ekeberg, O. (2010). Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics 8*, 43–60. *(Cited on page 101)*

Duarte, R., Zajzon, B., Schulte to Brinke, T., & Morrison, A. (2021). Functional neural architectures. doi: 10.5281/zenodo.5752597. *(Cited on page 31)*

Eccles, J., P, F., & Koketsu, K. (1954). Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones. *J. Physiol. 126*, 524–562. *(Cited on page 3)*

Eichenbaum, H. (2014). Time cells in the hippocampus: a new dimension for mapping memories. *Nat. Rev. Neurosci. 15*(11), 732–744. doi: 10.1038/nrn3827. *(Cited on page 45)*

Eliasmith, C., & Trujillo, O. (2014). The use and abuse of large-scale brain models. *Current Opinion in Neurobiology 25*, 1–6. doi: 10.1016/j.conb.2013.09.009. *(Cited on page 7)*

Eliav, T., Maimon, S. R., Sarel, A., Palgi, S., Las, L., & Ulanovsky, N. (2025). Fragmented replay of very large environments in the hippocampus of bats. *Cell 188*(15), 4091–4105.e16. doi: https://doi.org/10.1016/j.cell.2025.05.024. *(Cited on pages 41 and 46)*

Eppler, J. M., Helias, M., Muller, E., Diesmann, M., & Gewaltig, M. (2009). PyNEST: a convenient interface to the NEST simulator. *Front. Neuroinform. 2*, 12. doi: 10.3389/neuro.11.012.2008. *(Cited on page 70)*

EuroHPC, J. U. (2024). Governing board of the EuroHPC joint undertaking no 11/2024 amending the joint undertaking's work programme and budget for the year 2024 (amendment no 1). *ref. EC Regulations (EU) 2018/1488 and (EU) 2021/1173*. *(Cited on page 103)*

Euston, D., Tatsuno, M., & Mcnaughton, B. (2007). Fast-forward playback of recent memory sequences in prefrontal cortex during sleep. *Science (New York, N.Y.) 318*, 1147–50. doi: 10.1126/science.1148979. *(Cited on pages 7, 28, 37, 45, and 101)*

Felleman, D. J., & Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex 1*, 1–47. doi: 10.1093/cercor/1.1.1-a. *(Cited on page 5)*

Fernandez-Musoles, C., Coca, D., & Richmond, P. (2019). Communication sparsity in distributed spiking neural network simulations to improve scalability. *Front. Neuroinform. 13*, 19. doi: 10.3389/fninf.2019.00019. *(Cited on pages 9, 12, 50, and 51)*

Festa, D., Aschner, A., Davila, A., Kohn, A., & Coen-Cagli, R. (2021). Neuronal variability reflects probabilistic inference tuned to natural image statistics. *Nat. Commun. 12*(1), 3635. doi: 10.1038/s41467-021-23838-x. *(Cited on page 6)*

Fischl, B., & Dale, A. M. (2000). Measuring the thickness of the human cerebral cortex from magnetic resonance images. *Proc. Natl. Acad. Sci. USA 97*(20), 11050–11055. doi: 10.1073/pnas.200033797. *(Cited on page 5)*

Fries, P. (2005). A mechanism for cognitive dynamics: neuronal communication through neuronal coherence. *Trends. Cogn. Sci. 9*(10), 474–480. doi: 10.1016/j.tics.2005.08.011. *(Cited on page 29)*

Fujimoto, R. M. (1990). Parallel discrete event simulation. *Commun. ACM 33*(10), 30–53. doi: 10.1145/84537.84545. *(Cited on page 9)*

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., & Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, pp. 97–104. *(Cited on page 56)*

Gavornik, J. P., & Shouval, H. Z. (2011). A network of spiking neurons that can represent interval timing: mean field analysis. *J. Comput. Neurosci. 30*(2), 501–513. doi: 10.1007/s10827-010-0275-y. *(Cited on pages 44 and 45)*

Gewaltig, M.-O., & Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia J. 2*(4), 1430. doi: 10.4249/scholarpedia.1430. *(Cited on pages 9, 22, 31, 82, and 88)*

Gibbon, J. (1977). Scalar expectancy theory and weber's law in animal timing. *Psychological Review 84*(3), 279–325. doi: 10.1037/0033-295X.84.3.279. *(Cited on page 43)*

Girard, P., Hupé, J. M., & Bullier, J. (2001). Feedforward and feedback connections between areas v1 and v2 of the monkey have similar rapid conduction velocities. *J. Neurophysiol. 85*(3), 1328–1331. doi: 10.1152/jn.2001.85.3.1328. *(Cited on pages 5 and 50)*

Goddard, N., Hood, G., Howell, F., Hines, M., & De Schutter, E. (2001). Neosim: Portable large-scale plug and play modelling. *Neurocomputing 38-40*, 1657–1661. doi: https://doi.org/10.1016/S0925-2312(01)00528-8. Computational Neuroscience: Trends in Research 2001. *(Cited on pages 10 and 50)*

Goedeke, S., & Diesmann, M. (2008). The mechanism of synchronization in feed-forward neuronal networks. *New. J. Phys. 10*, 015007. doi: 10.1088/1367-2630/10/1/015007. *(Cited on page 40)*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. *(Cited on page 31)*

Graber, S., Mitchell, J., Kurth, A. C., Terhorst, D., Skaar, J.-E. W., Schöfmann, C. M., Kunkel, S., Trensch, G., Haug, N., Mallett, D., Andriyovich, P. Y., Otazu Porter, X., Lee, A. Y., & Plesser, H. E. (2024a). NEST 3.8. doi: 10.5281/ZENODO.12624783. *(Cited on page 31)*

Graber, S., Mitchell, J., Kurth, A. C., Terhorst, D., Skaar, J.-E. W., Schöfmann, C. M., Kunkel, S., Trensch, G., Haug, N., Mallett, D., Andriyovich, P. Y., Otazu Porter, X., Lee, A. Y., & Plesser, H. E. (2024b). Nest 3.8. doi: 10.5281/zenodo.12624784. *(Cited on page 89)*

Grundler, J. (2025). Inferring axonal conduction delays from anatomical data and analyzing their effects on network dynamics in brain-scale simulations of spiking neurons. Master thesis, RWTH Aachen University. *(Cited on page 75)*

Gupta, A. S., van der Meer, M. A., Touretzky, D. S., & Redish, A. D. (2010). Hippocampal replay is not a simple function of experience. *Neuron 65*(5), 695–705. doi: https://doi.org/10.1016/j.neuron.2010.01.034. *(Cited on page 46)*

Hahne, J., Diaz, S., Patronis, A., Schenck, W., Peyser, A., Graber, S., Spreizer, S., Vennemo, S. B., Ippen, T., Mørk, H., Jordan, J., Senk, J., Konradi, S., Weidel, P., Fardet, T., Dahmen, D., Terhorst, D., Stapmanns, J., Trensch, G., van Meegen, A., Pronold, J., Eppler, J. M., Linssen, C., Morrison, A., Sinha, A., Mitchell, J., Kunkel, S., Deepu, R., Hagen, E., Vierjahn, T., Kamiji, N. L., de Schepper, R., Machado, P., Albers, J., Klijn, W., Myczko, A., Mayner, W., Nagendra Babu, P., Jiang, H., Billaudelle, S., Vogler, B. S., Miotto, G., Kusch, L., Antonietti, A., Morales-Gregorio, A., Dolderer, J., Bouhadjar, Y., & Plesser, H. E. (2021). Nest 3.0. doi: 10.5281/zenodo.4739103. *(Cited on page 72)*

Hammarlund, P., & Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *J. Comput. Neurosci. 5*(4), 443–459. *(Cited on pages 10, 50, and 101)*

Hardy, N. F., Goudar, V., Romero-Sosa, J. L., & Buonomano, D. V. (2018). A model of temporal scaling correctly predicts that motor timing improves with speed.

Nature Communications 9(1), 4732. doi: 10.1038/s41467-018-07161-6. *(Cited on pages 28 and 41)*

Hass, J., & Durstewitz, D. (2016). Time at the center, or time at the side? assessing current models of time perception. *Current Opinion in Behavioral Sciences 8*, 238–244. doi: https://doi.org/10.1016/j.cobeha.2016.02.030. Time in perception and action. *(Cited on pages 42, 43, and 44)*

Hass, J., Blaschke, S., Rammsayer, T., & Herrmann, J. (2008). A neurocomputational model for optimal temporal processing. *Journal of Computational Neuroscience 25*, 449–64. doi: 10.1007/s10827-008-0088-4. *(Cited on pages 44, 45, and 103)*

Haug, N., Welle Skaar, J.-E., Mørk, H., Spreizer, S., Morrison, A., de Schepper, R., Kurth, A., Trensch, G., Terhorst, D., Lober, M., Albers, J., Linssen, C., Benelhedi, M. A., Villamar, J., Eppler, J. M., Vogelsang, J., Mitchell, J., Deepu, R., Kunkel, S., Nagendra Babu, P., Graber, S., Schöfmann, C. M., Vennemo, S. B., Grundler, J., & Plesser, H. E. (2023). Nest 3.5. doi: 10.5281/zenodo.8069926. *(Cited on page 89)*

Hawkins, J., & Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Front. Neural Circuits 10*, 23. doi: 10.3389/fncir.2016.00023. *(Cited on pages 8, 12, 18, 24, 31, and 97)*

Hawkins, J., & Blakeslee, S. (2007). *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan. *(Cited on pages 3, 8, and 18)*

Hawkins, J., Ahmad, S., Dubinsky, D., et al. (2011). Cortical learning algorithm and Hierarchical Temporal Memory. *Numenta Whitepaper 1*(68), 2. *(Cited on page 8)*

Hayot-Sasson, V., Hudson, N., Bauer, A., Gonthier, M., Foster, I., & Chard, K. (2025). Addressing reproducibility challenges in hpc with continuous integration. https://arxiv.org/abs/2508.21289. *(Cited on page 87)*

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. New York: John Wiley & Sons. doi: 10.1002/sce.37303405110. *(Cited on page 4)*

Herculano-Houzel, S. (2017). Numbers of neurons as biological correlates of cognitive capability. *Curr. Opin. Behav. Sci. 16*, 1–7. doi: https://doi.org/10.1016/j.cobeha.2017.02.004. Comparative cognition. *(Cited on pages 50 and 100)*

Herten, A., Achilles, S., Alvarez, D., Badwaik, J., Behle, E., Bode, M., Breuer, T., Caviedes-Voullième, D., Cherti, M., Dabah, A., Sayed, S. E., Frings, W., Gonzalez-Nicolas, A., Gregory, E. B., Mood, K. H., Hater, T., Jitsev, J., John, C. M., Meinke, J. H., Meyer, C. I., Mezentsev, P., Mirus, J.-O., Nassyr, S., Penke, C., Römmer, M., Sinha, U., Vieth, B. v. S., Stein, O., Suarez, E., Willsch, D., & Zhukov, I. (2024). Application-driven exascale: The jupiter benchmark suite. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–45. doi: 10.1109/SC41406.2024.00038. *(Cited on page 88)*

Hocquet, A., Wieber, F., Gramelsberger, G., Hinsen, K., Diesmann, M., Pasquini Santos, F., Landström, C., Peters, B., Kasprowicz, D., Borrelli, A., Roth, P., Lee, C. A. L., Olteanu, A., & Böschen, S. (2024). Software in science is ubiquitous yet overlooked. *Nature Computational Science*. doi: 10.1038/s43588-024-00651-2. *(Cited on pages 93 and 101)*

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol. 117*, 500–544. doi: 10.1113/jphysiol.1952.sp004764. *(Cited on page 3)*

Holtmaat, A., & Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. *Nature Reviews Neuroscience 10*(9), 647–658. doi: 10.1038/nrn2699. *(Cited on page 4)*

Holtmaat, A., Wilbrecht, L., Knott, G. W., Welker, E., & Svoboda, K. (2006). Experience-dependent and cell-type-specific spine growth in the neocortex. *Nature 441*, 979–983. doi:10.1038/nature04783. *(Cited on page 20)*

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA 79*, 2554–2558. *(Cited on page 7)*

Horn, D., & Usher, M. (1989). Neural networks with dynamical thresholds. *Phys. Rev. A 40*(2), 1036. doi: 10.1103/PhysRevA.40.1036. *(Cited on page 44)*

Ji, D., & Wilson, M. (2007). Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature neuroscience 10*, 100–7. doi: 10.1038/nn1825. *(Cited on pages 7, 28, 37, 45, and 101)*

Johansson, C., & Lansner, A. (2007). Imposing biological constraints onto an abstract neocortical attractor network model. *Neural Comput. 19*, 1871–1896. *(Cited on page 29)*

Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., Diesmann, M., & Kunkel, S. (2018). Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers. *Front. Neuroinform. 12*, 2. doi: 10.3389/fninf.2018.00002. *(Cited on pages 9, 12, 50, 73, and 88)*

Kampa, B. M., Letzkus, J. J., & Stuart, G. J. (2007). Dendritic mechanisms controlling spike-timing-dependent synaptic plasticity. *Trends Neurosci. 30*(9), 456–463. *(Cited on page 4)*

Kandel, E. R., Schwartz, J. H., & Jessel, T. M. (1991). *Principles of neural science* (3 ed.). New York: Elsevier. *(Cited on page 3)*

Kauth, K., Stadtmann, T., Brandhofer, R., Sobhani, V., & Gemmeke, T. (2020). Communication architecture enabling 100x accelerated simulation of biological neural networks. In *Proceedings of the Workshop on System-Level Interconnect: Problems and Pathfinding Workshop*, pp. 1–8. *(Cited on page 24)*

Kauth, K., Stadtmann, T., Sobhani, V., & Gemmeke, T. (2023). neuroaix-framework: design of future neuroscience simulation systems exhibiting execution of the

cortical microcircuit model 20× faster than biological real-time. *Front. Comput. Neurosci. 17*. doi: 10.3389/fncom.2023.1144143. *(Cited on page 24)*

Kayser, C., Montemurro, M. A., Logothetis, N. K., & Panzeri, S. (2009). Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns. *Neuron 61*(4), 597–608. doi: https://doi.org/10.1016/j.neuron.2009.01.008. *(Cited on page 43)*

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal 49*(2), 291–307. doi: 10.1002/j.1538-7305.1970.tb01770.x. *(Cited on page 9)*

Klinzing, J. G., Niethard, N., & Born, J. (2019). Mechanisms of systems memory consolidation during sleep. *Nature Neuroscience 22*, 1598 – 1610. https://api.semanticscholar.org/CorpusID:201657155. *(Cited on pages 37, 45, and 101)*

Kozloski, J., & Wagner, J. (2011). An ultrascalable solution to large-scale neural tissue simulation. *Front. Neuroinform. 5*, 15. doi: 10.3389/fninf.2011.00015. *(Cited on pages 10, 50, and 101)*

Krause, D. (2019). JUWELS: Modular tier-0/1 supercomputer at jülich supercomputing centre. *J. Large-scale Res. Facil. JLSRF 5*(A135). *(Cited on page 88)*

Kriener, B., Enger, H., Tetzlaff, T., Plesser, H. E., Gewaltig, M.-O., & Einevoll, G. T. (2014). Dynamics of self-sustained asynchronous-irregular activity in random networks of spiking neurons with strong synapses. *Front. Comput. Neurosci. 8*, 136. doi: 10.3389/fncom.2014.00136. *(Cited on pages 44 and 45)*

Kunkel, S., Schmidt, M., Eppler, J. M., Masumoto, G., Igarashi, J., Ishii, S., Fukai, T., Morrison, A., Diesmann, M., & Helias, M. (2014). Spiking network simulation code for petascale computers. *Front. Neuroinform. 8*, 78. doi: 10.3389/fninf.2014.00078. *(Cited on page 89)*

Kurth, A. C., Albers, J., Diesmann, M., & van Albada, S. J. (2025). Cell-type specific projection patterns promote balanced activity in cortical microcircuits. *BioRxiv*. doi: 10.1101/2024.10.03.616539. *(Cited on page 5)*

Kurth, A. C., Senk, J., Terhorst, D., Finnerty, J., & Diesmann, M. (2022). Sub-realtime simulation of a neuronal network of natural density. *Neuromorphic Comput. Eng. 2*(2), 021001. doi: 10.1088/2634-4386/ac55fc. *(Cited on pages 10 and 90)*

Laborieux, A., Ernoult, M., Hirtzlin, T., & Querlioz, D. (2021). Synaptic metaplasticity in binarized neural networks. *Nature Communications 12*(1). doi: 10.1038/s41467-021-22768-y. *(Cited on page 25)*

Lansner, A., Marklund, P., Sikström, S., & Nilsson, L.-G. (2013). Reactivation in working memory: an attractor network model of free recall. *PLOSONE 8*(8), e73776. doi: 10.1371/journal.pone.0073776. *(Cited on page 44)*

Lapicque, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarization. *J. Physiol. Pathol. Gen. 9*, 620–635. *(Cited on page 4)*

Larkum, M. E., Nevian, T., Sandler, M., Polsky, A., & Schiller, J. (2009). Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: A new unifying principle. *Science 325*(5941), 756–760. doi: 10.1126/science.1171958. *(Cited on page 32)*

Lashley, K. S. (1951). *The problem of serial order in behavior*, Volume 21. Bobbs-Merrill. *(Cited on page 3)*

Levitin, D. J., & Cook, P. R. (1996). Memory for musical tempo: Additional evidence that auditory memory is absolute. *Perception & Psychophysics 58*(6), 927–935. *(Cited on pages 7, 12, and 28)*

Liang, Q., Zeng, Y., & Xu, B. (2020). Temporal-sequential learning with a brain-inspired spiking neural network and its application to musical memory. *Frontiers in Computational Neuroscience Volume 14 - 2020*. doi: 10.3389/fncom.2020.00051. *(Cited on page 44)*

Liao, D., Hessler, N. A., & Malinow, R. (1995). Activation of postsynaptically silent synapses during pairing-induced ltp in ca1 region of hippocampal slice. *Nature 375*(6530), 400–404. *(Cited on pages 5 and 8)*

Linssen, C., Babu, P. N., Bouhadjar, Y., Ewert, L., Wybo, W., Lober, M., Feller, F., Rumpe, B., & Morrison, A. (2024). NESTML 8.0.0. doi: 10.5281/zenodo.12191059. *(Cited on pages 22 and 31)*

Linssen, C., Babu, P. N., Eppler, J. M., Koll, L., Rumpe, B., & Morrison, A. (2025). NESTML: a generic modeling language and code generation tool for the simulation of spiking neural networks with advanced plasticity rules. *Front. Neuroinform. 19*. doi: 10.3389/fninf.2025.1544143. *(Cited on pages 22, 31, and 71)*

Liu, Y., Dolan, R. J., Kurth-Nelson, Z., & Behrens, T. E. (2019). Human replay spontaneously reorganizes experience. *Cell 178*(3), 640–652.e14. doi: https://doi.org/10.1016/j.cell.2019.06.012. *(Cited on pages 7 and 28)*

Lober, M., Bouhadjar, Y., Diesmann, M., & Tetzlaff, T. (2026a). Learning sequence timing and control of replay speed in networks of spiking neurons. https://arxiv.org/abs/2605.22523. *(Cited on page vii)*

Lober, M., Bouhadjar, Y., Diesmann, M., & Tetzlaff, T. (2026b). Code and data for lober et al. (2026), learning sequence timing and control of replay speed in networks of spiking neurons. doi: 10.5281/zenodo.20290700. *(Cited on page 32)*

Lober, M., Diesmann, M., & Kunkel, S. (2026). Exploiting network topology in brain-scale simulations of spiking neural networks. *Neuromorphic Computing and Engineering 6*(2), 024024. doi: 10.1088/2634-4386/ae762e. *(Cited on page vii)*

Lundqvist, M., Rehn, M., Djurfeldt, M., & Lansner, A. (2006). Attractor dynamics in a modular network model of neocortex. *Netw. Comput. Neural Syst. 17*(3), 253–276. doi: 10.1080/09548980600774619. *(Cited on pages 29, 44, and 45)*

Lüscher, C., Noicoll, R., Malenka, R., & Muller, D. (2000). Synaptic plasticity and dynamic modulation of the postsynaptic membrane. *Nat. Neurosci. 3*(6), 545–550. *(Cited on page 8)*

Lyu, T., Sato, M., Aoki, S., Himeno, R., & Sun, Z. (2024). Cortex: Large-scale brain simulator utilizing indegree sub-graph decomposition on fugaku supercomputer. https://arxiv.org/abs/2406.03762. *(Cited on pages 10 and 101)*

Maass, W. (2000). On the computational power of winner-take-all. *Neural Comput. 12*(11), 2519–2535. *(Cited on pages 8 and 29)*

Maes, A., Barahona, M., & Clopath, C. (2020). Learning spatiotemporal signals using a recurrent spiking network that discretizes time. *PLOS Comput. Biol. 16*(1), e1007606. doi: 10.1371/journal.pcbi.1007606. *(Cited on page 44)*

Mainen, Z. F., & Sejnowski, T. J. (1995). Reliability of spike timing in neocortical neurons. *Science 268*, 1503–1506. *(Cited on pages 40 and 43)*

Malakasis, N., Chavlis, S., & Poirazi, P. (2023). Synaptic turnover promotes efficient learning in bio-realistic spiking neural networks. In *2023 57th Asilomar Conference on Signals, Systems, and Computers*, pp. 942–949. IEEE. *(Cited on page 20)*

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science 275*, 213–215. *(Cited on page 4)*

Martinez, R. H., Lansner, A., & Herman, P. (2019). Probabilistic associative learning suffices for learning the temporal structure of multiple sequences. *PLOSONE 14*(8), e0220161. doi: 10.1371/journal.pone.0220161. *(Cited on page 44)*

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE 78*(10), 1629–1636. doi: 10.1109/5.58356. *(Cited on page 10)*

Message Passing Interface Forum (2021). *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf. *(Cited on pages 9 and 51)*

Migliore, M., Cannia, C., Lytton, W. W., Markram, H., & Hines, M. (2006). Parallel network simulations with NEURON. *J. Comput. Neurosci. 21*(2), 119–223. doi: 10.1007/s10827-006-7949-5. *(Cited on page 53)*

Milojkovic, B. A. (2005). A strict correlation between dendritic and somatic plateau depolarizations in the rat prefrontal cortex pyramidal neurons. *J. Neurosci. 25*(15), 3940–3951. doi: 10.1523/jneurosci.5314-04.2005. *(Cited on page 32)*

Mohns, M., Diesmann, M., Grün, S., & Aertsen, A. (1999). Interaction of synchronous input activity and subthreshold oscillatons of membrane potential. In N. Elsner & U. Eysel (Eds.), *From Molecular Neurobiology to Clinical Neuroscience, Proceedings of the 27th Göttingen Neurobiology Conference*, Volume I, Stuttgart, pp. 100. Thieme. *(Cited on page 40)*

Monteiro, T., Rodrigues, F., Pexirra, M., Cruz, B., Gonçalves, A., Rueda-Orozco, P., & Paton, J. (2023). Using temperature to analyze the neural basis of a time-based decision. *Nature Neuroscience 26*, 1–10. doi: 10.1038/s41593-023-01378-5. *(Cited on page 42)*

Morrison, A., Mehring, C., Geisel, T., Aertsen, A., & Diesmann, M. (2005). Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Comput. 17*(8), 1776–1801. doi: 10.1162/0899766054026648. *(Cited on pages 9, 51, and 53)*

Mountcastle, V. (1978). An organizing principle for cerebral function: the unit module and the distributed system. *The mindful brain*. *(Cited on page 8)*

Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain 120*, 701–722. *(Cited on page 29)*

Oberkampf, W. L., & Roy, C. J. (2010). *Verification and Validation in Scientific Computing*. Cambridge University Press. doi: 10.1017/cbo9780511760396. *(Cited on pages 11 and 82)*

Palm, G. (1982). *Neural Assemblies. An Alternative Approach to Artificial Intelligence*, Volume 7 of *Studies of Brain Function*. Berlin: Springer-Verlag-Verlag. *(Cited on pages 29 and 44)*

Papoulis, A., & Pillai, S. U. (2002). *Probability, Random Variables, and Stochastic Processes* (4th ed.). Boston: McGraw-Hill. *(Cited on page 60)*

Pearce, O., Scott, A., Becker, G., Haque, R., Hanford, N., Brink, S., Jacobsen, D., Poxon, H., Domke, J., & Gamblin, T. (2023). Towards collaborative continuous benchmarking for hpc. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '23, New York, NY, USA, pp. 627–635. Association for Computing Machinery. doi: 10.1145/3624062.3624135. *(Cited on page 82)*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830. *(Cited on page 31)*

Pehle, C., Billaudelle, S., Cramer, B., Kaiser, J., Schreiber, K., Stradmann, Y., Weis, J., Leibfried, A., Müller, E., & Schemmel, J. (2022). The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Front. Neurosci. 16*, 795876. doi: 10.3389/fnins.2022.795876. *(Cited on pages 10 and 24)*

Pereira, U., & Brunel, N. (2020). Unsupervised learning of persistent and sequential activity. *Front. Comput. Neurosci. 13*, 97. doi: 10.3389/fncom.2019.00097. *(Cited on page 44)*

Pfeiffer, B. E., & Foster, D. J. (2013). Hippocampal place cell sequences depict future paths to remembered goals. *Nature 497*, 74 – 79. https://api.semanticscholar.org/CorpusID:175756. *(Cited on page 46)*

Plesser, H. E., Eppler, J. M., Morrison, A., Diesmann, M., & Gewaltig, M.-O. (2007). Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In A.-M. Kermarrec, L. Bougé, & T. Priol (Eds.), *Euro-Par 2007: Parallel Processing*, Volume 4641 of *Lecture Notes in Computer Science*, Berlin, pp. 672–681. Springer-Verlag. doi: 10.1007/978-3-540-74466-5_71. *(Cited on page 54)*

Plesser, H. E., Davison, A. P., Diesmann, M., Fukai, T., Gemmeke, T., Gleeson, P., Knight, J. C., Nowotny, T., René, A., Rhodes, O., Roque, A. C., Senk, J., Schwalger, T., Stadtmann, T., Tiddia, G., & van Albada, S. J. (2025). Building on models—a perspective for computational neuroscience. *Cereb. Cortex 35*(11), bhaf295. doi: 10.1093/cercor/bhaf295. *(Cited on page 101)*

Plotnikov, D., Blundell, I., Ippen, T., Eppler, J. M., Rumpe, B., & Morrison, A. (2016). NESTML: a modeling language for spiking neurons. In A. Oberweis & R. Reussner (Eds.), *Modellierung 2016*, Volume P-254 of *Lecture Notes in Informatics (LNI)*, pp. 93–108. Modellierung 2016, Karlsruhe (Germany), 17 Mar 2016 - 19 Mar 2016: Gesellschaft für Informatik e.V. (GI). http://juser.fz-juelich.de/record/826510. *(Cited on pages 22, 31, and 71)*

Potjans, T. C., & Diesmann, M. (2014). The cell-type specific cortical microcircuit: Relating structure and activity in a full-scale spiking network model. *Cereb. Cortex 24*(3), 785–806. doi: 10.1093/cercor/bhs358. *(Cited on pages 75 and 89)*

Pronold, J., Jordan, J., Wylie, B. J. N., Kitayama, I., Diesmann, M., & Kunkel, S. (2022a). Routing brain traffic through the von Neumann bottleneck: Efficient cache usage in spiking neural network simulation code on general purpose computers. *Parallel Comput. 113*, 102952. doi: 10.1016/j.parco.2022.102952. *(Cited on pages 9, 12, 50, 53, 78, and 90)*

Pronold, J., Jordan, J., Wylie, B. J. N., Kitayama, I., Diesmann, M., & Kunkel, S. (2022b). Routing brain traffic through the von Neumann bottleneck: Parallel sorting and refactoring. *Front. Neuroinform. 15*, 785068. doi: 10.3389/fninf.2021.785068. *(Cited on pages 53 and 90)*

Pronold, J., Morales-Gregorio, A., Rostami, V., & van Albada, S. J. (2024). Cortical multi-area model with joint excitatory-inhibitory clusters accounts for spiking statistics, inter-area propagation, and variability dynamics. *BioRxiv*. doi: 10.1101/2024.01.30.577979. *(Cited on page 77)*

Protze, J., Hermanns, M.-A., Müller, M. S., Nguyen, V. M., Jaeger, J., Saillard, E., Carribault, P., & Barthou, D. (2022). Mpi detach — towards automatic asynchronous local completion. *Parallel Computing 109*, 102859. doi: https://doi.org/10.1016/j.parco.2021.102859. *(Cited on page 79)*

Ramon y Cajal, S. (1899). *Textura del Systema Nervioso del Hombre y de los Vertebrados. (Texture of the Nervous System of Man and the Vertebrates)*, Volume 1. New York: Springer. *(Cited on page 3)*

Rhodes, O., Peres, L., Rowley, A. G. D., Gait, A., Plana, L. A., Brenninkmeijer, C., & Furber, S. B. (2019). Real-time cortical simulation on neuromorphic hardware. *Philos. Trans. R. Soc. A 378*(2164), 20190160. doi: 10.1098/rsta.2019.0160. *(Cited on pages 10 and 11)*

Riehle, A., Grün, S., Diesmann, M., & Aertsen, A. (1997). Spike synchronization and rate modulation differentially involved in motor cortical function. *Science 278*, 1950–1953. *(Cited on page 6)*

Rotter, S., & Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biol. Cybern. 81*(5-6), 381–402. doi: 10.1007/s004220050570. *(Cited on pages 22 and 32)*

Roxin, A., Riecke, H., & Solla, S. A. (2004). Self-sustained activity in a small-world network of excitable neurons. *Phys. Rev. Lett. 92*(19), 198101. *(Cited on page 44)*

Schiller, J., Major, G., Koester, H. J., & Schiller, Y. (2000). Nmda spikes in basal dendrites of cortical pyramidal neurons. *Nature 404*(6775), 285–289. doi: 10.1038/35005094. *(Cited on page 32)*

Schmidt, M., Bakker, R., Hilgetag, C. C., Diesmann, M., & van Albada, S. J. (2018a). Multi-scale account of the network structure of macaque visual cortex. *Brain Struct. Funct. 223*(3), 1409–1435. doi: 10.1007/s00429-017-1554-4. *(Cited on pages 5, 7, 12, 50, 51, 62, 74, and 77)*

Schmidt, M., Bakker, R., Shen, K., Bezgin, G., Diesmann, M., & van Albada, S. J. (2018b). A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLOS Comput. Biol. 14*(10), e1006359. doi: 10.1371/journal.pcbi.1006359. *(Cited on pages 12, 51, 62, 74, 77, and 89)*

Segundo, J., Moore, G., Stensaas, L., & Bullock, T. (1963). Sensitivity of neurones in aplysia to temporal pattern of arriving impulses. *J. Exp. Biol. 40*(4), 643–667. doi: 10.1242/jeb.40.4.643. *(Cited on page 6)*

Senk, J., Kriener, B., Djurfeldt, M., Voges, N., Jiang, H.-J., Schüttler, L., Gramelsberger, G., Diesmann, M., Plesser, H. E., & van Albada, S. J. (2022). Connectivity concepts in neuronal network modeling. *PLOS Comput. Biol. 18*(9), e1010086. doi: 10.1371/journal.pcbi.1010086. *(Cited on pages 19 and 30)*

Shadlen, M. N., & Newsome, W. T. (1998). The variable discharge of cortical neurons: Implications for connectivity, computation, and information coding. *J. Neurosci. 18*(10), 3870–3896. doi: 10.1523/jneurosci.18-10-03870.1998. *(Cited on page 6)*

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access 5*, 3909–3943. doi: 10.1109/access.2017.2685629. *(Cited on pages 11 and 82)*

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE 104*(1), 148–175. doi: 10.1109/JPROC.2015.2494218. *(Cited on page 31)*

Shapson-Coe, A., Januszewski, M., Berger, D. R., Pope, A., Wu, Y., Blakely, T., Schalek, R. L., Li, P. H., Wang, S., Maitin-Shepard, J., Karlupia, N., Dorkenwald, S., Sjostedt, E., Leavitt, L., Lee, D., Bailey, L., Fitzmaurice, A., Kar, R., Field, B., Wu, H., Wagner-Carena, J., Aley, D., Lau, J., Lin, Z., Wei, D., Pfister, H., Peleg, A., Jain, V., & Lichtman, J. W. (2021). A connectomic study of a petascale fragment of human cerebral cortex. *BioRxiv*. doi: 10.1101/2021.05.29.446289. *(Cited on page 5)*

Shapson-Coe, A., Januszewski, M., Berger, D. R., Pope, A., Wu, Y., Blakely, T., Schalek, R. L., Li, P. H., Wang, S., Maitin-Shepard, J., et al. (2024). A petavoxel fragment of human cerebral cortex reconstructed at nanoscale resolution. *Science 384*(6696), eadk4858. doi: 0.1126/science.adk4858. *(Cited on page 7)*

Siegel, S., Bouhadjar, Y., Tetzlaff, T., Waser, R., Dittmann, R., & Wouters, D. J. (2023). System model of neuromorphic sequence learning on a memristive crossbar array. *Neuromorphic Comput. Eng. 3*(2), 024002. doi: 10.1088/2634-4386/acca45. *(Cited on page 102)*

Speck, R., & Wyatt, C. (2024). Research software engineering. In M. Diesmann, J. Kowalski, & B. Rumpe (Eds.), *RWTH Themen - Research Software Engineering*, Volume 1/2024 of *RWTH Themen*, pp. 8–10. *(Cited on pages 11, 12, and 93)*

Stepanyants, A., Hirsch, J. A., Martinez, L. M., Kisvárday, Z. F., Ferecskó, A. S., & Chklovskii, D. B. (2007). Local potential connectivity in cat primary visual cortex. *Cereb. Cortex 18*(1), 13–28. doi: 10.1093/cercor/bhm027. *(Cited on page 5)*

Swadlow, H. A. (1988). Efferent neurons and suspected interneurons in binocular visual cortex of the awake rabbit: Receptive fields and binocular properties. *J. Neurophysiol. 59*(4), 1162–1187. doi: 10.1152/jn.1988.59.4.1162. *(Cited on page 6)*

Tetzlaff, C., Kolodziejski, C., Timme, M., & Wörgötter, F. (2011). Synaptic scaling in combination with many generic plasticity mechanisms stabilizes circuit connectivity. *Front. Comput. Neurosci. 5*, 47. *(Cited on page 5)*

Thibeault, C. M., Minkovich, K., O'Brien, M. J., Harris, F. C., & Srinivasa, N. (2013). Efficiently passing messages in distributed spiking neural network simulation. *Frontiers in Computational Neuroscience 7*, 77. doi: 10.3389/fncom.2013.00077. *(Cited on page 79)*

Thörnig, P., & von St. Vieth, B. (2021). JURECA: Data Centric and Booster Modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre. *JLSRF 7*, A182. doi: 10.17815/jlsrf-7-182. *(Cited on pages 76 and 88)*

Tiganj, Z., Jung, M. W., Kim, J., & Howard, M. W. (2017). Sequential firing codes for time in rodent medial prefrontal cortex. *Cereb. Cortex 27*(9), 5663–5671. doi: 10.1093/cercor/bhw336. *(Cited on page 45)*

Toro, R., Perron, M., Pike, B., Richer, L., Veillette, S., Pausova, Z., & Paus, T. (2008). Brain size and folding of the human cerebral cortex. *Cereb. Cortex 18*(10), 2352–2357. doi: 10.1093/cercor/bhm261. *(Cited on page 5)*

van Albada, S. J., Helias, M., & Diesmann, M. (2015). Scalability of asynchronous networks is limited by one-to-one mapping between effective connectivity and correlations. *PLOS Comput. Biol. 11*(9), e1004490. doi: 10.1371/journal.pcbi.1004490. *(Cited on pages 9, 11, and 82)*

van Albada, S. J., Pronold, J., van Meegen, A., & Diesmann, M. (2021). Usage and scaling of an open-source spiking multi-area model of monkey cortex. In *Lecture Notes in Computer Science*, pp. 47–59. Cham, Switzerland: Springer International Publishing. doi: 10.1007/978-3-030-82427-3_4. *(Cited on page 90)*

Viererbl, B., Denner, N., & Koch, T. (2022). "You don't meet anybody when walking from the living room to the kitchen": Informal communication during remote work. *Journal of Communication Management 26*(3), 331–348. doi: 10.1108/JCOM-10-2021-0117. *(Cited on page 82)*

Villamar, J., Vogelsang, J., Linssen, C., Kunkel, S., Kurth, A., Schöfmann, C. M., Benelhedi, M. A., Nagendra Babu, P., Eppler, J. M., de Schepper, R., Mitchell, J., Morrison, A., Haug, N., Diaz, S., Acimovic, J., Graber, S., Jiang, H.-J., Terhorst, D., Spreizer, S., Welle Skaar, J.-E., Stapmanns, J., Manninen, T., Krüger, M., Lehtimäki, M., Ito, S., Lee, A. Y., Lindahl, M., & Plesser, H. E. (2023). NEST 3.6. doi: 10.5281/zenodo.8344932. *(Cited on pages 22 and 70)*

Villamar, J., Kelbling, M., More, H. L., Denker, M., Tetzlaff, T., Senk, J., & Thober, S. (2025). Metadata practices for simulation workflows. *Sci. Data 12*(1), 1–18. doi: 10.1038/s41597-025-05126-1. *(Cited on pages 82, 88, and 93)*

Vogels, T. P., & Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *J. Neurosci. 25*(46), 10786–10795. *(Cited on page 44)*

Vogelsang, J., Lober, M., Schöfmann, C. M., Villamar, J., Terhorst, D., Senk, J., Plesser, H. E., Diesmann, M., Kunkel, S., & Kurth, A. C. (2026). Continuous benchmarking: Keeping pace with an evolving ecosystem of models and technologies. https://arxiv.org/abs/2604.15919. *(Cited on pages vii, 54, and 77)*

Voges, N., Guijarro, C., Aertsen, A., & Rotter, S. (2010). Models of cortical networks with long-range patchy projections. *J. Comput. Neurosci. 28*(1), 137–154. *(Cited on page 50)*

von Neumann, J. (1993). First draft of a report on the EDVAC. *IEEE Ann. Hist. Comput. 15*(4), 27–75. doi: 10.1109/85.238389. *(Cited on page 10)*

Von St. Vieth, B. (2021). JUSUF: Modular tier-2 supercomputing and cloud infrastructure at jülich supercomputing centre. *J. Large-scale Res. Facil. JLSRF 7*(A179). *(Cited on page 88)*

Wagner-Carena, J., Kate, S., Riordan, T., Abbasi-Asl, R., Aman, J., Amster, A., Bodor, A., Brittain, D., Buchanan, J., Buice, M., Bumbarger, D., Collman, F., da Costa, N., Denman, D., de Vries, S., Joyce, E., Kapner, D., King, C., Larkin, J., Lecoq, J., Mahalingam, G., Millman, D., Mölter, J., Morrison, C., Reid, R., Schneider-Mizell, C., Daniel, S., Suckow, S., Takasaki, K., Takeno, M., Torres, R., Vumbaco,

D., Waters, J., Wyrick, D., Yin, W., Zhuang, J., Mihalas, S., & Berteau, S. (2025). Hebb's vision: The structural underpinnings of hebbian assemblies. *bioRxiv*. doi: 10.1101/2025.04.24.649900. *(Cited on page 29)*

Wang, F., Kulkarni, S., Theilman, B., Rothganger, F., Schuman, C., Lim, S.-H., & Aimone, J. B. (2024). Scaling neural simulations in stacs. *Neuromorphic Computing and Engineering 4*(2), 024002. doi: 10.1088/2634-4386/ad3be7. *(Cited on page 10)*

Wennekers, T., & Palm, G. (2009). Syntactic sequencing in Hebbian cell assemblies. *Cogn. Neurodyn. 3*(4), 429–441. doi: 10.1007/s11571-009-9095-z. *(Cited on page 44)*

Xing, D., Shen, Y., Burns, S., Yeh, C.-I., Shapley, R., & Li, W. (2012). Stochastic generation of Gamma-band activity in primary visual cortex of awake and anesthetized monkeys. *J. Neurosci. 32*(40), 13873–13880. doi: doi:10.1523/JNEUROSCI.5644-11.2012. *(Cited on page 43)*

Yamaura, H., Igarashi, J., & Yamazaki, T. (2020). Simulation of a human-scale cerebellar network model on the K computer. *Front. Neuroinform. 14*, 16. doi: 10.3389/fninf.2020.00016. *(Cited on pages 10 and 101)*

Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., & Gao, J. (2022). Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. *ArXiv*. doi: 10.48550/arXiv.2203.03466. *(Cited on page 24)*

Zaytsev, Y. V., & Morrison, A. (2014). CyNEST: a maintainable Cython-based interface for the NEST simulator. *Front. Neuroinform. 8*, 23. doi: 10.3389/fninf.2014.00023. *(Cited on page 70)*

Zenke, F., Agnes, E. J., & Gerstner, W. (2015). Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *Nature communications 6*(1), 1–13. *(Cited on page 25)*

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., & Vasilescu, B. (2017). The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 60–71. IEEE. *(Cited on page 84)*

FUNDING

This thesis contributed to the following projects:

- NeuroSys as part of the initiative "Clusters4Future" funded by the Federal Ministry of Education and Research BMBF (03ZU1106CB, 03ZU2106CB),
- GRK 2416: MultiSenses-MultiScales funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation; 368482240/GRK2416),
- Joint Lab "Supercomputing and Modeling for the Human Brain" (SMHB),
- computing time granted by the JARA Vergabegremium and provided on the JARA Partition of the supercomputer JURECA at Forschungszentrum Jülich (computation grant JINB33),
- EBRAINS 2.0: A Research Infrastructure to Advance Neuroscience And Brain Health funded by the European Union's Horizon Europe Programme under the Specific Grant Agreement No. 101147319 (EBRAINS 2.0 Project),
- Joint Lab HiRSE the Helmholtz Platform for Research Software Engineering — an innovation pool project of the Helmholtz Association,
- HBP SGA3 - Human Brain Project Specific Grant Agreement 3 (2020-04-01 - 2023-03-31) funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 945539,
- HIDA-NORA Mobility Program

The funders had no role in the design of the study, collection, analysis, or interpretation of data, the decision to publish, or the preparation of this thesis.

EIDESSTATTLICHE ERKLÄRUNG

Ich, Melissa Lober,
erkläre hiermit, dass diese Dissertation und die darin dargelegten Inhalte die eigenen sind und selbstständig, als Ergebnis der eigenen originären Forschung, erarbeitet wurden. Hiermit erkläre ich an Eides statt:

- Diese Arbeit wurde vollständig oder größenteils in der Phase als Promovierender dieser Fakultät und Universität angefertigt;
- 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;
- Wenn immer andere eigene oder Veröffentlichungen Dritter herangezogen wurden, wurden diese klar benannt;
- 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;
- Alle wesentlichen Quellen von Unterstützung wurden benannt;
- 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.
- Ein Teil oder Teile dieser Arbeit wurden zuvor veröffentlicht und zwar in: *siehe Publications*

Aachen, May 2026

Melissa Lober

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Band / Volume 122
Controlled single-molecule manipulation
P. Kopp (2026), viii, 136 pp
ISBN: 978-3-95806-887-2

Band / Volume 123
Growth and Excitation Dynamics of Epitaxial 2D Materials and Molecular Layers Studied by Electron and Momentum Microscopy
M. E. Raths (2026), xvi, 161, XXIX pp
ISBN: 978-3-95806-892-6

Band / Volume 124
Ferroelectric and Resistive Switching in Epitaxial $Hf_{0.5}Zr_{0.5}O_2$
J. R. Knabe (2026), 145 pp
ISBN: 978-3-95806-895-7

Band / Volume 125
Temperature Influences on Frequency Mixing Magnetic Detection for mobile Drinking Water Analysis in Crisis Areas
M. Jessing (2026), xx, 113 pp
ISBN: 978-3-95806-902-2

Band / Volume 126
Light-matter interaction from ab-initio Keldysh formalism
M. D. Merte (2026), xvi, 184, A-27 pp
ISBN: 978-3-95806-906-0

Band / Volume 127
Towards high-quality decoupled Graphene heterostructures
M. Hutter (2026), VIII, 149 pp
ISBN: 978-3-95806-912-1 (Print)
ISBN: 978-3-95806-913-8 (E-Book)

Band / Volume 128
Adaptive-precision interatomic potentials for large-scale atomistic simulations
D. Immel (2026), xxiv, 135 pp
ISBN: 978-3-95806-934-3 (Print)
ISBN: 978-3-95806-935-0 (E-Book)

Band / Volume 129
Competing spin-orbit coupling and exchange interaction in a two-dimensional ferromagnet
X. L. Tan (2026), ii, 113 pp
ISBN: 978-3-95806-938-1 (Print)
ISBN: 978-3-95806-939-8 (E-Book)

Band / Volume 130
Analysis of Electrophysiological Data with Invertible Neural Networks
P. Bouss (2026), xxii, 187 pp
ISBN: 978-3-95806-944-2 (Print)
ISBN: 978-3-95806-945-9 (E-Book)

Band / Volume 131
Orbitronics with Two-Dimensional Rare-Earth Dichalcogenides from First Principles
M. Zeer (2026), xi, 159 pp
ISBN: 978-3-95806-960-2 (Print)
ISBN: 978-3-95806-961-9 (E-Book)

Band / Volume 132
Development of Methodology for Wash-free Bioassays Using Frequency Mixing Magnetic Detection Technique
F. Eivazi (2026), ii, 118 pp
ISBN: 978-3-95806-976-3 (Print)
ISBN: 978-3-95806-977-0 (E-Book)

Band / Volume 133
Graph Based Data Management Approaches for Energy Materials
M. Dreger (2026), xviii, 169 pp
ISBN: 978-3-95806-982-4 (Print)
ISBN: 978-3-95806-983-1 (E-Book)

Band / Volume 134
Sequence learning and computational efficiency in large-scale spiking neuronal networks
M. Lober (2026), xv, 140 pp
ISBN 978-3-69367-014-9 (Print)
ISBN 978-3-69367-015-6 (E-Book)

Weitere ***Schriften des Verlags im Forschungszentrum Jülich*** unter
http://wwwzb1.fz-juelich.de/verlagextern1/index.asp

Information
Band / Volume 134
ISBN 978-3-69367-015-6

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum