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 Kamalabad [1,2],
Thorsten Hater [4], Fabian Bösch [5], Christian Tetzlaff [1,2,3]

[1] III. Institute of Physics – Biophysics, University of Göttingen, Germany
[2] Department for Neuro- and Sensory Physiology, University Medical Center Göttingen, Germany
[3] Campus Institute Data Science (CIDAS), Göttingen, Germany
[4] Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany
[5] Swiss National Supercomputing Centre, ETH Zürich, Switzerland
✉ Correspondence: jannik.luboeinski@med.uni-goettingen.de

25 November 2024

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 the Plastic Arbor framework, supporting simulations of a large variety of spike-driven plasticity paradigms. To showcase the features of the new 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. Using the new framework, we have already been able to investigate the impact of dendritic structures on network dynamics across a timescale of several hours, showing a relation between the length of dendritic trees and the ability of the network to efficiently store information.

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.

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].

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, 11, 12, 13, 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, 18, 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 offers a powerful tool for capturing processes at the synaptic, dendritic, and neuronal levels, enabling the examination of their links to network dynamics. Although the core functionality for running network simulations has been available since the inception of the Arbor project [6], the mechanisms required for modeling plasticity-related processes have largely been absent. In our present work, we fill this gap by extending the Arbor simulator by the functionalities needed to model diverse spike-driven plasticity rules (e.g., [20, 21, 22]), yielding the Plastic Arbor framework. Thereby, we aim to 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 availability statement below, or see https://github.com/tetzlab/plastic_arbor).

In this article, we describe the novel functionalities in two different sections, where we present methods and results in a side-by-side fashion. The new technical features needed to implement diverse plasticity rules are described in the section 'Extensions of the Arbor core code'. In the section '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. Specifically, we 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.

2 Extensions of the Arbor core code

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

2.1 Spike-time detection to simplify computation

Many formulations of synaptic plasticity depend on the timing of pre- and postsynaptic spikes [23, 24, 20]. 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. subsection 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 subsections 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 in the postsynaptic neuron (e.g., somatic action potentials or dendritic spikes) and to transmit this information to another compartment or a synapse without explicit implementation of the physical transmission process (e.g., the backpropagation of action potentials). This is, inter alia, needed for STDP rules (cf. subsection 3.1) and calcium-based plasticity rules (cf. subsections 3.3–3.6).

2.2 Multiple postsynaptic variables depending on pre- and postsynaptic spiking

The Arbor documentation [25] 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 subsections 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

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

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 depicted in Eq. 3 using the Euler-Maruyama algorithm, a first-order stochastic Runge-Kutta method [26]. 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 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, the standard 64-bit implementation of the Mersenne-Twister algorithm in C++ necessitates the sequential updating of a state comprising at least 19968 bits ($k = 312$) to produce a series of independent random samples.

Consequently, we have opted for the utilization of counter-based PRNGs (CBPRNGs) [27]. 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, $u_{i,n} = \varphi(c_i(n), \kappa_i(n))$.

Owing to this characteristic and their stateless nature, CBPRNGs lend themselves well to parallelization on both CPU and GPU architectures.

Specifically, we employ the `Threefry-4x64-12` algorithm from the `random123` library [28]. 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 [29, 30]. 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 Supplementary Fig. S7). 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 individually 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 [31]. 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

5

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 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 availability statement at the end of this article.

Please note that since plasticity dynamics are 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. 1 shows an overview of essential features of Arbor's user interface, which is typically used to implement and simulate models of plastic neuronal networks.

3.1 Spike-timing-dependent plasticity

Spike-timing-dependent plasticity (STDP) is a phenomenon that is reported in a number of experimental studies [32, 33, 34, 23] and described by various theoretical models. 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 $t_{\mathrm{pre}}^n - t_{\mathrm{post}}^m$ $(m, n \in \mathbb{N})$ between pre- and postsynaptic spikes. In this case, plasticity does not directly

Arbor GUI:

Python frontend:

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

```python
# 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)
# ...
```

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

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

HPC integration:

```python
# 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)
# ...
```

```python
# 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)
# ...
```
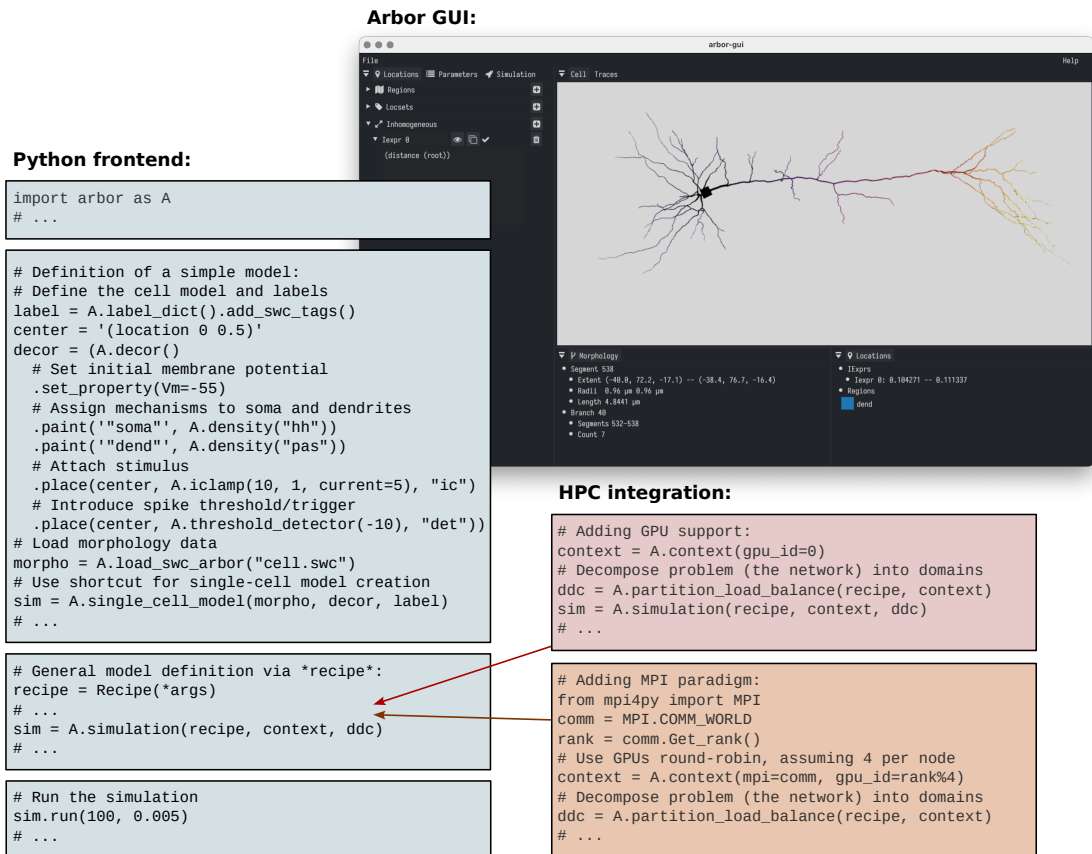
Figure 1: **Key features of Arbor's user interface:** graphical presentation of cell morphology with `Arbor GUI` (v0.8.0-dev-065b1c9); facilitated definition of models via the Python frontend (by setting up a so-called *recipe*; in some cases, there are also shortcuts that further accelerate the definition of certain simple models); modular integration of high performance computing (HPC) hardware such as graphics processing units (GPUs) and message passing interface (MPI) ranks.

depend on the spike rate. As a first step, we implemented a commonly used description [24] given by:

$$\frac{da_{\text{pre}}(t)}{dt} = -\frac{a_{\text{pre}}(t)}{\tau_{\text{pre}}} + A_{\text{pre}} \cdot \sum_n \delta(t - t_{\text{pre}}^n),\tag{6}$$

$$\frac{da_{\text{post}}(t)}{dt} = -\frac{a_{\text{post}}(t)}{\tau_{\text{post}}} + A_{\text{post}} \cdot \sum_m \delta(t - t_{\text{post}}^m),\tag{7}$$

$$\frac{dw(t)}{dt} = a_{\text{pre}}(t) \cdot \sum_m \delta(t - t_{\text{post}}^m) + a_{\text{post}}(t) \cdot \sum_n \delta(t - t_{\text{pre}}^n).\tag{8}$$

Here, the constants τ_{pre} and τ_{post} describe the decay of the eligibility traces $a_{\text{pre}}(t)$ and $a_{\text{post}}(t)$ induced by pre- and postsynaptic spikes, and $\delta(\cdot)$ represents the Dirac delta distribution. The amplitudes $A_{\text{pre}} > 0\,\mu S$ and $A_{\text{post}} < 0\,\mu S$ define the strengthening and weakening of the synaptic weight $w(t)$ that follow the occurrence of spikes, which takes more effect the 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 that was stimulated with Poisson input (see Fig. 2a). In addition, an inhibitory synapse was introduced to stabilize the dynamics. 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. subsection 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 and Supplementary Fig. S1a–c), 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 Supplementary Fig. S1d).

In addition to this experiment, we simulated a range of time delay settings to obtain a classical STDP time window, which shows a very good match with Brian 2 simulations as well (Fig. 2d).

Symbol	Value	Description	Refs.
w_0	$1.0\,\mu S$	Baseline of the synaptic weight	This study
τ_{pre}	$20.0\,ms$	Decay of the eligibility trace of presynaptic spikes	[24, 35]
τ_{post}	$\{10.0, 20.0\}\,ms$	Decay of the eligibility trace of postsynaptic spikes	[24, 35]
A_{pre}	$\{0.3, 0.01\}\,\mu S$	Strengthening amplitude	[24, 35]
A_{post}	$\{-0.2, -0.0105\}\,\mu S$	Weakening amplitude	[24, 35]
w_{max}	$10.0\,\mu S$	Maximum synaptic weight contributing to postsynaptic potentials	[35]

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 Supplementary Fig. S1a–d, and the second value was used to obtain the classical curve shown in Fig. 2d (also cf. [35]).
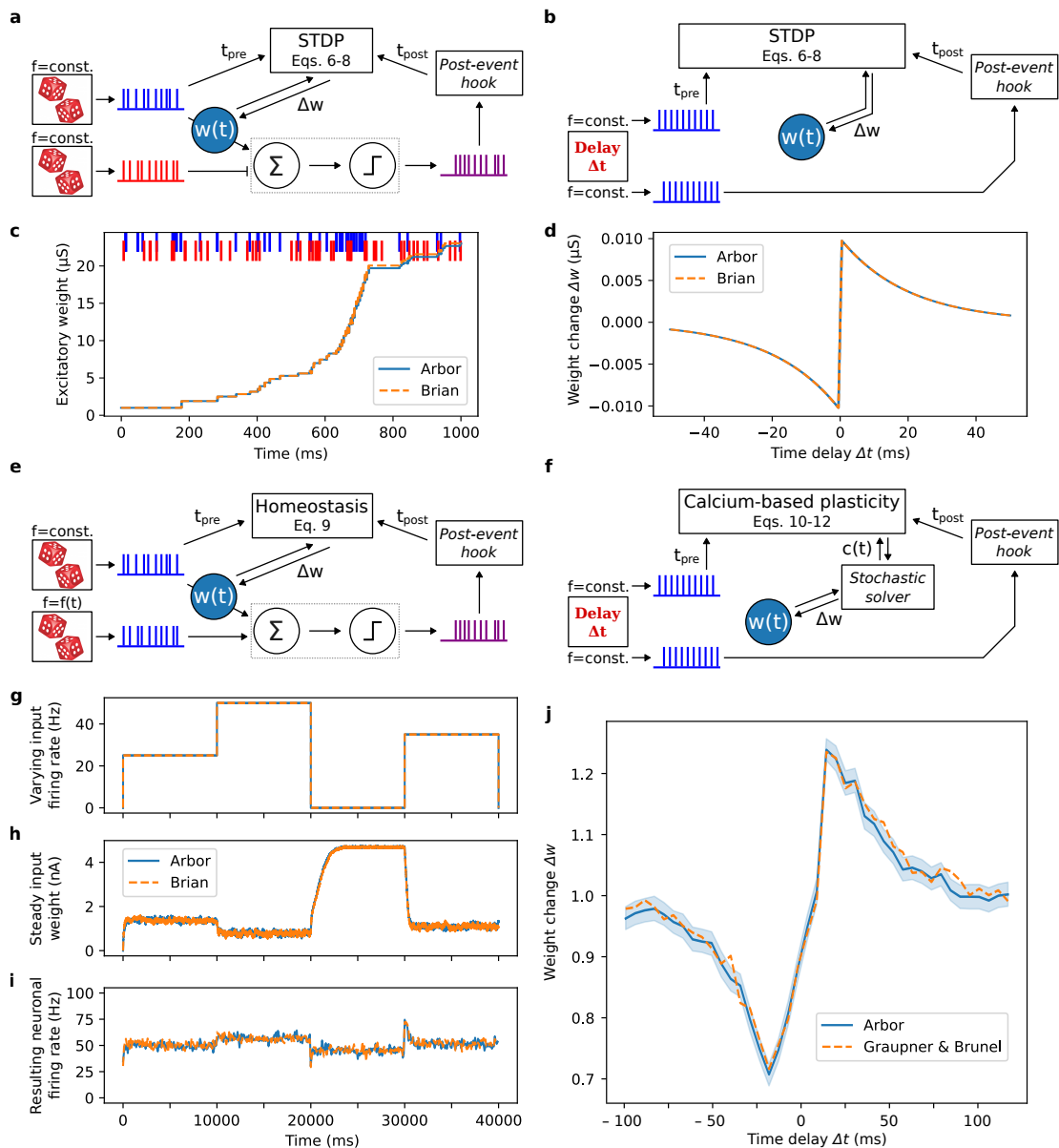
Figure 2: **Classical spike-timing dependent plasticity (STDP), spike-driven homeostasis, and calcium-driven synaptic plasticity in Arbor.** Arbor implementations (in lighter blue) are cross-validated by comparison to Brian 2 or a custom simulator (in orange). New features of the Arbor core code are 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 = 1.064 μS). **(d)** Classical STDP curve, obtained as detailed in (b) (CV > 0.999, RMSE = 0.001 μS). **(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.

10

Figure 2 *(previous page)*: **(f)** Paradigm of calcium-based, spike-timing- and rate-dependent synaptic plasticity, using the model by Graupner & Brunel [20]. 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 = 0.307 nA). **(i)** Firing rate of the neuron shown in (e) in the presence of homeostatic plasticity dynamics (CV = 0.508, RMSE = 1.981 Hz). **(j)** Calcium-driven synaptic plasticity as shown in (f). Reproduction of the numerical DP curve from Figure 2 of the related paper [20] (the mean is given by the dashed orange line). Every synapse is subject to 60 spike pairs presented at 1 Hz. 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 [36].

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 inputs 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. [37]). By using the newly implemented `POST_EVENT` hook (introduced in subsection 2.1), we could employ Arbor to simulate spike-driven homeostasis. To show this, we connect (similar to [38]) 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:

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

where the constants Δw_+ and Δw_- describe the weight changes upon the occurrence of pre- and postsynaptic spikes at times t_{pre}^n and t_{post}^m ($n, m \in \mathbb{N}$). 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 Supplementary Fig. S1e. We can see that in the case with homeostasis, the resulting firing rate is maintained at values around 50 Hz (Fig. 2i), while in the case without homeostasis (Supplementary Fig. S1e), 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 30 s), 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 [39].

3.3 Calcium-based synaptic plasticity

Following the implementation of a simple phenomenological STDP rule (subsection 3.1) as well as spike-driven homeostatic plasticity (subsection 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*,

11

Symbol	Value	Description	Refs.
w_{init}	$0.00\,\text{nA}$	Baseline weight of the plastic (fixed-rate input) synapse	[38]
w_{max}	$5.00\,\text{nA}$	Maximum weight of the plastic (fixed-rate input) synapse	[38]
Δw_+	$0.35\,\text{nA}$	Weight change due to presynaptic spike	[38]
Δw_-	$-0.35\,\text{nA}$	Weight change due to postsynaptic spike	[38]
w_{varying}	$3.50\,\text{nA}$	Weight of the varying-rate input synapse	[38]

Table 2: Parameters for the homeostatic plasticity model.

which is driven by pre- and postsynaptic spiking activity. It was presented by Graupner and Brunel in 2012 [20] and has since been used widely. In comparison to the phenomenological STDP rule, the calcium-based rule adapts the synaptic weight dynamics to depend on spike timing and spike rates, both of which have been shown to be important features of long-term synaptic plasticity [40].

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

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

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. Moreover, $\Theta[\cdot]$ denotes the Heaviside theta function, and

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

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. [41]). 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 subsection 2.3.

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

$$
\frac{dc(t)}{dt} = -\frac{c(t)}{\tau_{\text{c}}} + c_{\text{pre}} \cdot \sum_n \delta(t - t_{\text{pre}}^n - t_{\text{c,delay}}) + c_{\text{post}} \cdot \sum_m \delta(t - t_{\text{post}}^m),
\tag{12}
$$

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_{pre}^n and t_{post}^m, $t_{\text{c,delay}}$ 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 [20].

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	[20]
$t_{c,delay}$	13.7 ms	Delay of postsynaptic calcium influx after presynaptic spike	[20]
c_{pre}	1	Presynaptic calcium contribution, in vivo adjusted	[20]
c_{post}	2	Postsynaptic calcium contribution, in vivo adjusted	[20]
τ_c	20 ms	Calcium time constant	[20]
τ_w	150 s	Weight dynamics time constant	[20]
γ_p	321.808	Potentiation rate	[20]
γ_d	200	Depression rate	[20]
θ_p	1.3	Calcium threshold for potentiation	[20]
θ_d	1	Calcium threshold for depression	[20]
σ_{pl}	2.8248	Standard deviation for plasticity fluctuations	[20]

Table 3: Parameters for the implementation of the calcium-based plasticity model by Graupner & Brunel [20]. Note that as in the original mathematical model, the weights are kept without physical unit.

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 subsection 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 [21] that is based on observations at the level of a single neuron [20, 42].

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 [43]. Here, we consider a model that describes a number of such spines on a single dendritic branch [44]. The state and strength of these spines are subject to the previously mentioned calcium-based plasticity rule [21]. We describe the synaptic input to a specific spine via

$$I_{spine}^{Ca}(x, t_i) = \sum_i I_0 \cdot e^{-(t-t_i)/\tau_I} \cdot \Theta(t - t_i),$$

(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 10 ms. Note that we inject synaptic input at the top point of the spine heads, so the term $I_{\text{spine}}^{\text{Ca}}(x, t_i)$ 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:

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

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, each with head segments, as shown in Fig. 3a. 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 stand-alone simulator written in Python [45].

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 [50], 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. The custom code, however, incorporates distinct influx and outflux coefficients inspired by [50] and [54]. 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 [55, 56]. As a next step for our modeling demonstrations, we reproduce the results of standard protocols eliciting early-

14

Symbol	Value	Description	Refs.
γ_p	90	Potentiation rate	This study
γ_d	0.01	Depression rate	This study
θ_p	$0.11\,\mu\text{mol/l}$	Calcium threshold for potentiation	This study
θ_d	$0.05\,\mu\text{mol/l}$	Calcium threshold for depression	This study
r_{head}	$1.0\,\mu\text{m}$	Spine head radius	[46]
l_{head}	$1.0\,\mu\text{m}$	Spine head length	[46]
r_{dendrite}	$1.0\,\mu\text{m}$	Dendrite radius	[46]
l_{dendrite}	$80.0\,\mu\text{m}$	Dendrite length	This study
Δl_{comp}	$1.0\,\mu\text{m}$	Length of one compartment	This study
τ_C	$100\,\text{ms}$	Calcium decay time constant	[47]
τ_I	$1\,\text{ms}$	Injection current time constant	[48, 49]
γ	0.11	Fraction of current carried by Ca^{2+}	[50]
I_0	$4.0\,\text{pA}$	Injection current (Arbor implementation)	[50]
I_0	$5.5\,\text{pA}$	Injection current (stand-alone implementation)	[50]
D	$2.2 \cdot 10^{-10}\,\text{m}^2/\text{s}$	Calcium diffusion constant	[51, 52, 50]

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.

and late-phase plasticity at a single nerve fiber or single synapse (cf. the experimental results in [57]). To achieve this, we implemented the complex theoretical model from [22] in Arbor, requiring all of the new core components that we introduced in section 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++ [58] that was custom-developed and used in the scope of several previous studies [22, 59, 60, 61]. 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 [22, 61]. Also note that a UML sequence diagram of the model implementation is provided in Supplementary Fig. S5.

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:

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

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:

15
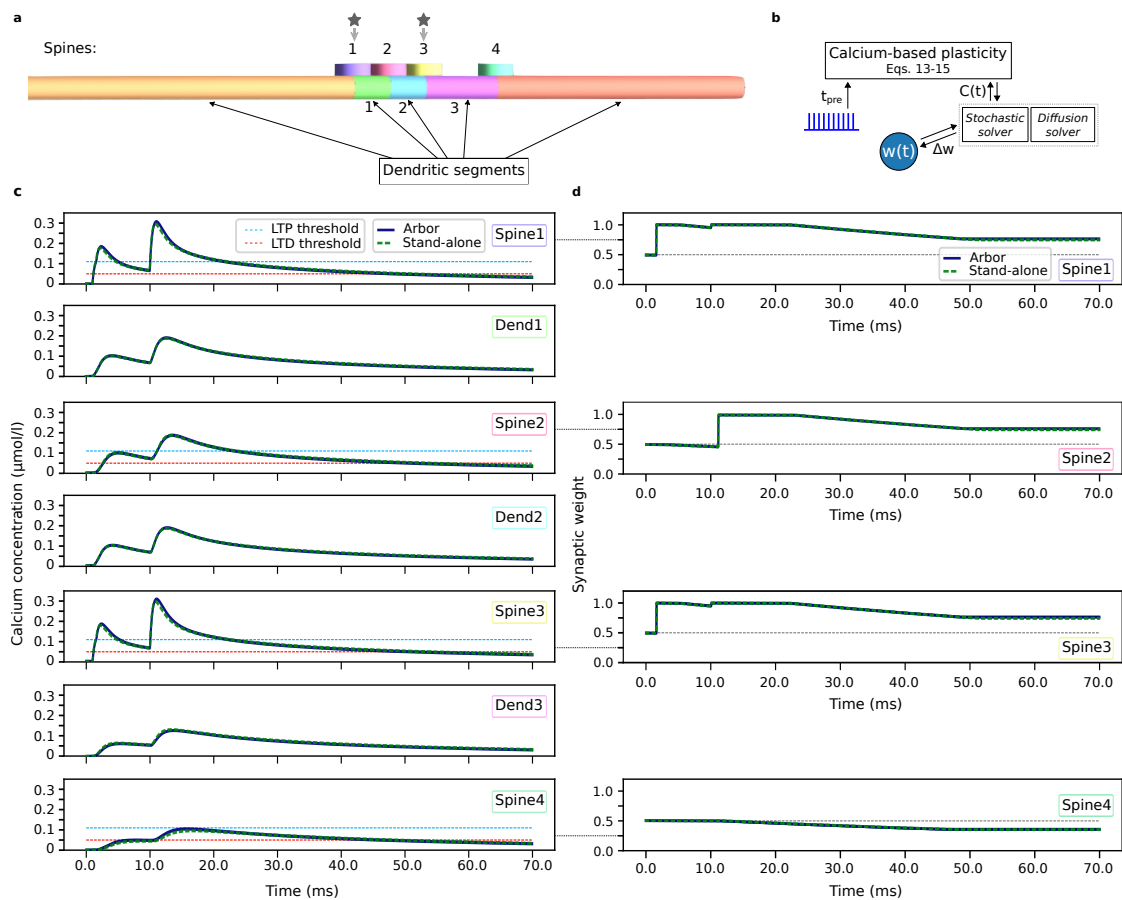
Figure 3: **Calcium-driven heterosynaptic plasticity in four spines on a dendritic branch.** Calcium is first introduced in spines 1 and 3 through synaptic input. Subsequently, calcium spatially distributes across the dendrite (according to Eq. 14), which influences synaptic plasticity at other synapses, promoting either depression or potentiation. **(a)** Illustration generated using `Arbor GUI` [53]. Each segment is represented by a different color, and a segment can consist of multiple compartments. Spines 1–4 are located at $x = -1.0, 0.0, 1.0, 3.0\,\mu\text{m}$, respectively. For the purpose of visualization, the morphology has been clipped at $-10\,\mu\text{m}$ and $+10\,\mu\text{m}$, 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. [44]). 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: $(0.991, 0.005\,\mu\text{mol/l})$; spine 2: $(0.998, 0.002\,\mu\text{mol/l})$; spine 3: $(0.989, 0.005\,\mu\text{mol/l}$; spine 4: $(0.967, 0.004\,\mu\text{mol/l}$; dendrite location 1: $(0.998, 0.002\,\mu\text{mol/l})$; dendrite location 2: $(0.998, 0.002\,\mu\text{mol/l})$; dendrite location 3: $(0.998, 0.003\,\mu\text{mol/l}))$. **(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: $(0.996, 0.003)$.

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

with scaling factor σ_pl and Gaussian white noise $\Gamma(t)$ (which has a mean value of zero and a variance of $1/dt$ [41]). Note that this calcium-driven model of early-phase plasticity is based on the model by Graupner & Brunel [20], which we considered in subsection 3.3. Adaptations by [62] and [22] have enabled the model to be compatible with synaptic tagging and capture models (also see [60, 63]). As in the original model (cf. Eq. 12), the calcium concentration depends on pre- and postsynaptic spikes at times t_pre^n and t_post^m, and is described by the following equation:

$$\frac{dc(t)}{dt} = -\frac{c(t)}{\tau_\mathrm{c}} + c_\mathrm{pre} \cdot \sum_n \delta(t - t_\mathrm{pre}^n - t_\mathrm{c,delay}) + c_\mathrm{post} \cdot \sum_m \delta(t - t_\mathrm{post}^m), \tag{19}$$

where τ_c is a time constant, c_pre and c_post are the spike-induced increases in the calcium concentration, $t_\mathrm{c,delay}$ 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 [62]:

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

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 [64, 62, 22]:

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

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 Supplementary Fig. S2). 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 Supplementary Fig. S3; 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

17

to match very well. In addition, we compared to Brian 2 again, which also shows a very good match (shown in Supplementary Fig. S3).

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 Supplementary Fig. S6). 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 [58, 65]) 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 [22]). 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 Supplementary Fig. S4 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 [22]. 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 10 s or 8 h 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 8 h. 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 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 subsection 3.7 and Supplementary Fig. S13). 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 [22, 63].

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 [66] and [67]). 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 subsection 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 [22]:

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

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 [22] with our Arbor implementation is shown in Figure 4h (also cf. Supplementary Fig. S10). 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 and Supplementary Figs. S2–S4), 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 biologically realistic functional dynamics (cf. [70, 71]).

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 [72, 73]. 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}. Note that otherwise, the electrical properties of the neurons are the same as in the single-compartment case presented in the previous subsection 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 $D_{\text{sps}} = 10^{-11}$ m^2/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

Symbol	Value	Description	Refs.
h_0	$4.20075\,\text{mV}$ $= 0.5\,\frac{\gamma_\text{p}}{\gamma_\text{p}+\gamma_\text{d}}$	Baseline value of the excitatory→excitatory coupling strength	[20, 62, 22]
$t_{c,\text{delay}}$	$0.0188\,\text{s}$	Delay of postsynaptic calcium influx after presynaptic spike	[20, 62, 22]
c_pre	1.0 (0.6)	Presynaptic calcium contribution (in vivo adjusted)	[20, 62, 68, 22]
c_post	0.2758 (0.1655)	Postsynaptic calcium contribution (in vivo adjusted)	[20, 62, 68, 22]
τ_c	$0.0488\,\text{s}$	Calcium time constant	[20, 62, 22]
τ_h	$688.4\,\text{s}$	Early-phase time constant	[20, 62, 22]
τ_p	$60\,\text{min}$	PRP time constant	[64, 62, 22]
τ_z	$60\,\text{min}$	Late-phase time constant	[64, 62, 22]
γ_p	1645.6	Potentiation rate	[20, 62, 22]
γ_d	313.1	Depression rate	[20, 62, 22]
θ_p	3.0	Calcium threshold for potentiation	[62, 22]
θ_d	1.2	Calcium threshold for depression	[62, 22]
σ_pl	$2.90436\,\text{mV}$	Standard deviation for plasticity fluctuations	[20, 62, 22]
p_max	$10.0\,\text{µmol/l}$	PRP synthesis scaling constant (equilibrium PRP concentration under ongoing PRP synthesis)	[64, 62, 22]
θ_pro	$2.10037\,\text{mV}$ $= 0.5\,h_0$	PRP synthesis threshold	[62, 22]
θ_tag	$0.840149\,\text{mV}$ $= 0.2\,h_0$	Tagging threshold	[62, 22]
f_int	$0.1\,\text{1/µmol}$	Late-phase factor accounting for PRP integration into the synapse	This study
r_comp	$1{\cdot}10^{-3}\,\text{µm}$	Radius of the single-compartment cell	This study
l_cell	$2{\cdot}10^{-3}\,\text{µm}$	Length of the single-compartment cell	This study

Table 5: Parameters for the model with calcium-based early-phase plasticity and STC-based late-phase plasticity based on [62] and [22]. 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 [69, 20]. For networks, the calcium contribution parameters are corrected by a factor of 0.6 to account for in vivo conditions (cf. [68]).

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 subsection 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 ($D_\mathrm{p} = 10^{-11}\,\mathrm{m^2/s}$), 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 Supplementary Fig. S8). However, although we adjusted the c_morpho parameter for the postsynaptic potentials to match those in the single-compartment paradigm, the electrical properties of the morphological structure show to have an impact 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. 4h and 5e,i). On the other hand, considering the long-term dynamics with very slow diffusion ($D_\mathrm{p} = 10^{-19}\,\mathrm{m^2/s}$), 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 ($D_\mathrm{p} = 10^{-15}\,\mathrm{m^2/s}$), PRPs reach the synapses after a certain time that arises from a complex interplay between synapses, soma, and dendrites (see the spatial distribution in Supplementary Fig. S9). These dynamics still enable functional memory recall and can serve to regulate the late-phase maintenance of synaptic changes. For increased size 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 Supplementary Fig. S11). 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 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.

Paradigm	Symbol	Value	Description	Refs.
(Any)	Δl_{comp}	$1.0\,\mu\mathrm{m}$	Length of one compartment	This study
Small cells	r_{comp}	$6.0\,\mu\mathrm{m}$	Effective radius of a compartment (used for dendrites as well as soma)	[70, 74, 75, 71]
	l_{soma}	$12.0\,\mu\mathrm{m}$	Length of the soma	[70, 74, 75]
Large cells	r_{comp}	$12.0\,\mu\mathrm{m}$	Effective radius of a compartment (used for dendrites as well as soma)	[70, 74, 75, 71]
	l_{soma}	$24.0\,\mu\mathrm{m}$	Length of the soma	[70, 74, 75]
Small dendrites	$l_{\mathrm{dendriteA}}$	$12.5\,\mu\mathrm{m}$	Length of apical dendritic branch	[76, 74, 71]
	$l_{\mathrm{dendriteB}}$	$5.0\,\mu\mathrm{m}$	Length of basal dendritic branch	[76, 74, 71]
	c_{morpho}	1.035	Correction factor for the altered impact of postsynaptic potentials due to the morphology	This study (referring to model in [22])
Large dendrites	$l_{\mathrm{dendriteA}}$	$25.0\,\mu\mathrm{m}$	Length of apical dendritic branch	[76, 74, 71]
	$l_{\mathrm{dendriteB}}$	$10.0\,\mu\mathrm{m}$	Length of basal dendritic branch	[76, 74, 71]
	c_{morpho}	1.020	Correction factor for the altered impact of postsynaptic potentials due to the morphology	This study (referring to model in [22])

Table 6: Cell morphology parameters for the network simulations of memory formation and consolidation with morphological neurons (subsection 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.
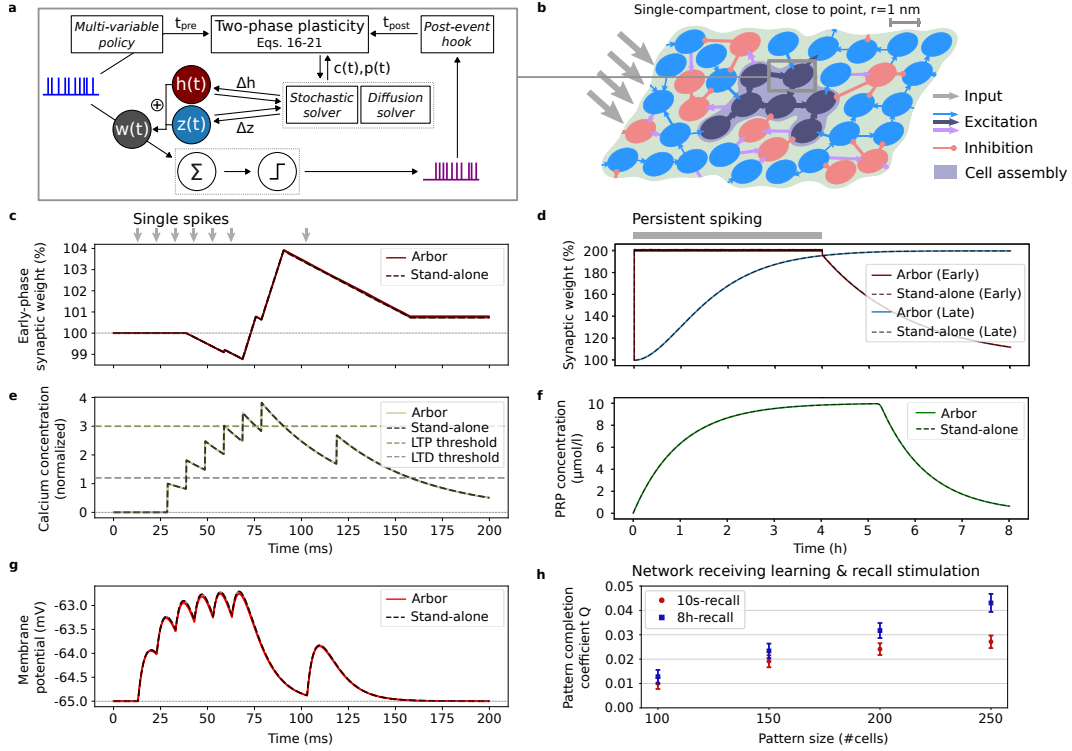
Figure 4: **Basic early- and late-phase plasticity with synaptic tagging and capture (STC), cross-validated with stand-alone simulator, and memory recall performance with single-compartment model. (a)** Paradigm of two-phase synaptic plasticity with calcium-based early phase and late phase described by synaptic tagging and capture (see [22]). 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 = 0.040 mV. **(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 = 0.226 mV; late phase: CV > 0.999, RMSE = 0.055 mV). **(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 = 0.107 μmol/l). **(g)** Membrane potential of the postsynaptic neuron (CV > 0.999, RMSE = 0.151 mV). 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 [58] by darker, dashed lines. For each curve, error bands represent the standard error of the mean (mostly too small to be visible). **(f)** Memory recall in networks of single-compartment neurons simulated with Arbor (qualitatively reproducing the point-neuron results of [22]). 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.
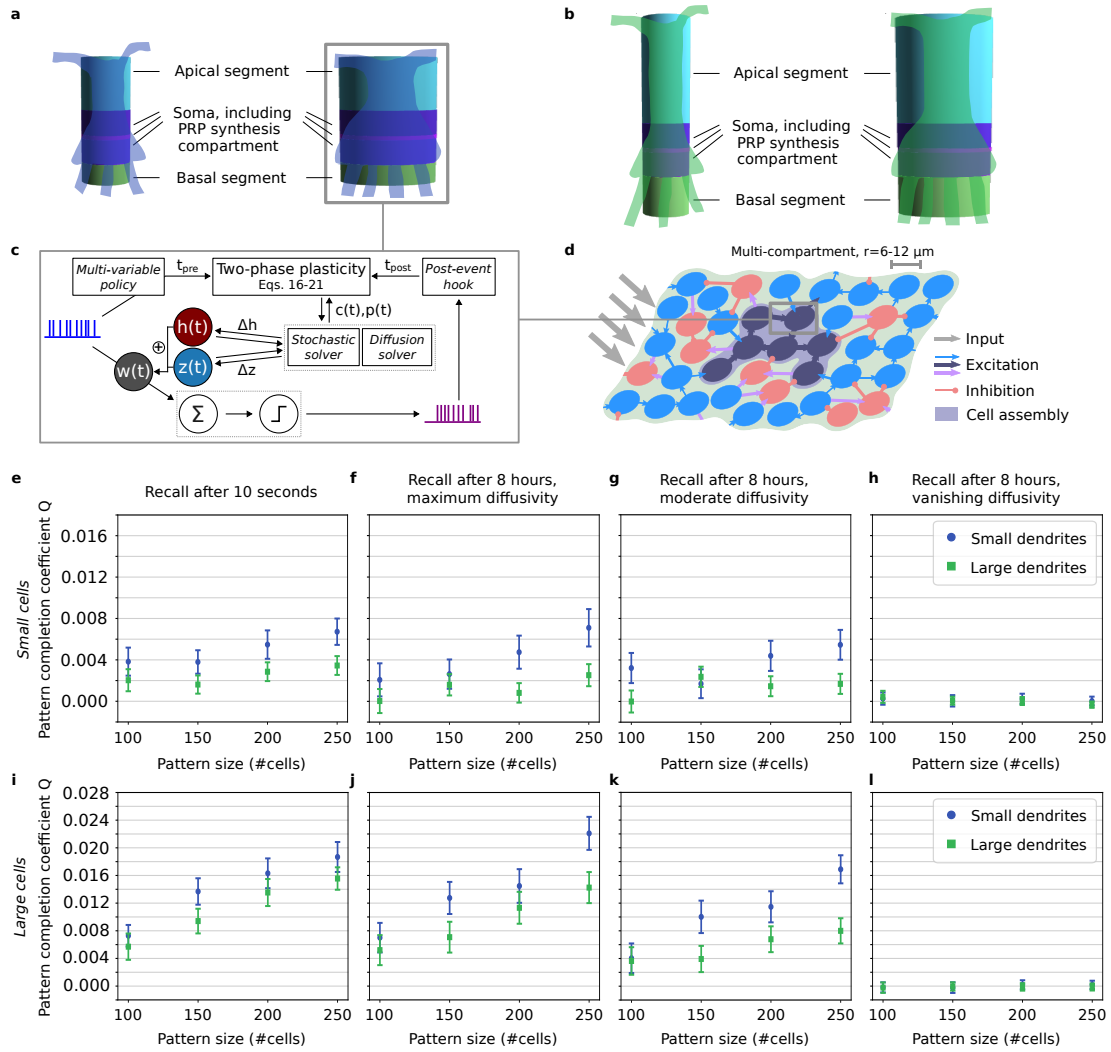
Figure 5: **Memory recall in a recurrent network of multi-compartment neurons after learning and after consolidation.** Results obtained with Arbor for networks of different kinds of multi-compartment neurons, demonstrating the impact of different values of the PRP diffusivity D_{p} on memory consolidation. Networks consist of 'small' cells (diameter of $6\,\mu m$) or of 'large' cells (diameter of $12\,\mu m$), with either small or large dendrites (in which cases each neuron comprises in total 31 or 48 compartments, respectively). The diameter and length values are given in Table 6. **(a,b)** Illustrations of used cell structures, generated using `Arbor GUI` [53]. 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 small dendrites, **(b)** the same with large 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)).

24

Figure 5 *(previous page)*: **(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 $10\,\mathrm{s}$ after learning (technically, $D_\mathrm{p} = 10^{-11}\,\mathrm{m^2/s}$, but late-phase plasticity does not occur on such short timescales). **(f)** Recall stimulation at $8\,\mathrm{h}$ after learning, $D_\mathrm{p} = 10^{-11}\,\mathrm{m^2/s}$. **(g)** Recall stimulation at $8\,\mathrm{h}$ after learning, $D_\mathrm{p} = 10^{-15}\,\mathrm{m^2/s}$. **(h)** Recall stimulation at $8\,\mathrm{h}$ after learning, $D_\mathrm{p} = 10^{-19}\,\mathrm{m^2/s}$. **(i-l)** Same as (e-h), but for large cells that consist of segments of twice the diameter.

3.7   Runtime and memory benchmarking

As a final step, we measure 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 (subsections 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 [77, 4] as well as to the custom stand-alone simulator [58] 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. 6a; comparing the main memory use with the GPU case is not meaningful since the GPU has its own memory). Another point to be mentioned is Arbor's support for single instruction, multiple data (SIMD) vectorization. As shown in Fig. 6a, 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 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 (Fig. 6a) and the memory consumption (Fig. 6b) only increase slightly when shifting from single-compartment neurons to neurons with 48 compartments. However, it is important to note that the considered multi-compartment network in general exhibits fewer spikes than the single-compartment version (Supplementary Fig. S12b; also cf. subsection 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 (Supplementary Fig. S12a).

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 (Supplementary Fig. S13). The runtimes of those simulations are much longer than for 10s-recall (cf. Fig. 6). As we used the 'fast-forward' approximation (see subsection 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.

4 Discussion

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 subsections 3.1 & 3.2, we presented plasticity rules that can be considered a basis for further spike-based mechanisms of synaptic plasticity. In subsections 3.3–3.5, we considered three different calcium-based rules in different scenarios. The results in subsection 3.3 constitute a reproduction of a widely-used calcium model that was directly fitted to experimental data [20]. The results in subsection 3.4 show how a model of calcium diffusion along dendrites can be employed to simulate heterosynaptic plasticity (cf. [21, 44]). In subsection 3.5, we used a calcium model as basis for a more complex model that captures early- and heterosynaptic late-phase plasticity [62, 22, 63]. Finally, we moved to the network level (subsections 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 [22] 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 large 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 [71]. 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 CoreNEU-RON [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 [78]. Further, GENESIS (GEneral NEural SImulation System) [79] is a simulator that has also been used widely for several decades, offering a platform for simulating both small-scale detailed models and large-scale network simulations. However, the use of GENESIS with respect to modern hardware backends is very limited. 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 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 [80]. CARLsim [81] 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 [82] 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 NeuronGPU [83] is optimized for neural network simulations on GPU but does currently not support multi-compartment models.

In summary, each of the existing simulators comes with its own strengths and weaknesses (also see [84, 85, 86]). 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. This focus makes Arbor an attractive option, particularly, for researchers who rely on high-performance computing for their simulations with networks of morphologically detailed neurons.

We have cross-validated all of our presented Arbor implementations either with Brian 2 [4] or with one of multiple stand-alone simulators that were custom-developed for previous studies [20, 58, 45]. At the single-synapse level, we did not find any significant differences between the results of the simulators (Figs. 2–4; Supplementary Figs. S1–S3). However, certain differences in the neuron model and in the numerical methods, as well as complex network effects, give rise to certain quantitative deviations in memory recall performance at the circuit level, while the qualitative behavior remained the same (Supplementary Fig. S10; also cf. subsection 3.5 and Supplementary Fig. S4).

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 by computing 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, Supplementary Fig. S12). This is not entirely unexpected, since Arbor has been designed particularly for networks of multi-compartment neurons.

The benchmarking results notwithstanding, there may be several ways to 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. [87], 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,

27

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 size of the dendritic branches, 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 [88, 89, 90, 91, 92].

In the future, most importantly, the extended Arbor simulation framework can enable researchers to conduct studies that examine the interplay between neuron-internal and network dynamics. 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. [93]). 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 [94], reservoir computing with heterogeneous networks [95, 96], or prototyping and benchmarking of neuromorphic algorithms (cf. [85, 97]).

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, 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. In the future, the Plastic Arbor framework may power a great variety of studies considering synaptic mechanisms and their interactions with neuronal dynamics and morphologies, from single synapses to large networks.
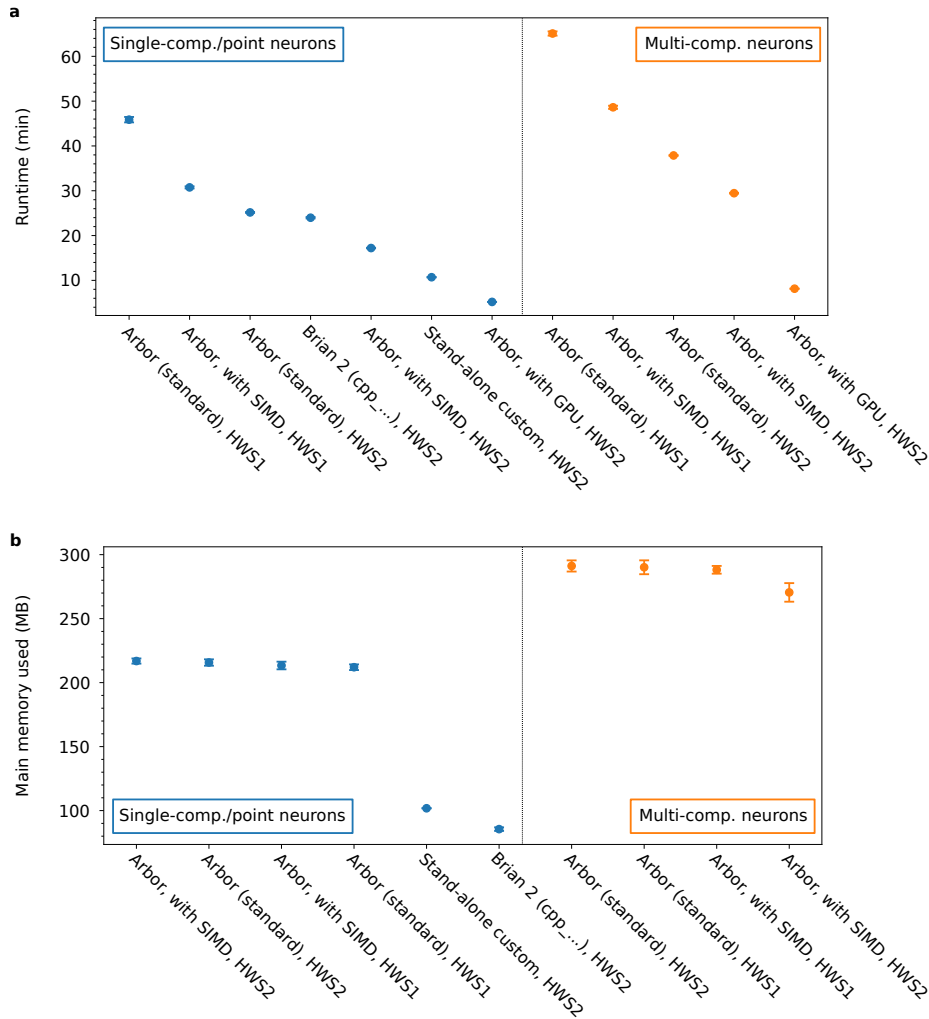
Figure 6: **Benchmarking results of runtime and memory use for Arbor and point-neuron simulators.** For 10s-recall paradigm in networks of 2000 neurons. The single-compartment simulations in Arbor as well the point-neuron simulations in Brian 2 (with `cpp_standalone` device) [77, 4] and in the custom stand-alone simulator [58] are conducted as described in subsection 3.5; they are represented by blue data points on the left. The Arbor simulations with multi-compartment/morphological neurons are conducted as described in subsection 3.6 and represented by orange data points on the right. Results are given for different hardware systems, HWS1: an older desktop computer (Intel Core i5-6600 CPU @ 3.30GHz, 1×8 GB DDR3-RAM, using 1 thread), HWS2: a newer compute server (AMD Ryzen Threadripper PRO 5995WX CPU, 8×32 GB DDR4-RAM, using 1 thread, in specified cases with NVIDIA T1000 8 GB GPU). For Arbor, results are distinguished between standard CPU execution, CPU with SIMD support, and with GPU support. **(a)** Total runtime of the simulations. Measurements were performed using `hyperfine` in version 1.15. **(b)** 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`. Data points represent the average over 10 trials; error bars represent the standard deviation.

29

Data 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 7 below. 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 [67]). For the simulation results of subsection 3.4, we used the development branch of pull request #2226 with the state of commit f0e456d due to a necessary fix in the diffusion processing; the branch shall be merged upon publication of this article (similarly, Arbor still considers diffusion constants in non-standard units, scaled by a factor of 10^{-7}, which will be corrected as well). For the benchmarking results in subsection 3.7 we used Arbor version 0.10.0. Note that while the mentioned Arbor versions are by now slightly outdated, we ensured that our extensive software tests from [67, 98] pass with these versions as well as with the release version 0.10.0 and the currently latest version 0.10.1-dev-7d1f82e, strongly indicating 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/simulator_comparison.

Section	Model description	Arbor implementation	Reference implementation(s)
3.1	Spike-timing-dependent plasticity	[99] (subdirectory `STDP/`)	[99]
3.2	Spike-driven homeostatic plasticity	[99] (subdirectory `spike_based_homeostasis/`)	[99]
3.3	Calcium-based synaptic plasticity	[7] (Arbor main repository, `calcium_stdp.py`)	[100]
3.4	Heterosynaptic calcium-based plasticity in dendrites	[45] (`Arbor_diff.py`)	[45] (`Custom.py`)
3.5	Synaptic tagging and capture, in individual synapses and in networks of single-compartment neurons	[66] (reduced code for single synapses) and [67]	[58]; [65] (reduced code for single synapses) and [77]
3.6	Synaptic memory consolidation in networks of morphological neurons	[98]	- (novel results)
3.7	Network models from 3.5 and 3.6	[67]; [98]	[58]; [77]

Table 7: Overview of the simulation code used to perform the simulations presented in this article.

Acknowledgments

Funding

Author contributions

Conceptualization: JL, SS, CT; Data curation: JL, SS, SSK; Formal Analysis: JL, SS, SSK; Funding acquisition: SS, CT; Investigation: JL, SS, SSK, CT; Methodology: JL, SS, SSK, CT; Project administration: CT; Resources: TH, CT; Software (Arbor core): JL, SS, TH, FB (with larger contributions by TH and FB); Software (sections 3.1–3.2): JL, SS; Software (section 3.3): SS, FB; Software (section 3.4): SSK; Software (sections 3.5–3.7): JL; Supervision: JL, SS, CT; Validation: JL, SS, SSK, TH, FB; Visualization: JL, SS, SSK, TH; Writing – original draft: JL, SS, SSK, TH, FB; Writing – review & editing: JL, SS, CT.

References

[1] Hines, M. A program for simulation of nerve equations with branching geometries. *International Journal of Bio-Medical Computing* **24**, 55–68 (1989).

[2] Kumbhar, P. *et al.* CoreNEURON: an optimized compute engine for the NEURON simulator. *Frontiers in Neuroinformatics* **13**, 63 (2019).

[3] Gewaltig, M.-O. & Diesmann, M. NEST (NEural Simulation Tool). *Scholarpedia* **2**, 1430 (2007).

[4] Stimberg, M., Brette, R. & Goodman, D. F. Brian 2, an intuitive and efficient neural simulator. *eLife* **8**, e47314 (2019).

[5] Awile, O. *et al.* Modernizing the NEURON simulator for sustainability, portability, and performance. *Frontiers in Neuroinformatics* **16**, 884046 (2022).

[6] Abi Akar, N. *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)*, 274–282 (IEEE, 2019).

[7] Cumming et al. Arbor Library v0.10.0 (2024). URL https://doi.org/10.5281/zenodo.13284959.

[8] Martin, S. J., Grimwood, P. D. & Morris, R. G. M. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual Review of Neuroscience* **23**, 649–711 (2000).

[9] Abraham, W. C., Jones, O. D. & Glanzman, D. L. Is plasticity of synapses the mechanism of long-term memory storage? *NPJ Science of Learning* **4**, 1–10 (2019).

[10] Smolen, P., Baxter, D. A. & Byrne, J. H. Molecular constraints on synaptic tagging and maintenance of long-term potentiation: a predictive model. *PLOS Computational Biology* **8**, e1002620 (2012).

[11] Gallimore, A. R., Kim, T., Tanaka-Yamamoto, K. & De Schutter, E. Switching on depression and potentiation in the cerebellum. *Cell Reports* **22**, 722–733 (2018).

[12] Mäki-Marttunen, T., Iannella, N., Edwards, A. G., Einevoll, G. T. & Blackwell, K. T. A unified computational model for cortical post-synaptic plasticity. *eLife* **9**, e55714 (2020).

[13] Becker, M. F. P. & Tetzlaff, C. The biophysical basis underlying the maintenance of early phase long-term potentiation. *PLOS Computational Biology* **17**, e1008813 (2021).

[14] Bonilla-Quintana, M. & Wörgötter, F. Exploring new roles for actin upon LTP induction in dendritic spines. *Scientific Reports* **11** (2021).

[15] Berridge, M. J. Dysregulation of neural calcium signaling in alzheimer disease, bipolar disorder and schizophrenia. *Prion* **7**, 2–13 (2013).

[16] Bonilla-Quintana, M. & Rangamani, P. Can biophysical models of dendritic spines be used to explore synaptic changes associated with addiction? *Physical Biology* **19**, 041001 (2022).

[17] Acharya, J. *et al.* Dendritic computing: branching deeper into machine learning. *Neuro-science* **489**, 275–289 (2022).

[18] Pagkalos, M., Makarov, R. & Poirazi, P. Leveraging dendritic properties to advance machine learning and neuro-inspired computing. *Current Opinion in Neurobiology* **85**, 102853 (2024).

[19] Zheng, H. *et al.* Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. *Nature Communications* **15**, 277 (2024).

[20] Graupner, M. & Brunel, N. Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. *Proceedings of the National Academy of Sciences of the USA* **109**, 3991–3996 (2012).

[21] Hiratani, N. & Fukai, T. Detailed dendritic excitatory/inhibitory balance through heterosynaptic spike-timing-dependent plasticity. *Journal of Neuroscience* **37**, 12106–12122 (2017).

[22] Luboeinski, J. & Tetzlaff, C. Memory consolidation and improvement by synaptic tagging and capture in recurrent neural networks. *Communications Biology* **4**, 275 (2021).

[23] Bi, G.-q. & Poo, M.-m. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience* **18**, 10464–10472 (1998).

[24] Song, S., Miller, K. D. & Abbott, L. F. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience* **3**, 919–926 (2000).

[25] Arbor documentation – Selection policy. https://docs.arbor-sim.org/en/latest/python/cell.html#arbor.selection_policy (2023). Accessed: 8 November 2024.

[26] Kloeden, P. E. & Platen, E. *Numerical Solution of Stochastic Differential Equations* (Springer Berlin, Heidelberg, 1992).

[27] Salmon, J. K., Moraes, M. A., Dror, R. O. & Shaw, D. E. 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 (Association for Computing Machinery, New York, NY, USA, 2011).

[28] Salmon, J. K., Morales, M. A. & Fenn, M. Random123: a library of counter-based random number generators. https://github.com/DEShawResearch/random123 (2010). Accessed: 23 November 2024.

[29] Rall, W. Theory of physiological properties of dendrites. *Annals of the New York Academy of Sciences* **96**, 1071–1092 (1962).

[30] Yates, S. Analytic solutions to the cable equation. https://github.com/arbor-sim/arbor/blob/07a9fa2ff3d4ed9173ecfb316a056a723e081de3/doc/math/cable_equation/cable_equation.tex (2019). Accessed: 23 November 2024.

[31] Sætra, M. J., Einevoll, G. T. & Halnes, G. An electrodiffusive, ion conserving pinsky-rinzel model with homeostatic mechanisms. *PLOS Computational Biology* **16**, e1007661 (2020).

[32] Gerstner, W., Kempter, R., Van Hemmen, J. L. & Wagner, H. A neuronal learning rule for sub-millisecond temporal coding. *Nature* **383**, 76–78 (1996).

[33] Magee, J. C. & Johnston, D. A synaptically controlled, associative signal for Hebbian plasticity in hippocampal neurons. *Science* **275**, 209–213 (1997).

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

[35] Introduction to Brian part 2: Synapses. https://brian2.readthedocs.io/en/stable/resources/tutorials/2-intro-to-brian-synapses.html (2021). Accessed: 14 June 2024.

[36] Arbor Tutorial: Spike Timing-dependent Plasticity Curve. https://docs.arbor-sim.org/en/stable/tutorial/calcium_stdp_curve.html (2023). Accessed: 14 June 2024.

[37] Zenke, F. & Gerstner, W. Hebbian plasticity requires compensatory processes on multiple timescales. *Philosophical Transactions of the Royal Society B* **372**, 20160259 (2017).

[38] Breitwieser, O. *Towards a Neuromorphic Implementation of Spike-based Expectation Maximization.* Master's thesis, Universität Heidelberg (2015). URL http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3240.

[39] Brian 2 example: "spike_based_homeostasis". https://brian2.readthedocs.io/en/latest/examples/synapses.spike_based_homeostasis.html (2021). Accessed: 14 June 2024.

[40] Sjöström, P. J., Turrigiano, G. G. & Nelson, S. B. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* **32**, 1149–1164 (2001).

[41] Gillespie, D. T. Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. *Physical Review E* **54** (1996).

[42] Lee, K. F., Soares, C., Thivierge, J.-P. & Béïque, J.-C. Correlated synaptic inputs drive dendritic calcium amplification and cooperative plasticity during clustered synapse development. *Neuron* **89**, 784–799 (2016).

[43] Stuart, G., Spruston, N. & Häusser, M. *Dendrites* (Oxford University Press, 2016), 3rd edn. URL https://doi.org/10.1093/acprof:oso/9780198745273.001.0001.

[44] Shafiee Kamalabad, S., Schmitt, S. & Tetzlaff, C. Calcium-based input timing learning. *Preprint at bioRxiv* (2024). URL https://doi.org/10.1101/2024.11.14.623617.

[45] Shafiee Kamalabad, S. Arbor simulation of calcium-dependent heterosynaptic plasticity (2024). URL https://github.com/Shirin1993/Arbor_diffusion. Accessed: 23 November 2024.

[46] Araya, R. Input transformation by dendritic spines of pyramidal neurons. *Frontiers in Neuroanatomy* **8**, 1–18 (2014).

[47] Yasuda, R. Biophysics of Biochemical Signaling in Dendritic Spines: Implications in Synaptic Plasticity (2017).

[48] Häusser, M. & Roth, A. Estimating the time course of the excitatory synaptic conductance in neocortical pyramidal cells using a novel voltage jump method. *Journal of Neuroscience* **17**, 7606–7625 (1997).

[49] Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. *Neuronal dynamics: From single neurons to networks and models of cognition* (Cambridge University Press, 2014).

[50] Biess, A., Korkotian, E. & Holcman, D. Barriers to diffusion in dendrites and estimation of calcium spread following synaptic inputs. *PLoS computational biology* **7**, e1002182 (2011).

[51] Allbritton, N. L., Meyer, T. & Stryer, L. Range of messenger action of calcium ion and inositol 1,4,5-trisphosphate. *Science* **258**, 1812–1815 (1992).

[52] Means, S. *et al.* Reaction diffusion modeling of calcium dynamics with realistic ER geometry. *Biophysical Journal* **91**, 537–557 (2006).

[53] Hater, T. *et al.* Arbor GUI v0.8 (2022). URL https://doi.org/10.5281/zenodo.7415130.

[54] Schuss, Z., Singer, A. & Holcman, D. The narrow escape problem for diffusion in cellular microdomains. *Proceedings of the National Academy of Sciences* **104**, 16098–16103 (2007).

[55] Frey, U. & Morris, R. G. M. Synaptic tagging and long-term potentiation. *Nature* **385**, 533–536 (1997).

[56] Redondo, R. L. & Morris, R. G. M. Making memories last: the synaptic tagging and capture hypothesis. *Nature Reviews Neuroscience* **12**, 17–30 (2011).

[57] Sajikumar, S., Navakkode, S., Sacktor, T. C. & Frey, J. U. Synaptic tagging and cross-tagging: the role of protein kinase Mζ in maintaining long-term potentiation but not long-term depression. *Journal of Neuroscience* **25**, 5750–5756 (2005).

[58] 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). URL https://doi.org/10.5281/zenodo.4429195.

[59] Luboeinski, J. & Tetzlaff, C. Organization and priming of long-term memory representations with two-phase plasticity. *Cognitive Computation* 275 (2022).

[60] Luboeinski, J. *The Role of Synaptic Tagging and Capture for Memory Dynamics in Spiking Neural Networks*. Dissertation, University of Göttingen (2021). URL http://doi.org/10.53846/goediss-463.

[61] Lehr, A. B., Luboeinski, J. & Tetzlaff, C. Neuromodulator-dependent synaptic tagging and capture retroactively controls neural coding in spiking neural networks. *Scientific Reports* **12**, 17772 (2022).

[62] Li, Y., Kulvicius, T. & Tetzlaff, C. Induction and consolidation of calcium-based homo- and heterosynaptic potentiation and depression. *PLOS One* **11**, e0161679 (2016).

[63] Luboeinski, J. & Tetzlaff, C. Modeling emergent dynamics arising from synaptic tagging and capture at the network level. In Sajikumar, S. & Abel, T. (eds.) *Synaptic Tagging and Capture: From Synapses to Behavior*, chap. 23, 471–503 (Springer, Cham, Switzerland, 2024), 2nd edn.

[64] 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 Computational Biology* **4**, e10000248 (2008).

[65] Luboeinski, J. Brian 2 simulation of the induction of early- and late-phase plasticity at a single synapse (2023). URL https://github.com/jlubo/brian_synaptic_plasticity_stc. Accessed: 23 November 2024.

[66] Luboeinski, J. Arbor simulation of the induction of early- and late-phase plasticity at a single synapse (2023). URL https://github.com/jlubo/arbor_2N1S. Accessed: 23 November 2024.

[67] Luboeinski, J. Arbor simulation of memory formation and consolidation in recurrent spiking neural networks with synaptic tagging and capture (2024). URL https://github.com/jlubo/arbor_network_consolidation. Accessed: 23 November 2024.

[68] Higgins, D., Graupner, M. & Brunel, N. Memory maintenance in synapses with calcium-based plasticity in the presence of background activity. *PLOS Computational Biology* **10**, e1003834 (2014).

[69] Wittenberg, G. M. & Wang, S. S.-H. Malleability of spike-timing-dependent plasticity at the CA3–CA1 synapse. *Journal of Neuroscience* **26**, 6610–6617 (2006).

[70] Jiang, S. *et al.* Anatomically revealed morphological patterns of pyramidal neurons in layer 5 of the motor cortex. *Scientific Reports* **10**, 7916 (2020).

[71] Carnevale, N. T. & Hines, M. L. *The NEURON book* (Cambridge University Press, 2006).

[72] Larkman, A. U. Dendritic morphology of pyramidal neurones of the visual cortex of the rat: Iii. spine distributions. *Journal of Comparative Neurology* **306**, 332–343 (1991).

[73] Gillon, C. J. *et al.* Responses of pyramidal cell somata and apical dendrites in mouse visual cortex over multiple days. *Scientific Data* **10**, 287 (2023).

[74] Van Aerde, K. I. & Feldmeyer, D. Morphological and physiological characterization of pyramidal neuron subtypes in rat medial prefrontal cortex. *Cerebral Cortex* **25**, 788–805 (2015).

[75] Džaja, D., Hladnik, A., Bičanić, I., Baković, M. & Petanjek, Z. Neocortical calretinin neurons in primates: increase in proportion and microcircuitry structure. *Frontiers in Neuroanatomy* **8**, 103 (2014).

[76] Carriba, P. & Davies, A. M. CD40 is a major regulator of dendrite growth from developing excitatory and inhibitory neurons. *eLife* **6**, e30442 (2017).

[77] Luboeinski, J. Brian 2 simulation of memory formation and consolidation in recurrent spiking neural networks based on synaptic tagging and capture (2024). URL https://github.com/jlubo/brian_network_plasticity. Accessed: 23 November 2024.

[78] NEURON documentation – CoreNEURON compatibility. https://nrn.readthedocs.io/en/8.2.2/coreneuron/compatibility.html (2022). Accessed: 8 November 2023.

[79] Bower, J. M. & Beeman, D. *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System* (Bower, Beeman, 2003), http://genesis-sim.org/GENESIS/iBoG/index.html, internet edn.

[80] Pagkalos, M., Chavlis, S. & Poirazi, P. Introducing the dendrify framework for incorporating dendrites to spiking neural networks. *Nature Communications* **14**, 131 (2023).

[81] Niedermeier, L. *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)*, 1–10 (IEEE, 2022).

[82] Panagiotou, S., Sidiropoulos, H., Soudris, D., Negrello, M. & Strydis, C. EDEN: A high-performance, general-purpose, NeuroML-based neural simulator. *Frontiers in Neuroinformatics* **16**, 724336 (2022).

[83] Golosio, B. *et al.* Fast simulations of highly-connected spiking cortical models using gpus. *Frontiers in Computational Neuroscience* **15**, 627620 (2021).

[84] Tikidji-Hamburyan, R. A., Narayana, V., Bozkus, Z. & El-Ghazawi, T. A. Software for brain network simulations: a comparative study. *Frontiers in Neuroinformatics* **11**, 46 (2017).

[85] Kulkarni, S. R., Parsa, M., Mitchell, J. P. & Schuman, C. D. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing* **447**, 145–160 (2021).

[86] Wang, C. *et al.* Brainpy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming. *eLife* **12**, e86365 (2023).

[87] Kobayashi, T., Kuriyama, R. & Yamazaki, T. Testing an explicit method for multi-compartment neuron model simulation on a GPU. *Cognitive Computation* 1–14 (2021).

[88] Sajikumar, S., Navakkode, S. & Frey, J. U. Identification of compartment- and process-specific molecules required for "synaptic tagging" during long-term potentiation and long-term depression in hippocampal CA1. *Journal of Neuroscience* **27**, 5068–5080 (2007).

[89] O'Donnell, C. & Sejnowski, T. J. Selective memory generalization by spatial patterning of protein synthesis. *Neuron* **82**, 398–412 (2014).

[90] Kastellakis, G., Silva, A. J. & Poirazi, P. Linking memories across time via neuronal and dendritic overlaps in model neurons with active dendrites. *Cell Reports* **17**, 1491–1504 (2016).

[91] Fonkeu, Y. *et al.* How mRNA localization and protein synthesis sites influence dendritic protein distribution and dynamics. *Neuron* **103**, 1109–1122 (2019).

[92] Sartori, F. *et al.* Statistical laws of protein motion in neuronal dendritic trees. *Cell Reports* **33** (2020).

[93] Lu, H., Diaz, S., Lenz, M. & Vlachos, A. Interplay between homeostatic synaptic scaling and homeostatic structural plasticity maintains the robust firing rate of neural networks. *Reviewed Preprint at eLife* (2023). URL https://doi.org/10.7554/eLife.88376.1.

[94] Senn, W. *et al.* A neuronal least-action principle for real-time learning in cortical circuits. *Reviewed Preprint at eLife* (2023). URL https://doi.org/10.7554/elife.89674.1.

[95] Perez-Nieves, N., Leung, V. C., Dragotti, P. L. & Goodman, D. F. Neural heterogeneity promotes robust learning. *Nature Communications* **12**, 5791 (2021).

[96] Morales, G. B., Mirasso, C. R. & Soriano, M. C. Unveiling the role of plasticity rules in reservoir computing. *Neurocomputing* **461**, 705–715 (2021).

[97] Khacef, L. *et al.* Spike-based local synaptic plasticity: A survey of computational models and neuromorphic circuits. *Neuromorphic Computing and Engineering* **3**, 042001 (2023).

[98] Luboeinski, J. Arbor simulation of memory formation and consolidation in recurrent networks of spiking multi-compartment neurons with synaptic tagging and capture (will be made available upon publication of the article).

[99] Schmitt, S. & Luboeinski, J. FIPPA project code repository (2024). URL https://github.com/tetzlab/FIPPA. Accessed: 23 November 2024.

[100] Graupner, M. Code related to: "Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location" (2020). URL https://github.com/mgraupe/CalciumBasedPlasticityModel/tree/main/Graupner2012PNAS. Accessed: 23 November 2024.