MCELL4 WITH BIONETGEN: A MONTE CARLO SIMULATOR OF RULE-BASED REACTION-DIFFUSION SYSTEMS WITH PYTHON INTERFACE

A PREPRINT

Adam Husar
Computational Neurobiology Lab
Salk Institute for Biological Studies
California, La Jolla 92037
adam@adam-husar.com

Mariam Ordyan
Institute for Neural Computations
University of California, San Diego
California, La Jolla, 92037
marordyan@gmail.com

Guadalupe C. Garcia
Computational Neurobiology lab
Salk Institute for Biological Studies
California, La Jolla 92037
ggarcia@salk.edu

Joel G. Yancey
Computational Neurobiology Lab
Salk Institute for Biological Studies
California, La Jolla 92037
joelgyancey@ucla.edu

Ali S. Saglam
Department of Computational and Systems Biology
University of Pittsburgh
Pittsburgh, PA 15260
als251@pitt.edu

James R. Faeder
Department of Computational and Systems Biology
University of Pittsburgh
Pittsburgh, PA 15260
faeder@pitt.edu

Thomas M. Bartol
Computational Neurobiology Lab
Salk Institute for Biological Studies
California, La Jolla 92037
bartol@salk.edu

Terrence J. Sejnowski
Institute for Neural Computations
University of California, San Diego
California, La Jolla 92037
terry@salk.edu

May 1, 2023

Corresponding author: bartol@salk.edu

ABSTRACT

Biochemical signaling pathways in living cells are often highly organized into spatially segregated volumes, membranes, scaffolds, subcellular compartments, and organelles comprising small numbers of interacting molecules. At this level of granularity stochastic behavior dominates, well-mixed continuum approximations based on concentrations break down and a particle-based approach is more accurate and more efficient. We describe and validate a new version of the open-source MCell simulation program (MCell4), which supports generalized 3D Monte Carlo modeling of diffusion and chemical reaction of discrete molecules and macromolecular complexes in solution, on surfaces representing membranes, and combinations thereof. The main improvements in MCell4 compared to the previous versions, MCell3 and MCell3-R, include a Python interface and native BioNetGen reaction language (BNGL) support. MCell4's Python interface opens up completely new possibilities of interfacing with external simulators allowing creation of sophisticated event-driven multiscale/multiphysics simulations. The native BNGL support through a new open-source library libBNG (also introduced in this paper) provides the capability to run a given BNGL model spatially resolved in MCell4 and, with appropriate simplifying assumptions, also in the BioNetGen simulation environment, greatly accelerating and simplifying model validation and comparison.

Contents

1 Introduction **3**

1.1 Particle-Based Reaction Dynamics Tools . . . 3

1.2 MCell4 Python Application Programming Interface . . . 4

1.3 Motivation for a New BioNetGen Library . . . 4

1.4 MCell4 Features . . . 4

1.4.1 Python/C++ API for Model Creation and Model Execution . . . 5

1.4.2 The Reaction Language is Now BNGL . . . 5

1.4.3 Ability to Go Back and Forth between MCell4 and BNG Simulator Environments . . . 5

1.4.4 Other Advanced Features . . . 5

1.5 Model Creation and Visualization in CellBlender . . . 5

2 Methods **5**

2.1 MCell4: a Bird's Eye View . . . 5

2.2 Python API Generator: a Closer Look . . . 7

2.3 MCell4 Model Structure . . . 7

2.3.1 MCell4 Python API Example . . . 8

2.4 Graph-Based Approach To Protein Modeling . . . 8

2.4.1 Extension of BNGL for Volume-Surface Reactions . . . 8

2.4.2 Units and Interoperability between MCell4 and BioNetGen . . . 11

2.4.3 Example of an MCell4 Model with BioNetGen Specification . . . 12

3 Results **12**

3.1 Testing & Validation . . . 12

3.1.1 SNARE Complex . . . 13

3.1.2 Event-Driven Release of Neurotransmitter by SNARE Complex . . . 13

3.1.3 CaMKII Model with Large Reaction Network . . . 15
3.1.4 Volume-Surface and Surface-Surface Reactions: Membrane Localization Model . . . 16
3.1.5 Stochastic Fluctuations in a System with Multiple Steady States: Autophosphorylation 17
3.2 Performance . . . 17
3.3 Hybrid Simulation Example . . . 18

4 Conclusions **19**
4.1 Summary . . . 19
4.2 Availability and Future Directions . . . 22
4.3 Acknowledgements . . . 22

1 Introduction

Living cells are complex structures in which biomolecules and biochemical processes are spatially organized and span the extracellular space, plasma membrane, cytosol and subcellular organelles. These biochemical processes are intrinsically multiscale since they are based on molecular interactions on a small scale leading to emergent behavior of cells on a larger scale. Due to the dynamic nature of biochemical processes on different temporal and spatial scales, appropriate mathematical tools are required to understand the underlying dynamics and to dissect the mechanisms that control system behavior [1]. Overall, understanding how cellular design dictates function is essential to understanding life and disease, in the brain, heart, or elsewhere. MCell (Monte Carlo Cell) is a biochemistry simulation tool that uses spatially realistic 3D cellular models and stochastic Monte Carlo algorithms to simulate the movements and interactions of discrete molecules within and between cells[2, 3, 4, 5]. Here we describe MCell4, a new version of MCell.

One of the most important new features in MCell4 is a flexible Python application programming interface (API), that allows coupling between MCell and other simulation engines or custom code. By itself MCell performs particle-based reaction-diffusion simulations in the spatial and temporal scales of nm to 10s of µm and µs to 10s of seconds. But MCell4's Python API facilitates the generation of multiscale hybrid models, as we demonstrate here with an example.

A second important addition to MCell4 is an efficient support for rule-based modeling using the BioNetGen (BNG) Language (BNGL). BNG is an open source software package for representing and simulating biochemical reactions [6]. While powerful, BNG models are non-spatial. Support for models expressed using BNGL in MCell4 can help dissect the role space plays in different scenarios. This is not a trivial task since both the time scales of diffusion and reactions [7] as well as the spatial localization of proteins condition the results.

We first present the design principles of MCell4 and its API, and next we introduce the new BioNetGen library. And finally we demonstrate some of the new features in MCell 4 with examples and present a hybrid model coupling spatial simulations in MCell with ordinary differential equations (ODEs).

1.1 Particle-Based Reaction Dynamics Tools

In particle-based reaction-diffusion simulations, each molecule is represented as an individual agent. Molecules diffuse either in volumes or on membranes and may affect each other by reacting upon collision. A review of the currently maintained particle-based stochastic simulators that describes Smoldyn [7], eGFRD [8], SpringSaLaD [9], ReaDDy [10], and MCell3 was recently published in [11].

MCell is a particle-based simulator that represents molecules as point particles. The typical simulation time-step in MCell is 1 µs, and the simulated times can stretch from seconds to minutes. Briefly, MCell operates as follows. As a volume molecule diffuses a single step by random Brownian motion, all molecules within a given radius along its trajectory, or at the point of collision on a surface, are considered for a reaction. For surface molecules, the molecule first diffuses one step, and then its neighbors are evaluated for reaction. There is no volume exclusion for molecules diffusing in 3D volumes, and molecules on surfaces (in membranes) occupy a fixed area. MCell allows defining arbitrary geometry, and complex models such as a 180 μm^3 3DEM reconstruction of hippocampal neuropil have been used to construct a geometrically-precise

simulation of 100s of neuronal synapses at once [5]. A detailed description of the mathematical foundations of MCell's algorithms can be found here [2, 3, 4].

MCell3-R [12] is an extension of MCell that supports BNGL [13] and allows modeling of protein complexes or polymers by using rule-based definition of reactions. MCell3-R uses a library called NFSim [14] to compute the products of reaction rules for reactions described in BNGL.

MCell4 is a new C++ implementation of MCell, providing a versatile Python interface in addition to many other improvements. In particular it is significantly faster when dealing with complex reaction networks expressed as rules in BNGL. And most of MCell's features introduced previously [4] have been retained. First, let us briefly go over the motivation for the new features in MCell4.

1.2 MCell4 Python Application Programming Interface

We had two important motivations behind the creation of the MCell4 Python API: 1) to give the users the freedom to customize their models in a full-featured modern programming language, and 2) create an easy way to couple MCell4 with other simulation platforms to allow multi-scale, multi-physics simulations.

The main goal when designing the new API for MCell4 was to allow definition of complex models combining many reaction pathways distributed over complex geometry. Thus, one of the main requirements was to enable modularity with reusable components that can be independently validated. One can then build complex models by combining existing or new modules.

Similar to the approach in the PySB modeling framework [15], a model in MCell4 is seen as a piece of software, and the same processes used in software development can also be applied to biological model development. The most important ones are: 1) incremental development where the model is built step by step, relying on solid foundations of modeling done and validated before, 2) modularity that provides the capability to create self-contained, reusable libraries, 3) unit testing and validation to verify that parts of the model behave as expected, and 4) human-readable and writable model code that can be stored using git or other code version control software which, besides being essential for incremental development, also allows code reviews [16] so that other team members can inspect the latest changes to the model and can contribute their own modules to the growing code base.

1.3 Motivation for a New BioNetGen Library

NFSim [14] is a C++ library that provides BioNetGen support, implements the network-free method, and is used in MCell3-R [12]. To use a BNGL model in MCell3-R, the BNGL file first needs to be parsed by the BioNetGen compiler; then, a converter generates MDL, MDLR, and XML files. These files then constitute the model to be simulated in MCell3-R. The main disadvantage with this approach is that the original BNGL file is not a part of the MCell3-R model anymore, and every time changes are made to it, the converter must be run again, and any potential changes made by hand to our MCell3-R model files will be lost. There are also performance and memory consumption issues with MCell3-R when the simulated system has a huge number of potential reactions.

For a seamless integration of BNGL in MCell4, a better solution was needed. Therefore, we implemented a new library for the BioNetGen language that contains a BNGL parser and a network-free BNG reaction engine whose main purpose is to compute the reaction products given a reaction rule and reactants. This BNG library (libBNG) was designed with independence from MCell4 in mind so that it can be used in other simulation tools. libBNG does not support all special features and keywords of the BioNetGen tool suite yet, most notably, BNGL functions are not supported, but the set of supported features is sufficient for any MCell4 model. And note that when needed, functions can be represented in MCell4's Python code. Sources of libBNG are available under the MIT license here: [17].

1.4 MCell4 Features

Here we briefly enumerate some of the features of MCell4. In the results section we have chosen a few relevant examples specifically to demonstrate some of these features. When appropriate, we will indicate which example uses the mentioned feature.

1.4.1 Python/C++ API for Model Creation and Model Execution

All models can now be created in Python. While CellBlender (see section 1.5) allows creation of some of the simpler models without the need to write Python code by hand, for more complicated customized models at least part of the model needs to include a custom Python script. And while CellBlender allows inclusion of custom python scripts in a model, for simplicity and explanatory power, all the examples presented in the results section of this paper are written purely in Python.

1.4.2 The Reaction Language is Now BNGL

In MCell4 the reaction language is BNGL [13]. Thus, MCell4 fully supports rule-based reactions and all models use this feature.

Most importantly, the support for BNGL and NFSim means that MCell4 allows direct, agent-based evaluation of reaction rules and thus enables spatially-resolved network-free simulations of interactions between and among volume and surface molecules. The CaMKII model in the results section 3.1.3, for example, would not be possible without the spatial network-free simulation allowed by MCell4.

1.4.3 Ability to Go Back and Forth between MCell4 and BNG Simulator Environments

The new BNG library [17] allows direct loading and parsing of a BNG model and using it in realistic 3D cellular geometry. This allows simulation and comparison of results of non-spatial (simulated with BioNetGen solvers) and spatial (simulated with MCell4) implementations of the same BNG model. This is demonstrated with several of the examples in the results section, namely: SNARE (3.1.1), and CaMKII (3.1.3).

And if the spatial features are found to be unimportant for a given model, and simulation speed is of more concern, then by keeping the BNGL file as a separate module, one can run it with the BNG simulator when appropriate. See section 2.4.3 for an example.

1.4.4 Other Advanced Features

Among the more advanced features introduced in MCell4 is the possibility to include transcellular and transmembrane interactions between surface molecules located on separate membranes. MCell4 also allows both coarse-grained and fine-grained customization of models through time-step customization and event-driven callbacks. Callbacks allow to run custom Python code in the event of a reaction or molecule-wall collision. An example of use of callbacks to implement release of neurotransmitter by activation of a SNARE complex is given in section-3.1.2.

Finally, the new Python API introduced in MCell4 entails the ability to run multi-scale multi-physics hybrid simulations taking advantage of all the existing Python packages. For an example of a hybrid model see section 3.3.

1.5 Model Creation and Visualization in CellBlender

CellBlender is a Blender [18] addon that allows creation and visualization of MCell4 models. CellBlender was updated from its previous MCell3 version and the main new features are: automatic generation of well structured Python code for complete MCell4 models from their CellBlender representations, execution and visualization of these models, and visualization of simulation runs from externally created Python-only models. CellBlender offers an easy way how to start with MCell through built-in examples (Fig. 1 shows a Rat Neuromuscular Junction model example), and tutorials [19].

2 Methods

2.1 MCell4: a Bird's Eye View

We will briefly review MCell4's architecture and fundamental apects of its API, starting with Fig. 2.

MCell advances simulation time by running iterations. Duration of an iteration is given by a user-defined time step (usually 1 µs). Scheduler keeps track of events to be run in each iteration and the main simulation loop implemented in the World object asks scheduler to handle subsequent events (Fig. 3) until the desired number of iterations was run.

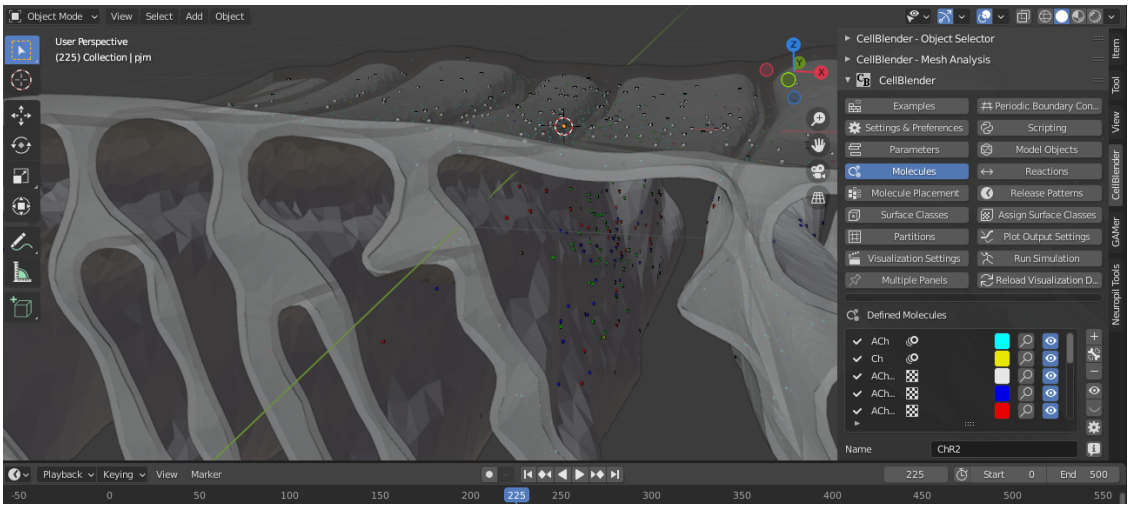

Figure 1: MCell4 models can be created, executed, and visualized using CellBlender, an addon for Blender. The capabilities of Blender are indispensible for creating complex geometries for MCell4 models.

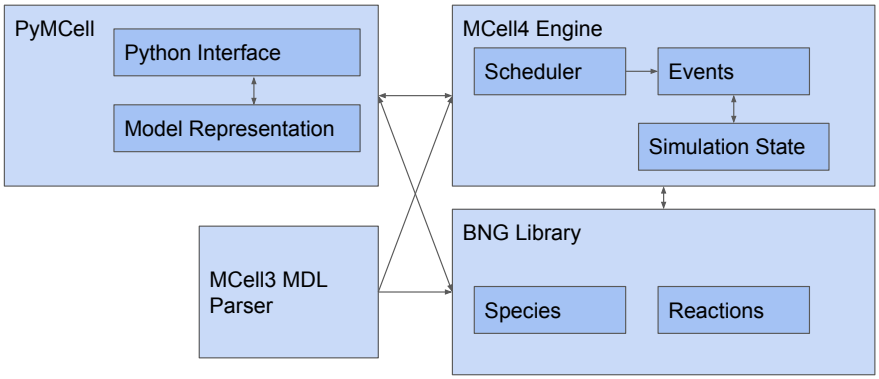

Figure 2: MCell4 is composed of four main components: 1) PyMCell library provides Python interface and contains classes to hold the model representation, 2) MCell4 engine implements the simulation algorithms, 3) BNG (BioNetGen) library provides methods to resolve BioNetGen reactions, and 4) MDL (Model Description Language) parser for backwards compatibility with MCell3.

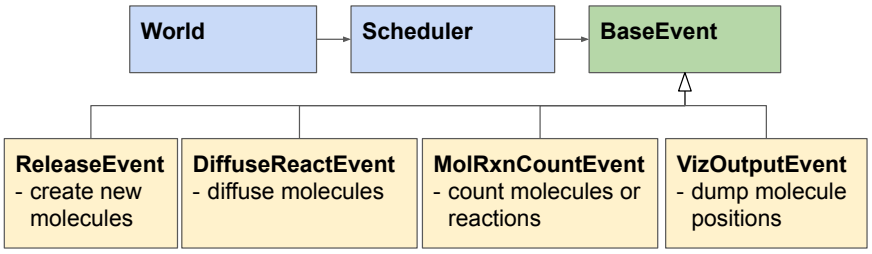

Figure 3: Scheduler executes time step iterations which consists of discrete events executed in this order: 1) ReleaseEvent creates new molecules, 2) MolRxnCountEvent counts numbers of molecules of or how many times a reaction occurred, 3) VizOutputEvent stores molecule locations for visualization in CellBlender, and 4) DiffuseReactEvent diffuses molecules, checks collisions, and executes reactions. Only the DiffuseReactEvent has to be executed each time step to move time, other events listed here are optional.

2.2 Python API Generator: a Closer Look

The MCell4 physics engine is implemented in C++. To ensure a reliable correspondence between a model representation in python and in C++, the quality of the user experience when creating a model, and a well maintained documentation, we have have developed a Python API generator, which reads a file in the YAML format and automatically generates all the base C++ classes, their corresponding Python API representations, code for informative error messages, and documentation.

Thanks to the API generator schematically represented in Fig. 4, when new features are added to MCell4, one only needs to modify the single API definition in the YAML format to ensure that both the API and the documentation reflect the new features.

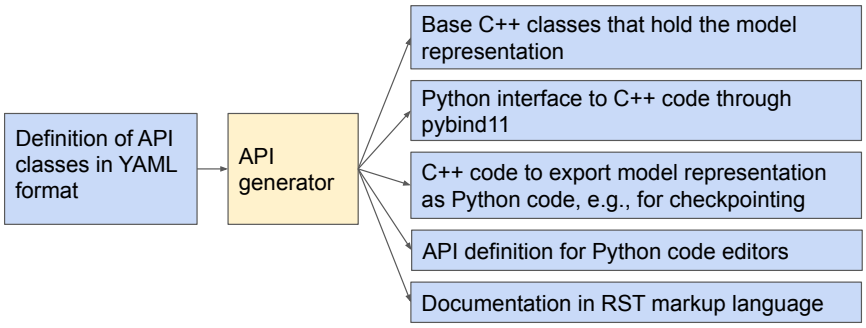

Figure 4: API generator reads a high-level definition of the MCell4 Python interface and generates code and documentation. Automatic generation of a tool's API gives the possibility to easily modify or extend the API while making sure that all parts including documentation stay consistent. The API generator is a general tool that can be also used (with minor modifications) for other software tools that combine C++ and Python [20].

2.3 MCell4 Model Structure

An important aspect to allow reusability (e.g., [21]) is to have a predefined model structure. This way, every piece of code for a given component (such as reaction definitions, geometry, initial model state, and observables) is in a file with a specified name and follows a predefined coding style. Such standardized model structure (shown in Fig. 5) aids in the reuse of code and simplifies the job of modelers by allowing them to leverage existing model parts. Another advantage of a predefined model structure is the capability to combine parts of existing models into one model (Fig. 6).

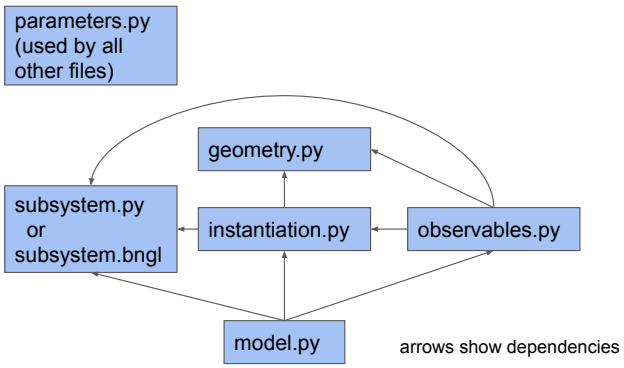

Figure 5: The main files of a base MCell4 model are: 1) parameters.py with all the model parameters, 2) subsystem.py that captures information on species and reactions in a way that this subsystem module is independent of a particular model and can be used as a reusable module, 3) geometry.py with a definition of 3D geometry objects, 4) instantiation.py that usually defines the initial model state, i.e., which geometry objects are created in the simulation and the number and positions of molecules to be released at a given time, 5) observables.py with lists of characteristics to be measured during simulation, and 6) model.py where all the parts of the model are being put together and that allows to define a simulation loop with optional interactions with external simulators. Model.py is the only required file.

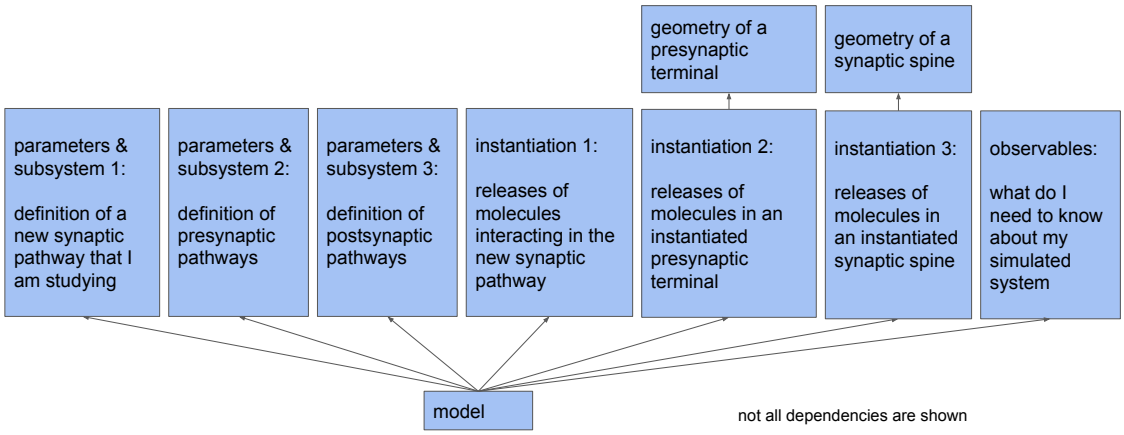

Figure 6: Modularity of the model allows inclusion of multiple subsystem definitions into a single model. In the example shown here, individual modules are used to construct a model of a new synaptic pathway that is affected by other processes. The complete model includes the modules that individually define the presynaptic terminal with its presynaptic pathways and postsynaptic spine with its postsynaptic pathways.

2.3.1 MCell4 Python API Example

A simple model example that shows the MCell4 API with Subsystem, Instantiation, and Model classes is shown in Fig. 7. Because of this example's simplicity, we do not show the division into separate files as it was shown in Fig. 5.

2.4 Graph-Based Approach To Protein Modeling

BNGL [23] allows intuitive modeling of protein complexes by representing them as undirected graphs. Such graphs contain two types of nodes: *elementary molecules* and *components*. Component nodes represent *binding sites* of this protein and can also express *state* of the whole protein or its binding site. A graph representing a *single protein* is an elementary molecule node with component nodes connected to it through *edges*. To form a dimer, two individual components of different proteins are bound by creating an edge between them. A graph with one or more elementary molecules with their components is called a *complex*. A *reaction rule* defines a graph transformation that manipulates the graph of reactants. A reaction rule usually manipulates edges to connect or diconnect complexes or change state of a component. It can also modify the elementary molecules such as in a reaction A + B -> C where we do not care about the reaction details and do not need to model individual binding sites. An example of applying a reaction rule that connects complexes and changes state is shown in Fig. 8. Note that what we call an "elementary molecule type" here is called a "molecule type" in BioNetGen. In MCell, "molecules" are whole molecules such as protein complexes that act as individual agents in the simulation. For better clarity, we adopt the name "elementary molecule" for the base building blocks of complexes. The tool SpringSaLaD [9] uses the same distinction.

This graph-based approach is essential when dealing with combinatorial complexity. To make an ordinary differential equation (ODE) model of a protein with 10 sites where each can be unphosphorylated, phosphorylated or bound to another protein requires 3^{10} ODEs [24]. Compared to this, a BNGL model of the same protein will have just six reversible reaction rules (assuming no interaction between these 10 sites). Such a model can then be simulated using network-free simulation methods [25].

2.4.1 Extension of BNGL for Volume-Surface Reactions

BNGL compartments [26] allow defining hierarchical volumes or surfaces where simulated molecules are located. To model a transport reaction that moves a volume molecule from one compartment through a channel (located in a membrane) into another volume compartment, one must specify the direction of this transport. We will show such a reaction using hierarchy of compartments in Fig. 9.

In BNGL, a reaction that defines the transport of A from compartment EC into CP using transporter T is represented with the following rule:

```
A@EC + T@PM -> A@CP + T@PM
```

MCell4 Python API

```
import mcell as m

subsystem = m.Subsystem()
a = m.Species(
    name = 'a',                    # this species will be called 'a',
    diffusion_constant_3d = 1e-6   # molecules of 'a' are volume
                                   # molecules and diffuse in 3D space
)
subsystem.add_species(a)

instantiation = m.Instantiation()
# ReleaseSite defines which and how many molecules will be released
# either when simulation starts (default) or at a predefined time
rel_a = m.ReleaseSite(
    name = 'rel_a',
    complex = a,             # molecules of which species to release
    number_to_release = 10,  # copy number
    location = (0, 0, 0)     # all these molecules will be released
                             # at (x, y, z) location (0, 0, 0)
)
instantiation.add_release_site(rel_a)

model = m.Model()
model.add_subsystem(subsystem)         # include information on species
model.add_instantiation(instantiation) # include molecule release site

model.initialize()                     # initialize simulation state
model.run_iterations(10)               # simulate 10 iterations
model.end_simulation()                 # final simulation step
```

Figure 7: Example of a simple MCell4 model that releases 10 volume molecules of species 'a' and simulates their diffusion for 10 iterations with a default time step of 1 µs. Please note that for this and following examples, a system variable PYTHONPATH must be set so that the Python interpreter knows where to find the MCell4 library [22].
Alternatively one can append to python's search path from within the model file with: "import sys; sys.path.append("/path/to/mcell4/libs")".

To model multiple instances of cells or organelles, this definition needs to be replicated with different compartments like this:

```
A@EC + T@PM1 -> A@CP1 + T@PM1
A@EC + T@PM2 -> A@CP2 + T@PM2
...
```

MCell3 uses general a specification of orientations [4] where the rule above is represented as:

```
A' + T' -> A, + T'
```

A' (apostrophe) means that the molecule A hits molecule T from the outside (defined below) of the compartment, T' means that the surface molecule T must be oriented towards the outside. And for products, A, (comma) says to create A on the inside and T' means that T will still be oriented towards the outside. Geometry objects in MCell are composed of triangles and outside is where the normal vector of the triangle points to. More details on the MCell3 molecule orientations can be found in [4].

To avoid the repetition of reaction rules for each compartment and to keep the BNG language consistent (the MCell3 solution is not compatible with the BNGL grammar), we defined an extension to the BNG language

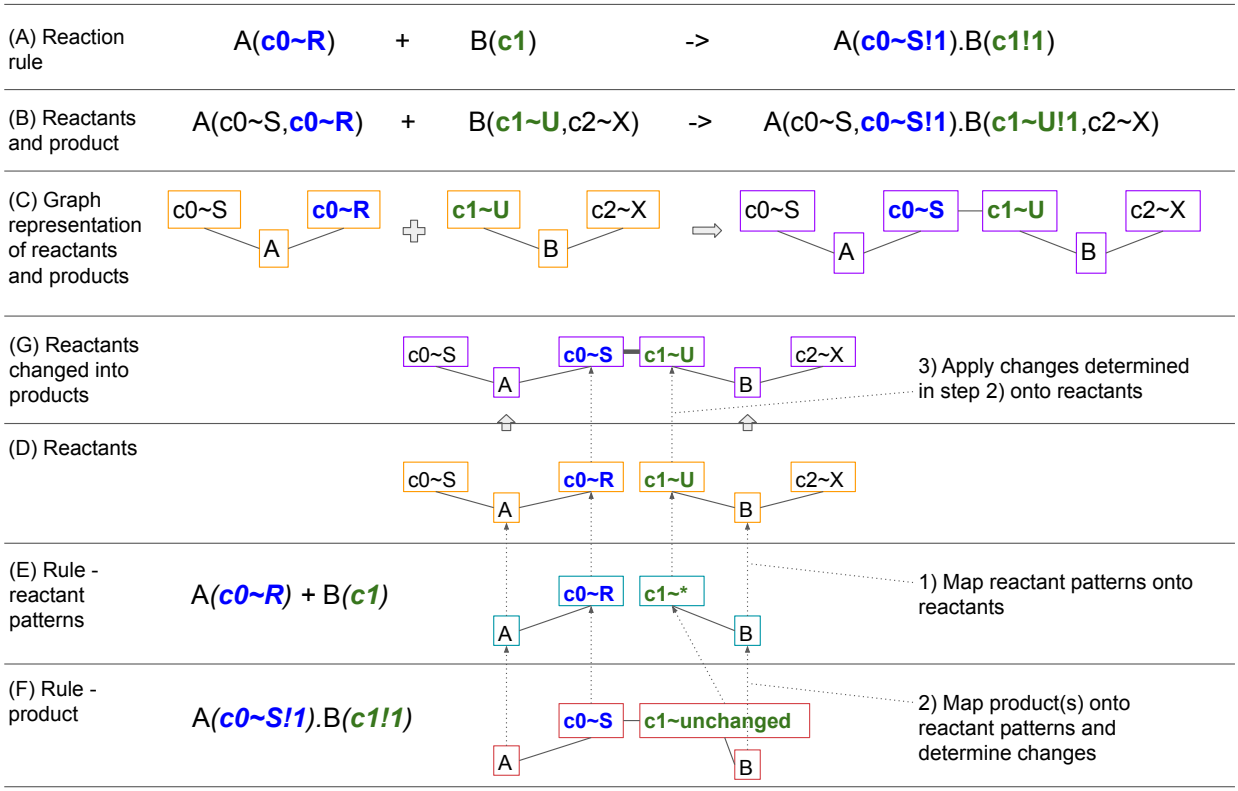

Figure 8: Example of a graph transformation with BNG reaction rules. In this example, reactants are defined with molecule types A(c0∼R∼S,c0∼R∼S) and B(c1∼U∼V,c2∼X∼Y) where A and B are names of the molecule types, c0 is a component of A that can be in one of the states R and S, and similarly c2 and c3 are components of B. (A) is the example reaction rule, (B) are example species reactants and products in the BNGL syntax, and (C) shows a graph representation of the rule in (B).
Application of the rule is done in the following steps: 1) a mapping from each molecule and each component from reactant patterns (E) onto reactants (D) is computed (dotted arrows), if the state of a component is set in the pattern, the corresponding reactant's component state must match. The next step 2) is to compute a mapping of the rule product pattern (F) onto reactant patterns (E). The difference between the reaction rule product pattern and the reactant patterns tells what changes need to be made to generate the product. In this example, a bond between A's component c0 with state R and B's component c1 is created. The state of A's component c0 is changed to S. Once the mappings are computed, we follow the arrows leading from the reaction rule product pattern (F) to reactant patterns (E) and then to reactants (D) and 3) perform changes on the reactants resulting in the product graph (G). Each graph component of the product graph is a separate product and there is exactly one product in this example.

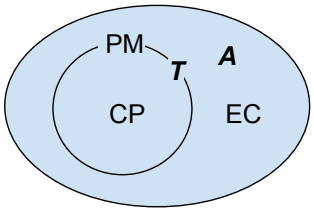

Figure 9: Example of compartments, EC is extracellular space, PM is the plasma membrane, and CP is cytoplasm. A is a molecule that diffuses freely in 3D space, and T is located in a membrane.

that uses compartment classes called @IN and @OUT. The original BNG reaction with specific compartments is then more generally defined as:

```
A@OUT + T -> A@IN + T
```

When this rule is applied to reactants A@EC and T@PM, we know that the compartment OUT for this specific rule usage is EC, and the inside of surface compartment PM is CP, thus IN is CP. We insert this information to the rule A@OUT + T -> A@IN + T and get A@EC + T@PM -> A@CP + T@PM which is the same as the example rule we started with.

One more issue to deal with is the orientation of the transporter in the membrane. One might need to model flippases and floppases (e.g., [27]) that change the orientation of a receptor in a membrane. In MCell3, this is handled by orientations where comma means heading inwards, and apostrophe means outwards. In MCell4, when a molecule is created in a membrane, its orientation is always heading outwards (T' in the MCell3 notation). If one needs to define orientation explicitly, a component of an elementary molecule can be used. One can extend the definition of the molecule type T to contain a component 'o' with two states called INWARDS and OUTWARDS. Our rule constrained to a specific state of the transporter will be then:

```
A@OUT + T(o~OUTWARDS) -> A@IN + T(o~OUTWARDS)
```

To flip the orientation, a standard BNGL rule F + T(o∼OUTWARDS) -> F + T(o∼INWARDS) can be used; here, F is a surface molecule flippase.

To summarize, we introduced an extension to the BNG language where compartment classes @IN and @OUT allow defining general volume+surface molecule reaction rules that can be applied regardless of specific compartments.

2.4.2 Units and Interoperability between MCell4 and BioNetGen

Usage of the BNGL language offers an excellent interchange format since the same model definition can be executed by MCell, BioNetGen that implements various simulation approaches such as ODE, SSA, PLA, and NFSim, and other tools. However, BioNetGen does not have pre-described units and the user is free to use any unit system they deem suitable and are compatible with the underlying algorithms. To allow model interchange, we define a set of units to be used when BNGL models are used by MCell4 and when the model is exported for BioNetGen as shown in Table 1.

Simulation tool and mode of usage	**Volume-volume or volume-surface bi-molecular reaction rate**	**Surface-surface bimolecular reaction rate**	**Unimolecular reaction rate**	**Compartment volume**	**Seed species (initial molecule release) value**
MCell4 with default units	$M^{-1}\,s^{-1}$	$\mu m^2\,N^{-1}s^{-1}$	s^{-1}	μm^3	N
MCell with BNG units; BioNetGen ODE, SSA, PLA	$\mu m^3\,N^{-1}s^{-1}$	$\mu m^3 N^{-1}s^{-1}$	s^{-1}	μm^3	N
BioNetGen NFSim	$N^{-1}s^{-1}$	$N^{-1}s^{-1}$	s^{-1}	ignored	N

Table 1: Units used by MCell and suggested units for BioNetGen. Unit N represents the number of molecules, M is molar concentration. BioNetGen interprets membranes (2D compartments) as thin volumes of thickness 10 nm. BioNetGen NFSim does not fully support compartmental BNGL yet and the volume of the compartment must be incorporated into the rate units of the reactions occurring in that compartment, therefore the NFSim's bimolecular reaction rate unit does not contain a volumetric component. Aditional units that MCell uses are: distances are in µm and diffusion constants are in $cm^2\,s^{-1}$.

An MCell model is defined by a combination of Python and BNGL code. Although the recommended approach is to capture all the reaction rules and initial molecule releases using BNGL, it might be beneficial to use Python code for these definitions as well (e.g., to generate reaction networks programmatically). There are also spatial model aspects that cannot be captured by BNGL. To simplify model validation, MCell4 provides automated means to export the model defined both by Python and by BNGL into pure BNGL.

A best-effort approach is used during this export. All model features that can be exported into BNGL are exported and error messages are printed for the model aspects that have no equivalent in BNGL. If no essential model aspects were skipped, the exported model can be used for validation of MCell simulation results. Verifying results using multiple tools can show errors in the model or in the simulation tools and such validation is a recommended step in an MCell model development.

2.4.3 Example of an MCell4 Model with BioNetGen Specification

To demonstrate the support for BNGL in MCell4, we show a simple example (Fig. 11) that loads information on species and reaction rules, molecule releases, and compartment information from a BNGL file (Fig. 10).

BNGL

```
begin parameters
  # provide diffusion constant for used molecule species
  MCELL_DIFFUSION_CONSTANT_3D_A 1.0e-6
  MCELL_DIFFUSION_CONSTANT_3D_B 2.0e-6
  MCELL_DIFFUSION_CONSTANT_3D_C 1.3e-6
end parameters

begin compartments
  # 3D (volume) compartment with volume 1um^3
  CP 3 1
end compartments

begin seed species
  # release 100 molecules of A and 100 of B in compartment CP
  A@CP 100
  B@CP 100
end seed species

begin reaction rules
  # a simple rule telling that when A and B react, C is the product
  # rate is assumed to be in um^3*1/N*1/s
  A + B -> C 100
end reaction rules
```

Figure 10: BNGL file that defines a compartment CP, tells to release 100 of A's and B's into it, and when A and B react, C is the product.

One aspect deserving a mention is that the file in Fig. 10 is a standard BNGL file that can be used directly by other tools such as BioNetGen so no extra conversion steps are needed for the BNGL file to be used elsewhere. This allows for fast validation of the modeled reaction network with BioNetGen's ODE or other solvers and checking it against spatial simulation results in MCell without the need to have multiple representations of the same model.

3 Results

3.1 Testing & Validation

We performed extensive testing and validation to make sure MCell4 provides correct results. As the main references were the previous versions MCell3 [4] and MCell3-R [12] used. One can obtain identical results for MCell3/MCell3-R and MCell4 by using specific compilation options. These options ensure that the molecules are simulated in the same order. A testsuite containing more than 350 MCell3 and MCell3-R tests that verify correct results was created. Simulation results were also validated against BioNetGen with ODE solver [6] and with NFSim [14] by running equivalent models with MCell4 and with BioNetGen with up to 1024 different random seeds. The diffusion constants for MCell4 were set to a high value to emulate a well-mixed solution. The averages of last iteration counts of molecules of given species were then compared.

MCell4 Python API

```
import mcell as m

model = m.Model()

# specify that this model uses BioNetGen units (see Table 1)
model.config.use_bng_units = True

# load the information on species (diffusion constants),
# reaction rules, also creates compartment CP as a box with
# volume 1um^3 and creates release sites for molecules A and B
model.load_bngl('sybsystem.bngl')

model.initialize()                      # initialize simulation state
model.run_iterations(10)                # simulate 10 iterations
model.end_simulation()                  # final simulation step
```

Figure 11: MCell4 Python code that demonstrates loading of the BNGL file from Fig.10 (referenced as subsystem.bngl). Here, we are loading the whole BNGL file. It is also possible to load only specific parts of the BNGL file such as only reaction rules or compartment and molecule release information and also to replace BNGL compartments with actual 3D geometry.

More than 45 of such tests are a part of the MCell4 testsuite [28]. Some of these tests also serve as examples are are referenced from MCell4's API reference manual [29].

3.1.1 SNARE Complex

We implemented a cooperative dual Ca^{2+} sensor model for neurotransmitter release, the SNARE complex [30] as another example of an MCell4 implementation with BioNetGen Specification. The model accounts for the binding of up to five calcium ions to the sensor and the synchronous or asynchronous release of neurotransmitters. An adapted version of the model was previously implemented in an older version of MCell [31]. The model is composed of 18 state variables, calcium ions and 63 reactions. There are different possible implementations of the model in BNGL language. The one presented here is compatible with MCell4, and allows to simulate the model in BioNetGen and MCell4 without modifying the code. It consists of three molecules types and ten reaction rules (Fig. 12). The snare complex (represented as **snare**) is an elementary molecule that has eight components: five **s**, that represent the binding site for calcium molecules in the synchronous sensor; two **a** components that represent the binding sites for calcium in the asynchronous sensor; and one component called **dv** with two states ($\sim 0 \sim 1$), that represents a docked vesicle to the snare complex (~ 1) or its absence (~ 0). Additionally there are calcium ions (Ca^{2+}) that bind and unbind to the complex. The release of neurotransmitters is emulated with the variable V_release(), which captures the timing of the release but does not release neurotransmitters. Fig 13A shows the states of the model, and the synchronous and asynchronous release. Assuming well-mixed conditions, a large volume containing the surface complexes, a large number of complexes and a constant calcium concentration, the results obtained with ODE BioNetGen simulations and the spatial model in MCell4 give qualitative similar results (Fig 13B). The source code for this example can be found [32].

3.1.2 Event-Driven Release of Neurotransmitter by SNARE Complex

To actually release neurotransmitter in an event-driven manner at the times captured by the variable V_release() in the SNARE example above, we employ a new feature in MCell4: callbacks. One of the most powerful new features of MCell4 is the ability to specify python code that is executed (i.e. called) each time a user-specified reaction or wall collision event occurs during a simulation, thus the term "callback". In the MCell4 Python API the code to be executed, driven by the event, is expressed as a function and this function is referred to as a "callback function".

BNGL

```
begin compartments
  # Plasma membrane (PM) 2D compartment with volume 0.01 um x SA um^2
  PM 2 6e-4
  # Cytoplasm (CP) 3D volume compartment with volume 1e-3um^3
  CP 3 1e-3 PM
end compartments

begin molecule types
  snare(s~0~1~2~3~4~5,a~0~1~2,dv~0~1)
  Ca
  V_release()
end molecule types

begin species
  # SNARE complex are released in the PM
  snare(s~0,a~0,dv~1)@PM 70
  # Fixed calcium number in the cytosol
  Ca@CP Ca0
end species

begin observables
  Molecules SNARE_sync snare(s~5)
  Molecules SNARE_async snare(a~2)
  Molecules V_release V_release()
end observables

begin reaction rules
  # Calcium binding to the synchronous component of the sensor
  snare(s~0)@PM + Ca@CP <-> snare(s~1)@PM 5*ksp, 1*b^0*ksm
  snare(s~1)@PM + Ca@CP <-> snare(s~2)@PM 4*ksp, 2*b^1*ksm
  snare(s~2)@PM + Ca@CP <-> snare(s~3)@PM 3*ksp, 3*b^2*ksm
  snare(s~3)@PM + Ca@CP <-> snare(s~4)@PM 2*ksp, 4*b^3*ksm
  snare(s~4)@PM + Ca@CP <-> snare(s~5)@PM 1*ksp, 5*b^4*ksm

  # Calcium binding to asynchronous component of the sensor
  snare(a~0)@PM + Ca@CP <-> snare(a~1)@PM 2*kap, 1*b^0*kam
  snare(a~1)@PM + Ca@CP <-> snare(a~2)@PM 1*kap, 2*b^1*kam

  # Synchronous vesicle release
  sync: snare(s~5,dv~1)@PM -> snare(s~5,dv~0)@PM + V_release()@CP gamma
  # Asynchronous vesicle release
  async: snare(dv~1,a~2)@PM -> snare(dv~0,a~2)@PM + V_release()@CP a*gamma
  # Vesicle docking to SNARE
  snare(dv~0) -> snare(dv~1) k_dock
 end reaction rules

end model
```

Figure 12: BNGL compartmental code implementation of the snare complex model. One 3D compartment is defined cytosol (CP) and its associated plasma membrane (PM). Molecule types are defined, and their released sites specified: snare molecules are released in the PM, and Calcium ions in the Cytosol. This is followed by the observables, and the reaction rules governing the interactions.

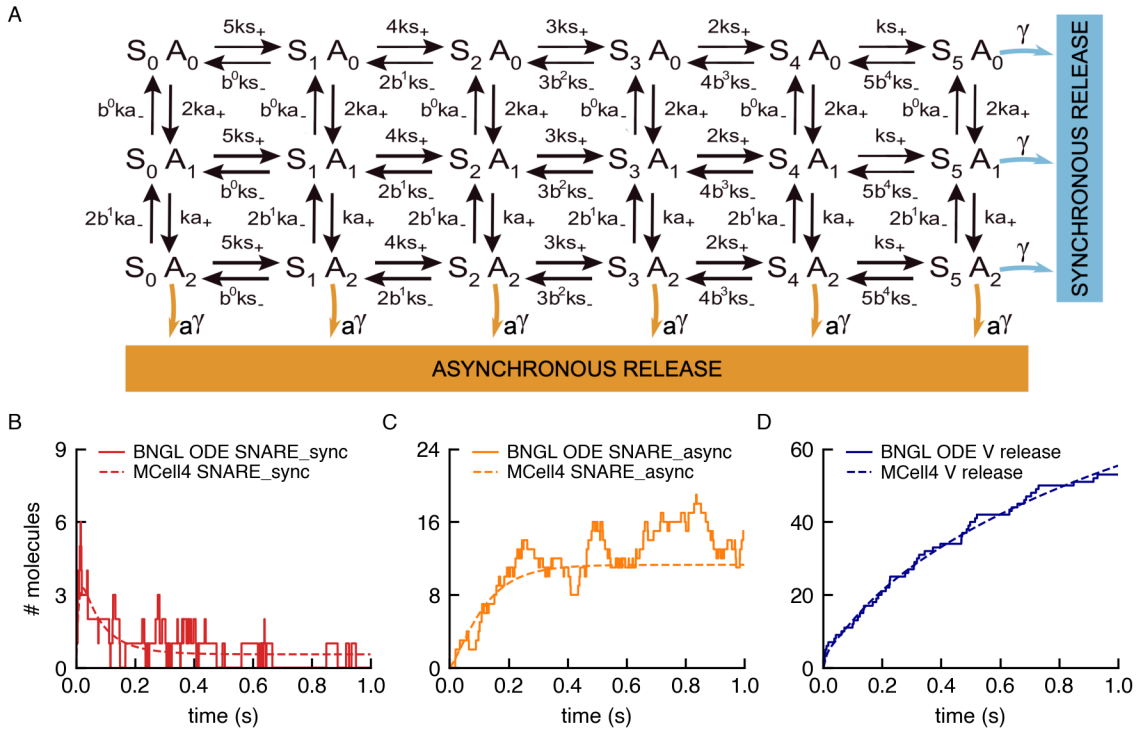

Figure 13: (A) Schematic diagram of the state variables of the SNARE complex model. It consists of 36 states, S and A represent the synchronous and asynchronous components of the complex, which can be in five and two different states respectively (B-D) Independent Simulations of the model with ODEs in BioNetGen and in MCell4.

In the case of the SNARE complex we can create a callback function which will release a given number of neurotransmitter molecules, at the time the synchronous release or asynchronous release reactions happen, and we can localize the release with the individual SNARE complex causing the release. Here we briefly describe how this is accomplished. The full details and Python source code of the working MCell4 model can be found at https://github.com/mcellteam/article_mcell4_1/tree/master/snare_complex/snare_w_callback.

There are two types of callback functions allowed in the MCell4 Python API, namely reaction callback functions and wall hit callback functions. For the SNARE complex example we have created a reaction callback function which will be called upon the stochastic occurrence of the "sync" or "async" labeled reactions specified in Fig. 12. We name our reaction callback function "release_event_callback". We associate our callback function with the labeled reactions using the "register_reaction_callback()" command provided by the API. During simulation of the model, whenever a sync or async reaction occurs, the MCell4 physics kernel will execute "release_event_callback()". But how will our callback function know which species of neurotransmitter to release, how much, and where? Of course we could design our callback function to be inflexible and always release 1000 molecules of glutamate at xyz location (0,0,0). But to accommodate greater flexibility in the design of callback functions the register_reaction_callback() command also allows us to associate additional metadata (called "context") to be passed to the callback function. In our example we have created a Python class called "ReleaseEventCallbackContext" which will contain the name of the species and number to be released as well as the relative release location, and release_event_callback() can then make use of this context to perform the desired operations. See file "customization.py" in the working model for complete details.

3.1.3 CaMKII Model with Large Reaction Network

To demonstrate correct results for systems with large reaction networks, we used a CaMKII dodecamer model which is an extension of a model described in [33].

The protein complex CaMKII dodecamer is composed of two CaMKII hexameric rings stacked on top of each other. Each CaMKII monomer with calmodulin (CaM) binding site can be in one of 18 states. Then the total number of states in which the dodecamer CaMKII complex can be is $18^{12}/12 \approx 10^{12}$ (the division by 12 is to remove symmetric states). This is an example of the combinatorial complexity mentioned in section 2.4 where it is simply not feasible to generate the whole reaction network upfront and a network-free approach must be used. Fig. 14 shows the results of this validation.

Here we also show an extension of the aforementioned model [33], where we can now observe the effects of the geometry of the compartment on the simulation results thanks to MCell4. Figure 15 shows three different variations of the model, where the first variation is the homogeneously distributed molecules in the compartment (equivalent to the well-mixed versioned published previously, Figure 15 A), and two variations with a small subcompartment, located near the top of the larger compartment, which in our model is not transparent to CaMKII and CaM molecules. In the first variation all the molecules are still homogeneously distributed throughout the compartment, but the the CaMKII and CaM molecules in the subcompartment and the rest of the compartment do not mix (Figure 15 B), and in the second variation half of CaMKII molecules are placed in the subcompartment and the other half in the remainder of the compartment, while CaM is still distributed homogeneously throughout the entire volume (Figure 15).

We sought to observe the effect of these three conditions on CaMKII phosphorylation as a result of Ca^{2+} influx into the compartment. In all three conditions a Ca^{2+} influx is simulated from a single point source located in the center of the top face of the compartment. As in [33] the Ca^{2+} influx was such that at the peak the free calcium concentration was $\sim 10\mu M$, and fall back to nearly steady state levels within 100. As we can see these spatial differences have a small but significant effect on CaMKII phosphorylation levels in response Ca^{2+} influx. These differences would have been impossible to investigate without the combination of the network-free simulations and the diffusion in space accomplished in MCell4.

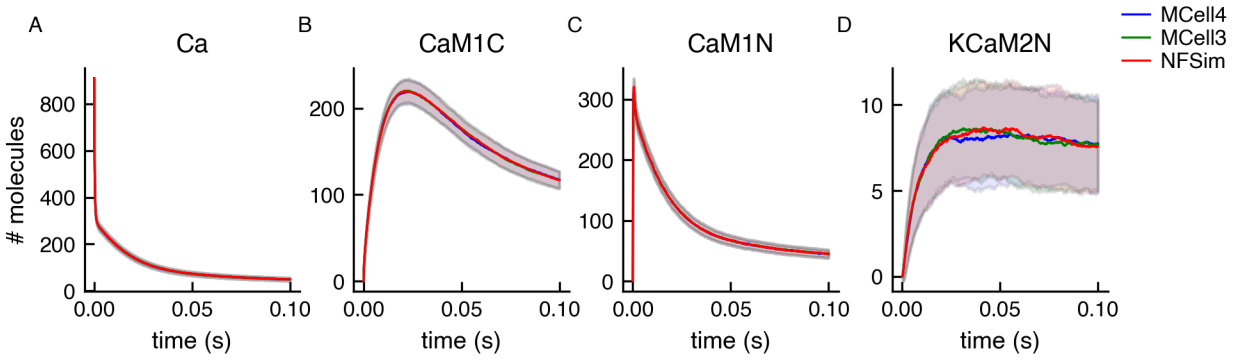

Figure 14: Validation of MCell4 simulation against BioNetGen NFSim (BNG) and MCell3R using CaMKII model. The input BNGL model for NFSim was obtained by automatic BNGL export from the MCell4 model. The simulation ran for 100000 iterations (0.1 s), lines in the graphs show averages from 256 runs with different random seeds, and bands represent one standard deviation. Molecules in MCell3R and MCell4 use diffusion constant $10^{-3} cm^2/s$ to emulate a well-mixed solution (the usual value is around $10^{-6} cm^2/s$). Graph titles represents these BNGL observable patterns: CaM1C – CaM(C$\sim$ 1, $N\sim$ 0,camkii), CaM1N – CaM(C $\sim$ 0,N $\sim$ 1,camkii), KCaM2N – CaMKII(Y286 $\sim$ U,cam!1).CaM(C $\sim$ 0,N $\sim$ 2,camkii!1). The model starts far from equilibrium; therefore there is an initial jump in the Ca and CaM(C $\sim$ 0,N $\sim$ 1,camkii) concentrations. The molecule names are explained in [33].

3.1.4 Volume-Surface and Surface-Surface Reactions: Membrane Localization Model

We used a membrane localization model from [34] (section 2A) to validate volume-surface and surface-surface reactions.

The model analyzes how membrane localization stabilizes protein-protein interactions: a pair of protein binding partners A and B that can also localize to the membrane surface by binding a lipid M and this binding to a membrane constrains the space where the molecules diffuse and thus promotes complex formation. The model uses a simulation box of dimensions $0.47 \times 0.47 \times 5\mu m^3$. Surface molecules M are released on one of the smaller sides of this box, and edges of this bottom side are set to be reflective, so the surface molecules cannot diffuse to other sides.

MCell divides surface areas into tiles. A maximum of one molecule can occupy one tile at a time - this tiling models volume exclusion for surface molecules. A parameter surface grid density sets the density of the tiles. The initial density of surface molecules in the model is 17000 $molecules/\mu m^2$, and we set the surface grid density to 40000 $tiles/\mu m^2$.

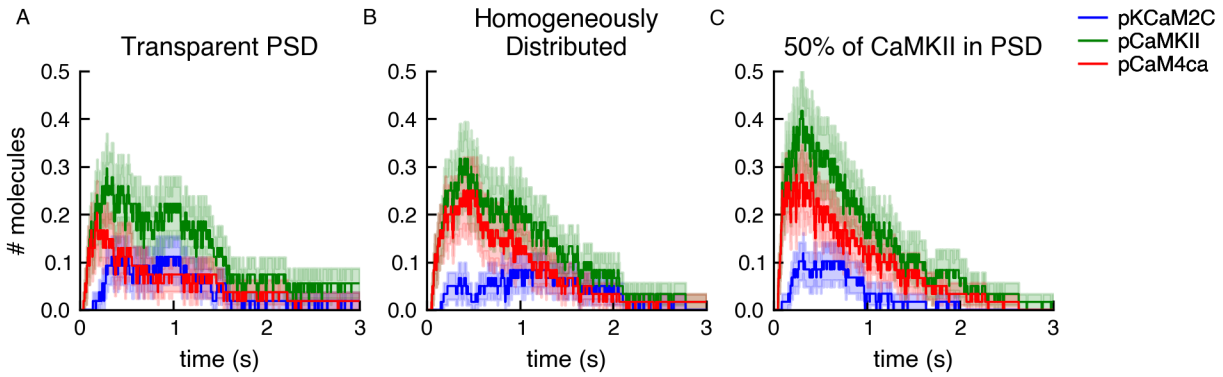

Figure 15: The effect of trapping CaMKII and CaM inside a subcompartment on CaMKII phosphorylation. Three different conditions where simulated. (A) All molecules are homogeneously distributed throughout the whole compartment. (B) With a small subcompartment, near the top of the larger compartment which is reflective to CaMKII and CaM, but is transparent to calcium ions and PP1. All the molecules are homogeneously distributed throughout both compartments. (C) The subcompartment is reflective to CaMKII and CaM and 50% of the CaMKII molecules are trapped inside the subcompartment, and the rest of the molecules are distributed homogeneously throughout the remainder of the larger compartment. The plots show an average of 60 runs, bands represent standard error of the mean.

3.1.5 Stochastic Fluctuations in a System with Multiple Steady States: Autophosphorylation

Another validation model from [34] (section 2B) shows stochastic fluctuations in a system with multiple steady states. A deterministic ODE solution does not show these multiple steady states and almost immediately stabilizes in one of them. In Fig. 17 we show the output of MCell and automatically exported BNGL model (more details on BNGL export are in 2.4.2) simulated with NFSim along with steady states shown with ODE solutions.

3.2 Performance

With relatively small reaction networks (less than 100 or so reactions), the performance of MCell4 is similar to MCell3 as shown in Fig. 18 (A). MCell3 was already heavily optimized, so there were not many opportunities for further optimizations. MCell3 contains optimization of cache performance that speeds up models with large geometries, and this optimization is not present in MCell4. This is why MCell3 is faster than MCell4 for the Neuropil (full model) with about 4 million triangles that define its geometry. The situation is different when comparing MCell4 and MCell3-R with models that use large BNGL-defined reaction networks 18 (B). MCell3-R uses NFSim as a library to compute reaction products for BNG reactions. With large reaction networks containing possibly 10^{10} of reactions or more, MCell3-R keeps all those computed reactions in

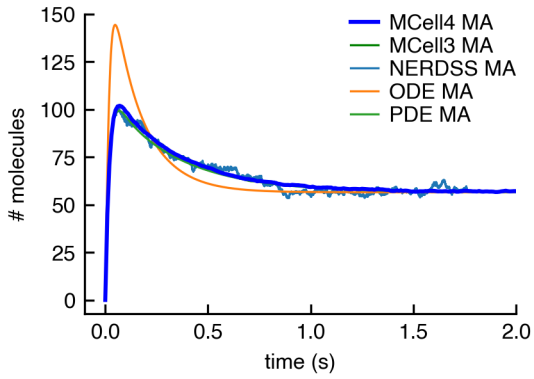

Figure 16: Simulation results for the membrane localization model. The plot shows copy numbers of a surface molecule MA (surface molecule M with a bound volume molecule A). MCell4 and MCell3 results show a good match with the NERDDS simulator (used data from [34], data end at time 1.75 s), and the same equilibrium is reached as computed with ODE and PDE solutions produced by VCell (used VCell models from [34] and simulated with VCell 7.2.0.39). MCell3 and MCell4 results are an average of 512 runs with different random seeds.

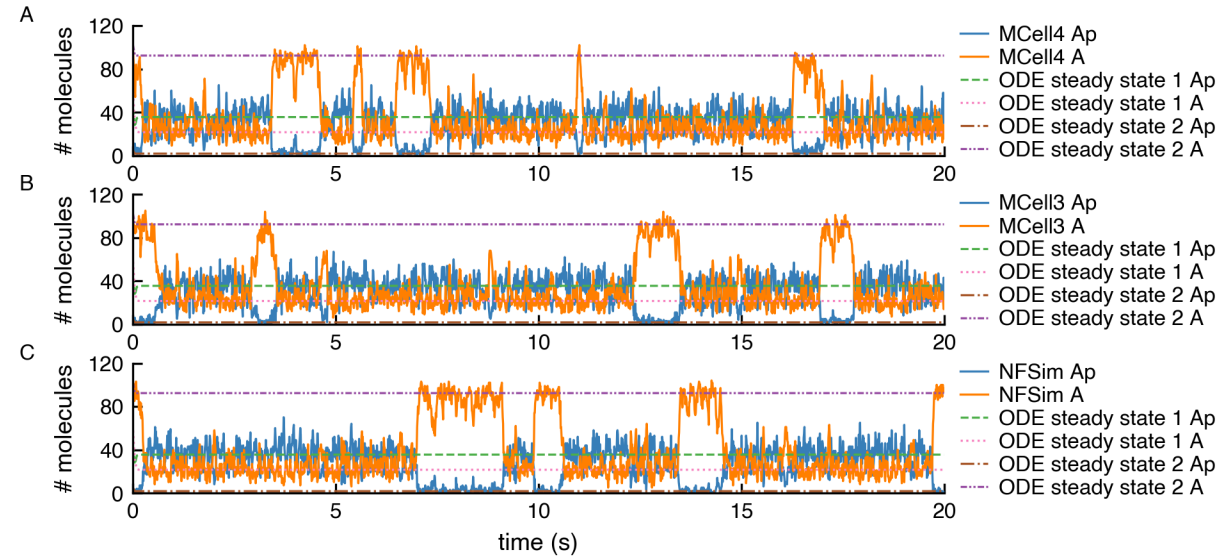

Figure 17: Stochastic simulation allows to model systems that exhibit switching between multiple steady states. Copy numbers of unphosphorylated kinase A and its phosphorylated variant Ap are shown for a single simulation run of MCell4, MCell3, and NFSim. The NFSim model was obtained by automatically exporting the MCell4 model into BNGL. The graphs also show solutions obtained with a deterministic ODE model for which data from [34] were used. Results demonstrate that the MCell results correctly reach one of the stable steady states for which copy numbers are shown with the ODE result. The simulation stays in such a state, and then due to stochastic behavior, another state switch occurs.

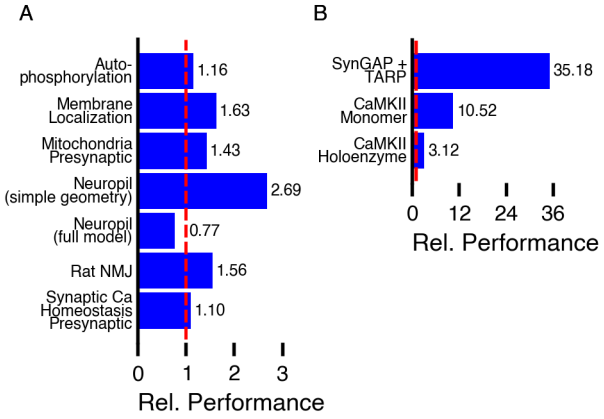

Figure 18: For selected benchmarks, we measured elapsed time for how long the simulation ran starting from the second iteration (after all initializations) and ending when the simulation finished. Time was measured on AMD Ryzen 9 3900X@3.8GHz. Both MCell3 and MCell4 use a single execution thread. Relative performance shown in the graphs is computed as time for MCell3 or MCell3-R divided by time for MCell4. The sources of the models are as follows: Presynaptic Ca Homeostasis [31]; Rat Neuromuscular Junction [2] model with updated geometry (shown in Fig 1), Neuropil [5]; Mitochondrion Model [35]; Membrane Localization [34]; Autophosphorylation [34]; CaMKII Monomers [33]; CaMKII Holoenzyme [33]; SynGAP with TARP (not yet published).

memory and gradually slows down due to this. Such reaction networks can be easily defined in BNGL, e.g., for polymerization used in the SynGAP with TARP model. We tried to implement reaction cache cleanup in MCell3-R, but it has shown to be very difficult, and this attempt was abandoned. MCell4 with BNG library keeps track of how many molecules of each species are there in the simulated system and periodically removes from cache reactions and species that are not used. This allows simulating very complex reaction networks with a potentially infinite number of species and reactions without excessive memory and performance impact.

3.3 Hybrid Simulation Example

The MCell4's Python API allows interacting with a running MCell4 simulation. A demonstration that will be shown in this section models one selected molecular species in Python code using a differential equation and the remaining species in MCell as particles. As a basis for this demonstration of hybrid simulation, we used a circadian clock model also published in article [34], originally based on article [36].

The model simulates the behavior of an activator protein A and repressor protein R that are produced from mRNA transcribed from a single copy of a gene (one for each protein). Coupling of A and R expression is driven by positive feedback of the activator A, which binds to each gene's promoters to enhance transcription. Protein R also binds to A to effectively degrade it, and all proteins and mRNA are also degraded spontaneously at a constant rate.

Compared to the original model in [36], authors of [34] speeded-up the reaction rates of the model from hours to seconds by multiplying the reaction rates by 3600. Since the purpose of this example is the demonstration of a hybrid model in MCell4 and its validation that requires many runs, we made another change that accelerates the simulation, namely we reduced the simulation volume to 0.25 µm by a factor of 268 and multiplied unimolecular reaction rates by the same factor. The kinetics of bimolecular reactions is speeded-up by the same rate just by reducing the volume.

In the hybrid model, protein R is simulated as a concentration (using well-mixed approach), the other species are simulated as particles. We used an MCell4 model as a basis. In this base model, there are several reactions that consume or produce R (Fig. 19). These needed to be replaced with reactions that do not model R as a particle (Fig. 20). The main simulation loop pseudo-code is shown in in Fig. 21.

BNGL Reactions

```
A_and_R_to_AR:             A + R -> AR          AR_kon       # 1/M*1/s
R_to_0:                    R -> 0               R_koff       # 1/s
mRNA_R_to_mRNA_R_plus_R:   mRNA_R -> mRNA_R + R mRNA_R_koff  # 1/s
AR_to_R:                   AR -> R              AR_koff      # 1/s
```

Figure 19: Reaction rules affecting protein R in particle-only model.

BNGL Reactions

```
A_to_AR:                   A -> AR              A_koff       # 1/s
# R_to_0:                  - modeled as ODE
# mRNA_R_to_mRNA_R_plus_R: - modeled as ODE
AR_to_0:                   AR -> 0              AR_koff      # 1/s
```

Figure 20: Reaction rules affecting protein R in hybrid model.

To validate that the results of the hybrid variant are correct, we ran 1024 instances of stochastic simulations with different initial random seeds. We also compared the effect of two different diffusion constant values when using MCell. Results that show average oscillation frequencies are shown in Fig. 22 and copy numbers of molecules A and R in Fig. 23.

When using a fast diffusion constant of $10^{-7} cm^2/s$ for all molecules, all simulation approaches produce practically the same results. A significant advantage of using hybrid modeling is that for this specific example, the MCell hybrid model's simulation speed is 4x faster. This is thanks to: 1) Allowing 5x longer time step by not having to model the fastest reactions for particle-based molecules. The time step for the particle-only model has to be $10^{-7}s$ to precisely model these fast reactions. 2) Not having to model species R as particles.

This is a relatively simple example where we compute the ODE separately with Python code, but it shows the strength of this approach where one can couple other engines to MCell4 and achieve multi-scale simulations.

4 Conclusions

4.1 Summary

We have described MCell4, a particle-based reaction-diffusion tool based on Monte Carlo algorithms that allows spatially realistic simulation of volume and surface molecules in a detailed 3D geometry. MCell4

MCell4 Pseudo-code

```
num_R = 0.0               # in N, initial copy number of Rs,
                          # modeled as a floating-point value

T_STEP = 5e-7             # in us, simulation time step
NA = 6.0221409e+23        # in N/mol, Avogardo's constant
VOLUME = 4.188993 * 1e-15 # in l, simulated volume

for i in range(ITERATIONS):
    # 1) Run particle-based simulation for 1 time step
    model.run_iterations(1)

    # 2)   Update the concentration-based copy number of Rs
    # 2.1) Rs consumed by original reaction A + R -> AR
    dR_due_A_to_AR =
        -model.get_number_of_reactions_in_last_iteration('A_to_AR')

    # 2.2) Rs consumed by original reaction R -> 0
    dR_due_R_to_0  =
        -(num_R * R_koff * TIME_STEP)

    # 2.3) Rs produced by original reaction mRNA_R -> mRNA_R + R
    dR_due_mRNA_R  =
        model.get_number_of_molecules('mRNA_R') * mRNA_R_koff * T_STEP

    # 2.4) Rs produced by original reaction AR -> R
    dR_due_AR_to_0 =
        model.get_number_of_reactions_in_last_iteration('AR_to_0')

    # 2.5) Update the copy number of Rs
    num_R +=
        dR_due_A_to_AR + dR_due_R_to_0 + dR_due_mRNA_R + dR_due_AR_to_0

    # 3) Update rate of reaction A -> AR (originally A + R -> AR):
    # Sets the rate A_koff using concentration of R effectively
    # converting a bimolecular reaction rate from 1/M*1/s to a
    # unimolecular rate in 1/s.
    # Concentration is here computed with copy number of Rs
    # truncated to the closest integer to avoid reactions happening
    # when there is less than 1.0 Rs.
    concentration_R = floor(numR) / NA / VOLUME   # in 1/M
    model.set_reaction_rate('A_to_AR', concentration_R * AR_kon)
```

Figure 21: Pseudo-code of the main simulation loop that: 1) runs an iteration of particle-based simulation, 2) updates the copy number of R based on the current MCell state, and 3) updates rate of reaction A -> AR that was originally a bimolecular reaction A + R -> AR. N is a unit representing the copy number. This code was rewritten to show the actual computations in a more comprehensible way, the actual MCell4 Python code is available in the GitHub repository accompanying this article [32].

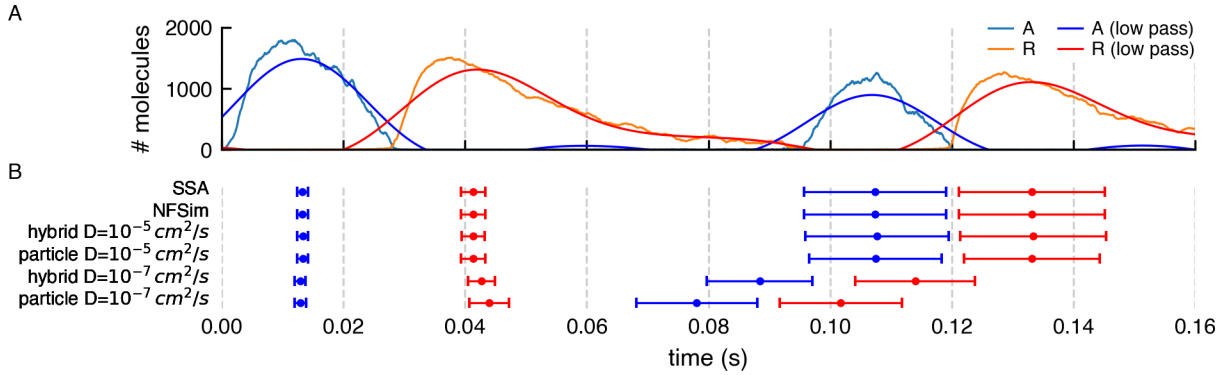

Figure 22: (A) Result of a stochastic simulation of the circadian clock model with NFSim. Copy numbers of molecules A and R show periodic oscillation. A low pass frequency filter was used to smooth the values of A and R. The reason for the smoothing was to get a numerical value related to the actual peak, these peaks from low-pass filtered data do not represent actual average peaks but can be better used as a proxy for the actual time of a peak that can be compared afterwards. (B) The error bars capture the mean and standard deviation of such low pass filtered peak times for different model and simulation algorithm variants. Each of the variants was run 1024 times. One can see that SSA, NFSim, and the hybrid MCell model with a fast diffusion constant $D = 10^{-5} cm^2/s$ variants give practically the same results. Hybrid MCell model with slower $D = 10^{-7} cm^2/s$ shows faster oscillation than the non-spatial models SSA and NFSim, and MCell with fast diffusion constant. The pure particle-based MCell model with $D = 10^{-7} cm^2/s$ shows the fastest oscillations.

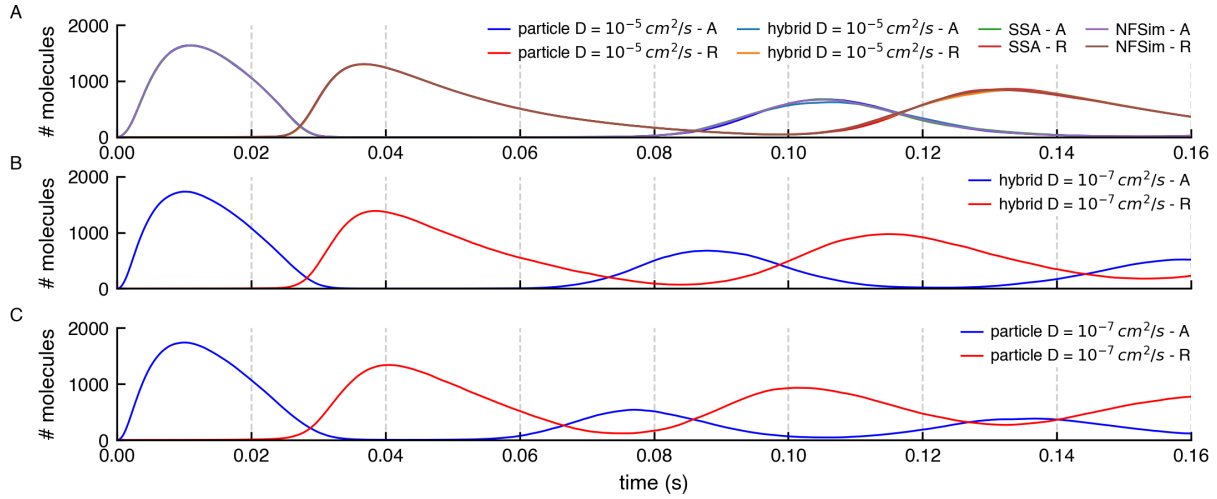

Figure 23: (A) Average copy number values for A and R proteins from 1024 runs for NFSim, SSA, and MCell with fast diffusion constant match each other. To get even better match, one would need to run more than 1024 runs because stochastic molecular simulations show high variability when the copy number of some of the species is low which is true here for both A and R. (B), (C) Average copy numbers for MCell simulations with slow diffusion constant. Shown as separate plots from (A) to highlight the spatial simulation effect when molecules diffuse slowly.

builds on features of MCell3 (and MCell3-R). On top of that, it provides a Python API that permits controlling the simulation through Python code.

With Python API, one can change geometry, reaction rates, create or remove molecules, execute reactions, etc., this all during simulation. This powerful new feature allows construction and execution of multi-scale hybrid models.

Another distinction between MCell3 and MCell4 is that the reactions are now natively written in BioNetGen language. This allows a seamless transition between MCell4 and BNG simulation environments and has dramatically improved the ability to run network free simulations compared to the NFSim engine used previously in MCell3-R [12, 14].

As we have demonstrated here through example models, MCell4 has introduced many new features including the ability to create a fully spatial network-free model with realistic geometry, the ability to go back and forth between MCell4 and BNG environments, and transmembrane or transcellular interactions between surface molecules.

MCell4 is a significant improvement on the previous version in terms of simulation speed, number of features, as well as usability, and allows to simulate new systems that could not be modeled previously.

4.2 Availability and Future Directions

MCell4 is available under the MIT license. For easy usage, a package containing MCell, Blender, Blender plugin CellBlender, and other tools is available along with detailed documentation and tutorial online at [37]. A part of MCell4 is a new C++ library for parsing the BioNetGen language and provides methods to process BioNetGen reactions. This library libBNG is also available under the MIT license [17].

MCell4 does not support the definition of spatially extended complexes that could be useful for instance when modeling the post-synaptic density [5] or actin filament networks [38] where simply replacing these polymers with a single point in space is inadequate. Furthermore, improved volume exclusion will be important. We have plans to combine particle-based simulation with concentration or well-mixed simulation algorithms such as SSA [39] or the finite element method that uses PDEs (partial differential equations), e.g., [40]. Such hybrid modeling will provide means to simulate longer timescales while still being spatially accurate and able to correctly handle cases when the copy number of molecules is low. All these features will be the focus of future developments.

4.3 Acknowledgements

The authors thank Dr. Mary Kennedy for deep insights into of all things CaMKII. We thank Dr. Padmini Rangamani for discussions on boundary conditions and the biophysics of diffusion near membranes. We heartily thank Dr. Markus Dittrich, Dr. Burak Kaynak, Dr. Oliver Ernst, Dr. Rex Kerr, Jacob Czech, Jed Wing, Don Spencer, and Robert Kuczewski whose insights on API design for discrete event simulation have guided development of MCell over the years, laying the foundation for MCell4. And we thank Jorge Aldana for his expert technical support of the computing infrastructure in the Computational Neurobiology Laboratory at Salk. Funding for this research was provided by NIH MMBioS P41-GM103712, NIH CRCNS R01-MH115556, NIH CRCNS R01-MH129066, NSF NeuroNex DBI-1707356, and NSF NeuroNex DBI-2014862.

References

[1] Jeremy Gunawardena. Models in biology:'accurate descriptions of our pathetic thinking'. *BMC biology*, 12(1):1–11, 2014.

[2] Joel R Stiles, D Van Helden, Thomas M Bartol, and M M Salpeter. Miniature endplate current rise times $< 100\ \mu$s from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proc. Natl. Acad. Sci.*, 93(12):5747–5752, 1996.

[3] Joel R Stiles and Thomas M Bartol. *Monte Carlo Methods for Simulating Realistic Synaptic Microphysiology Using MCell*, chapter 4. CRC Press, 2001.

[4] Rex A Kerr, Thomas M Bartol, Boris Kaminsky, Markus Dittrich, Jen-Chien Jack Chang, Scott B Baden, Terrence J Sejnowski, and Joel R Stiles. Fast monte carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces. *SIAM journal on scientific computing*, 30(6):3126–3149, 2008.

[5] Thomas M Bartol, Daniel X Keller, Justin P Kinney, Chandrajit L Bajaj, Kristen M Harris, Terrence J Sejnowski, and Mary B Kennedy. Computational reconstitution of spine calcium transients from individual proteins. *Frontiers in synaptic neuroscience*, 7:17, 2015.

[6] Leonard A Harris, Justin S Hogg, José-Juan Tapia, John AP Sekar, Sanjana Gupta, Ilya Korsunsky, Arshi Arora, Dipak Barua, Robert P Sheehan, and James R Faeder. Bionetgen 2.2: advances in rule-based modeling. *Bioinformatics*, 32(21):3366–3368, 2016.

[7] Steven S Andrews. Smoldyn: particle-based simulation with rule-based modeling, improved molecular interaction and a library interface. *Bioinformatics*, 33(5):710–717, 2017.

[8] Thomas R Sokolowski, Joris Paijmans, Laurens Bossen, Thomas Miedema, Martijn Wehrens, Nils B Becker, Kazunari Kaizu, Koichi Takahashi, Marileen Dogterom, and Pieter Rein Ten Wolde. egfrd in all dimensions. *The Journal of chemical physics*, 150(5):054108, 2019.

[9] Bashar Ibrahim, Richard Henze, Gerd Gruenert, Matthew Egbert, Jan Huwald, and Peter Dittrich. Spatial rule-based modeling: a method and its application to the human mitotic kinetochore. *Cells*, 2(3):506–544, 2013.

[10] Moritz Hoffmann, Christoph Fröhner, and Frank Noé. Readdy 2: Fast and flexible software framework for interacting-particle reaction dynamics. *PLoS computational biology*, 15(2):e1006830, 2019.

[11] Steven S Andrews. Particle-based stochastic simulators. *Encyclopedia of Computational Neuroscience*, 10:978–1, 2018.

[12] Jose-Juan Tapia, Ali Sinan Saglam, Jacob Czech, Robert Kuczewski, Thomas M Bartol, Terrence J Sejnowski, and James R Faeder. MCell-R: A particle-resolution network-free spatial modeling framework. *Methods in molecular biology (Clifton, NJ)*, 1945:203, 2019.

[13] Michael L Blinov, James R Faeder, Byron Goldstein, and William S Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

[14] Michael W Sneddon, James R Faeder, and Thierry Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. *Nature methods*, 8(2):177–183, 2011.

[15] Carlos F Lopez, Jeremy L Muhlich, John A Bachman, and Peter K Sorger. Programming biological models in Python using PySB. *Molecular systems biology*, 9(1):646, 2013.

[16] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721. IEEE, 2013.

[17] BioNetGen library on GitHub. `https://github.com/mcellteam/libbng`. Accessed: 2022-05-07.

[18] Blender website. `https://www.blender.org/`. Accessed: 2022-05-07.

[19] CellBlender Tutorials and Examples. `https://mcell.org/tutorials_iframe.html`. Accessed: 2022-05-07.

[20] MCell4 API Generator sources on GitHub. `https://github.com/mcellteam/mcell/tree/master/libmcell/definition`. Accessed: 2022-05-07.

[21] Stewart Robinson, Richard E Nance, Ray J Paul, Michael Pidd, and Simon JE Taylor. Simulation model reuse: definitions, benefits and obstacles. *Simulation modelling practice and theory*, 12(7-8):479–494, 2004.

[22] MCell4 Installation Documentation. https://mcell.org/mcell4_documentation/installation.html#setting-system-variable-mcell-path-and-adding-python-3-9-to-path. Accessed: 2022-05-07.

[23] John AP Sekar and James R Faeder. Rule-based modeling of signal transduction: a primer. *Computational Modeling of Signaling Networks*, pages 139–218, 2012.

[24] Marc R Birtwistle. Analytical reduction of combinatorial complexity arising from multiple protein modification sites. *Journal of The Royal Society Interface*, 12(103):20141215, 2015.

[25] Lily A Chylek, Leonard A Harris, Chang-Shung Tung, James R Faeder, Carlos F Lopez, and William S Hlavacek. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 6(1):13–36, 2014.

[26] Leonard A Harris, Justin S Hogg, and James R Faeder. Compartmental rule-based modeling of biochemical systems. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 908–919. IEEE, 2009.

[27] David L Daleke. Regulation of transbilayer plasma membrane phospholipid asymmetry. *Journal of lipid research*, 44(2):233–242, 2003.

[28] MCell4 testsuite on GitHub. https://github.com/mcellteam/mcell_tests. Accessed: 2022-05-07.

[29] MCell4 Python API Reference. https://mcell.org/mcell4_documentation/generated/api.html. Accessed: 2022-05-07.

[30] Jianyuan Sun, Zhiping P Pang, Dengkui Qin, Abigail T Fahim, Roberto Adachi, and Thomas C Südhof. A dual-ca 2+-sensor model for neurotransmitter release in a central synapse. *Nature*, 450(7170):676–682, 2007.

[31] Suhita Nadkarni, Thomas M Bartol, Terrence J Sejnowski, and Herbert Levine. Modelling vesicular release at hippocampal synapses. *PLoS Comput Biol*, 6(11):e1000983, 2010.

[32] MCell4 GitHub repository with models and data shown in this article. https://github.com/mcellteam/article_mcell4_1. Accessed: 2022-05-07.

[33] Mariam Ordyan, Tom Bartol, Mary Kennedy, Padmini Rangamani, and Terrence Sejnowski. Interactions between calmodulin and neurogranin govern the dynamics of camkii as a leaky integrator. *PLoS computational biology*, 16(7):e1008015, 2020.

[34] Margaret E Johnson, Athena Chen, James R Faeder, Philipp Henning, Ion I Moraru, Martin Meier-Schellersheim, Robert F Murphy, Thorsten Prüstel, Julie A Theriot, and Adelinde M Uhrmacher. Quantifying the roles of space and stochasticity in computer simulations for cell biology and cellular biochemistry. *Molecular Biology of the Cell*, 32(2):186–210, 2021.

[35] Guadalupe C Garcia, Thomas M Bartol, Sébastien Phan, Eric A Bushong, Guy Perkins, Terrence J Sejnowski, Mark H Ellisman, and Alexander Skupin. Mitochondrial morphology provides a mechanism for energy buffering at synapses. *Scientific reports*, 9(1):1–12, 2019.

[36] José MG Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002.

[37] MCell website. www.mcell.org. Accessed: 2022-05-07.

[38] Nicholas M Cronin and Kris A DeMali. Dynamics of the actin cytoskeleton at adhesion complexes. *Biology*, 11(1), 2022.

[39] Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[40] Michael L Blinov, James C Schaff, Dan Vasilescu, Ion I Moraru, Judy E Bloom, and Leslie M Loew. Compartmental and spatial rule-based modeling with virtual cell. *Biophysical Journal*, 113(7):1365–1372, 2017.