PLOS Computational Biology

SOFTWARE

Plastic Arbor: A modern simulation framework for synaptic plasticity—From single synapses to networks of morphological neurons

Jannik Luboeinski[1,2,3]*, **Sebastian Schmitt**[1,2], **Shirin Shafiee**[1,2], **Thorsten Hater**[4], **Fabian Bösch**[5], **Christian Tetzlaff**[1,2,3]

1 III. Institute of Physics—Biophysics, University of Göttingen, Göttingen, Germany, **2** Department for Neuro- and Sensory Physiology, University Medical Center Göttingen, Göttingen, Germany, **3** Campus Institute Data Science (CIDAS), Göttingen, Germany, **4** Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany, **5** Swiss National Supercomputing Centre, ETH Zürich, Zürich, Switzerland

* jannik.luboeinski@med.uni-goettingen.de

OPEN ACCESS

Citation: Luboeinski J, Schmitt S, Shafiee S, Hater T, Bösch F, Tetzlaff C (2026) Plastic Arbor: A modern simulation framework for synaptic plasticity—From single synapses to networks of morphological neurons. PLoS Comput Biol 22(2): e1013926. https://doi.org/10.1371/journal.pcbi.1013926

Editor: Joanna Jędrzejewska-Szmek, Instytut Biologii Doswiadczalnej im M Nenckiego Polskiej Akademii Nauk, POLAND

Received: January 20, 2025

Accepted: January 19, 2026

Published: February 12, 2026

Funding: For this work, CT received funding from the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) through

Abstract

Arbor is a software library designed for efficient simulation of large-scale networks of biological neurons with detailed morphological structures. It combines customizable neuronal and synaptic mechanisms with high-performance computing, supporting multi-core CPU and GPU systems.

In humans and other animals, synaptic plasticity processes play a vital role in cognitive functions, including learning and memory. Recent studies have shown that intracellular molecular processes in dendrites significantly influence single-neuron dynamics. However, for understanding how the complex interplay between dendrites and synaptic processes influences network dynamics, computational modeling is required.

To enable the modeling of large-scale networks of morphologically detailed neurons with diverse plasticity processes, we have extended the Arbor library to support simulations of a large variety of spike-driven plasticity paradigms. To showcase the features of the extended framework, we present examples of computational models, beginning with single-synapse dynamics, progressing to multi-synapse rules, and finally scaling up to large recurrent networks. While cross-validating our implementations by comparison with other simulators, we show that Arbor allows simulating plastic networks of multi-compartment neurons at nearly no additional cost in runtime compared to point-neuron simulations. In addition, we demonstrate that Arbor is highly efficient in terms of runtime and memory use as compared to other simulators. Using the extended framework, as an example, we investigate the impact of dendritic structures on network dynamics across a timescale of several hours, finding a relation between the length of dendritic trees and the ability of the network to efficiently store information.

grants SFB1286 (C01, Z01) and TE 1172/7-1, as well as from European Commission Horizon 2020 through grant no. 899265 (ADOPD). Further, CT and SSc received funding from European Commission Horizon 2020 through grant no. 945539 (HBP SGA3). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. We acknowledge support by the Open Access Publication Funds/transformative agreements of the Göttingen University.

Competing interests: The authors have declared that no competing interests exist.

By our extension of Arbor, we aim to provide a valuable tool that will support future studies on the impact of synaptic plasticity, especially, in conjunction with neuronal morphology, in large networks.

Author summary

Understanding how synaptic plasticity shapes the dynamics of the brain requires simulation tools that provide efficient, scalable computation to integrate detailed neuronal morphology into large networks of neurons. Therefore, we extended the open-source Arbor simulator to enable modeling a comprehensive range of spike-driven plasticity mechanisms. Using progressively more complex examples – from a single synapse to recurrent networks of thousands of morphological neurons – we demonstrate that the extended Arbor framework matches the results of established simulators. At the same time, it uncloses a regime of new investigations. Benchmarking on modern hardware shows that adding morphology incurs only modest runtime and memory overhead as compared to point-neuron models. Moreover, a comparison with the widely used NEURON framework shows that Arbor is markedly more efficient, even when computing additional plasticity dynamics. Leveraging this performance, as an example, we explore how dendritic length and cell size influence long-term memory recall, revealing that larger dendritic trees can impair, whereas larger cell diameters can enhance, pattern-completion performance. The extended Arbor framework and the considered examples of plasticity models are freely available and shall serve to provide a powerful platform for researchers investigating the interplay of cellular structure, synaptic plasticity, and network dynamics.

1 Introduction

Over the past decades, a number of open-source software simulators have been developed to facilitate the investigation of biological neural networks. Current prominent examples are (Core) NEURON [1,2], NEST [3], and Brian 2 [4]. While NEST and Brian 2 are widely used for the simulation of large-scale networks of point neurons, NEURON is a well-established tool for modeling neurons with detailed morphology, with its first version already released in the 1980s [1]. Notably, NEURON enables to flexibly create realistic neuron models in its own script language called NMODL. Over the past decades, NEURON was extended to include more features as well as to keep up with the fast development in computing hardware [2,5]. This has, however, yielded a complex code base that constrains usability, flexibility, and the optimization for modern hardware backends. To overcome these limitations, Arbor has been developed, which is a new simulator library for networks of neurons with detailed morphology. With a Python frontend and support for various model-descriptive formats, including NMODL, Arbor facilitates the implementation and customization of neuron and synapse models. At the same time, Arbor offers heavily optimized execution on different hardware systems [6]. It supports, in particular,

modern backend architectures such as multi-core central processing units (CPUs), graphics processing units (GPUs), and message passing interface (MPI) ranks [6], and is openly developed and available on GitHub [7]. Besides CoreNEURON, Arbor is the only simulator with comprehensive GPU and MPI support for multi-compartment models.

Synaptic plasticity is the ability of synapses to strengthen or weaken in response to neural activity, which is essential for learning and memory [8,9]. So-called long-term synaptic plasticity is crucial for both short- and long-term memory and appears in two types: long-term potentiation (LTP) and long-term depression (LTD), which strengthen and weaken synaptic connections, respectively. These processes mainly involve changes in the distribution of postsynaptic receptors and in the structure of dendritic spines, driven by complex biochemical and biophysical mechanisms within synapses as well as in dendrites [10–14]. Their dysregulation has further been linked to neurological disorders like Alzheimer's disease and schizophrenia [15,16]. On the other hand, recent machine learning approaches also make use of synaptic and dendritic processes to improve the performance of large-scale neural networks [17–19]. Thus, investigations on the functional impact of synapses and dendrites on network dynamics are becoming more and more fundamental to advance research in areas of neuroscience, medicine, and machine learning.

Arbor constitutes a powerful tool that enables to capture processes at the synaptic, dendritic, and neuronal levels and to examine their interaction with network dynamics. The core functionality for running network simulations, with neurons of several hundreds of compartments and custom ion channels defined via the NMODL language, has been available since the inception of the Arbor project [6]. Arbor has been shown to outperform NEURON in terms of speed and to provide efficient weak and strong scaling (see [6] and the S1 Appendix of the present study). Furthermore, a recent study has demonstrated Arbor's capabilities of scaling efficiently to at least 50 PFLOPS systems with 4000 A100 GPUs [20]. Nevertheless, the mechanisms required for modeling plasticity-related processes have largely been absent. In our present work, we fill this gap extending the Arbor simulator by the general functionality needed to model diverse spike-driven plasticity rules (e.g., [21–23]), which constitutes the 'Plastic Arbor' framework. By this, we provide a basis for investigating the functional impact of plasticity dynamics in *large* networks of morphological neurons via our publicly available code (see the data and code availability statement below).

In this article, we aim to present Arbor's new set of features to model plasticity in large-scale networks of multi-compartment neurons, as well as to demonstrate that Arbor enables highly efficient simulation of such networks in terms of runtime and memory use. We describe the novel functionality in two different sections, where we present methods and results in a side-by-side fashion. The new technical features needed to implement plasticity rules are described in the section 'Design and Implementation: Extensions of the Arbor core code'. In the section 'Results: Computational modeling with synaptic plasticity', we present examples for the newly implemented technical features introduced before, with increasing complexity of the implemented models. We start from the level of simulating a widely-used spike-timing-dependent plasticity rule in a single synapse, then consider plasticity rules involving multiple synapses, move on to more complex plasticity rules including several hidden states, and finally reproduce findings from large recurrently connected networks. We cross-validate all Arbor implementations with the Brian 2 simulator [4] or with model-specific custom simulators. Importantly, the newly extended functionality of Arbor has enabled us to provide first predictions about the network dynamics underlying synaptic memory consolidation across networks of neurons of different morphology. We specifically provide insights into how the performance in a pattern completion task at the network level depends on the length of the dendrites as well as on the overall size of the employed neurons. Finally, we provide benchmarking results showcasing Arbor's runtime and memory efficiency compared to other simulators.

2 Design and implementation: Extensions of the Arbor core code

In this section, we present the new extensions of the Arbor core code that are necessary for the implementation of synaptic plasticity models in Arbor. Examples of related models are presented in the following Sect 3.

2.1 Spike-time detection to simplify computation

Many formulations of synaptic plasticity depend on the timing of pre- and postsynaptic spikes [21,24,25]. In neuroscience, spike-timing-dependent plasticity (STDP) serves as a valuable phenomenological model that can encapsulate the intricacies of synaptic plasticity with respect to its dependence on pre- and postsynaptic spike timing in a computationally efficient manner (cf. Sect 3.1). The simplicity of its formulation, focusing on the relative timing of spikes, makes STDP a practical choice for computational models that serve to understand learning processes in complex neural systems, without the necessity of a detailed molecular blueprint. Nevertheless, to capture the molecular and cellular mechanisms that underlie synaptic modification, more detailed models such as the calcium-based models used in Sects 3.3–3.6 of this article are needed.

To enable the implementation of models with reduced complexity, such as STDP, we have introduced a hook named `POST_EVENT` that serves to detect spiking events and to provide a buffer with all spike events of the postsynaptic neuron (e.g., somatic action potentials or dendritic spikes). This information can thus be conveyed to another compartment or synapses without explicit implementation of the physical transmission process (e.g., the backpropagation of action potentials). This is, inter alia, needed for STDP rules (cf. Sect 3.1) and calcium-based plasticity rules (cf. Sects 3.3–3.6).

2.2 Multiple postsynaptic variables depending on pre- and postsynaptic spiking

The Arbor documentation [26] defines a selection policy as 'Enumeration used for selecting an individual item from a group of items sharing the same label.' (where, for example, the items might be synapse objects with the label 'exc_input_synapse'). Already present in previous Arbor versions, the `round_robin` policy enables to iterate over the items of an object group in a round-robin fashion, e.g., to iterate over synapses connecting to the same postsynaptic compartment.

We added the new selection policy `round_robin_halt`, which enables to halt at the current item of the group until the `round_robin` policy is called again. This functionality is crucial to implement the independent update of multiple postsynaptic variables that depend on pre- and postsynaptic spiking. This is required, for instance, for large-scale network models including spike-driven postsynaptic calcium dynamics (see Sects 3.5 and 3.6) that shall occur alongside the usual postsynaptic voltage dynamics. In such a case, the `round_robin_halt` policy serves to target both dynamics without having to define explicit labels for every individual connection in the network, which can save a tremendous amount of compute resources.

2.3 Computation with stochastic differential equations

For numerous learning mechanisms, in particular also for some of the plasticity rules that are considered in the computational experiments presented here, random processes are required. Such processes are often described by stochastic differential equations (SDEs). In general, the coupled first-order equations for a vector of stochastic state variables $\mathbf{X}$ can be expressed in their differential form as

$$d\mathbf{X}(t) = \mathbf{f}(t, \mathbf{X}(t))dt + \sum_{i=0}^{M-1} \mathbf{g}_i(t, \mathbf{X}(t))dB_i(t), \tag{1}$$

where the vector-valued function $\mathbf{f}$ denotes the deterministic differential, and the last term represents the stochastic contribution. Here, the M functions $\mathbf{g}_i$ with units $[g_i] = [X]/\sqrt{t}$ are associated with standard Wiener processes B_i, where $B_i(0) = 0$ almost surely, and $B_i(t) \sim \mathcal{N}(0, t)$ with units $[B_i] = \sqrt{t}$.

The stochastic integral is defined by Itô's non-anticipative generalization of the Riemann– Stieltjes summation

$$S_i = \int_{t_0}^{t_0+s} \mathbf{g}_i(\tau, \mathbf{X}(\tau))dB_i(\tau) = \lim_{N\to\infty} \sum_{n=0}^{N-1} \mathbf{g}_i(t_i, \mathbf{X}(t_i))\,(B(t_i) - B(t_{i-1}))\,, \tag{2}$$

where $t_0 < t_1 < \cdots < t_{N-1} = t_0 + s$, and $N \in \mathbb{N}$. By introducing stationary Gaussian white noise W_i such that $W_i(t)dt = dB_i(t)$, the system of equations can be expressed using more common shorthand notation as

$$\mathbf{X}'(t) = \mathbf{f}(t, \mathbf{X}(t)) + \sum_{i=0}^{M-1} \mathbf{g}_i(t, \mathbf{X}(t))W_i(t). \tag{3}$$

We equipped Arbor with the capability to numerically solve the system of SDEs given in Eq 3 using the Euler-Maruyama algorithm, a first-order stochastic Runge-Kutta method [27]. The algorithm for integrating the stochastic dynamics from discrete time step t_k to $t_{k+1} = t_k + \Delta t$ comprises two steps:

1. Drawing random variables $\Delta\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}\Delta t)$, where $\mathbf{Q}$ denotes the correlation matrix of the white noises W_i,
2. Computing $\hat{\mathbf{X}}(t_{k+1}) = \hat{\mathbf{X}}(t_k) + f(t_k, \hat{\mathbf{X}}(t_k))\Delta t + \sum_{i=0}^{M-1} \mathbf{g}i(t_k, \hat{\mathbf{X}}(t_k))\Delta Wi$.

Currently, we assume uncorrelated noise, $\mathbf{Q} = \mathbb{I}$. Hence, to generate M independent random samples for every instantiation of every stochastic process at each time step, a normally distributed noise source of sufficient quality is required.

Traditional pseudorandom number generators (PRNGs) prove inadequate for this context, as they typically generate a sequence of samples through the evaluation of a recurrence relation φ of order k. Here, the nth sample of a series u_n is contingent upon the k preceding values: $u_n = \varphi(n, u_{n-1}, u_{n-2}, \cdots, u_{n-k})$. For instance, to produce a series of independent random samples, the standard 64-bit implementation of the Mersenne-Twister algorithm in C++ necessitates the sequential updating of a state comprising at least 19968 bits ($k = 312$).

Therefore, for efficient, in particular, highly parallelized implementations, other solutions such as counter-based PRNGs (CBPRNGs) [28] are more suitable. In CBPRNGs, each sample can be independently drawn by modulating the input to the generator function. This input may be subdivided into a counter $c(n)$ and a key $\kappa(n)$, thus enabling the construction of a distinct input for every required source of white noise, $u_{i,n} = \varphi(c_i(n), \kappa_i(n))$. Due to this approach that eliminates the need for lookup tables, the initialization cost is basically zero, and the evaluation cost is similar compared to that of traditional PRNGs such as Mersenne-Twister [28]. Owing to these characteristics and their stateless nature, CBPRNGs lend themselves well to parallelization on both CPU and GPU architectures.

Specifically, we employ the `Threefry-4x64-12` CBPRNG algorithm from the `random123` library [29]. This algorithm's input width, at 2×256 bits, affords ample capacity for uniquely encoding the white noise sources. `Threefry-4x64-12` yields four independent, uniformly distributed values per invocation, which we cache for each noise source and refresh only upon depletion.

To generate random numbers following a normal distribution, we employ the Box-Muller method, ensuring uniform cache depletion across all noise sources as opposed to rejection-sampling based techniques.

In preceding versions of Arbor (before v0.8), the inclusion of random processes required modifying the C++ code produced by Arbor's `modcc` compiler. This approach hindered the effective utilization of CBPRNGs and mandated the manual crafting of solvers for SDEs. Presently, however, we have augmented Arbor's NMODL dialect with a specialized solver method, denoted `stochastic`, alongside a mechanism for specifying independent noise sources via the `WHITE_NOISE` code block. These enhancements enable the seamless handling of SDEs as described above.

2.4 Diffusion of arbitrary particles

Arbor's comprehension of neuronal morphology is built on the *cable model* of neuronal dynamics:

$$\frac{1}{C_\mathrm{m}}\frac{\partial}{\partial t}U_\mathrm{m} = \frac{\partial}{\partial x}\frac{1}{R_\mathrm{l}}\frac{\partial}{\partial x}U_\mathrm{m} + I_\mathrm{m}, \tag{4}$$

which describes the evolution of the membrane potential U_m depending on time and one spatial dimension [30,31]. In this equation, R_l denotes the axial (longitudinal) resistance and C_m the membrane capacity. The current I_m accounts for the radial transport of charges across the membrane via ion-channels (for more details, see Fig G in S1 Appendix). The term $\frac{\partial}{\partial x}\frac{1}{R_\mathrm{l}}\frac{\partial}{\partial x}U_\mathrm{m}$ describes a longitudinal current along the dendritic segment that results in charge equalization.

It is usually assumed that this model is valid for a thin layer around the membrane where all changes to individual ionic concentrations – commonly labeled X_i and X_o for the intra- and extracellular concentration of ion species X – are equalized to that of a surrounding internal or external buffer solution. This buffering is modeled as an infinitely fast process, such that any alterations are visible only on timescales of the numerical timestep. The trans-membrane current I_m can be expressed as a function of individual ion species X:

$$I_\mathrm{m} = \sum_\mathrm{X} g_\mathrm{X} \cdot (U_\mathrm{m} - E_\mathrm{X}),$$

with

$$E_\mathrm{X} = \frac{RT}{z_\mathrm{X}F} \cdot \ln\left(\frac{X_o}{X_i}\right),$$

where the ion channel models produce the conductivities g_X and the reversal potentials E_X, with universal gas constant R, Faraday constant F, temperature T, and charge number z_X.

Note that models of neuronal dynamics that include the resolution of individual ions in the evolution of the membrane potential are tractable but computationally more demanding than the cable model [32]. The diffusion of particles along dendrites is, nevertheless, a critical element for many computational neuron models. As mentioned before, a rigorous model for the transport of ions is feasible, but involves a different equation for charge equalization as opposed to Eq 4. Namely, it requires handling the changes in the intra- and extracellular concentration of particles through molar fluxes (including the buffering because now the associated timescale has become relevant), and – closing the feedback loop – the Nernst equation for computing the individual reversal potentials. The alternative of mixing models would lead to a flawed formulation, where particles are transported by diffusion despite already being included in the longitudinal currents of Eq 4. Thus, we decided to implement diffusion of arbitrary particles in Arbor as if the relevant species were strictly neutral, i.e. no additional flow of charges is considered, neither along the dendrite nor across the membrane. The physical model for diffusion of the concentration X of the specific particle species is then simply given by

$$\frac{\partial}{\partial t}X = \frac{\partial}{\partial x}D\frac{\partial}{\partial x}X + \phi, \tag{5}$$

where we have the diffusivity D and the molar flux ϕ across the membrane, from, or to internal stores. This equation is in shape identical to the cable equation, which allows us to leverage Arbor's existing, highly optimized solver. The diffusive model is decoupled from the cable model and exposed via a separate quantity X_d to NMODL. Users can — if desired — retrofit the interaction with the cable model by assigning the appropriate mechanisms formulated in NMODL:

$$I_\mathrm{m} = \sum_\mathrm{X} I_\mathrm{X} = \sum_\mathrm{X} g_\mathrm{X} \cdot (U - E_\mathrm{X}),$$
$$\frac{\partial}{\partial t} X_d = \phi_\mathrm{X} + \frac{I_\mathrm{X}}{z_\mathrm{X} F},$$
$$E_\mathrm{X} = \frac{RT}{z_\mathrm{X} F} \cdot \ln\left(\frac{X_o}{X_d}\right),$$

yielding a closed model together with Eqs (4) and (5), however, at the cost of the inherent issues described above.

3 Results: Computational modeling with synaptic plasticity

In this section, we present representative models of synaptic plasticity that we implemented in Arbor, serving as examples to demonstrate the full functionality of the newly implemented features. As we have made the respective code freely available, readers can simply reuse or adapt the models for their own investigations with Arbor. For references to the code bases used for the specific simulations, please see the data and code availability statement at the end of this article.

Furthermore, we provide benchmarking results to compare the runtime and memory performance of the simulation of different network models in Arbor and other simulators.

Please note that since the implementation of plasticity dynamics is the main target of this paper, we only present the model equations that correspond to those. For parts of the considered models other than the plasticity dynamics, please refer to the cited literature and provided code.

Fig 1A,B shows an overview of essential features of the Arbor user interface, which can be used to implement and simulate models of plastic neuronal networks. Fig 1C summarizes the main contribution of our present work: a simulation framework that enables to target arbitrary mechanisms of synaptic plasticity by allowing to implement a mathematical model for the weight dynamics dw/dt in the NMODL domain-specific language or in C++ code. Note that while this allows to implement a broad range of plasticity mechanisms, including such with a lot of biochemical and biophysical detail, we focus here on plasticity mechanisms as they are typically used by researchers considering network models.

3.1 Spike-timing-dependent plasticity

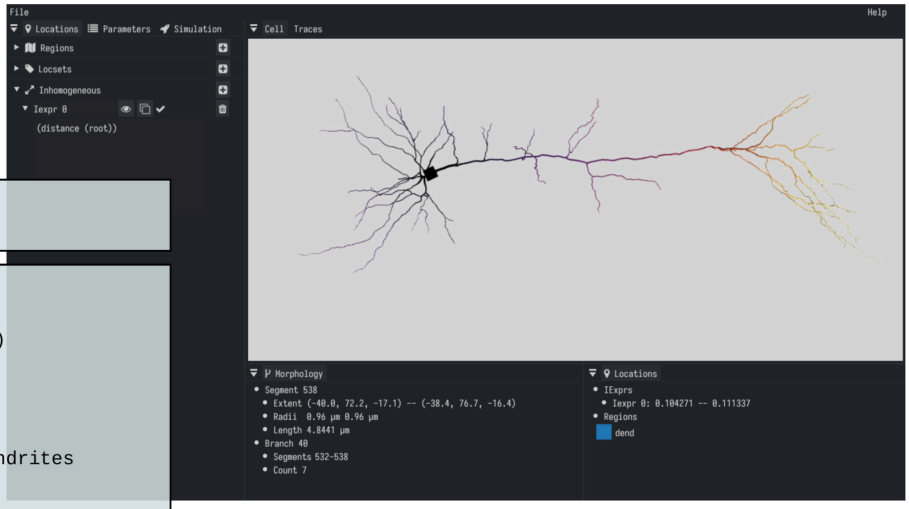
Spike-timing-dependent plasticity (STDP) is a phenomenon that is reported in a number of experimental studies [24,33–35] and described by various theoretical models (cf. [36]). The earlier models of STDP have provided a relatively simple form of synaptic plasticity that only depends on *the specific timing of pre- and postsynaptic spikes* via the temporal differences $t^n_\mathrm{pre} - t^m_\mathrm{post}$ ($m, n \in \mathbb{N}$) between pre- and postsynaptic spikes. In this case, plasticity does not directly depend on the spike rate. As a first step, we implemented a commonly used description [25] given by:

$$\frac{da_\mathrm{pre}(t)}{dt} = -\frac{a_\mathrm{pre}(t)}{\tau_\mathrm{pre}} + A_\mathrm{pre} \cdot \sum_n \delta(t - t^n_\mathrm{pre}), \tag{6}$$
$$\frac{da_\mathrm{post}(t)}{dt} = -\frac{a_\mathrm{post}(t)}{\tau_\mathrm{post}} + A_\mathrm{post} \cdot \sum_m \delta(t - t^m_\mathrm{post}), \tag{7}$$
$$\frac{dw(t)}{dt} = a_\mathrm{pre}(t) \cdot \sum_m \delta(t - t^m_\mathrm{post}) + a_\mathrm{post}(t) \cdot \sum_n \delta(t - t^n_\mathrm{pre}). \tag{8}$$

Here, the constants τ_pre and τ_post describe the decay of the eligibility traces $a_\mathrm{pre}(t)$ and $a_\mathrm{post}(t)$ induced by pre- and postsynaptic spikes, and $\delta(\cdot)$ represents the Dirac delta distribution. The amplitudes $A_\mathrm{pre} > 0\mu\mathrm{S}$ and $A_\mathrm{post} < 0\mu\mathrm{S}$ define the strengthening and weakening of the synaptic weight $w(t)$ that follow the occurrence of spikes, which takes more effect the

A

Arbor GUI



B

Python frontend:

```
import arbor as A
# ...
```

```
# Definition of a simple model:
# Define the cell model and labels
label = A.label_dict().add_swc_tags()
center = '(location 0 0.5)'
decor = (A.decor()
  # Set initial membrane potential
  .set_property(Vm=-55)
  # Assign mechanisms to soma and dendrites
  .paint('"soma"', A.density("hh"))
  .paint('"dend"', A.density("pas"))
  # Attach stimulus
  .place(center, A.iclamp(10, 1, current=5), "ic")
  # Introduce spike threshold/trigger
  .place(center, A.threshold_detector(-10), "det"))
# Load morphology data
morpho = A.load_swc_arbor("cell.swc")
# Use shortcut for single-cell model creation
sim = A.single_cell_model(morpho, decor, label)
# ...
```

```
# General model definition via *recipe*:
recipe = Recipe(*args)
# ...
sim = A.simulation(recipe, context, ddc)
# ...
```

```
# Run the simulation
sim.run(100, 0.005)
# ...
```

HPC integration:

```
# Adding GPU support:
context = A.context(gpu_id=0)
# Decompose problem (the network) into domains
ddc = A.partition_load_balance(recipe, context)
sim = A.simulation(recipe, context, ddc)
# ...
```

```
# Adding MPI paradigm:
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
# Use GPUs round-robin, assuming 4 per node
context = A.context(mpi=comm, gpu_id=rank%4)
# Decompose problem (the network) into domains
ddc = A.partition_load_balance(recipe, context)
# ...
```

C

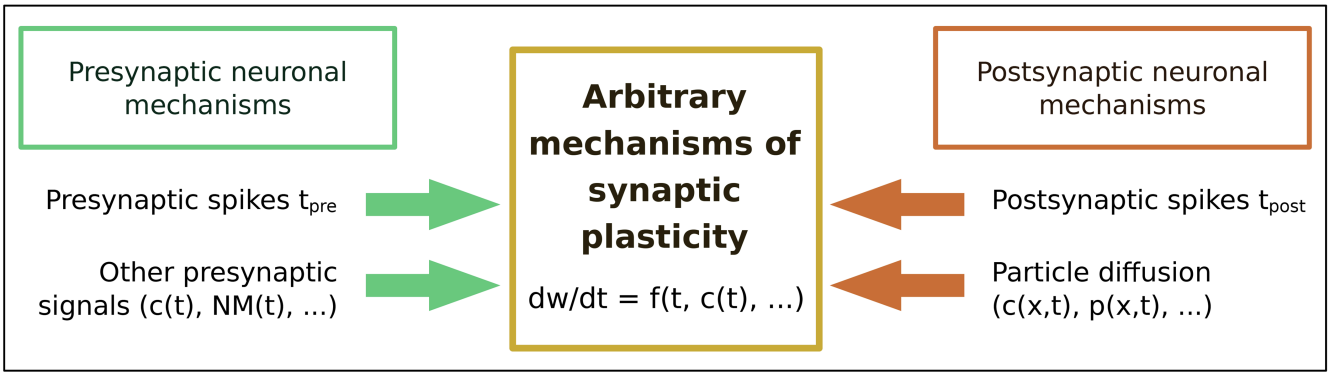


Fig 1. **Overview of the extended Arbor framework. (A)** Graphical presentation of cell morphologies via `Arbor GUI` (v0.8.0-dev-065b1c9 shown here). **(B)** Key code features for model simulation with Arbor. On the left: facilitated definition of models via the Python frontend by setting up a so-called *recipe* (where in some cases, there are shortcuts that further accelerate the definition of certain simple models). On the right: modular integration of high performance computing (HPC) hardware such as graphics processing units (GPUs) and message passing interface (MPI) ranks. **(C)** General approach of the new plasticity framework, enabling to simulate a wide variety of synaptic plasticity mechanisms in arbitrary model paradigms (examples of which are provided in Sect 3).

https://doi.org/10.1371/journal.pcbi.1013926.g001

closer together pre- and postsynaptic spikes occur in time. Note that the synaptic weight $w(t)$ is updated for each complete pair of pre- and postsynaptic spikes. The synaptic weight is initialized at the baseline value w_0, and its contribution to postsynaptic potentials is clipped at a value $w_{\max}$. For the parameter values, see Table 1.

To simulate this model with Arbor, we implemented a single, conductance-based excitatory synapse. This synapse was connected to a Leaky Integrate-and-Fire (LIF) neuron and was stimulated with Poissonian spike input (see Fig 2A). In addition, an inhibitory synapse was introduced to stabilize the dynamics, also driven by Poissonian spike input. Both the excitatory synapse and the LIF neuron were implemented as custom mechanisms in Arbor (defined via NMODL scripts). Note that for this implementation, the `POST_EVENT` hook, which we added to Arbor's NMODL dialect, is needed (cf. Sect 2.1) – the hook is called whenever a threshold detector on a cell is triggered. In the present case, this is when the postsynaptic neuron spikes.

We evaluated the functionality of the STDP implementation by comparing between Arbor and the Brian 2 simulator, which provides a suitable cross-validation due to the different implementations of the numerical solver as well as the algorithmic data representation. We observe a good match between the presented measures recorded in Brian 2 and Arbor (Fig 2C; Fig A in S1 Appendix), which we quantified via the values of coefficient of variation (CV) and root mean square error (RMSE), computed using `scikit-learn` in version 1.3.2. Regarding the spike times, we also encounter only minimal differences, which we quantified by computing the spike time mismatch (detailed in Fig A in S1 Appendix).

In addition to this experiment, we simulated a range of time delay settings to obtain a classical STDP time window, which also shows a very good match with Brian 2 simulations (Fig 2D) as well as with the theoretical expectation (cf. Fig A in S1 Appendix).

3.2 Spike-driven homeostatic plasticity

After implementing an STDP rule at a single synapse, we now consider a type of synaptic plasticity that depends on *multiple presynaptic stimuli onto the same postsynaptic neuron*.

Spike-based homeostatic plasticity describes the finding that the strength of a synapse adapts according to the spiking activity of the postsynaptic neuron – hence, the synaptic strength is up- or downregulated to maintain a certain activity of the neuron (cf. [39]). By using the newly implemented `POST_EVENT` hook (introduced in Sect 2.1), we could employ Arbor to simulate spike-driven homeostasis. To show this, we connect (similar to [40]) an LIF neuron to two Poisson-stimulus inputs – one with a varying spike rate and the other with a fixed spike rate. The weight of the synapse for the varying-rate input is kept static, while the weight of the synapse for the fixed-rate input is plastic. The plasticity of the latter synapse should cause the neuron, in a homeostatic manner, to maintain a firing rate that is determined by the parameters of the plasticity rule. This is realized by the following weight dynamics for the plastic synapse:

$$\frac{dw(t)}{dt} = \Delta w_+ \cdot \sum_n \delta(t - t^n_{\text{pre}}) + \Delta w_- \cdot \sum_m \delta(t - t^m_{\text{post}}), \tag{9}$$

Table 1. Parameters for the plain STDP model. In the case that two values are given, the first value was used for the detailed analysis shown in Fig 2C and in Fig A in S1 Appendix, and the second value was used to obtain the classical curve shown in Fig 2D (also cf. [37]).

Symbol	Value	Description	Refs.
w_0	1.0μS	Baseline of the synaptic weight	This study
τ_{pre}	20.0ms	Decay of the eligibility trace of presynaptic spikes	[25,37]
τ_{post}	{10.0, 20.0}ms	Decay of the eligibility trace of postsynaptic spikes	[25,37]
A_{pre}	{0.3, 0.01}μS	Strengthening amplitude	[25,37]
A_{post}	{−0.2, −0.0105}μS	Weakening amplitude	[25,37]
$w_{\max}$	10.0μS	Maximum synaptic weight contributing to postsynaptic potentials	[37]

https://doi.org/10.1371/journal.pcbi.1013926.t001

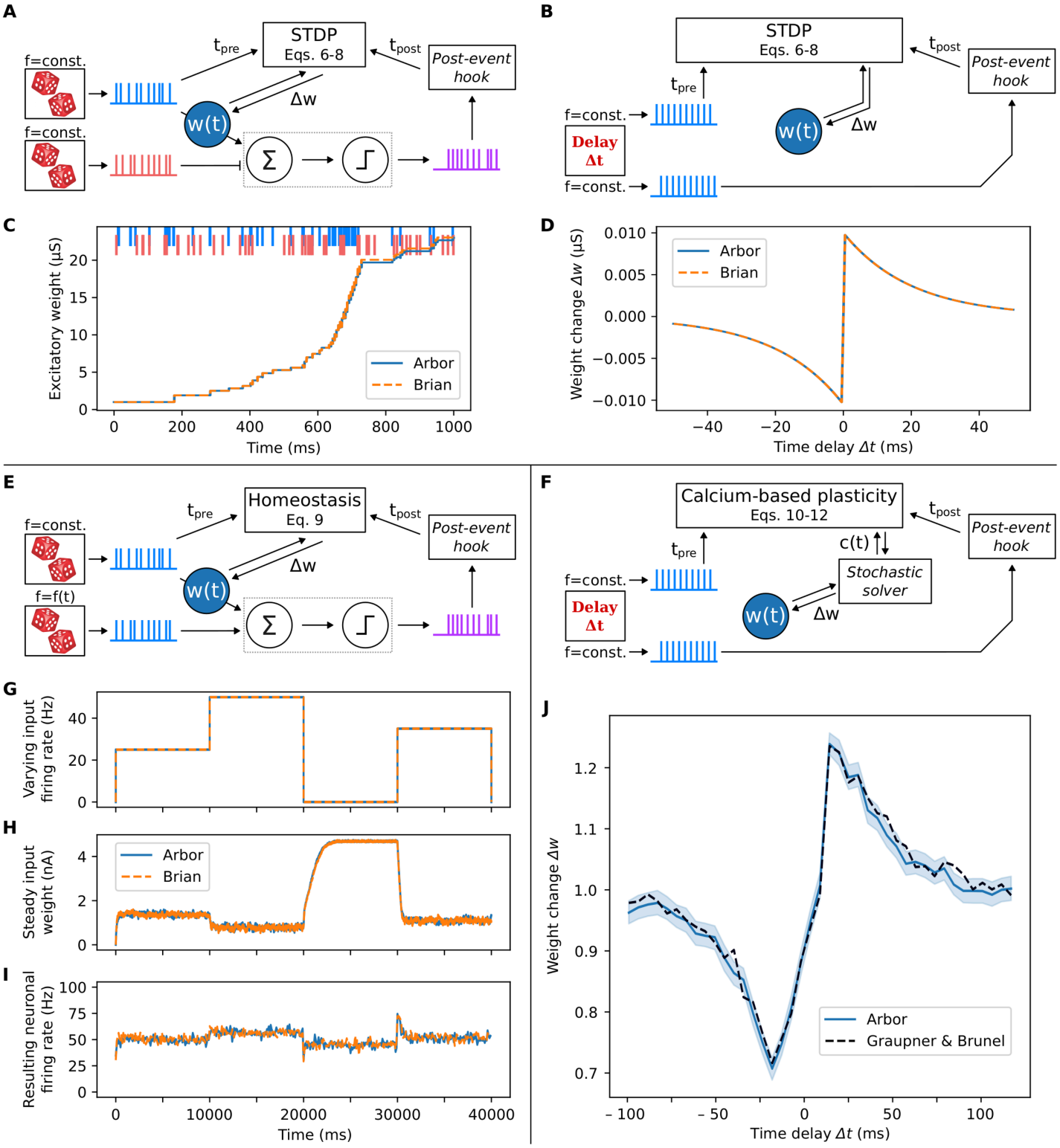


Fig 2. Classical spike-timing dependent plasticity (STDP), spike-driven homeostasis, and calcium-driven synaptic plasticity in Arbor. Arbor implementations (in lighter blue) are cross-validated by comparison to Brian 2 (in orange) or a custom simulator. New features of the Arbor core code are highlighted in italic. **(A)** STDP paradigm where two Poisson spike sources stimulate an excitatory and an inhibitory synapse connecting to a single neuron (spikes shown in blue and red, respectively). The excitatory connection undergoes STDP (results shown in (C)). Image of dice from Karen

Arnold/publicdomainpictures.net. **(B)** STDP paradigm where two regular spike trains, phase-shifted by delay Δt, drive the weight dynamics of a single synapse (results shown in (D)). **(C)** Strength of the excitatory synapse, subject to STDP, as shown in (A) (goodness of fit between Arbor and Brian 2: CV > 0.999, RMSE = 1.064μS). **(D)** Classical STDP curve, obtained as detailed in (B) (CV > 0.999, RMSE = 0.001μS). **(E)** Homeostasis with two Poisson spike sources connected to an LIF neuron via current-based delta synapses (results shown in (G–I), averaged over 50 trials). One of these Poisson inputs spikes at a fixed rate and is plastic, while the other spikes at a varying rate and is static. (F) Paradigm of calcium-based, spike-timing- and rate-dependent synaptic plasticity, using the model by Graupner & Brunel [21]. Two regular spike trains, phase-shifted by delay Δt, drive the stochastic weight dynamics of a single synapse (results shown in (J)). **(G)** Time course of the varying rate of the input in (E). **(H)** Strength of the plastic synapse, subject to the dynamics given in (E) (CV = 0.996, RMSE = 0.307nA). **(I)** Firing rate of the neuron shown in (E) in the presence of homeostatic plasticity dynamics (CV = 0.508, RMSE = 1.981Hz). **(J)** Calcium-driven synaptic plasticity as shown in (F). Reproduction of the numerical DP curve from Fig 2 of the related paper [21] (the mean is given by the dark dashed line). Every synapse is subject to 60 spike pairs presented at 1Hz. Arbor results were averaged over 4000 trials, the solid blue line indicates the mean and the shaded region the 95% confidence interval. Quantification of deviation between the mean curves: CV = 0.987, RMSE = 0.123. Note that the generation of this plot is now also demonstrated in an Arbor tutorial [38].

https://doi.org/10.1371/journal.pcbi.1013926.g002

where the constants Δw_+ and Δw_- describe the weight changes upon the occurrence of pre- and postsynaptic spikes at times t^n_{pre} and t^m_{post} ($n, m \in \mathbb{N}$). The weight is initialized at value w_{init} (see Table 2).

The resulting weight and firing rate dynamics, along with the varying input rate, are shown in Fig 2G–I and in Fig A in S1 Appendix. We can see that in the case with homeostasis, the resulting firing rate is maintained at values around 50Hz (Fig 2I), while in the case without homeostasis (cf. Fig A in S1 Appendix), the resulting firing rate is mainly imposed by the input rate. Note that for the time period with input rate 0 (from $t = 20$s to 30s), the homeostatic weight adjustment can only happen to a limited extent since we do not allow the weights to increase beyond $w_{\max}$. We cross-validated our Arbor implementation with Brian 2, drawing from an existing example implementation [41].

3.3 Calcium-based synaptic plasticity

Following the implementation of a simple phenomenological STDP rule (Sect 3.1) as well as spike-driven homeostatic plasticity (Sect 3.2), we now consider a more complex rule of synaptic plasticity that requires multiple postsynaptic variables. This rule describes the potentiation and depression of synaptic strength depending on *the postsynaptic calcium concentration*, which is driven by pre- and postsynaptic spiking activity. It was presented by Graupner and Brunel in 2012 [21] and has since been used widely. In comparison to the phenomenological STDP rule, the calcium-based rule adapts the synaptic weight dynamics such that they depend on spike timing and spike rates, both of which have been shown to be important features of long-term synaptic plasticity [42].

The change of the synaptic weight in this model is given by the following equation:

$$\begin{aligned}\tau_w \frac{dw_{ji}(t)}{dt} = &- w_{ji}(t) \cdot \big(1 - w_{ji}(t)\big) \cdot \big(w_* - w_{ji}(t)\big) \\ &+ \gamma_{\text{p}} \cdot \big(1 - w_{ji}(t)\big) \cdot \Theta\left[c_{ji}(t) - \theta_{\text{p}}\right] \\ &- \gamma_{\text{d}} \cdot w_{ji}(t) \cdot \Theta\left[c_{ji}(t) - \theta_{\text{d}}\right] \\ &+ \xi(t),\end{aligned} \tag{10}$$

Table 2. Parameters for the homeostatic plasticity model.

Symbol	Value	Description	Refs.
w_{init}	0.00nA	Baseline weight of the plastic (fixed-rate input) synapse	[40]
$w_{\max}$	5.00nA	Maximum weight of the plastic (fixed-rate input) synapse	[40]
Δw_+	0.35nA	Weight change due to presynaptic spike	[40]
Δw_-	−0.35nA	Weight change due to postsynaptic spike	[40]
w_{varying}	3.50nA	Weight of the varying-rate input synapse	[40]

https://doi.org/10.1371/journal.pcbi.1013926.t002

where τ_w is a time constant, w_* defines the boundary between the basins of attraction for potentiation and depression, γ_p and γ_d are the potentiation and depression rates, $c_{ji}(t)$ is the calcium concentration at the postsynaptic site, and θ_p and θ_d are thresholds for triggering potentiation and depression, respectively (cf. Table 3). Moreover, $\Theta[\cdot]$ denotes the Heaviside theta function, and

$$\xi(t) = \sqrt{\tau_w \left(\Theta\left[c(t) - \theta_p\right] + \Theta\left[c(t) - \theta_d\right]\right)} \cdot \sigma_{pl} \cdot \Gamma(t) \tag{11}$$

is a noise term with scaling factor σ_{pl} and Gaussian white noise $\Gamma(t)$, which has a mean value of zero and a variance of $1/dt$ (cf. [43]). Note that to implement the noise term, support for stochastic differential equations was needed, which we have added to the Arbor core code as described above in Sect 2.3.

Finally, the dynamics of the calcium concentration is given by the following equation:

$$\frac{dc(t)}{dt} = -\frac{c(t)}{\tau_c} + c_{pre} \cdot \sum_n \delta(t - t^n_{pre} - t_{c,delay}) + c_{post} \cdot \sum_m \delta(t - t^m_{post}), \tag{12}$$

where τ_c is a time constant, c_{pre} and c_{post} are increases in the intracellular calcium concentration of the dendritic spine induced by pre- and postsynaptic spikes at times t^n_{pre} and t^m_{post}, $t_{c,delay}$ is the delay of the presynaptic contributions, and $\delta(\cdot)$ is the Dirac delta distribution.

Fig 2J shows the results of our Arbor implementation for the weight change over the delay between pre-and postsynaptic spikes (analogously to Fig 2D), cross-validated with the numerical results by the original study [21].

3.4 Heterosynaptic calcium-based plasticity in dendrites

As a next step, we use a calcium-based plasticity rule slightly different to the one in the previous subsection, with the aim to simulate the *spread of calcium in a dendritic branch*, which enables us to model *heterosynaptic plasticity*. This model serves as an example of our diffusion extension for the Arbor core code, which has been described in Sect 2.4.

Homosynaptic plasticity and heterosynaptic plasticity are two forms of plasticity that play crucial roles in shaping neural connections. Homosynaptic plasticity involves changes within a specific neural pathway or synapse in response to repeated stimulation or learning, leading to the strengthening or weakening of that connection depending on factors such as frequency or duration of stimulation. On the other hand, heterosynaptic plasticity is a broader phenomenon where stimulation of one synapse induces changes in other, unstimulated synapses. Here, we consider a calcium-based heterosynaptic plasticity rule [22] that is based on observations at the level of a single neuron [21,44].

Table 3. Parameters for the implementation of the calcium-based plasticity model by Graupner & Brunel [21]. Note that as in the original mathematical model, the weights are kept without physical unit.

Symbol	Value	Description	Refs.
w_0	0.0 or 1.0	Initial value of the synaptic weight (drawn randomly)	This study
w_*	0.5	Boundary between the basins of attraction for potentiation and depression	[21]
$t_{c,delay}$	13.7ms	Delay of postsynaptic calcium influx after presynaptic spike	[21]
c_{pre}	1	Presynaptic calcium contribution, in vivo adjusted	[21]
c_{post}	2	Postsynaptic calcium contribution, in vivo adjusted	[21]
τ_c	20ms	Calcium time constant	[21]
τ_w	150s	Weight dynamics time constant	[21]
γ_p	321.808	Potentiation rate	[21]
γ_d	200	Depression rate	[21]
θ_p	1.3	Calcium threshold for potentiation	[21]
θ_d	1	Calcium threshold for depression	[21]
σ_{pl}	2.8248	Standard deviation for plasticity fluctuations	[21]

https://doi.org/10.1371/journal.pcbi.1013926.t003

Dendrites are crucial components for the information processing in neurons, as they receive signals from other neurons and integrate them to generate a particular response. Spiny structures on the dendrites can serve to receive synaptic inputs and at the same time undergo plastic changes [45]. Here, we consider a model that describes a number of such spines on a single dendritic branch [46]. The state and strength of these spines are subject to the previously mentioned calcium-based plasticity rule [22]. We describe the synaptic input to a specific spine via

$$I^{\mathrm{Ca}}_{\mathrm{spine}}(x, t_i) = \sum_i I_0 \cdot e^{-(t-t_i)/\tau_I} \cdot \Theta(t - t_i), \tag{13}$$

which induces an elevated level of calcium at the target spine via monoexponential contributions with amplitude I_0 and time constant τ_I. However, due to the calcium diffusion dynamics in our system, other unstimulated and inactive spines will also experience changes in their calcium level. These changes depend on the spine location with respect to the stimulated spine(s) as well as the temporal characteristics of the stimulation (i.e., frequency, duration, and delay). In our simulations, pre-synaptic spike events arrive at active spines at times t_i as a regular spike train with a time interval of 10ms. Note that we inject synaptic input at the top point of the spine heads, so the term $I^{\mathrm{Ca}}_{\mathrm{spine}}(x, t_i)$ is zero for all other regions (in particular, the dendritic shaft). The calcium diffusion in the dendritic branch and in the spines is then described by the following equation:

$$\frac{\partial C(x,t)}{\partial t} = D\frac{\partial^2 C(x,t)}{\partial x^2} - \frac{C(x,t)}{\tau_C} + w_i \cdot I^{\mathrm{Ca}}_{\mathrm{spine}}(x, t_i), \tag{14}$$

where $C(x,t)$ is the calcium concentration, D is the diffusion constant, and τ_C is the calcium decay time constant. The synaptic strength is computed as outlined below:

$$\begin{aligned}\frac{dw_i(t)}{dt} = &(1 - w_i(t)) \cdot \gamma_p \cdot \Theta\left[C(t) - \theta_p\right] \\ &- w_i(t) \cdot \gamma_d \cdot \Theta\left[C(t) - \theta_d\right],\end{aligned} \tag{15}$$

where w_i is the synaptic weight of spine i, the constants γ_p and γ_d quantify the rate of synaptic strengthening or weakening during potentiation and depression, and θ_p and θ_d are the calcium threshold values for triggering potentiation and depression, respectively.

The dendritic branch in our model includes four spines as shown in Fig 3A, each consisting of one compartment accounting for both head and neck (also cf. Table 4). The spines are kept such simple to maintain the focus on the heterosynaptic plasticity dynamics driven by calcium diffusion across the dendritic branch. Synaptic inputs are applied to spines 1 and 3, which increases the level of calcium in these spines. Next, by means of diffusion, the level of calcium in spines 2 and 4 increases as well (Fig 3C). This causes the synaptic weight at these spines to change in a manner that depends on the spatial proximity to the active spines. Fig 3D shows that spine 2 undergoes heterosynaptic potentiation as a result of its proximity to the active spines 1 and 3, whereas spine 4 undergoes heterosynaptic depression due to its remote location and consequently lower levels of calcium. The Arbor results are cross-validated with a custom standalone simulator written in Python [47].

Although the results from both simulators match very well, there is a specific difference between the models that should be mentioned. Namely, the two implementations use different models of the diffusion dynamics with respect to the spine. The custom simulation code utilizes a diffusion equation for the dendrite based on a model by [52], and a time-dependent ordinary differential equation for the spine. Thereby, it does not consider spatial diffusion between spine and dendrite but instead features rate factors that govern calcium exchange between the two segments. In contrast to that, Arbor considers diffusion throughout the whole morphological structure, including the spines, which are modeled as small sub-branches of the dendritic tree. The custom code, however, incorporates distinct influx and outflux coefficients inspired by [52] and [56].

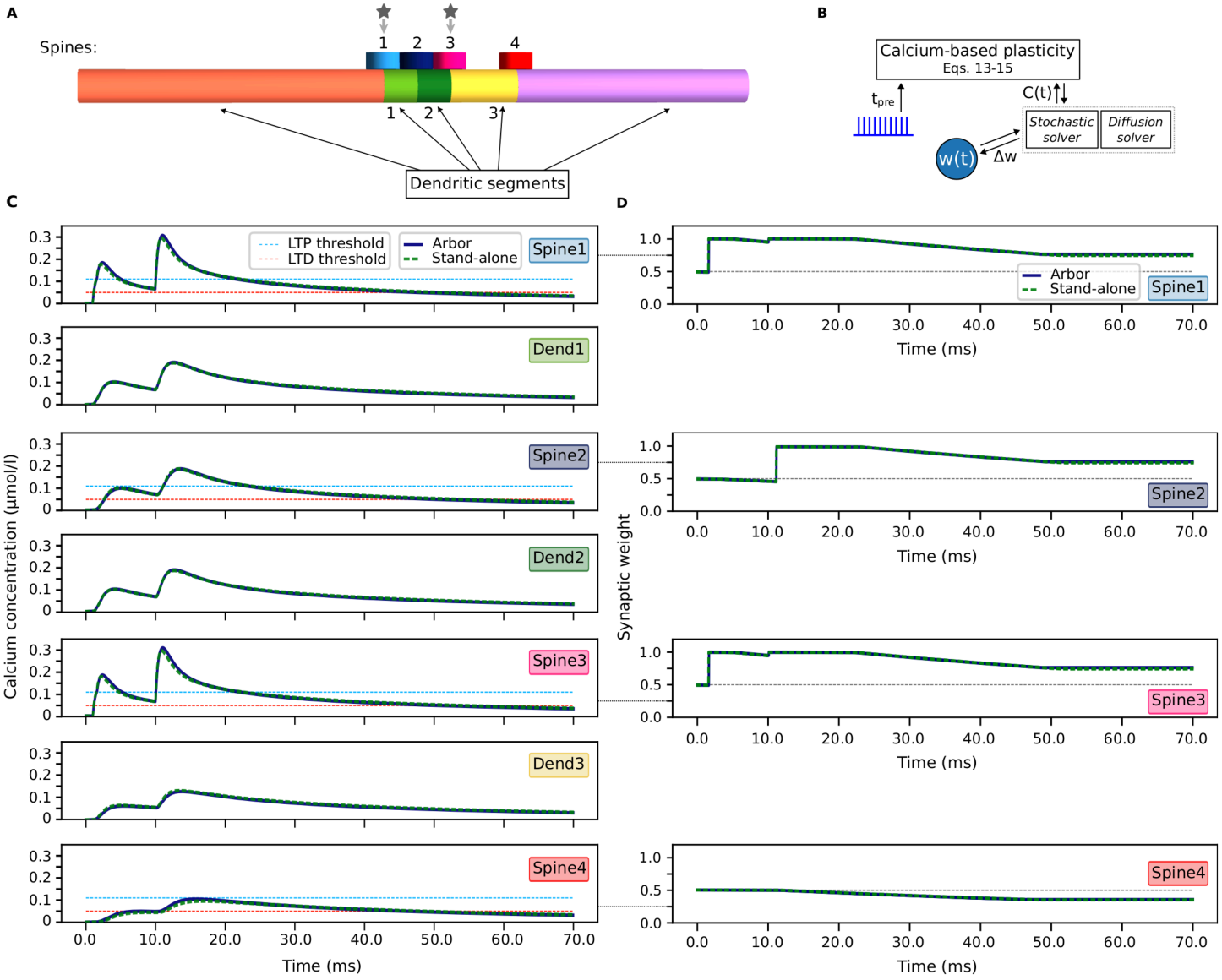


Fig 3. Calcium-driven heterosynaptic plasticity in four spines on a dendritic branch. Calcium is first introduced in spines 1 and 3 through synaptic input. Subsequently, calcium spatially distributes across the dendrite (according to Eq 14), which influences synaptic plasticity at other synapses, promoting either depression or potentiation. The parameter values are provided in Table 4. **(A)** Illustration generated using `Arbor GUI` [55]. Each segment is represented by a different color, and a segment can consist of multiple compartments. Spines 1–4 are located at $x = -1.0, 0.0, 1.0, 3.0\mu m$, respectively. For the purpose of visualization, the morphology has been clipped at $-10\mu m$ and $+10\mu m$, before scaling the dendrite along the x-axis by 2. **(B)** Paradigm of synaptic plasticity that depends on a spike-timing- and rate-dependent, diffusive calcium concentration (cf. [46]). Regular spike trains induce calcium injection in specific spines, eventually leading to weight changes (results shown in (C–D)). New features of the Arbor core code are highlighted in italic. **(C)** The change in the calcium level of each spine, and of the dendritic segments in between, in response to the stimulation to spines 1 and 3. Quantification of deviation between the simulators given by (CV, RMSE): spine 1: (0.991, 0.005µmol/l); spine 2: (0.998, 0.002µmol/l); spine 3: (0.989, 0.005µmol/l); spine 4: (0.967, 0.004µmol/l; dendrite location 1: (0.998, 0.002µmol/l); dendrite location 2: (0.998, 0.002µmol/l); dendrite location 3: (0.998, 0.003µmol/l)). **(D)** Synaptic weight changes, which follow the calcium level of the spines. Spines 1–3 undergo synaptic potentiation (elevated synaptic weights), while spine 4 undergoes depression (reduced synaptic weight). Quantification of deviation between the simulators (CV, RMSE): spine 1: (0.982, 0.015); spine 2: (0.993, 0.014); spine 3: (0.981, 0.016); spine 4: (0.996, 0.003).

https://doi.org/10.1371/journal.pcbi.1013926.g003

Table 4. Parameters for the calcium-based heterosynaptic plasticity model (also cf. Fig 3). Note that the injection current amplitude I_0 varies across implementations due to the differences mentioned in the main text.

Symbol	Value	Description	Refs.
γ_p	90	Potentiation rate	This study
γ_d	0.01	Depression rate	This study
θ_p	0.11μmol/l	Calcium threshold for potentiation	This study
θ_d	0.05μmol/l	Calcium threshold for depression	This study
r_{head}	1.0μm	Spine head and neck radius	[48]
l_{head}	1.0μm	Spine length (including head and neck)	[48]
r_{dendrite}	1.0μm	Dendrite radius	[48]
l_{dendrite}	80.0μm	Dendrite length	This study
Δl_{comp}	1.0μm	Length of one compartment	This study
τ_C	100ms	Calcium decay time constant	[49]
τ_I	1ms	Injection current time constant	[50,51]
γ	0.11	Fraction of current carried by Ca^{2+}	[52]
I_0	4.0pA	Injection current (Arbor implementation)	[52]
I_0	5.5pA	Injection current (stand-alone implementation)	[52]
D	$2.2 \cdot 10^{-10}\text{m}^2/\text{s}$	Calcium diffusion constant	[52–54]

https://doi.org/10.1371/journal.pcbi.1013926.t004

To maintain consistency, we neglected the possibility of different rates in the custom code and used a unified rate for the diffusion between the dendrite and the spines. Accordingly, we needed to adjust the amplitude of the injected current in the custom code to align with Arbor (cf. Table 4).

3.5 Synaptic tagging and capture, in individual synapses and in networks of single-compartment neurons

The early and late phase, i.e., the induction and maintenance, of long-term synaptic plasticity are described by the so-called *synaptic tagging and capture* (STC) hypothesis [57,58]. As a next step for our modeling demonstrations, we reproduce the results of standard protocols eliciting early- and late-phase plasticity at a single nerve fiber or single synapse (cf. the experimental results in [59]). To achieve this, we implemented the complex theoretical model from [23] in Arbor, requiring all of the new core components that we introduced in Sect 2. We cross-validated our Arbor implementation by comparing its results to results from a stand-alone simulator for synaptic memory consolidation written in C++ [60] that was custom-developed and used in the scope of several previous studies [23,61–63]. Note that since the stand-alone simulator considers idealized point-neuron dynamics, we also implemented an approximate point neuron in Arbor, by integration of the current flow over the surface of a very small cylinder (cf. Table 5).

In the following, we provide the mathematical description of the used plasticity model. The parameter values can be found in Table 5. For the other parts of the model, please refer to the code or the original studies [23,63]. Also note that a UML sequence diagram of the model implementation is provided in Fig E in S1 Appendix.

The total synaptic weight

$$w = h + h_0 \cdot z \tag{16}$$

consists of two variable contributions, accounting for the two-phase nature of STC mechanisms. The first contribution is given by the early-phase weight h, while the second one is the late-phase weight z. The factor h_0 is used to normalize z such that it has the same dimension as h. The early-phase weight is described by the following differential equation:

$$\begin{aligned}\tau_h \frac{dh(t)}{dt} = 0.1\,(h_0 - h(t)) + \gamma_{\text{p}} \cdot (10\,\text{mV} - h(t)) \cdot \Theta\left[c(t) - \theta_{\text{p}}\right] \\ -\gamma_{\text{d}} \cdot h(t) \cdot \Theta\left[c(t) - \theta_{\text{d}}\right] + \xi(t),\end{aligned} \tag{17}$$

Table 5. Parameters for the model with calcium-based early-phase plasticity and STC-based late-phase plasticity based on [64] and [23]. The calcium concentration in this model is a dimensionless quantity since it is only considered in the synapses (see main text). We use parameters for the calcium-based early-phase model that were fitted on hippocampal slice data [21,71]. For networks, the calcium contribution parameters are corrected by a factor of 0.6 to account for in vivo conditions (cf. [70]).

Symbol	Value	Description	Refs.
h_0	$4.20075\text{mV} = 0.5\,\frac{\gamma_p}{\gamma_p+\gamma_d}$	Baseline value of the excitatory→excitatory coupling strength	[21,23,64]
$t_{c,delay}$	0.0188s	Delay of postsynaptic calcium influx after presynaptic spike	[21,23,64]
c_{pre}	1.0 (0.6)	Presynaptic calcium contribution (in vivo adjusted)	[21,23,64,70]
c_{post}	0.2758 (0.1655)	Postsynaptic calcium contribution (in vivo adjusted)	[21,23,64,70]
τ_c	0.0488s	Calcium time constant	[21,23,64]
τ_h	688.4s	Early-phase time constant	[21,23,64]
τ_p	60min	PRP time constant	[23,64,66]
τ_z	60min	Late-phase time constant	[23,64,66]
γ_p	1645.6	Potentiation rate	[21,23,64]
γ_d	313.1	Depression rate	[21,23,64]
θ_p	3.0	Calcium threshold for potentiation	[23,64]
θ_d	1.2	Calcium threshold for depression	[23,64]
σ_{pl}	2.90436mV	Standard deviation for plasticity fluctuations	[21,23,64]
p_{max}	10.0μmol/l	PRP synthesis scaling constant (equilibrium PRP concentration under ongoing PRP synthesis)	[23,64,66]
θ_{pro}	$2.10037\text{mV} = 0.5\,h_0$	PRP synthesis threshold	[23,64]
θ_{tag}	$0.840149\text{mV} = 0.2\,h_0$	Tagging threshold	[23,64]
f_{int}	0.1l/μmol	Late-phase factor accounting for PRP integration into the synapse	This study
r_{comp}	$1{\cdot}10^{-3}$μm	Radius of the single-compartment cell	This study
l_{cell}	$2{\cdot}10^{-3}$μm	Length of the single-compartment cell	This study

https://doi.org/10.1371/journal.pcbi.1013926.t005

where τ_h is a time constant, γ_p is the potentiation rate, γ_d is the depression rate, and $c(t)$ is the calcium concentration at the postsynaptic site. Finally, $\xi(t)$ constitutes a noise term that depends on the occurrence of potentiation or depression:

$$\xi(t) = \sqrt{\tau_h \left(\Theta\left[c(t)-\theta_p\right] + \Theta\left[c(t)-\theta_d\right]\right)} \cdot \sigma_{pl} \cdot \Gamma(t) \tag{18}$$

with scaling factor σ_{pl} and Gaussian white noise $\Gamma(t)$ (which has a mean value of zero and a variance of $1/dt$ [43]). Note that this calcium-driven model of early-phase plasticity is based on the model by Graupner & Brunel [21], which we considered in Sect 3.3. Adaptations by [64] and [23] have enabled the model to be compatible with synaptic tagging and capture models (also see [62,65]). As in the original model (cf. Eq 12), the calcium concentration depends on pre- and postsynaptic spikes at times t^n_{pre} and t^m_{post}, and is described by the following equation:

$$\frac{dc(t)}{dt} = -\frac{c(t)}{\tau_c} + c_{pre} \cdot \sum_n \delta(t - t^n_{pre} - t_{c,delay}) + c_{post} \cdot \sum_m \delta(t - t^m_{post}), \tag{19}$$

where τ_c is a time constant, c_{pre} and c_{post} are the spike-induced increases in the calcium concentration, $t_{c,delay}$ is the delay of the presynaptic contributions, and $\delta(\cdot)$ is the Dirac delta distribution. Note that calcium does not have a particular dimension here since it is only considered in the point-approximated synapses. The late-phase synaptic weight, which depends on the early-phase weight $h(t)$, is given by [64]:

$$\begin{aligned}\tau_z \frac{dz(t)}{dt} = \;& p(t) \cdot f_{int} \cdot (1 - z(t)) \cdot \Theta\left[(h(t) - h_0) - \theta_{tag}\right] \\ & - p(t) \cdot f_{int} \cdot (z + 0.5) \cdot \Theta\left[(h_0 - h(t)) - \theta_{tag}\right],\end{aligned} \tag{20}$$

where τ_z is a time constant, $p(t)$ is the concentration of plasticity-related products or proteins (PRPs), f_{int} accounts for the integration of PRPs into the synaptic structure, and θ_{tag} is the tagging threshold. The synapse is considered tagged if the change in early-phase weight $|h(t) - h_0|$ exceeds the tagging threshold. Late-phase potentiation or depression occurs when the synapse is both tagged and PRPs are abundant ($p(t) > 0$). The synthesis of PRPs depends on another threshold crossing and is described by [23,64,66]:

$$\tau_p \frac{dp(t)}{dt} = -p(t) + p_{\text{max}} \cdot \Theta\left[\left(\sum_{\text{synapses}} |h(t) - h_0|\right) - \theta_{\text{pro}}\right] \tag{21}$$

with time constant τ_p, the PRP synthesis threshold θ_{pro}, and the PRP synthesis scaling constant p_{max}. Note that for a single synapse, the sum in the threshold condition reduces to the early-phase weight change of that individual synapse only.

As a first step for our model implementation, we considered basic dynamics of a single synapse with early-phase plasticity, and compared the results from Arbor and from the stand-alone simulator. The resulting curves match very well, as shown in Fig 4C,E,G (also see Fig B in S1 Appendix). To rule out any significant deviations that might be caused by the different numerical methods used by the two simulators, we further checked the validity of both approaches by comparing to a Brian 2 implementation (see Fig C in S1 Appendix; since Brian 2 comes with a Heun solver, which can solve stochastic differential equations that contain multiplicative noise with very high precision, it can provide a benchmark for the accuracy of other simulations).

Next, we considered basic dynamics of a single synapse with late-phase plasticity, which is shown in Fig 4D,F. Under continuous strong stimulation, the early-phase weight reaches its maximum after some time, and the late-phase weight subsequently converges to roughly the same value. Eventually, the early-phase weight decays. Again, we compared the results that we obtained from our Arbor implementation with the stand-alone simulator, finding the curves to match very well. In addition, we compared to Brian 2 again, which also shows a very good match (shown in Fig C in S1 Appendix).

Following the implementation of the basic dynamics of early- and late-phase synaptic plasticity, we used Arbor to reproduce the outcome of experimental standard protocols for early- and late-phase plasticity. Again, our results obtained with Arbor are in agreement with the results from the stand-alone simulator (see Fig F in S1 Appendix). In summary, by matching the aforementioned results, we could prove the validity of our Arbor model implementation (as well as the validity of the stand-alone and Brian 2 implementations [60,67]) with respect to the simulation of early- and late-phase synaptic plasticity.

Next, as a major test for the newly implemented plasticity functionality in Arbor, we employed a recurrent spiking neural network model comprising plastic synapses (which had before been used in [23]). The network consists of 2000 single-compartment neurons that exhibit synaptic connections with a probability of 0.1. A fraction of 1600 of these neurons are excitatory. The synapses within this excitatory population are plastic and follow the model that we considered above (Eqs 16–21). The remaining neurons are inhibitory and connected via static synapses. See Fig D in S1 Appendix for cross-validation plots of the resulting spike transmission.

Using this network model, we aimed to simulate the ‘10s-recall’ and ‘8h-recall’ paradigms that had been investigated in [23]. These paradigms comprise the formation of a cell assembly that retains the memory of a given learning stimulus, and the recall of this memory upon stimulation of half of the original neurons either 10s or 8h later. Here, we chose groups of 100, 150, 200, or 250 neurons receiving the learning stimulus to form the ‘core’ of the learned cell assembly. Subsequently, this cell assembly could become consolidated by the synaptic tagging and capture mechanisms, which we tested via recall stimulation after 8h. Simulating such long biological time spans in reasonable compute time required us to implement a ‘fast-forward’ computation mechanism for phases of slow network dynamics. For this, we first implemented a state-saving mechanism, which enables to stop the Arbor simulation at an arbitrary point, save the synaptic weights and PRP concentrations, and then set up a new Arbor *recipe* with the previous state (note that Arbor already provides a

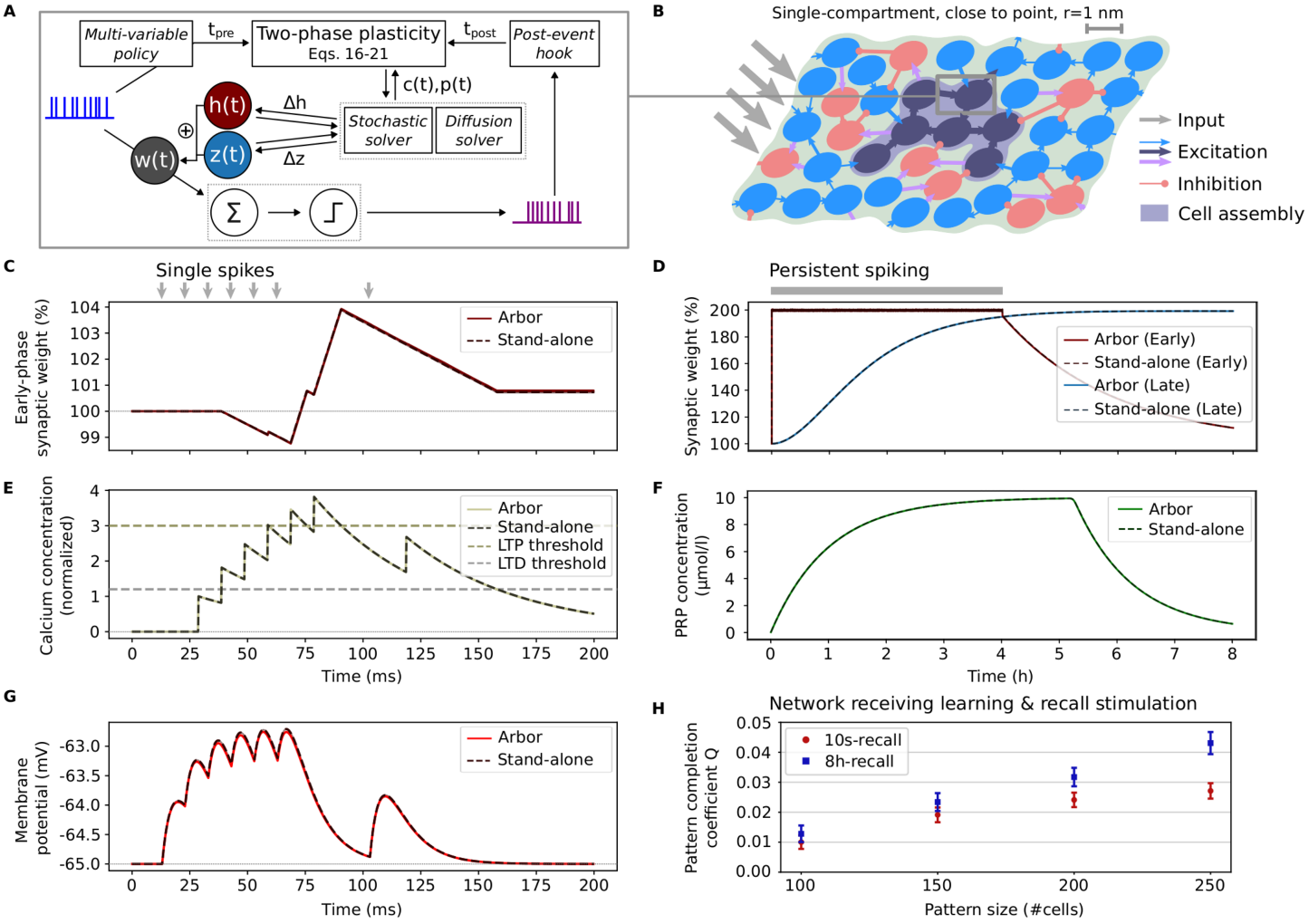


Fig 4. Basic early- and late-phase plasticity with synaptic tagging and capture (STC), cross-validated with stand-alone simulator, and memory recall performance with single-compartment model. (A) Paradigm of two-phase synaptic plasticity with calcium-based early phase and late phase described by synaptic tagging and capture (see [23]). Specific spiking input drives the weight dynamics, which further depend on stochastic dynamics and diffusion of PRP (results shown in (C–G)). New features of the Arbor core code are highlighted in italic. **(B)** Fraction of a neuronal network consisting of excitatory (blue and dark blue circles) and inhibitory neurons (red circles). Following external input, the synapses between excitatory neurons undergo plastic changes implemented as detailed in (A), forming a Hebbian cell assembly (related results in (H)). **(C)** Averaged noisy early-phase synaptic weight (cf. Eq 17). The synapse receives spiking input at pre-defined times (indicated by bold gray arrows). Goodness of fit between the mean curves: CV = 0.999, RMSE = 0.040mV. **(D)** Limit cases of early- and late-phase synaptic weight (cf. Eqs 17 and 20). The presynaptic neuron is stimulated to spike at maximal rate (indicated by gray bar). The late-phase weight has been shifted for graphical reasons (cf. Eq 20; early phase: CV = 0.201, RMSE = 0.226mV; late phase: CV > 0.999, RMSE = 0.055mV). **(E)** Postsynaptic calcium concentration, successively crossing the thresholds for depression (LTD) and potentiation (LTP) (cf. Eq 19; CV > 0.999, RMSE = 0.065). **(F)** The postsynaptic PRP concentration rises until it reaches its maximum through the continued stimulation (cf. Eq 21; CV = 0.998, RMSE = 0.107µmol/l). **(G)** Membrane potential of the postsynaptic neuron (CV > 0.999, RMSE = 0.151mV). Basic early-phase plasticity dynamics (C,E,G): average across 10 batches, each consisting of 100 trials. Baseline levels are represented by fine, dotted lines. Basic late-phase plasticity dynamics (D,F): average across 10 batches, each consisting of 10 trials. Noise seeds were drawn independently for each trial. Results of Arbor are represented by continuous lines, results of the stand-alone simulator [60] by darker, dashed lines. For each curve, error bands represent the standard error of the mean (mostly too small to be visible). **(H)** Memory recall in networks of single-compartment neurons simulated with Arbor (qualitatively reproducing the point-neuron results of [23]). Pattern completion is measured by the coefficient Q (see Eq 22) for stimulated patterns of varied size (a varied number of neurons are stimulated for learning/recall). Average over 100 network realizations; error bars represent the 95% confidence interval.

https://doi.org/10.1371/journal.pcbi.1013926.g004

checkpointing feature, but this does not go as far as to enable changing parts of the recipe). The simulation is then continued with long timesteps for the slow network dynamics without considering spiking and calcium dynamics, which results in much shorter compute time (also cf. runtime results in Sect 3.7 and in Fig M in S1 Appendix). Finally, before performing memory recall, the compute mode is switched again to simulate the network dynamics in full detail. Note that a similar approach had previously been taken with the stand-alone simulator [23,65].

For the plasticity dynamics, we followed the formulation presented above (Eqs 16–21), but since the considered neurons did now hold multiple synapses, an adaptation of the technical implementation became necessary to simulate global PRP dynamics. Specifically, we could no longer compute the PRP dynamics in the NMODL mechanisms of the synapses (cf. the code in [68] and [69]). Instead, we had to compute the weight change sum from Eq 21 in the soma. For this, we needed to model the signaling of the weight changes from the synapses to the soma. We did this by implementing a putative substance that diffuses across the compartments of the neuron. We call this substance the 'signal triggering PRP synthesis' (SPS). In the case of a single-compartment neuron, naturally, the SPS reaches the soma instantaneously (the multi-compartment case is considered in Sect 3.6). In every timestep, the amount of SPS is compared against the threshold θ_{pro} (cf. Eq 21), letting PRP synthesis take place as long as the threshold is crossed. Subsequently, produced PRPs diffuse across the neuron to reach the synapses, where they can give rise to late-phase weight changes.

Finally, to measure the performance in recalling the input pattern defined by the learning stimulus, we use the following quantity [23]:

$$Q := \frac{\bar{\nu}_{\text{ans}} - \bar{\nu}_{\text{ctrl}}}{\bar{\nu}_{\text{as}}}. \tag{22}$$

For this, the population of excitatory neurons is divided into three subpopulations: assembly neurons that are stimulated by both recall and learning stimulus ('as'), assembly neurons that are not stimulated by recall but were stimulated by learning stimulus ('ans'), and control neurons that are stimulated by neither recall nor learning stimulus ('ctrl'). The mean firing rates in the three subpopulations upon 10s- and 8h-recall, computed using time windows of 0.5 s centered at $t_{\text{recall}} = 20.1$ s and $t_{\text{recall}} = 28810.1$ s, are denoted by $\bar{\nu}_{\text{as}}$, $\bar{\nu}_{\text{ans}}$, and $\bar{\nu}_{\text{ctrl}}$, respectively. Thus, values of $Q > 0$ indicate that the pattern is successfully recalled.

The qualitative reproduction of the results from [23] with our Arbor implementation is shown in Fig 4H (also cf. Fig J in S1 Appendix). While we previously found that elementary dynamics of the used plasticity rule match very well for Arbor, the stand-alone simulator, and Brian 2 (Fig 4C–G; Figs B–D in S1 Appendix), the behavior of the full network can not be reproduced in full detail. We attribute this to three factors: First, the different simulators use different numerical solving methods. Second, the neurons in the stand-alone simulator and in Brian 2 are actual point neurons, whereas in Arbor we only consider *approximate* point neurons, which are described by a very small but finite-sized cylinder. And finally, the high complexity of the network dynamics further amplifies the existing differences between the simulators. Thus, although the qualitative behavior is maintained, not all quantitative deviations can be eliminated.

3.6 Synaptic memory consolidation in networks of morphological neurons

Now, we are finally going to demonstrate how we can exploit Arbor's capabilities to simulate networks of multi-compartment neurons with synaptic plasticity. To this end, we extend the size of the cylindrical compartment considered in the previous subsection, split it into two cylinders, add to their middle a compartment for PRP synthesis, and use this as the soma (see Fig 5A,B). We further add two cylinders to represent dendritic branches – one to account for an apical dendrite and one to account for basal dendrites. These branches will have synapses at their tips and are meant to approximate the impact of apical and basal dendritic input onto the soma. The parameters of the morphology are given in Table 6. For simplification, all compartments have the same diameter, and we chose the diameter value to yield a cross-sectional area that effectively provides biologically realistic functional dynamics (cf. [72,73]). Nevertheless, we should note that for a variety of

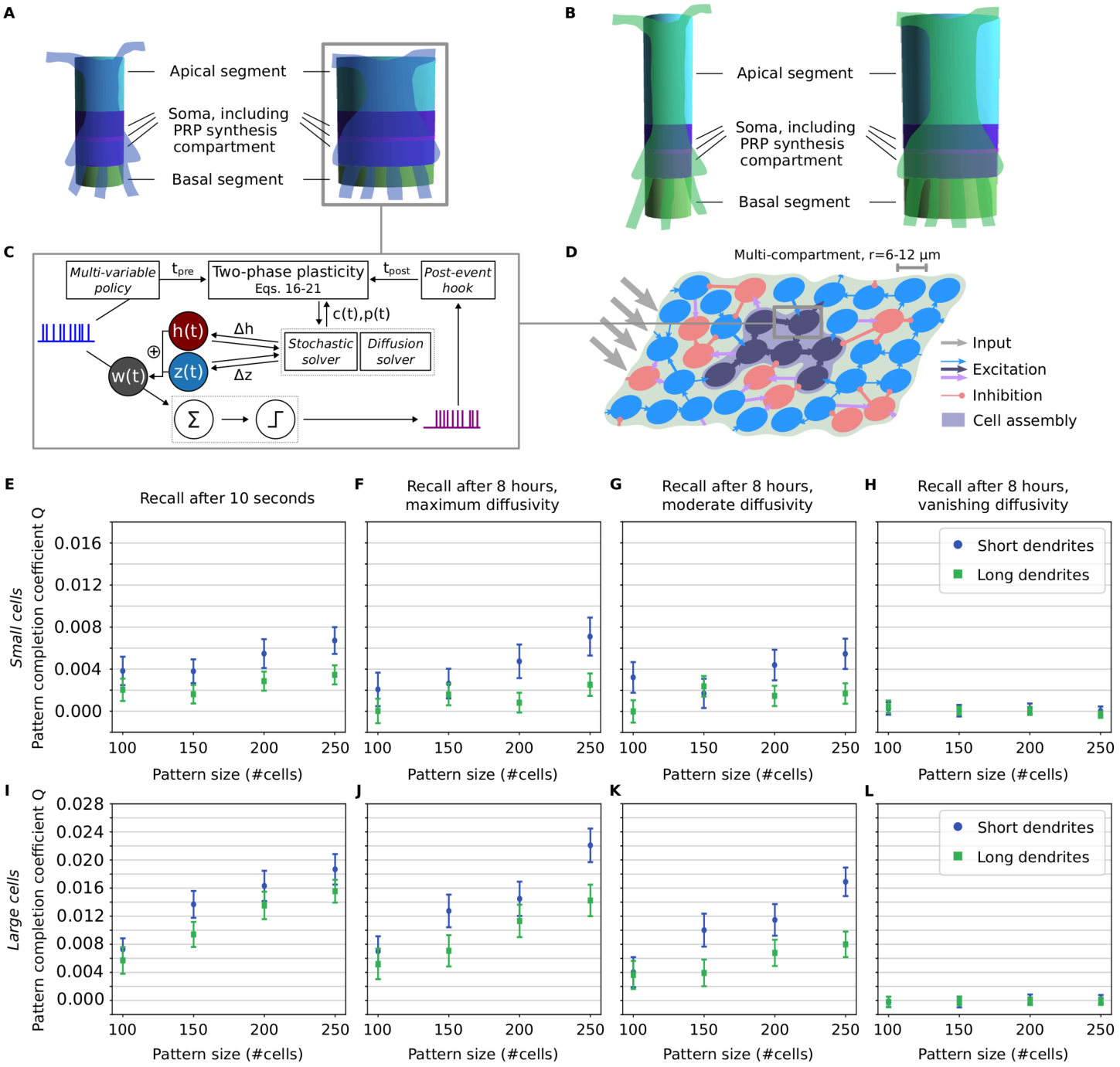


Fig 5. **Memory recall in a recurrent network of multi-compartment neurons after learning and after consolidation.** Results obtained with Arbor for networks of different kinds of multi-compartment neurons, demonstrating the impact of different values of the PRP diffusivity D_p on memory consolidation. Networks consist of 'small' cells (radius of 6µm) or of 'large' cells (radius of 12µm), with either short or long dendrites (in which cases each neuron comprises in total 31 or 48 compartments, respectively). The radius and length values are given in Table 6. **(A,B)** Illustrations of used cell structures, generated using `Arbor GUI` [55]. Each segment is represented by a different color. A segment can consist of a multitude of compartments. Overlaid with illustrations of more realistic neuron structures that would have roughly similar functional properties. **(A)** a small (left) and a large (right) cell with short dendrites, **(B)** the same with long dendrites (cf. Table 6). **(C)** Paradigm of two-phase synaptic plasticity with calcium-based early phase and late phase described by synaptic tagging and capture. The impact of the diffusion of PRPs can be examined using different morphological neuron structures. New features of the Arbor core code are highlighted in italic. **(D)** Fraction of a neuronal network consisting of excitatory multi-compartment

(blue and dark blue circles) and inhibitory neurons (red circles). Following external input, the synapses between excitatory neurons undergo plastic changes implemented as detailed in (C), forming a Hebbian cell assembly (related results in (E–L)). (E–H) Memory recall measured by pattern completion coefficient Q (see Eq 22) for a stimulated subset of varied size (a varied pattern of neurons are stimulated for learning/recall). Value $Q > 0$ indicates the successful recall of a memory representation. Average over 100 network realizations. Error bars represent the 95% confidence interval. **(E)** Recall stimulation at 10s after learning (technically, $D_p = 10^{-11}\mathrm{m}^2/\mathrm{s}$, but late-phase plasticity does not occur on such short timescales). **(F)** Recall stimulation at 8h after learning, $D_p = 10^{-11}\mathrm{m}^2/\mathrm{s}$. **(G)** Recall stimulation at 8h after learning, $D_p = 10^{-15}\mathrm{m}^2/\mathrm{s}$. **(H)** Recall stimulation at 8h after learning, $D_p = 10^{-19}\mathrm{m}^2/\mathrm{s}$. **(I–L)** Same as (E–H), but for large cells that consist of segments of twice the diameter.

https://doi.org/10.1371/journal.pcbi.1013926.g005

Table 6. Cell morphology parameters for the network simulations of memory formation and consolidation with morphological neurons (Sect 3.6). We investigated each combination of the cell and dendrite sizes. The values are chosen to approximate the effective functional dynamics that arise from the structures of real neurons (essentially, pyramidal cells) in hippocampus or neocortex. See the main text for more details.

Paradigm	Symbol	Value	Description	Refs.
(Any)	Δl_{comp}	1.0μm	Length of one compartment	This study
Small cells	r_{comp}	6.0μm	Effective radius of a compartment (used for dendrites as well as soma)	[72,73,76,77]
	l_{soma}	12.0μm	Length of the soma	[72,76,77]
Large cells	r_{comp}	12.0μm	Effective radius of a compartment (used for dendrites as well as soma)	[72,73,76,77]
	l_{soma}	24.0μm	Length of the soma	[72,76,77]
Short dendrites	$l_{\mathrm{dendriteA}}$	12.5μm	Length of apical dendritic branch	[73,76,78]
	$l_{\mathrm{dendriteB}}$	5.0μm	Length of basal dendritic branch	[73,76,78]
	c_{morpho}	1.035 for 'small cells', 1.030 for 'large cells'	Correction factor for the altered impact of postsynaptic potentials due to the morphology	This study (referring to model in [23])
Long dendrites	$l_{\mathrm{dendriteA}}$	25.0μm	Length of apical dendritic branch	[73,76,78]
	$l_{\mathrm{dendriteB}}$	10.0μm	Length of basal dendritic branch	[73,76,78]
	c_{morpho}	1.020 for 'small cells', 1.018 for 'large cells'	Correction factor for the altered impact of postsynaptic potentials due to the morphology	This study (referring to model in [23])

https://doi.org/10.1371/journal.pcbi.1013926.t006

scientific investigations, a more refined morphology may be needed. Here, however, we focused on showcasing the capabilities of Arbor to implement synaptic plasticity mechanisms in multi-compartment neurons, using the mentioned simplified morphologies.

The network is structured such that the apical dendrites receive external input, while the basal dendrites account for the recurrent connectivity of the excitatory neurons within the simulated network. This is grounded by findings on the neocortical layer structure of the neocortex, where basal dendrites receive the inputs from within a layer and apical dendrites receive inputs from other layers [74,75]. Finally, the inhibitory neurons form connections directly onto the soma. To account for the propagation of excitatory postsynaptic potentials along the morphology of the basal dendrites, we introduced a correction factor c_{morpho} for the impact at the soma. Note that otherwise, the electrical properties of the neurons are the same as in the single-compartment case presented in the previous Sect 3.5.

To implement the plasticity dynamics, as in the previous subsection, we use again a putative SPS substance that diffuses from the synapses across the whole neuron with the purpose to signal weight changes to the soma. For the sake of simplicity, we assume that the diffusion of the SPS towards the soma happens very fast, with a diffusivity of $D_{\mathrm{sps}} = 10^{-11}\mathrm{m}^2/\mathrm{s}$. In every timestep, we compare the concentration of the SPS in the center of the soma against a PRP synthesis threshold (cf. Eq 21). PRP synthesis will take place as long as the threshold is crossed. Howsoever, note that here we compare to the *concentration* and not to the *amount* of SPS. This enables a more efficient NMODL implementation, but requires the renormalization of the threshold parameter by scaling it with the total volume of the neuron:

$$\theta^*_{\text{pro}} = \frac{\theta_{\text{pro}}}{V_{\text{tot}}} = \frac{\theta_{\text{pro}}}{\sum_i V_i}, \tag{23}$$

where V_i are the volumes of the individual compartments (the neurons used here comprise up to 48 compartments). Also note that unless the diffusion happens instantaneously, in the multi-compartment case, the SPS concentration measured in the soma will never perfectly reflect the total amount of SPS in the whole neuron, which constitutes an essential difference to the single-compartment case.

For the diffusion of PRPs within the simulated neurons, we again use the same mechanism as described in the previous Sect 3.5. However, here we simulate 'real' diffusion across a spatial morphology structure (cf. Eq 5), for which we decided to consider three diffusivity values (see Fig 5F–H,J–L). For the fastest considered diffusion ($D_p = 10^{-11}\text{m}^2/\text{s}$), the PRPs reach all parts of the neuron almost instantaneously, such that there is no difference to a single-compartment model in this respect (cf. the distribution of PRPs in Fig H in S1 Appendix). However, although we adjusted the c_{morpho} parameter to match the impact of a single postsynaptic potential with the single-compartment paradigm, the electrical properties of the morphological structure show to have an effect on the firing rate of the neurons. This is demonstrated by the results for single- and multi-compartment neurons on memory recall after 10 seconds, which naturally does not depend on PRPs (Figs 5H and 6E,I). On the other hand, considering the long-term dynamics with very slow diffusion ($D_p = 10^{-19}\text{m}^2/\text{s}$), the PRPs will not reach the target synapses within the time window of the synaptic tag and thus cannot elicit late-phase plasticity. For moderate diffusion values ($D_p = 10^{-15}\text{m}^2/\text{s}$), PRPs reach the synapses after a certain time that arises from a complex interplay between synapses, soma, and dendrites (see the spatial distribution in Fig I in S1 Appendix). These dynamics still enable functional memory recall and can serve to regulate the late-phase maintenance of synaptic changes. For increased length of the dendrites, however, the memory recall performance tends to become worse (see Fig 5F–H,J–L; also cf. the results for the mutual information in Fig K in S1 Appendix). Interestingly, increasing the size of the neurons (measured by the diameter of the soma and dendrites) has a converse effect: the memory recall is improved (Fig 5J–L).

Although our present study has been focused on simulation methods, these last findings may provide interesting theoretical insights into the role of neuronal structure and dynamics for cognitive functionality at the network level. As the presented results show, Arbor enables to seamlessly move from single- to multi-compartment neurons in a complex network model, leaving the remaining parts of the model unchanged. In the future, the framework that we have developed can be used as a basis for further investigations on neural networks involving diffusion dynamics in multi-compartment neurons.

3.7 Runtime and memory benchmarking with the synaptic memory consolidation model

Employing the network model introduced in the previous section, we scrutinize the resources required by Arbor to simulate networks of single- and multi-compartment neurons in different computing environments. To this end, we consider the networks of 2000 single- or multi-compartment neurons with the 10s-recall paradigm that we used before (Sects 3.5 & 3.6). After running these network models with different Arbor backends on different hardware systems, we compare their runtime and memory use. In the single-compartment case, we also compare to Brian 2 with `cpp_standalone` device [4,79] as well as to the custom stand-alone simulator [60] considered before.

Fig 6 shows that the point-neuron simulators (Brian 2 and stand-alone) need shorter runtimes and less memory compared to Arbor. This is to be expected: Since point-neuron simulators consider neurons without any geometrical structure, they theoretically require much less calculation steps than Arbor, which accounts for at least one finite-size compartment per neuron. Furthermore, the custom stand-alone simulator is highly optimized for the particular model, and can therefore be thought to set an upper bound. Nevertheless, we found that Arbor's capability of employing GPU hardware can boost its runtime to be even faster than Brian 2 and the custom simulator (Fig 6). Another point to be mentioned is Arbor's support for single instruction, multiple data (SIMD) vectorization. As shown in Fig 6, switching on SIMD vectorization provides a small but solid improvement in runtime. This comes without any cost for the end user, given that they have a CPU that

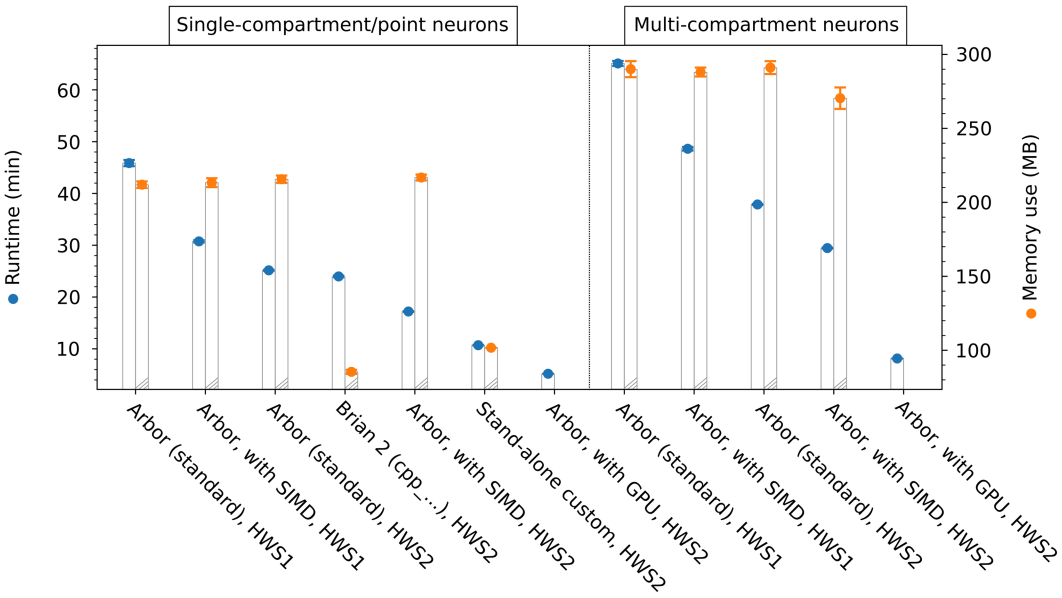


Fig 6. Benchmarking results of runtime and memory use with the synaptic memory consolidation model in Arbor and point-neuron simulators. For 10s-recall paradigm in networks of 2000 neurons. The single-compartment simulations in Arbor as well the point-neuron simulations in Brian 2 (with `cpp_standalone` device) [4,79] and in the custom stand-alone simulator [60] are conducted as described in Sect 3.5; they are represented by data points on the left-hand side. The Arbor simulations with multi-compartment/morphological neurons of 48 compartments are conducted as described in Sect 3.6 and represented by data points on the right-hand side. Results are given for different hardware systems, HWS1: an older desktop computer (Intel Core i5-6600 CPU @ 3.30GHz, 1 × 8GB DDR3-RAM, using 1 thread), HWS2: a newer compute server (AMD Ryzen Threadripper PRO 5995WX CPU, 8 × 32GB DDR4-RAM, using 1 thread, in specified cases with NVIDIA T1000 8GB GPU). For Arbor, results are distinguished between standard CPU execution, CPU with SIMD support, and with GPU support. The respective left bars with blue data points show the total runtime of the simulations (comprising initialization and state propagation phases). Measurements were performed using `hyperfine` in version 1.15. The respective right bars with orange data points show the use of main memory, given by the maximum over time of the number of 'dirty' bytes, including private and shared memory, as returned from `pmap`. Note that for the GPU cases considering the main memory use is not meaningful, since the GPU has its own additional memory. Data points represent the average over 10 trials; error bars represent the standard deviation. Also cf. Fig L in S1 Appendix.

https://doi.org/10.1371/journal.pcbi.1013926.g006

supports SIMD, which has been the industry standard for many years. The only drawback of SIMD usage in Arbor might be its negative impact on code readability when developing custom mechanism code in C++, which is rather a niche case.

Importantly, our results show that Arbor allows to shift from single-compartment neurons to morphological neurons at *almost no cost*: both the runtime and the memory consumption only increase slightly when shifting from single-compartment neurons (Fig 6, left-hand side) to neurons with 48 compartments (Fig 6, right-hand side). However, it is important to note that the considered multi-compartment network in general exhibits fewer spikes than the single-compartment version (cf. Fig L in S1 Appendix; also cf. Sect 3.6). Thus, given that the number of spikes is a critical factor for the runtime, we checked if the runtime per spike follows as similar trend, and indeed found that this measure also exhibits a slight increase only (Fig L in S1 Appendix).

Hence, while Arbor performs well in single-compartment neuron simulations, it excels in multi-compartment neuron models, providing all the necessary functionality to simulate morphological neurons with electrical cable properties and diffusing particles.

Finally, note that we also benchmarked results for 8h-recall simulations in Arbor (Fig M in S1 Appendix). The runtimes of those simulations are much longer than for 10s-recall (cf. Fig 6). As we used the 'fast-forward' approximation (see Sect 3.5), there are only $10-20\%$ more timesteps than in the 10s-recall paradigm, so only a small fraction of the increase in runtime can be attributed to additional timesteps. Instead, we found the longer runtimes to be essentially due to a large overhead needed for switching between full and fast-forward computation, specifically, the setting of a large number of

probes to store the state of the whole network. To counter this, in the future, we plan to augment our framework by introducing new mechanisms that serve to retain the simulation state.

3.8 Runtime and memory benchmarking with large-scale networks

To underline our claims of Arbor's scalability and efficiency, in this final section, we consider a set of technical benchmarks based on the so-called busyring benchmark. This is a tunable, well-understood workload that enables stressing various parts of the underlying hardware architecture. It has been used in this capacity as an acceptance benchmark for the novel JUPITER system at the Jülich Supercomputing Centre (JSC) [80].

The basic unit of work in the busyring benchmark is a ring of k neurons tuned to propagate a single spike indefinitely (see Fig 7E). Connections within the ring are realized via conductance-based exponential synapses with a uniform delay t_{delay} and weight $w \neq 0$. Thus, each neuron spikes at frequency $\nu = \frac{1}{k \cdot t_{\text{delay}}}$. In addition, a set of s connections are placed between ring neurons and random endpoints, where $w = 0$. These connections generate computational load in the code paths responsible for spike transmission and delivery, yet have no effect on the target neurons. The cell model is generated randomly as a tree of given depth, which is done individually per cell to avoid early-stage optimizations. The neuron is then endowed with Hodgkin-Huxley dynamics [81] on the soma region and a plain leak current on the remaining morphology. We call this neuron model `simple-branchy`. To consider the computational load of plasticity dynamics, we further implemented a variant of the benchmark where the synapses targeted by $w = 0$ connections use an STDP model (cf. Sect 3.1).

We measured the resulting wallclock times on our hardware system 'HWS2' that we already used in Sect 3.7, using the parameter values from Table 7. Next, we determined the optimal runtime on this multi-core CPU system by selecting the best performance across the given ranges of MPI ranks and CPU threads, and subsequently compared the results for CoreNEURON, Arbor, and Arbor with STDP synapses (Fig 7A–C). From this, we observe a tremendous speedup provided by Arbor, even if STDP dynamics are considered. Note that our STDP implementation relies on the `POST_EVENT` hook, which requires additional memory as well as additional runtime for storing and sorting and for dispatching the spike events to the synapses (while particular STDP rules may probably be implemented even more efficiently, the `POST_EVENT` hook offers more general functionality). Fig 7D shows that the speedup is also maintained across network sizes, where both CoreNEURON and Arbor scale almost linearly. Furthermore, disentangling the setup and state propagation phases of the simulations, we find almost linear scaling for these particular measures as well (Fig 7F). Exact runtime values, also for different dendritic tree depths, are provided in Table 8. Note that Arbor's performance may be increased even further by appropriately adjusting the `cpu_group_size` setting for each particular case (here, all results are for `cpu_group_size=1`).

Regarding the memory use, we also observe a much better efficiency of Arbor as compared to CoreNEURON, while optimal results are achieved for the lowest rank and thread numbers (Fig N in S1 Appendix). Again both simulators scale almost linearly (see Table A in S1 Appendix for more details).

When additionally utilizing the existing GPU in the HWS2 setup described above, further speed gains can be achieved. In general, due to their large numbers of cores, it is expected that GPUs will accelerate simulations of large networks, which is supported by our results. Comparing Fig 7F and Fig 7G, we see that for large networks (here: 32768 neurons), especially if they also feature plasticity dynamics, the GPU brings benefits for both setup and propagation phase (see dark gray bars in Fig 7G; also cf. Tables 8 and 9). However, for smaller networks of 1024 neurons, using the GPU can even slow down the simulation. For the exact runtime values with GPU, including such for different dendritic tree depths, see Table 9. Note that for CoreNEURON, none of the simulations finished because in all cases the compute system ran out of memory.

By choosing different values for t_{delay}, k, and s, the workload can be adapted to investigate the network behavior of different real-world models. Considering different cell models further allows the emulation of their computational workload in

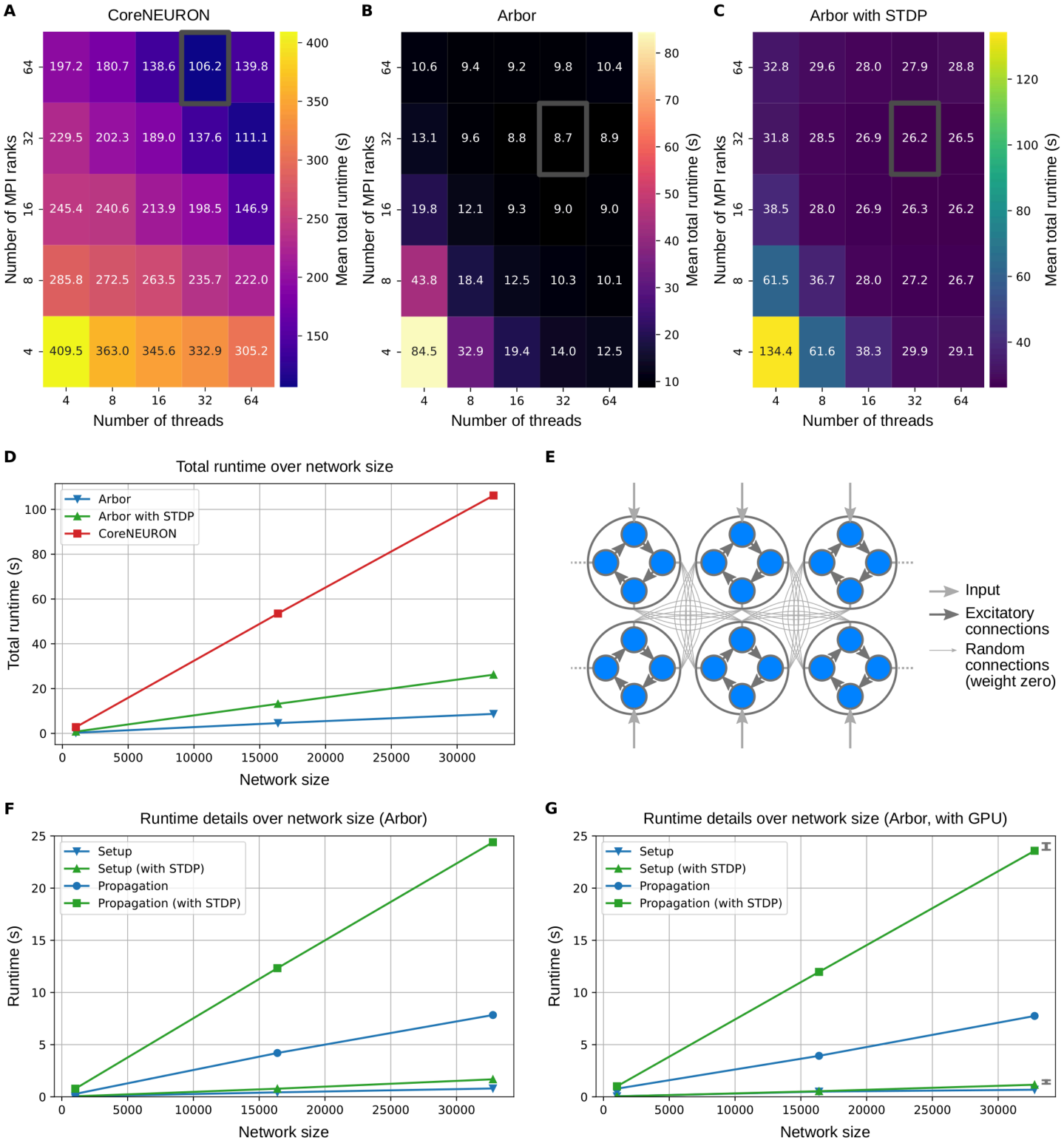


Fig 7. Benchmarking of simulation runtime for large-scale networks. (A–C) Total wallclock time to initialize and execute a simulation with 32768 cells over 200ms in Arbor and CoreNEURON. A busyring network of `simple-branchy` cells with tree depth 2 is used, run on the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM) with **(A)** CoreNEURON, **(B)** Arbor with SIMD, **(C)** Arbor with SIMD with STDP mechanisms for the random synapses. The respectively fastest paradigm for each implementation is highlighted by the gray box. **(D)** Scaling of

the fastest results for the total runtime over network size. **(E)** Sketch of the busyring network consisting of rings of integrate-and-fire neurons (shown as blue disks), connected internally via excitatory synapses, and across the whole network via random synapses of weight zero. One neuron of each ring receives external stimulation. **(F)** Scaling of the setup and propagation runtime related to the total runtimes in **(D)**. **(G)** Scaling of the setup and propagation runtime for fastest total runtime using an additional NVIDIA T1000 8GB GPU. For the case with STDP, the GPU-mediated speedup is indicated by dark gray bars. All values are averaged over 10 trials, with coefficient of variation CV < 0.06 in all cases. See Tables 8 and 9 for more details.

https://doi.org/10.1371/journal.pcbi.1013926.g007

Table 7. Parameters for benchmarking large networks. Used for benchmarking large networks with busyring in Arbor and CoreNEURON (STDP parameters are only used in Arbor). For CPU threads and MPI ranks, all combinations of powers of 2 from the given ranges were considered (the ranges were chosen according to the 64-core CPU of the HWS2 system).

Description	Symbol	Value
Number of CPU threads	-	4 – 64
Number of MPI ranks	-	4 – 64
Simulated time	t_{duration}	200ms
Timestep	Δt	0.025ms
Synaptic delay	t_{delay}	5ms
Synaptic time constant	τ	2ms
Decay of the STDP eligibility trace of presynaptic spikes	τ_{pre}	10.0ms
Decay of the STDP eligibility trace of postsynaptic spikes	τ_{post}	10.0ms
STDP strengthening amplitude	A_{pre}	0.01μS
STDP weakening amplitude	A_{post}	−0.01μS
Maximum synaptic weight for STDP	w_{max}	10.0μS
Ring size	k	4
Random connections per cell	s	1000

https://doi.org/10.1371/journal.pcbi.1013926.t007

Table 8. Wallclock time measurements for busyring benchmark. Total runtime results are provided for networks of `simple-branchy` cells as reported by Arbor and CoreNEURON (using the fastest paradigm as detailed in Fig 7A–C). The shares of the setup and state propagation phases are given in brackets, respectively. Results are collected with the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM, GPU not used). All values are averaged over 10 trials, with coefficient of variation CV < 0.06 in all cases. Arbor with SIMD. In CoreNEURON, the most extensive simulation did not finish (d.n.f.) due to exceeded memory. See Table 9 for results with GPU.

Number of cells	Tree depth	Number of synapses	Number of compartments	Runtime of CoreNEURON (s)	Runtime of Arbor (s)	Runtime of Arbor with STDP (s)
1024	0	1.03M	1024	2.83 (0.75+2.07)	0.23 (0.04+0.19)	0.73 (0.05+0.68)
	2	1.03M	46116	2.81 (0.78+2.04)	0.32 (0.04+0.28)	0.84 (0.05+0.79)
	10	1.03M	1672700	6.23 (0.65+5.58)	2.67 (0.19+2.48)	3.22 (0.21+3.01)
16384	0	16.41M	16384	51.49 (5.28+46.21)	3.24 (0.37+2.86)	11.36 (0.75+10.61)
	2	16.41M	737408	53.52 (5.32+48.20)	4.64 (0.41+4.23)	13.18 (0.81+12.37)
	10	16.41M	26616896	163.11 (7.47+155.64)	43.07 (2.35+40.72)	52.48 (2.58+49.90)
32768	0	32.81M	32768	103.61 (9.57+94.04)	6.11 (0.80+5.31)	22.49 (1.48+21.00)
	2	32.81M	1475268	106.17 (9.90+96.27)	8.68 (0.82+7.86)	26.16 (1.72+24.45)
	10	32.81M	52957704	d.n.f.	85.75 (4.65+81.09)	104.66 (5.05+99.60)

https://doi.org/10.1371/journal.pcbi.1013926.t008

network models. The relation of both network and cell parameters can then provide a holistic image of the performance of varying network models. In addition to the `simple-branchy` cell with Hodgkin-Huxley dynamics that we considered above, Arbor's busyring implementation features another pre-configured cell model called `complex`. This model comprises a set of eight channel types on the soma, including kinetic schemes and calcium concentration models, as well as five channel types on the remaining morphology. In this way, the model approximates a cell from the mouse visual cortex [82]. A runtime comparison between `complex` and `simple-branchy` is shown in Table B in S1 Appendix.

The scaling behavior of Arbor enables the simulation of even larger and more complex network models than the ones we considered so far in this study. As an outlook, we show in Fig O in S1 Appendix first results of strong scaling on state-of-the-art supercomputing systems at JSC (the preview system JEDI and the current flagship system JUWELS) with an extremely large network of 10^6 cells of the `complex` type (nevertheless, without plasticity dynamics). The parameters for

Table 9. Wallclock time measurements for busyring benchmark with GPU. Total runtime results are provided for networks of `simple-branchy` cells as reported by Arbor (using the fastest paradigm as detailed in Fig 7B,C). The shares of the setup and state propagation phases are given in brackets, respectively. Results are collected with the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM, with NVIDIA T1000 8GB GPU). All values are averaged over 10 trials, with coefficient of variation CV < 0.06 in all cases. Arbor with SIMD. In CoreNEURON, due to exceeded memory, none of the simulations finished. See Table 8 for results without GPU.

Number of cells	Tree depth	Number of synapses	Number of compartments	Runtime of Arbor (s)	Runtime of Arbor with STDP (s)
1024	0	1.03M	1024	0.32 (0.02+0.30)	0.81 (0.03+0.78)
	2	1.03M	46116	0.44 (0.02+0.41)	1.03 (0.03+1.00)
	10	1.03M	1672700	3.13 (0.14+2.99)	3.65 (0.15+3.50)
16384	0	16.41M	16384	3.10 (0.29+2.81)	10.94 (0.50+10.44)
	2	16.41M	737408	4.46 (0.32+4.15)	12.60 (0.56+12.04)
	10	16.41M	26616896	43.51 (3.07+40.45)	52.38 (3.33+49.05)
32768	0	32.81M	32768	5.85 (0.55+5.30)	21.51 (1.05+20.47)
	2	32.81M	1475268	8.48 (0.70+7.78)	24.84 (1.13+23.71)
	10	32.81M	52957704	86.62 (6.18+80.44)	104.46 (6.71+97.75)

https://doi.org/10.1371/journal.pcbi.1013926.t009

both systems are given in Table C in S1 Appendix. As the NVIDIA H100 GPU model used in JEDI offers roughly twice the memory bandwidth and more than twice the floating-point performance of the A100 of JUWELS, we compare one JEDI node to two nodes of JUWELS. Within the strong scaling range over an eightfold increase of nodes in the simulation, we observe a scaling efficiency ϵ of well over 80%. The scaling efficiency is defined by

$$\epsilon = \frac{T(n_0)}{T(n) \cdot n},$$

where $T(n)$ is the wallclock time measured in the strong scaling series with n nodes and reference $n_0 = 4$.

4 Availability and future directions

4.1 Discussion and outlook

In this work, we have aimed to demonstrate the versatility of the extended Arbor simulator in modeling synaptic plasticity mechanisms within large neuronal networks. Specifically, we considered Arbor implementations of homosynaptic, homeostatic, and heterosynaptic plasticity mechanisms in different setups. In Sects 3.1 & 3.2, we presented plasticity rules that can be considered a basis for further spike-based mechanisms of synaptic plasticity. In Sects 3.3–3.5, we considered three different calcium-based rules in different scenarios. The results in Sect 3.3 constitute a reproduction of a widely-used calcium model that was directly fitted to experimental data [21]. The results in Sect 3.4 show how a model of calcium diffusion along dendrites can be employed to simulate heterosynaptic plasticity (cf. [22,46]). In Sect 3.5, we used a calcium model as basis for a more complex model that captures early- and heterosynaptic late-phase plasticity [23,64,65]. Finally, we moved to the network level (Sects 3.5 & 3.6), where we built on the complex two-phase learning rule introduced before. Here, we first reproduced previous results on memory recall [23] using single-compartment neurons, and then extended the neurons by additional morphological segments accounting for dendritic structure. By this, we could demonstrate how (Plastic) Arbor serves to seamlessly gather computational insights into the impact of morphological neuron structures in large networks. In particular, our results from the multi-compartment model provide new insights showing that long dendritic structures can have a deteriorating impact on memory function at the network level, and that the PRP transport velocity in these structures might only play a minor role (Fig 5E–H). Furthermore, a large cell diameter can have a converse effect, yielding enhanced memory recall (Fig 5I–L).

In the following, we will briefly discuss the pros and cons of the most prominent neural network simulators, and compare their target use cases with Arbor. With its first version released in the 1980s and widespread usage, NEURON has

significantly advanced the understanding of the brain by facilitating the computational study of complex neuronal processes [73]. Nevertheless, as mentioned above, its engine under the hood is outdated when it comes to high-performance computing, especially, for network simulations involving detailed neuron models. While CoreNEURON [2,5] constitutes an approach to address this, it has restricted flexibility and usability due to its dependence on the NEURON environment, and it lacks support for certain important features of NMODL [83]. Further, GENESIS (GEneral NEural SImulation System) [84] is a simulator that has also been used widely for several decades, offering a platform for multi-scale simulations with particular focus on detailed electrical and chemical interactions. While GENESIS is very limited with respect to modern hardware backends, the MOOSE simulator is developed as its modern successor for the efficient simulation of biochemically detailed dynamics [85]. Regarding alternatives that are more focused on point-neuron simulations, NEST (NEural Simulation Tool) is a widely used simulator designed for large-scale simulations of spiking neural networks. It is particularly known for its scalability and efficiency in simulating large-scale networks. However, its lack of support for multi-compartment neurons has been one of the initial reasons to develop Arbor, which was therefore originally named 'NestMC'. Brian 2 is another widely used network simulator that enables the definition of models directly via differential equations [4]. Furthermore, Brian 2 enables high flexibility by generating an intermediate abstract code, processed by so-called device interfaces. This allows to seamlessly exchange the underlying numerical backend architecture. While Brian 2 does not yet come with comprehensive support for multi-compartment neurons, a new extension called Dendrify specifically focuses on detailed dendritic morphology and may provide a valuable tool for the investigation of dendritic processing [86]. CARLsim [87] is another framework for the parallelized simulation of large-scale spiking neural networks, but it has been optimized for real-time simulations and neuromorphic hardware implementation. Accordingly, the single-compartment Izhikevich neuron is the most biologically realistic neuron model that is supported by CARLsim. Finally, the recently released simulator EDEN [88] appears to offer high flexibility by supporting NeuroML model descriptions, however, it still lacks support for GPU backends. As opposed to that, the equally new simulator NEST GPU (previously named NeuronGPU) [89,90] is optimized for neural network simulations on GPU, however, it does not support multi-compartment models. Similarly, the GeNN simulator also offers highly optimized GPU simulation for networks of point neurons [91].

In summary, each of the existing simulators comes with its own strengths and weaknesses (also see [92–94]). The decision to use one specific simulator should depend on the particular needs of a research project, the available compute resources, as well as the expertise of the involved researchers. For example, Brian 2 can be considered quite user-friendly for quickly setting up networks of point neurons, NEST exhibits unique performance for very large network simulations, and for NEURON there are many existing implementations of morphological neuron models. Arbor, finally, was designed to easily define networks of morphological neurons and then to map the internal modeling primitives to available compute resources in an optimized manner. As we have shown, this approach enables very good runtime and memory efficiency and makes Arbor an attractive option, particularly, for researchers who rely on high-performance computing for their simulations with networks of morphologically detailed neurons and existing NEURON models.

We have cross-validated all of our presented Arbor implementations either with Brian 2 [4] or with one of multiple standalone simulators that were custom-developed for previous studies [21,47,60]. At the single-synapse level, we did not find any significant differences between the results of the simulators (Figs 2–5; Figs A–C in S1 Appendix). However, this does not preclude that certain differences may arise in other simulation paradigms, where the validity of the particular simulators that are used should be carefully evaluated (also cf. [95]). Furthermore, differences in the neuron model and in the numerical methods, as well as complex network effects, have shown to give rise to certain quantitative deviations in memory recall performance at the circuit level, while the qualitative behavior remained the same (Fig J in S1 Appendix; also cf. Sect 3.5 and Fig D in S1 Appendix).

Regarding the use of compute resources for a large network with plastic connections, we found that Arbor performs well both in terms of runtime and memory use. Compared to optimized point-neuron simulators, Arbor only uses slightly

more resources when computing on CPU, and can even outperform those simulators when using its capability to compute on GPU (Fig 6). However, Arbor particularly shines when it comes to simulating networks of multi-compartment neurons, which necessitates almost no additional cost compared to single-compartment neurons (shown by the runtime per spike, Fig L in S1 Appendix). This is not entirely unexpected, since Arbor has been designed particularly for networks of multi-compartment neurons. Furthermore, we found that Arbor outperforms its main competitor in this regard, CoreNEURON, in terms of runtime and memory use across all considered network sizes, even with an additional plasticity workload (Tables 8 & 9, Table B in S1 Appendix).

The benchmarking results notwithstanding, there may be several ways to even further improve the performance of Arbor. First, to compute the dynamics of given models, Arbor uses implicit solving methods, which have the advantage of being stable but come at the cost of runtime performance. In specific cases, these methods may be replaced by faster explicit algorithms. Kobayashi et al. [96], for instance, have shown how an explicit method with adaptive time steps and second-order accuracy can serve to avoid heavy memory access, which can be helpful particularly when using GPUs. Second, the introduction of exact point neurons may eliminate the need to simulate a spatial neuron model if this is not needed. Note that meanwhile, a LIF neuron feature has been added to Arbor, although still being in a test stage.

By the final model simulation results that we have presented in Fig 5, we could gain first new insights into the functional interplay between, on the one hand, network parameters such as the size of a stimulated pattern, and on the other hand, PRP diffusion within neurons. Specifically, the results suggest that the functionality of pattern recall is not very sensitive to the diffusion speed. Nevertheless, if the diffusion is too slow, as expected, all memory functionality will vanish, which also occurs if the pattern size is too small. Moreover, if we increase the length of the dendritic branches (in a range that roughly relates to lengths of main branches in cortical pyramidal cells [73,76,78]), the memory recall capability is impaired (Fig 5F–H), indicating that additional mechanisms may be needed to obtain a stable memory system. Somehow in contrast to that, a larger cell and dendrite diameter improves memory recall (Fig 5J–L). By targeting the impact that PRP diffusion within neurons has on the dynamics of large networks, our findings may complement the picture that other studies have drawn of the functional role of spatial PRP dynamics within neurons [97–101]. Nevertheless, it is important to mention that in this work, we considered simplified neuronal morphologies in order to focus on the principles of simulating the considered synaptic plasticity mechanisms. In general, at the cost of requiring more computational resources, Arbor allows to use much more detailed realistic morphologies, which can be loaded from SWC, NeuroML 2, or NeuroLucida ASCII files as they are extracted from many experimental datasets.

In the future, most importantly, the extended Arbor simulation framework can enable researchers to conduct studies that examine the interplay between neuron-internal, synaptic, and network dynamics. These may, for instance, be related to memory consolidation [23,65,102] or working memory [103,104]. Moreover, Arbor allows the implementation of models that include changes in the connectivity structure of networks. Using this together with the tools that we have presented here will enable researchers to also study the interactions between structural and functional plasticity processes at the network level (cf. [105–109]). Another important goal will be to add the models that have so far been implemented in Arbor to open databases, such as the Open Source Brain repository or ModelDB, for broader adoption and community-driven validation. It would also be intriguing to conduct collaborative studies where Arbor is used to simulate electrophysiological and network properties, while at the same time using calcium imaging experiments to validate diffusion dynamics. This could serve to validate models as given in Sect 3.4 [46]. Furthermore, spine density distributions derived from in vivo experiments [110,111] could be used, for example, to test diffusion models in more detail. Finally, the large number of available SBML kinetic definitions [112] may provide a vast source of models, however, according Arbor implementations will require to reconcile temporal and spatial scales. The central interfaces that Arbor offers to implement ion channels in the cable model are ionic (transmembrane) currents and ionic concentrations. Thereby, SBML models targeting any of these variables can be considered viable for translation to NMODL or for connecting to Arbor cell models directly.

It should further be noted that by considering synaptic plasticity processes in neural networks, simulation software contributes essentially to the understanding of biological mechanisms as well as to the development of artificial intelligence applications. In this light, further applications of our extended version of Arbor may target, for example, paradigms of somato-dendritic mismatch error reduction [113], reservoir computing with heterogeneous networks [114,115], or prototyping and benchmarking of neuromorphic algorithms (cf. [93,116,117]).

Due to its modern computing architecture and inherent support of multi-compartment neurons, the Arbor simulator constitutes an important tool for the computational modeling of neuronal networks. Through the newly introduced extensions, enabling to simulate synaptic plasticity, we have increased the range of Arbor's use cases substantially. Furthermore, as an example, we have provided a first demonstration of how the extended framework enables to gain new insights into the functional impact of morphological neuronal structure at the network level. With its newly extended functionality, the Arbor framework is able to power a great variety of future studies considering synaptic mechanisms and their interactions with neuronal morphologies, from single synapses to large networks, in a highly efficient manner.

Data and code availability

All data presented in this study can be reproduced using publicly available simulation code and analysis scripts, which are listed in the following.

The Arbor framework can be installed as described on https://arbor-sim.org. The code can be retrieved, e.g., from [7]. Our simulation code for the different subsections is referenced in Table 10 below. The code of the Arbor implementations can also be found at https://doi.org/10.5281/zenodo.18088337. Note that since Arbor is still under development, some parts of our model implementations may be subject to changes when using later Arbor versions, however, we intend to keep the model implementations updated along with the Arbor core code.

Due to the ongoing development of the Arbor core code, we ran our extensive simulations with different Arbor versions. For the simulation results presented in subsections 3.1–3.3 as well as 3.5–3.6, we used Arbor version 0.9.1-dev-2f4c325 (for instructions on how to install this exact version, see the README.md file that comes with our model code [69]). For the simulation results of subsection 3.4, we used the development branch of pull request #2226 with the state of commit f0e456d, which has been merged since Arbor version 0.10.1-dev-257e74f (note that at that state of commit, Arbor still used diffusion constants in non-standard units, scaled by a factor of 10^{-7}). For the benchmarking results in subsection 3.7 we used Arbor version 0.10.0, and for those in subsection 3.8 we used the development branch of pull request #2489

Table 10. Overview of the simulation code used to perform the simulations presented in this article. The code of the Arbor implementations can also be found at https://doi.org/10.5281/zenodo.18088337.

Section	Model description	Arbor implementation	Reference implementation(s)
3.1	Spike-timing-dependent plasticity	[119] (subdirectory `STDP/`)	[119] (subdirectory `STDP/`)
3.2	Spike-driven homeostatic plasticity	[119] (subdirectory `spike_based_homeostasis/`)	[119] (subdirectory `spike_based_homeostasis/`)
3.3	Calcium-based synaptic plasticity	[7] (Arbor main repository, `calcium_stdp.py`)	[120]
3.4	Heterosynaptic calcium-based plasticity in dendrites	[47] (`Arbor_diff.py`)	[47] (`Custom.py`)
3.5	Synaptic tagging and capture, in individual synapses and in networks of single-compartment neurons	[68] (reduced code for single synapses) and [69]	[60]; [67] (reduced code for single synapses) and [79]
3.6	Synaptic memory consolidation in networks of morphological neurons	[118]	- (novel results)
3.7	Network models from 3.5 and 3.6	[69]; [118]	[60]; [79]
3.8	Busyring network benchmark	[121]	[122]

https://doi.org/10.1371/journal.pcbi.1013926.t010

with the state of commit 65a4a3a, which shall be merged upon publication of this article. Note that while all of the mentioned Arbor versions are by now slightly outdated, we ensured that our extensive software tests from [69,118] passed with the used versions including the release version 0.10.0, which strongly indicates that the simulation results will remain the same.

The scripts that we used to perform the comparisons across simulators in subsection 3.5 can be found here: https://github.com/jlubo/simulatorcomparison.

Supporting information

List of Legends of S1 Appendix

Fig A: More details on classical spike-timing dependent plasticity (STDP) and spike-driven homeostasis. Related to Fig 2 in the main article. Arbor implementations (in lighter blue) are cross-validated by comparison to Brian 2 (in orange) or theory. In (a–d), two Poisson spike sources stimulate an inhibitory and an excitatory synapse connecting to a single neuron (spikes are shown in red and blue, respectively). The excitatory connection undergoes STDP. **(a)** Membrane potential of the neuron (goodness of fit between the curves: CV = 0.923, RMSE = 1.265mV). **(b)** Conductance of the excitatory synapse (CV = 0.996, RMSE = 0.486μS). **(c)** Conductance of the inhibitory synapse (CV = 0.997, RMSE = 0.148μS). **(d)** Spike time mismatch measured by $\left(t^{\text{Arbor}} - t^{\text{Brian}}\right)/t^{\text{Brian}} \cdot 100$, where t^{Arbor} and t^{Brian} are the postsynaptic spike times obtained from the Arbor and Brian 2 simulations, respectively. The result indicates that the difference in spike timing due to different implementations of the Poisson process is below 0.1%. **(e)** Classical STDP curve, compared with theoretical expectation $A_{\text{pre}} \cdot \exp(-\Delta t/\tau_{\text{pre}})$ for $\Delta t > 0$ and $A_{\text{post}} \cdot \exp(\Delta t/\tau_{\text{post}})$ otherwise (CV > 0.999, RMSE = 0.001μS). **(f)** As opposed to Fig 2I in the main article, the resulting firing rate of the neuron in the absence of homeostatic plasticity is shown (CV = 0.972, RMSE = 1.674Hz).

Fig B: Basic early- and late-phase plasticity dynamics with synaptic tagging and capture (STC), cross-validated with stand-alone simulator. Also note the the cross-validation with the stand-alone simulator in the main article (Fig 4) and with Brian 2 in Fig C. **(a)** Averaged noisy early-phase synaptic weight (see Eq 17) in the main article). Stimulating spikes reach the synapse at pre-defined times (indicated by bold gray arrows). Goodness of fit between the mean curves: CV = 0.999, RMSE = 0.040mV. **(b)** Limit cases of early- and late-phase synaptic weight (see Eq 17 and Eq 20 in the main article). The presynaptic neuron is stimulated with a strong current to spike at maximal rate (duration of the stimulation indicated by gray bar). The late-phase weight has been shifted for graphical reasons (also cf. Eq 20; early phase: CV = 0.201, RMSE = 0.221mV; late phase: CV > 0.999, RMSE = 0.055mV). **(c)** Standard deviation of the noisy early-phase synaptic weight (RMSE = 0.067mV), and**(d)** standard deviation of early- and late-phase synaptic weight (early phase: RMSE = 0.133mV; late phase: RMSE = 0.004mV), demonstrating the matching of the stochastic properties of the two solvers.**(e)** Postsynaptic calcium concentration, which successively crosses the thresholds for depression (LTD) and potentiation (LTP) (cf. Eq Eq 19 in the main article; CV > 0.999, RMSE = 0.065). **(f)** The postsynaptic PRP concentration rises until it reaches its maximum due to the continued stimulation (cf. Eq 21 in the main article; CV = 0.998, RMSE = 0.107μmol/l). **(g)** Membrane potential of the postsynaptic neuron (CV > 0.999, RMSE = 0.151mV). Continuous lines (specified in the legends) represent the dynamics simulated in Arbor. Darker, dashed lines represent the results of the stand-alone simulator [60]. Fine, dotted lines represent the baseline level of the respective quantity. Basic early-phase plasticity dynamics (a,c,e,g): average across 10 batches, each consisting of 100 trials. Basic late-phase plasticity dynamics (b,d,f): average across 10 batches, each consisting of 10 trials. The noise seeds were drawn independently for each trial. Error bands represent the standard error of the mean (often too small to be visible).

Fig C: Basic early- and late-phase plasticity dynamics with synaptic tagging and capture (STC), cross-validated with the Brian 2 simulator. Also see the the cross-validation with the stand-alone simulator in Fig B and in the main article (Fig 4). **(a)** Averaged noisy early-phase synaptic weight (see Eq 17) in the main article). Stimulating spikes reach the

synapse at pre-defined times (indicated by bold gray arrows). **(b)** Limit cases of early- and late-phase synaptic weight (see Eq 17 and Eq 20 in the main article). The presynaptic neuron is stimulated with a strong current to spike at maximal rate (duration of the stimulation indicated by gray bar). The late-phase weight has been shifted for graphical reasons (also cf. Eq 20). **(c)** Standard deviation of the noisy early-phase synaptic weight, and **(d)** standard deviation of early- and late-phase synaptic weight, demonstrating the matching of the stochastic properties of the two solvers. **(e)** Postsynaptic calcium concentration, which successively crosses the thresholds for depression (LTD) and potentiation (LTP) (cf. Eq Eq 19 in the main article). **(f)** The postsynaptic PRP concentration rises until it reaches its maximum due to the continued stimulation (cf. Eq 21 in the main article). **(g)** Membrane potential of the postsynaptic neuron. Continuous lines (specified in the legends) represent the dynamics simulated in Arbor. Darker, dashed lines represent the results of [67], using Brian 2 with `cpp_standalone` device [4]. Fine, dotted lines represent the baseline level of the respective quantity. Basic early-phase plasticity dynamics (a,c,e,g): average across 10 batches, each consisting of 100 trials. Basic late-phase plasticity dynamics (b,d,f): average across 10 batches, each consisting of 10 trials. The noise seeds were drawn independently for each trial. Error bands represent the standard error of the mean (often too small to be visible).

Fig D: Elementary motifs of spike transmission in a pre-defined sparsely connected network, cross-validated with stand-alone simulator. Continuous lines (specified in the legends) represent the dynamics simulated in Arbor. Darker dashed lines represent the results of the stand-alone simulator [60]. Stochastic variables in the model have been replaced by deterministic mean dynamics ($\sigma_{\text{pl}} = 0$). The connectivity matrix `connections_default.txt` from [60] is used. **(a)** Response of neuron 68 to a spike in neuron 6 (excitatory→excitatory); **(b)** response of neuron 1760 to a spike in neuron 6 (excitatory→inhibitory); **(c)** response of neuron 17 to a spike in neuron 1615 (inhibitory→excitatory); **(d)** response of neuron 1690 to a spike in neuron 1615 (inhibitory→inhibitory).

Fig E: UML sequence diagram for the single-compartment synaptic tagging and capture model. The diagram describes the technical specification that is necessary to implement the model from [23]. Essentially, the spiking neuron model produces pre- and postsynaptic spikes that give rise to calcium-based early-phase plasticity, which then elicits synaptic tagging, synthesis of plasticity-related products (PRPs), and late-phase plasticity.

Fig F: Impact of classical stimulation protocols at a single synapse. Different types of long-term synaptic plasticity are induced depending on the stimulation protocol [59,64]: **(a)** late-phase potentiation through strong tetanic (STET) stimulation, **(b)** early-phase potentiation through weak tetanic (WTET) stimulation, **(c)** late-phase depression through strong low-frequency (SLFS) stimulation, **(d)** early-phase depression through weak low-frequency (WLFS) stimulation. See [23] for details of the protocols. All protocols affect the early-phase weight (dark red lines) and lead to the crossing of the tag threshold (dotted red lines), whereas only the 'strong' protocols lead to the crossing of the PRP synthesis threshold (dashed green lines), thereby enabling changes in late-phase weight (blue lines). The total synaptic weight (orange lines) is the sum of early- and late-phase weight. These results were obtained with Arbor using the code from [68] (which has the same basis as our network code [69,118]). In addition, for comparison, the total synaptic weight obtained from point-neuron simulations with the stand-alone simulator [60] is shown in grated dark shading (overlapping with the total weight from Arbor simulations). Average over 100 trials; sampling rate: 30/min. Compartment measures in Arbor as detailed in Table 5 in the main article. Error bands represent the relative standard deviation of early-phase, late-phase, and total synaptic weight. The late-phase weight has further been shifted for graphical reasons (cf. Eq 20 in the main article).

Fig G: Formal characteristics of the diffusion along a dendritic cable in Arbor. (a) A single compartment and its equivalent circuit in the cable model. **(b)** Important quantities of the diffusion mechanisms.

Fig H: Diffusion of arbitrary particles along a dendrite in Arbor at different times (maximum diffusivity). Example for paradigm of large dendrites, small cells, and maximum diffusivity $D_{\text{p}} = 10^{-11}\text{m}^2/\text{s}$ (cf. Table 6 in the main article).

(a,b) Plasticity-related product or protein (PRP) concentration in the center of the soma, and summed amount of the signal triggering PRP synthesis (SPS) across the whole cell. Furthermore, the amount of SPS in the whole cell is estimated from the concentration of SPS in the center of the soma, which is the actual driver of PRP synthesis (PS) in the model. As long as this quantity is above the PS threshold, PS happens (cf. Eqs 21 & 23 in the main article). **(c,d)** PRP concentration at the basal end of the soma, and summed amount of PRP across the whole cell (under ongoing PS, converges towards $p_{\max}$ times the total volume of the cell). **(e,f)** PRP concentration along apical dendrite. **(g,h)** PRP concentration along basal dendrite.

Fig I: Diffusion of arbitrary particles along a dendrite in Arbor at different times (moderate diffusivity). Example for paradigm of large dendrites, small cells, and moderate diffusivity $D_p = 10^{-15}\mathrm{m}^2/\mathrm{s}$ (cf. Table 6 in the main article). **(a,b)** Plasticity-related product or protein (PRP) concentration in the center of the soma, and summed amount of the signal triggering PRP synthesis (SPS) across the whole cell. Furthermore, the amount of SPS in the whole cell is estimated from the concentration of SPS in the center of the soma, which is the actual driver of PRP synthesis (PS) in the model. As long as this quantity is above the PS threshold, PS happens (cf. Eqs 21 & 23 in the main article). **(c,d)** PRP concentration at the basal end of the soma, and summed amount of PRP across the whole cell (under ongoing PS, converges towards $p_{\max}$ times the total volume of the cell). **(e,f)** PRP concentration along apical dendrite. **(g,h)** PRP concentration along basal dendrite.

Fig J: Memory recall in recurrent networks of single-compartment neurons and point neurons, after learning and after consolidation. Plots show the pattern completion, measured by coefficient Q (see Eq 22 in the main article) for a stimulated subset of varied size (a varied number of neurons are stimulated for learning/recall). Values at 10s are shown in red and values at 8h in blue (after learning and after consolidation, respectively). Value $Q > 0$ indicates the successful recall of a memory representation. Results in **(a)** obtained with Arbor, in **(b)** from the custom stand-alone simulator [60], and in **(c)** from Brian 2 with `cpp_standalone` device [79] (no data for 8h due to missing fast-forward computation implementation). Data averaged over 100 network realizations (unlike in [23]). Error bars represent the 95% confidence interval.

Fig K: Memory recall in a recurrent neural network after learning and after consolidation. Analogous to Fig 5 in the main article, a measure of mutual information is considered here. Results obtained with Arbor for networks of different kinds of multi-compartment neurons, demonstrating the impact of different values of the PRP diffusivity D_p on memory consolidation. Networks consist of 'small' cells (radius of 6μm) or of 'large' cells (radius of 12μm), with either small or large dendrites (in which cases each neuron comprises in total 31 or 48 compartments, respectively). The radius and length values are given in Table 6 in the main article. **(a,b)** Illustrations of used cell structures, generated using `Arbor GUI` [55]. Each segment is represented by a different color. A segment can consist of a multitude of compartments. Overlaid with illustrations of more realistic neuron structures that would have roughly similar functional properties. **(a)** Small (left) and large (right) cell with small dendrites, **(b)** the same with large dendrites. **(c-f)** Memory recall measured by the mutual information between the distribution of neuronal firing rates during learning and during recall stimulation (see [23]), for a stimulated subset of varied size (a varied number of neurons are stimulated for learning/recall). Average over 100 network realizations. Error bars represent the 95% confidence interval. **(c)** Recall stimulation at 10s after learning (technically, $D_p = 10^{-11}\mathrm{m}^2/\mathrm{s}$, but late-phase plasticity does not occur at such fast timescales). **(d)** Recall stimulation at 8h after learning, $D_p = 10^{-11}\mathrm{m}^2/\mathrm{s}$. **(e)** Recall stimulation at 8h after learning, $D_p = 10^{-15}\mathrm{m}^2/\mathrm{s}$. **(f)** Recall stimulation at 8h after learning, $D_p = 10^{-19}\mathrm{m}^2/\mathrm{s}$. **(g-j)** Same as (c-f), but for large cells that consist of segments of twice the diameter.

Fig L: Benchmarking results of total runtime per spike, for 10s-recall paradigm in networks of 2000 neurons, related to Arbor simulations in Fig 6 in the main article. **(a)** Total runtime (including setup and propagation phase) per spike, accounting for the fact that the spike numbers in the single-compartment and multi-compartment simulations are different (cf. panel (b)). The data points show the runtime as given in Fig 6 in the main article, divided by the total number of

spikes. **(b)** Total number of spikes in the different simulation paradigms. Data points represent the average over 10 trials; error bars represent the standard deviation.

Fig M: Benchmarking results of total runtime and memory use for 8h-recall paradigm. Networks of 2000 neurons (also see Fig 6 in the main article) in Arbor. The single-compartment simulations ('SC') are conducted as described in subection 3.5 in the main article. The simulations with multi-compartment/morphological neurons ('MC') are conducted as described in subsection 3.6. Results are given for execution with and without SIMD support on the HWS1 system (Intel Core i5-6600 CPU @ 3.30GHz, 1 × 8GB DDR3-RAM, using 1 thread). **(a)** Total runtime of the simulations, including setup and propagation phase. Measurements were performed using `hyperfine` in version 1.15. **(b)** Memory consumption, given by the maximum over time of the number of 'dirty' bytes, including private and shared memory, as returned from the `pmap` tool. Data points represent the average over 10 trials; error bars represent the standard deviation.

Fig N: Benchmarking of total memory use for large-scale networks. Total memory use, in megabytes, to initialize and execute a simulation with 32768 cells over 200ms in Arbor and CoreNEURON. A busyring network of `simple-branchy` cells with tree depth 2 is used, run on the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM) with **(a)** CoreNEURON, **(b)** Arbor with SIMD, **(c)** Arbor with SIMD with additional STDP mechanisms for the random synapses. The respectively lowest memory use for each implementation is highlighted by the gray box. **(d)** Scaling of the best results over network size. **(e)** Sketch of the busyring network consisting of rings of integrate-and-fire neurons (shown as blue disks), connected internally via excitatory synapses, and across the whole network via random synapses of weight zero. One neuron of each ring receives external stimulation. All values are averaged over 10 trials, with coefficient of variation CV < 0.004 in all cases. See Table A for more details.

Fig O: Strong scaling on the JUWELS and JEDI supercomputing systems. Benchmarking results for the total wall-clock time, including setup and state propagation phases, of a simulation with 10^6 `complex` cells over 200ms. We show strong scaling, i.e., the cell count is subdivided across the available GPUs. For reference, the intervals of 80% efficiency are shown, where 100% efficiency represents perfect scaling with the additional hardware. The data was obtained during the pre-production phase of the JUPITER supercomputing system on the JEDI preview system. Parameter values are provided in Table 7 in the main article; further parameters and hardware characteristics are provided in Table C.

Table A: Total memory use for busyring benchmark. Measurements are provided for networks of `simple-branchy` cells as reported by Arbor and CoreNEURON (using the lowest-memory paradigm analogously to the runtime optimization detailed in Fig 7B,C in the main article). Results are collected with the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM, GPU not used). All values are averaged over 10 trials, with coefficient of variation CV < 0.004 in all cases. Arbor with SIMD. In CoreNEURON, the most extensive simulation did not finish (d.n.f.) due to exceeded memory. For the corresponding runtime results, cf. Table 8 in the main article.

Table B: Wallclock time measurements for busyring benchmark with different cell types. Total runtime results are provided as reported by Arbor (using the lowest-memory paradigm as detailed in Fig N). All networks consist of 1024 neurons and 1.03 million synapses. The shares of the setup and state propagation phases are given in brackets, respectively. Results are collected with the HWS2 system (AMD Ryzen Threadripper PRO 5995WX CPU with 64 cores, 8 × 32GB DDR4-RAM, GPU not used). All values are averaged over 10 trials, with coefficient of variation CV < 0.06 in all cases. Arbor with SIMD. Also see Tables 8 and 9 in the main article for other paradigms.

Table C: Parameters and hardware characteristics for investigating strong scaling on supercomputing systems. We use one MPI rank per GPU and the full set of 4 GPUs per node when present in the node architecture. See Table 7 in the main article for further parameter values.
(PDF)

Acknowledgments

We would like to thank the other Arbor developers for their support and for their dedication to continuously improve the software. We would further like to thank Silvio Rizzoli, Arash Golmohammadi, and Marcel Stimberg for helpful discussions.

Author contributions

Conceptualization: Jannik Luboeinski, Sebastian Schmitt, Christian Tetzlaff.

Data curation: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee.

Formal analysis: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee.

Funding acquisition: Sebastian Schmitt, Christian Tetzlaff.

Investigation: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Christian Tetzlaff.

Methodology: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Thorsten Hater, Christian Tetzlaff.

Project administration: Christian Tetzlaff.

Resources: Thorsten Hater, Christian Tetzlaff.

Software: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Thorsten Hater, Fabian Bösch, Christian Tetzlaff.

Supervision: Jannik Luboeinski, Sebastian Schmitt, Christian Tetzlaff.

Validation: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Thorsten Hater.

Visualization: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Thorsten Hater.

Writing – original draft: Jannik Luboeinski, Sebastian Schmitt, Shirin Shafiee, Thorsten Hater, Fabian Bösch.

Writing – review & editing: Jannik Luboeinski, Sebastian Schmitt, Christian Tetzlaff.

References

1. Hines M. A program for simulation of nerve equations with branching geometries. Int J Biomed Comput. 1989;24(1):55–68. https://doi.org/10.1016/0020-7101(89)90007-x PMID: 2714879
2. Kumbhar P, Hines M, Fouriaux J, Ovcharenko A, King J, Delalondre F, et al. CoreNEURON: an optimized compute engine for the NEURON simulator. Front Neuroinform. 2019;13:63. https://doi.org/10.3389/fninf.2019.00063 PMID: 31616273
3. Gewaltig MO, Diesmann M. NEST (NEural Simulation Tool). Scholarpedia. 2007;2:1430.
4. Stimberg M, Brette R, Goodman DF. Brian 2, an intuitive and efficient neural simulator. Elife. 2019;8:e47314. https://doi.org/10.7554/eLife.47314 PMID: 31429824
5. Awile O, Kumbhar P, Cornu N, Dura-Bernal S, King JG, Lupton O, et al. Modernizing the NEURON simulator for sustainability, portability, and performance. Front Neuroinform. 2022;16:884046. https://doi.org/10.3389/fninf.2022.884046 PMID: 35832575
6. Akar NA, Cumming B, Karakasis V, Kusters A, Klijn W, Peyser A, et al. Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2019. p. 274–82. https://doi.org/10.1109/empdp.2019.8671560
7. Cumming et al. Arbor library v0.10.0. 2024. https://doi.org/10.5281/zenodo.13284959
8. Martin SJ, Grimwood PD, Morris RG. Synaptic plasticity and memory: an evaluation of the hypothesis. Annu Rev Neurosci. 2000;23:649–711. https://doi.org/10.1146/annurev.neuro.23.1.649 PMID: 10845078
9. Abraham WC, Jones OD, Glanzman DL. Is plasticity of synapses the mechanism of long-term memory storage?. NPJ Sci Learn. 2019;4:9. https://doi.org/10.1038/s41539-019-0048-y PMID: 31285847
10. Smolen P, Baxter DA, Byrne JH. Molecular constraints on synaptic tagging and maintenance of long-term potentiation: a predictive model. PLoS Comput Biol. 2012;8(8):e1002620. https://doi.org/10.1371/journal.pcbi.1002620 PMID: 22876169

11. Gallimore AR, Kim T, Tanaka-Yamamoto K, De Schutter E. Switching on depression and potentiation in the cerebellum. Cell Rep. 2018;22(3):722–33. https://doi.org/10.1016/j.celrep.2017.12.084 PMID: 29346769
12. Mäki-Marttunen T, Iannella N, Edwards AG, Einevoll GT, Blackwell KT. A unified computational model for cortical post-synaptic plasticity. eLife. 2020;9:e55714. https://doi.org/10.7554/eLife.55714 PMID: 32729828
13. Becker MFP, Tetzlaff C. The biophysical basis underlying the maintenance of early phase long-term potentiation. PLoS Comput Biol. 2021;17(3):e1008813. https://doi.org/10.1371/journal.pcbi.1008813 PMID: 33750943
14. Bonilla-Quintana M, Wörgötter F. Exploring new roles for actin upon LTP induction in dendritic spines. Sci Rep. 2021;11(1):7072. https://doi.org/10.1038/s41598-021-86367-z PMID: 33782451
15. Berridge MJ. Dysregulation of neural calcium signaling in Alzheimer disease, bipolar disorder and schizophrenia. Prion. 2013;7(1):2–13. https://doi.org/10.4161/pri.21767 PMID: 22895098
16. Bonilla-Quintana M, Rangamani P. Can biophysical models of dendritic spines be used to explore synaptic changes associated with addiction?. Phys Biol. 2022;19(4):10.1088/1478-3975/ac6cbe. https://doi.org/10.1088/1478-3975/ac6cbe PMID: 35508164
17. Acharya J, Basu A, Legenstein R, Limbacher T, Poirazi P, Wu X. Dendritic computing: branching deeper into machine learning. Neuroscience. 2022;489:275–89. https://doi.org/10.1016/j.neuroscience.2021.10.001 PMID: 34656706
18. Pagkalos M, Makarov R, Poirazi P. Leveraging dendritic properties to advance machine learning and neuro-inspired computing. Curr Opin Neurobiol. 2024;85:102853. https://doi.org/10.1016/j.conb.2024.102853 PMID: 38394956
19. Zheng H, Zheng Z, Hu R, Xiao B, Wu Y, Yu F, et al. Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. Nat Commun. 2024;15(1):277. https://doi.org/10.1038/s41467-023-44614-z PMID: 38177124
20. Ibáñez M, et al. Scalable spike transmission in large-scale brain network simulations. In: Proceedings of 2026 International Parallel & Distributed Processing Symposium (IDPDS).
21. Graupner M, Brunel N. Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. Proc Natl Acad Sci U S A. 2012;109(10):3991–6. https://doi.org/10.1073/pnas.1109359109 PMID: 22357758
22. Hiratani N, Fukai T. Detailed dendritic excitatory/inhibitory balance through heterosynaptic spike-timing-dependent plasticity. J Neurosci. 2017;37(50):12106–22. https://doi.org/10.1523/JNEUROSCI.0027-17.2017 PMID: 29089443
23. Luboeinski J, Tetzlaff C. Memory consolidation and improvement by synaptic tagging and capture in recurrent neural networks. Commun Biol. 2021;4(1):275. https://doi.org/10.1038/s42003-021-01778-y PMID: 33658641
24. Bi GQ, Poo MM. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. J Neurosci. 1998;18(24):10464–72. https://doi.org/10.1523/JNEUROSCI.18-24-10464.1998 PMID: 9852584
25. Song S, Miller KD, Abbott LF. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nat Neurosci. 2000;3(9):919–26. https://doi.org/10.1038/78829 PMID: 10966623
26. Arbor documentation – selection policy. 2023. Accessed 2024 November 8. https://docs.arbor-sim.org/en/latest/python/cell.html#arbor.selection_policy
27. Kloeden PE, Platen E. Numerical solution of stochastic differential equations. Berlin, Heidelberg: Springer; 1992.
28. Salmon JK, Moraes MA, Dror RO, Shaw DE. Parallel random numbers: as easy as 1, 2, 3. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, 2011.
29. Salmon JK, Morales MA, Fenn M. Random123: a library of counter-based random number generators. 2010. Accessed 2024 November 23. https://github.com/DEShawResearch/random123
30. Rall W. Theory of physiological properties of dendrites. Ann N Y Acad Sci. 1962;96:1071–92. https://doi.org/10.1111/j.1749-6632.1962.tb54120.x PMID: 14490041
31. Yates S. Analytic solutions to the cable equation. 2019. Accessed 2024 November 23. https://github.com/arbor-sim/arbor/blob/07a9fa2ff3d4ed9173ecfb316a056a723e081de3/doc/math/cable_equation/cable_equation.tex
32. Sætra MJ, Einevoll GT, Halnes G. An electrodiffusive, ion conserving Pinsky-Rinzel model with homeostatic mechanisms. PLoS Comput Biol. 2020;16(4):e1007661. https://doi.org/10.1371/journal.pcbi.1007661 PMID: 32348299
33. Gerstner W, Kempter R, van Hemmen JL, Wagner H. A neuronal learning rule for sub-millisecond temporal coding. Nature. 1996;383(6595):76–81. https://doi.org/10.1038/383076a0 PMID: 8779718
34. Magee JC, Johnston D. A synaptically controlled, associative signal for Hebbian plasticity in hippocampal neurons. Science. 1997;275(5297):209–13. https://doi.org/10.1126/science.275.5297.209 PMID: 8985013
35. Markram H, Lübke J, Frotscher M, Sakmann B. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. Science. 1997;275(5297):213–5. https://doi.org/10.1126/science.275.5297.213 PMID: 8985014
36. Morrison A, Diesmann M, Gerstner W. Phenomenological models of synaptic plasticity based on spike timing. Biol Cybern. 2008;98(6):459–78. https://doi.org/10.1007/s00422-008-0233-1 PMID: 18491160
37. Introduction to Brian part 2: synapses. 2021. Accessed 2024 June 14. https://brian2.readthedocs.io/en/stable/resources/tutorials/2-intro-to-brian-synapses.html
38. Arbor Tutorial: Spike Timing-dependent Plasticity Curve. 2023. Accessed 2024 June 14. https://docs.arbor-sim.org/en/stable/tutorial/calcium_stdp_curve.html

39. Zenke F, Gerstner W. Hebbian plasticity requires compensatory processes on multiple timescales. Philos Trans R Soc Lond B Biol Sci. 2017;372(1715):20160259. https://doi.org/10.1098/rstb.2016.0259 PMID: 28093557
40. Breitwieser O. Towards a neuromorphic implementation of spike-based expectation maximization. Universität Heidelberg. 2015. http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3240
41. Brian 2 example: Spike based homeostasis. 2021. Accessed 2024 June 14. https://brian2.readthedocs.io/en/latest/examples/synapses.spike_based_homeostasis.html
42. Sjöström PJ, Turrigiano GG, Nelson SB. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. Neuron. 2001;32:1149–64. https://doi.org/10.1016/S0896-6273(01)00542-6 PMID: 11754844
43. Gillespie D. Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Topics. 1996;54(2):2084–91. https://doi.org/10.1103/physreve.54.2084 PMID: 9965289
44. Lee KFH, Soares C, Thivierge J-P, Béïque J-C. Correlated synaptic inputs drive dendritic calcium amplification and cooperative plasticity during clustered synapse development. Neuron. 2016;89(4):784–99. https://doi.org/10.1016/j.neuron.2016.01.012 PMID: 26853305
45. Stuart G, Spruston N, Häusser M. Dendrites. Oxford University Press; 2016. https://doi.org/10.1093/acprof:oso/9780198745273.001.0001
46. Shafiee S, Schmitt S, Tetzlaff C. Calcium-based input timing learning. bioRxiv. 2024. https://doi.org/10.1101/2024.11.14.623617
47. Shafiee S. Arbor simulation of calcium-dependent heterosynaptic plasticity. 2024. Accessed 2024 November 23. https://github.com/Shirin1993/Arbor_diffusion
48. Araya R. Input transformation by dendritic spines of pyramidal neurons. Front Neuroanat. 2014;8:141. https://doi.org/10.3389/fnana.2014.00141 PMID: 25520626
49. Yasuda R. Biophysics of biochemical signaling in dendritic spines: implications in synaptic plasticity. 2017. https://doi.org/10.1016/j.bpj.2017.07.029 PMID: 28866426
50. Häusser M, Roth A. Estimating the time course of the excitatory synaptic conductance in neocortical pyramidal cells using a novel voltage jump method. J Neurosci. 1997;17(20):7606–25. https://doi.org/10.1523/JNEUROSCI.17-20-07606.1997 PMID: 9315883
51. Gerstner W, Kistler WM, Naud R, Paninski L. Neuronal dynamics: from single neurons to networks and models of cognition. Cambridge University Press; 2014.
52. Biess A, Korkotian E, Holcman D. Barriers to diffusion in dendrites and estimation of calcium spread following synaptic inputs. PLoS Comput Biol. 2011;7(10):e1002182. https://doi.org/10.1371/journal.pcbi.1002182 PMID: 22022241
53. Allbritton NL, Meyer T, Stryer L. Range of messenger action of calcium ion and inositol 1,4,5-trisphosphate. Science. 1992;258(5089):1812–5. https://doi.org/10.1126/science.1465619 PMID: 1465619
54. Means S, Smith AJ, Shepherd J, Shadid J, Fowler J, Wojcikiewicz RJH, et al. Reaction diffusion modeling of calcium dynamics with realistic ER geometry. Biophys J. 2006;91(2):537–57. https://doi.org/10.1529/biophysj.105.075036 PMID: 16617072
55. Hater T. Arbor GUI v0.8. 2022. https://doi.org/10.5281/zenodo.7415130
56. Schuss Z, Singer A, Holcman D. The narrow escape problem for diffusion in cellular microdomains. Proc Natl Acad Sci U S A. 2007;104(41):16098–103. https://doi.org/10.1073/pnas.0706599104 PMID: 17901203
57. Frey U, Morris RG. Synaptic tagging and long-term potentiation. Nature. 1997;385(6616):533–6. https://doi.org/10.1038/385533a0 PMID: 9020359
58. Redondo RL, Morris RGM. Making memories last: the synaptic tagging and capture hypothesis. Nat Rev Neurosci. 2011;12(1):17–30. https://doi.org/10.1038/nrn2963 PMID: 21170072
59. Sajikumar S, Navakkode S, Sacktor TC, Frey JU. Synaptic tagging and cross-tagging: the role of protein kinase Mzeta in maintaining long-term potentiation but not long-term depression. J Neurosci. 2005;25(24):5750–6. https://doi.org/10.1523/JNEUROSCI.1104-05.2005 PMID: 15958741
60. Luboeinski J, Lehr A. Simulation code and analysis scripts for memory formation and consolidation with synaptic tagging and capture in recurrent spiking neural networks. 2024. https://doi.org/10.5281/zenodo.4429195
61. Luboeinski J, Tetzlaff C. Organization and priming of long-term memory representations with two-phase plasticity. Cogn Comput. 2022;15(4):1211–30. https://doi.org/10.1007/s12559-022-10021-7
62. Luboeinski J. The Role of Synaptic Tagging and Capture for Memory Dynamics in Spiking Neural Networks. University of Göttingen; 2021. http://dx.doi.org/10.53846/goediss-463
63. Lehr AB, Luboeinski J, Tetzlaff C. Neuromodulator-dependent synaptic tagging and capture retroactively controls neural coding in spiking neural networks. Sci Rep. 2022;12(1):17772. https://doi.org/10.1038/s41598-022-22430-7 PMID: 36273097
64. Li Y, Kulvicius T, Tetzlaff C. Induction and consolidation of calcium-based homo- and heterosynaptic potentiation and depression. PLoS One. 2016;11(8):e0161679. https://doi.org/10.1371/journal.pone.0161679 PMID: 27560350
65. Luboeinski J, Tetzlaff C. Modeling emergent dynamics arising from synaptic tagging and capture at the network level. In: Sajikumar S, Abel T, editors. Synaptic tagging and capture: from synapses to behavior. 2nd ed. Cham, Switzerland: Springer; 2024. p. 471–503. https://doi.org/10.1007/978-3-031-54864-2_23
66. Clopath C, Ziegler L, Vasilaki E, Büsing L, Gerstner W. Tag-trigger-consolidation: a model of early and late long-term-potentiation and depression. PLoS Comput Biol. 2008;4(12):e1000248. https://doi.org/10.1371/journal.pcbi.1000248 PMID: 19112486

67. Luboeinski J. Brian 2 simulation of the induction of early- and late-phase plasticity at a single synapse. 2023. Accessed 2024 November 23. https://github.com/jlubo/brian_synaptic_plasticity_stc
68. Luboeinski J. Arbor simulation of the induction of early- and late-phase plasticity at a single synapse. 2023. Accessed 2024 November 23. https://github.com/jlubo/arbor_2N1S
69. Luboeinski J. Arbor simulation of memory formation and consolidation in recurrent spiking neural networks with synaptic tagging and capture. 2024. Accessed 2024 December 17. https://github.com/jlubo/arbor_network_consolidation
70. Higgins D, Graupner M, Brunel N. Memory maintenance in synapses with calcium-based plasticity in the presence of background activity. PLoS Comput Biol. 2014;10(10):e1003834. https://doi.org/10.1371/journal.pcbi.1003834 PMID: 25275319
71. Wittenberg GM, Wang SS-H. Malleability of spike-timing-dependent plasticity at the CA3-CA1 synapse. J Neurosci. 2006;26(24):6610–7. https://doi.org/10.1523/JNEUROSCI.5388-05.2006 PMID: 16775149
72. Jiang S, Guan Y, Chen S, Jia X, Ni H, Zhang Y, et al. Anatomically revealed morphological patterns of pyramidal neurons in layer 5 of the motor cortex. Sci Rep. 2020;10(1):7916. https://doi.org/10.1038/s41598-020-64665-2 PMID: 32405029
73. Carnevale NT, Hines ML. The NEURON book. Cambridge University Press; 2006.
74. Larkman AU. Dendritic morphology of pyramidal neurones of the visual cortex of the rat: III. Spine distributions. J Comp Neurol. 1991;306(2):332–43. https://doi.org/10.1002/cne.903060209 PMID: 1711059
75. Gillon CJ, et al. Responses of pyramidal cell somata and apical dendrites in mouse visual cortex over multiple days. Scientific Data 2023;10:287.
76. van Aerde KI, Feldmeyer D. Morphological and physiological characterization of pyramidal neuron subtypes in rat medial prefrontal cortex. Cereb Cortex. 2015;25(3):788–805. https://doi.org/10.1093/cercor/bht278 PMID: 24108807
77. Džaja D, Hladnik A, Bičanić I, Baković M, Petanjek Z. Neocortical calretinin neurons in primates: increase in proportion and microcircuitry structure. Front Neuroanat. 2014;8:103. https://doi.org/10.3389/fnana.2014.00103 PMID: 25309344
78. Carriba P, Davies AM. CD40 is a major regulator of dendrite growth from developing excitatory and inhibitory neurons. Elife. 2017;6:e30442. https://doi.org/10.7554/eLife.30442 PMID: 29111976
79. Luboeinski J. Brian 2 simulation of memory formation and consolidation in recurrent spiking neural networks based on synaptic tagging and capture. 2024. Accessed 2024 November 23. https://github.com/jlubo/brian_network_plasticity
80. Herten A, et al. Application-driven exascale: The JUPITER benchmark suite. In: Proceedings of The International Conference for High Performance Computing Networking, Storage, and Analysis (SC '24), Atlanta, GA (USA). 2024. https://juser.fz-juelich.de/record/1033607.
81. Hodgkin AL, Huxley AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol. 1952;117(4):500–44. https://doi.org/10.1113/jphysiol.1952.sp004764 PMID: 12991237
82. Allen Institute. Accessed 2025 June 6. https://celltypes.brain-map.org/experiment/electrophysiology/643625553
83. NEURON documentation – CoreNEURON compatibility. 2022. Accessed 2023 November 8. https://nrn.readthedocs.io/en/8.2.2/coreneuron/compatibility.html
84. Bower JM, Beeman D. The book of Genesis: exploring realistic neural models with the General Neural Simulation System. internet ed. 2003. http://genesis-sim.org/GENESIS/iBoG/index.html.internet.edn.
85. Ray S, Bhalla US. PyMOOSE: interoperable scripting in python for MOOSE. Front Neuroinform. 2008;2:6. https://doi.org/10.3389/neuro.11.006.2008 PMID: 19129924
86. Pagkalos M, Chavlis S, Poirazi P. Introducing the Dendrify framework for incorporating dendrites to spiking neural networks. Nat Commun. 2023;14(1):131. https://doi.org/10.1038/s41467-022-35747-8 PMID: 36627284
87. Niedermeier L, Chen K, Xing J, Das A, Kopsick J, Scott E, et al. CARLsim 6: an open source library for large-scale, biologically detailed spiking neural network simulation. In: 2022 International Joint Conference on Neural Networks (IJCNN). 2022. p. 1–10. https://doi.org/10.1109/ijcnn55064.2022.9892644
88. Panagiotou S, Sidiropoulos H, Soudris D, Negrello M, Strydis C. EDEN: a high-performance, general-purpose, NeuroML-based neural simulator. Front Neuroinform. 2022;16:724336. https://doi.org/10.3389/fninf.2022.724336 PMID: 35669596
89. Golosio B, Tiddia G, De Luca C, Pastorelli E, Simula F, Paolucci PS. Fast simulations of highly-connected spiking cortical models using GPUs. Front Comput Neurosci. 2021;15:627620. https://doi.org/10.3389/fncom.2021.627620 PMID: 33679358
90. Golosio B, Villamar J, Tiddia G, Pastorelli E, Stapmanns J, Fanti V, et al. Runtime construction of large-scale spiking neuronal network models on GPU devices. Applied Sciences. 2023;13(17):9598. https://doi.org/10.3390/app13179598
91. Knight JC, Nowotny T. Larger GPU-accelerated brain simulations with procedural connectivity. Nat Comput Sci. 2021;1(2):136–42. https://doi.org/10.1038/s43588-020-00022-7 PMID: 38217218
92. Tikidji-Hamburyan RA, Narayana V, Bozkus Z, El-Ghazawi TA. Software for brain network simulations: a comparative study. Front Neuroinform. 2017;11:46. https://doi.org/10.3389/fninf.2017.00046 PMID: 28775687
93. Kulkarni SR, Parsa M, Mitchell JP, Schuman CD. Benchmarking the performance of neuromorphic and spiking neural network simulators. Neurocomputing. 2021;447:145–60. https://doi.org/10.1016/j.neucom.2021.03.028
94. Wang C, Zhang T, Chen X, He S, Li S, Wu S. BrainPy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming. Elife. 2023;12:e86365. https://doi.org/10.7554/eLife.86365 PMID: 38132087

95. Plesser HE. Commentary: Accelerating spiking neural network simulations with PymoNNto and PymoNNtorch. Front Neuroinform. 2024;18:1446620. https://doi.org/10.3389/fninf.2024.1446620 PMID: 39507425
96. Kobayashi T, Kuriyama R, Yamazaki T. Testing an explicit method for multi-compartment neuron model simulation on a GPU. Cogn Comput. 2021;15(4):1118–31. https://doi.org/10.1007/s12559-021-09942-6
97. Sajikumar S, Navakkode S, Frey JU. Identification of compartment- and process-specific molecules required for "synaptic tagging" during long-term potentiation and long-term depression in hippocampal CA1. J Neurosci. 2007;27(19):5068–80. https://doi.org/10.1523/JNEUROSCI.4940-06.2007 PMID: 17494693
98. O'Donnell C, Sejnowski TJ. Selective memory generalization by spatial patterning of protein synthesis. Neuron. 2014;82(2):398–412. https://doi.org/10.1016/j.neuron.2014.02.028 PMID: 24742462
99. Kastellakis G, Silva AJ, Poirazi P. Linking memories across time via neuronal and dendritic overlaps in model neurons with active dendrites. Cell Rep. 2016;17(6):1491–504. https://doi.org/10.1016/j.celrep.2016.10.015 PMID: 27806290
100. Fonkeu Y, et al. How mRNA localization and protein synthesis sites influence dendritic protein distribution and dynamics. Neuron. 2019;103:1109–22. https://doi.org/10.1016/j.neuron.2019.06.022 PMID: 31350097
101. Sartori F, Hafner A-S, Karimi A, Nold A, Fonkeu Y, Schuman EM, et al. Statistical laws of protein motion in neuronal dendritic trees. Cell Rep. 2020;33(7):108391. https://doi.org/10.1016/j.celrep.2020.108391 PMID: 33207192
102. Zenke F, Laborieux A. Theories of synaptic memory consolidation and intelligent plasticity for continual learning. arXiv preprint 2024. https://doi.org/10.48550/arXiv.2405.16922
103. Eriksson J, Vogel EK, Lansner A, Bergström F, Nyberg L. Neurocognitive architecture of working memory. Neuron. 2015;88(1):33–46. https://doi.org/10.1016/j.neuron.2015.09.020 PMID: 26447571
104. Masse NY, Rosen MC, Freedman DJ. Reevaluating the role of persistent neural activity in short-term memory. Trends Cogn Sci. 2020;24(3):242–58. https://doi.org/10.1016/j.tics.2019.12.014 PMID: 32007384
105. O'Donnell C, Nolan MF, van Rossum MCW. Dendritic spine dynamics regulate the long-term stability of synaptic plasticity. J Neurosci. 2011;31(45):16142–56. https://doi.org/10.1523/JNEUROSCI.2520-11.2011 PMID: 22072667
106. Fauth M, Wörgötter F, Tetzlaff C. Long-term information storage by the interaction of synaptic and structural plasticity. The rewiring brain. Elsevier; 2017. p. 343–60. https://doi.org/10.1016/b978-0-12-803784-3.00016-0
107. Gallinaro JV, Gašparović N, Rotter S. Homeostatic control of synaptic rewiring in recurrent networks induces the formation of stable memory engrams. PLoS Comput Biol. 2022;18(2):e1009836. https://doi.org/10.1371/journal.pcbi.1009836 PMID: 35143489
108. Lu H, Diaz S, Lenz M, Vlachos A. The interplay between homeostatic synaptic scaling and homeostatic structural plasticity maintains the robust firing rate of neural networks. eLife Sciences Publications, Ltd; 2024. https://doi.org/10.7554/elife.88376.2
109. Kaster M, Czappa F, Butz-Ostendorf M, Wolf F. Building a realistic, scalable memory model with independent engrams using a homeostatic mechanism. Front Neuroinform. 2024;18:1323203. https://doi.org/10.3389/fninf.2024.1323203 PMID: 38706939
110. Yang G, Pan F, Gan W-B. Stably maintained dendritic spines are associated with lifelong memories. Nature. 2009;462(7275):920–4. https://doi.org/10.1038/nature08577 PMID: 19946265
111. Attardo A, Fitzgerald JE, Schnitzer MJ. Impermanence of dendritic spines in live adult CA1 hippocampus. Nature. 2015;523(7562):592–6. https://doi.org/10.1038/nature14467 PMID: 26098371
112. SBML: Systems biology markup language. 2025. Accessed 2025 December 18. https://sbml.org/about/core/
113. Senn W, Dold D, Kungl AF, Ellenberger B, Jordan J, Bengio Y, et al. A neuronal least-action principle for real-time learning in cortical circuits. eLife Sciences Publications, Ltd.; 2024. https://doi.org/10.7554/elife.89674.2
114. Golmohammadi A, Luboeinski J, Tetzlaff C. Skewed neuronal heterogeneity enhances efficiency on various computing systems. arXiv preprint 2025. https://doi.org/10.48550/arXiv.2412.05126
115. Morales GB, Mirasso CR, Soriano MC. Unveiling the role of plasticity rules in reservoir computing. Neurocomputing. 2021;461:705–15. https://doi.org/10.1016/j.neucom.2020.05.127
116. Boahen K. Dendrocentric learning for synthetic intelligence. Nature. 2022;612(7938):43–50. https://doi.org/10.1038/s41586-022-05340-6 PMID: 36450907
117. Khacef L, Klein P, Cartiglia M, Rubino A, Indiveri G, Chicca E. Spike-based local synaptic plasticity: a survey of computational models and neuromorphic circuits. Neuromorph Comput Eng. 2023;3(4):042001. https://doi.org/10.1088/2634-4386/ad05da
118. Luboeinski J. Arbor simulation of memory formation and consolidation in recurrent networks of spiking multi-compartment neurons with synaptic tagging and capture. 2024. Accessed 2024 December 17. https://github.com/jlubo/arbor_network_consolidation_mc
119. Schmitt S, Luboeinski J. FIPPA project code repository. 2024. Accessed 2024 November 23. https://github.com/tetzlab/FIPPA
120. Graupner M. Code related to: "Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location" 2020. Accessed 2024 November 23. https://github.com/mgraupe/CalciumBasedPlasticityModel/tree/main/Graupner2012PNAS
121. Hater T, et al. Arbor implementation of busyring benchmark with STDP. 2025. Accessed 2025 June 25. https://github.com/arbor-sim/arbor/commit/65a4a3a797f99096ff3619829b5d9e06e513f2c9
122. Luboeinski J, et al. CoreNEURON implementation of busyring benchmark. 2025. Accessed 2025 August 7. https://github.com/jlubo/coreneuron_busyring