A Software Framework for Validating Neuroscience Models

Shailesh Appukuttan, Lungsi Sharma, Pedro Garcia-Rodriguez, Andrew Davison

A Software Framework for Validating Neuroscience Models

Shailesh Appukuttan, Lungsi Sharma, Pedro Garcia-Rodriguez and Andrew P. Davison

February 24, 2022

1 Introduction

The development of computational models and their use as the basis for more complex hypothesis has become a common practice in neuroscience. The Hodgkin-Huxley model introduced over half a century ago played a pivotal role in establishing the significance of mathematical modeling in neuroscience. This revolution has been enabled by the substantial progress in computer technology and computing power. However, because the universe of possible problem is far-ranging, practitioners of modeling often focus on the qualitative simulation of the next more difficult problem class. Achieving quantitative accuracy is usually overlooked.

For a morphological and bio-physically detailed neural model may be used in basic research or medical applications, a previous assessment of its performance is required, including a validation with respect to experimental data. The need for validation tests in brain modelling can be motivated by several factors. The kind of available data and the anatomical structure of a brain region can impose serious constraints to generalization and applicability range of those models. Digitally reconstruction of just an axon/dendrite of a neuron from a microscopic image can take months and may render incomplete/broken morphologies. Neurons are diverse in both morphology, connection patterns and electrical activity and just sparse data may be available. For instance, there are incomplete reconstructions of dendrites for Striatum region (basal ganglia), and sparser data on dendrite's than on soma's electrophysiology or number of synapses for Hippocampus. Blurred borders between layers and more complicated geometry than in Neocortex (no functional columns but functional *lamellae*) introduces more uncertainty when modelling Hippocampus regions.

In computational neuroscience most reported model validations are published alongside the model being developed. Many of these validations do not get reproduced and the model often do not get validated against new experimental data. There have been some signs acknowledging the lack of systematic approach to validating models and as a consequence an unmethodically developed model in terms of the scientific method. Bhattacharya and Chowdhury (Bhattacharya and Chowdhury, 2015) reports a collection of commonalities and distinctions of validation approaches for computational models targeting neurological and psychiatric disorders. Based on the idea of unit-testing in computer science SciUnit is a test-driven framework for formally validating scientific models against data (Omar et al., 2014b; Sarma et al., 2016). FindSim (Viswan et al., 2018) is a framework for optimizing models of neurophysiology and cellular signaling by mapping experimental protocols and fitting the given model parameters based on the simulation results. It runs from command line with eye on executing commands in batch mode for large parallel optimization. The framework lacks systematic tracking. Building upon the basic features of SciUnit this paper will introduce a validation framework and service that emphasizes model testing and tracking the test and its result.

A major shortcoming in computational neuroscience lies in the absence of a well-defined system for the validation and comparison of these models. We are attempting to address the above challenges by developing test suites that automate comparison between different models, and allow tracking the performance of a model over time as it is refined.

1.1 The Problem of Validation.

Due to the amorphous nature of the word validation we define it in a technical context. Validation is defined as the process of assessing the quality of the simulated model within its domain of applicability. This involves estimating the degree or range of accuracy consistent with the intended application of the model (Schlesinger, 1979; Mehta, 1996). The *model* includes the code as well as the conceptual modeling assumptions (Leijnse and Hassanizadeh, 1994; Roache, 1998d). In this paper we are speaking of validating computer models rather than codes. This is a scrupulous but necessary distinction (Tsang, 1991). The task of checking the code comes under the term *verification* (Roache, 1998d). Another distinction between validation and verification is that in principle verification is done for the code and then for specific calculations while the task of validation continues with the extension of the parameter ranges and improved (or new) experiments.

1.1.1 The Difficulty of Comparing with Experiments.

In physical and natural science validation is extremely important but a challenging task. Although experiments play a decisive role in judging the quality of a model uncritical acceptance of experimental values may lead to both false invalidation (negative conclusion) and false validation. Difficulties with experiments include

- Accepting a single experiment as the final word (Marvin, 1995).
- Experimental uncertainty increases with progression to more complex predicted quantities (Aeschliman and Oberkampf, 1998).
- Limitations of experimental data collected from publications.

The limitation of the facility and instrumentation is one of the reasons why the validator should be cautious relying on single experiments or measurement procedures. Whether the validation uses one or more experimental data because experimental techniques can play a critical role in the measurement process their error estimates in addition to those for the measured quantities may help quantify the accuracy of the experimental data (Coleman and Steele, 1995; Coleman et al., 1995; W. G. Steele and Coleman, 1996).

Validation in computational neuroscience is commonly based on experiments from published data. Missing experimental details often lead to poor calculation comparisons (Barber, 1998). In cases where model development and experiments were performed by the same team an external validator often gets access to the experiments as published data. However, the extent to which one may infer from performing the validation is usually limited for data collected from publications. As Roach (Roache, 1998b) puts it

> Almost invariably, critical details are missing from published data, particularly for archival journal publications where discussion is limited in the interest of reducing paper length. It is critically important that the boundary and/or initial conditions assumed by the [model]] be accurately known from the experiment.

1.1.2 Technical and scientific challenges in validating Neuroscience models

Validating increasingly complex models of neural systems with respect to experimental data is both a scientific and a technical challenge. Technical challenges include interfacing validation test definitions with model implementations, tracking model/test versions, and automation of the validation process. Validation can face the lack of automated workflows for model validation, and of standardization in model interfaces (which impedes comparisons between models). There is a need to support a wide range of model/experimental formats *e.g.*, Python/HOC scripts with no standardization, NeuroML, Allen Institute and Blue Brain Project formats. Furthermore, several categories of validations must coexist, *e.g.*, anatomical/structural, morphology and electrical patterns, functional and behavioural. As a general rule, the more complex/detailed is the model, the more tests are needed to validate it.

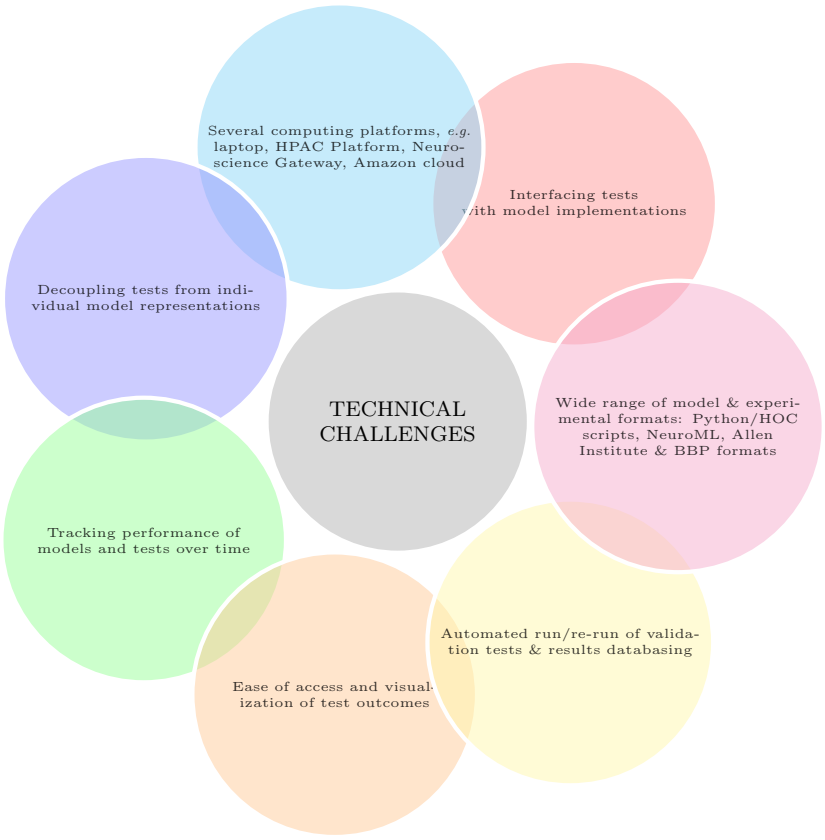

Figure 1: Some technical challenges in Neuroscience validations

So, (1) it should be as easy as possible to run/re-run validation tests; (2) validation tests should be decoupled from individual models representation; (3) record of several tests results (scores) must be available; (4) discoverability should be possible, *i.e.*, finding validation tests that are appropriate for the model; (5) entering runs in the database should be automatic, and may be done on laptop, HPAC Platform, Neuroscience Gateway, Amazon cloud, etc (Figure 1)

On the other hand, model validation implies scientific challenges. Several levels of modelling can be found, *e.g.*, sub-cellular, single neuron, spike-based, network models, population-based network models. Intensive data searching or meta-analysis to generate new ones from existing references is common, reconciling conflicting experimental findings, and deciding on the weighting to be given to different validations, to name but a few. In addition, choosing appropriate statistical tests for quantifying the differences between model predictions and experimental results can be a serious handicap (Figure 2, left).

So far, validation in Neuroscience has most often been done in an *ad hoc* manner, with most studies reporting only qualitative comparisons between simulation results and experimental findings, and different modeling studies using different datasets for validation, which makes comparison of models difficult. For a quantitative validation, a prediction error must be calculated to quantify the model's performance by the match between model prediction and experimental observation. Each model can be assigned a *score*, defined as the relative prediction error respect to the uncertainty level present in the data and in the model (numerical errors of the integration algorithm, inputs uncertainty, etc). The score is a real number which summarizes the behavior of the model with respect to some specific experimental data. A score can also be the statistic of a probability distribution (Figure 2, right).

Unfortunately, other scoring alternatives may be inaccessible in most cases. For instance, the application of traditional statistical hypothesis testing leading to a $p-$value, to judge the relevance of the discrepancy between experiment and model, requires the specification of a probability distribution for the reference data under consideration. However, many studies report too few observations to accurately determine the statistical description needed.

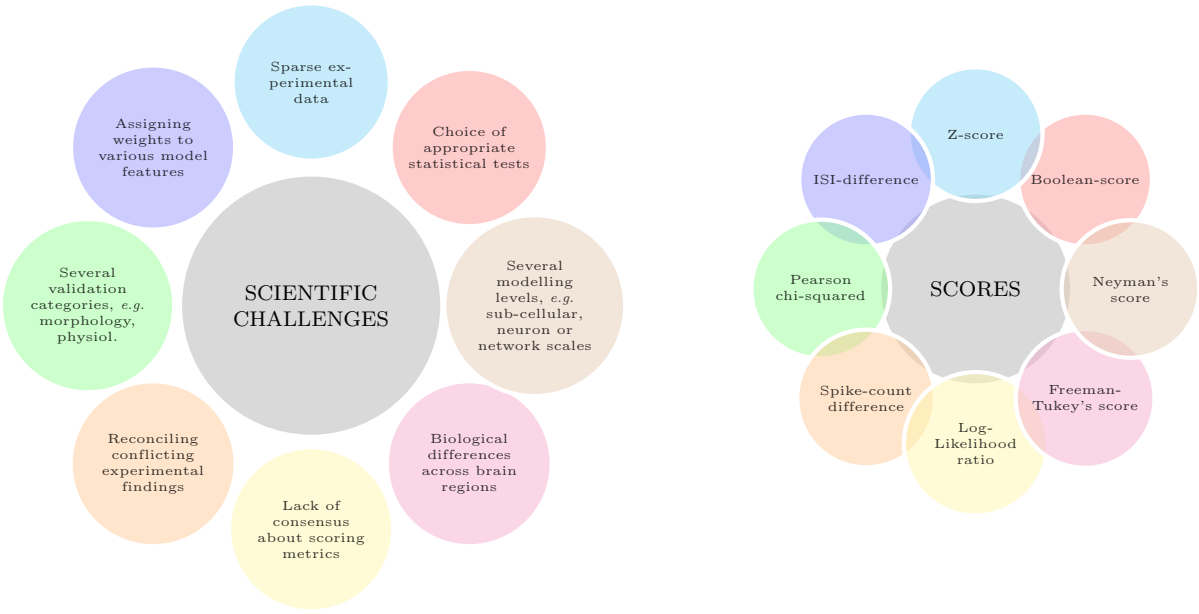

Figure 2: Quantitative validation in Neuroscience. Left: Some scientific challenges. Right: Discrepancy between model prediction and experimental data is quantified by cost- (*aka* fitness- or objective-) functions: scores

1.1.3 What scientific validation of models is not

Validating a model concerns its generalization or predictive power, *i.e.* its ability to replicate observations obtained in experimental scenarios other than those used in constructing the model, *e.g.* a new stimulation protocol. In fact, validation must be considered as a modelling step applied on a previously optimized mathematical model, rather than on a set of equations whose successful performance for a minimal experimental scenario has not been guaranteed yet.

	Validation	Optimization
Compares model prediction and experimental data	✓	✓
Use of error measures (*e.g.* cost-functions) to quantify models's performance	✓	✓
About model's generality, *i.e.* ability to replicate new data	✓	✗
Parameter searching	✗	✓
Implies several simulations	✗	✓
Used anytime that new data/model's version is available	✓	✗
Stops after reaching a desired level of accuracy	✗	✓
Fixed model's parameterizations	✓	✗
Re-use of fitting data	✗	✓

Table 1: Similarities and differences between model validation and model optimization

Consequently, differently to model optimization, a validation test does not concern with computer-intensive parameters searching and rarely implies the need for time-consuming simulations. Indeed, a model with a fixed set of parameter values is tested in the scope of additional experimental data not previously used to fit the model (Table 1).

1.1.4 Systematic Tracking of Validation History.

Scientific method tells us that model development is intertwined with the continuous process of new experimentation. The lack of a validation history management system hinders and limits the potential progress in further development of the model. The need to organize validation tests and its results have always existed. However,

with the growing number of models and experimental data a more systematic approach to track the validation is necessary. In addition to record-keeping of the validation results tracking the validation test is equally critical. Availability of new experimental data may either result in a new validation test or modification of an available test.

2 Methods

2.1 Framework Requirements

Although in practice the distinction among the user groups might be blurred it is useful to view the division of labor for validation among three classes of users: code developers, modelers and applied users (see Figure 1). The code developers takes the responsibility for verifying the code. In addition to the general task of debugging codes, users from this group look for program correctness (Jay, 1984). In other words, "correctness or veracity of the prediction, without bringing in supporting topics such as what is being predicted or how it is done" (Roache, 1998d). Therefore, during the modeling process users from the modeling group will become code developers. The verification task is therefore a prerequisite to the validation task. The responsibility for the modelers involves model calibration as well as its validation.

The applied users are specialists in applications but not necessarily experts in coding or modeling. Experimentalists would come under this domain. The work done by the modelers and the experimentalist are interrelated because "[c]alibration and [v]alidations are specific to a particular design application and user community" (Roache, 1998a).

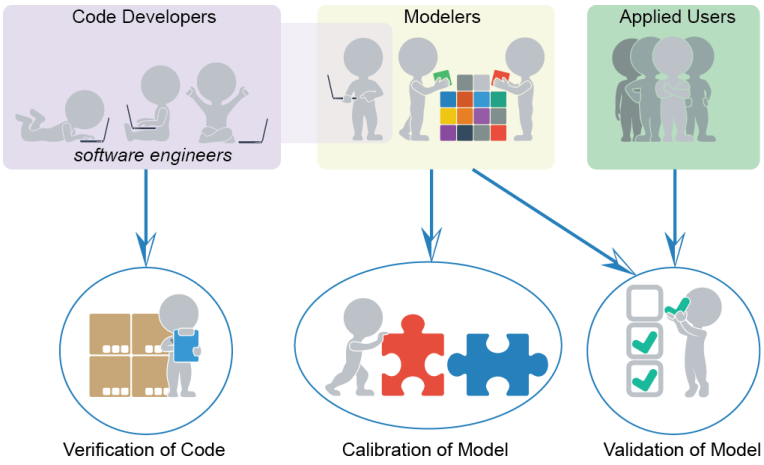

Figure 3: User Groups.

Since validation is often a long-term project the user must be able to determine versions on a particular analysis, retrieve an older version, and when applicable maintain it.

- Traceability
- Retrievability
- Maintainability

The thing being versioned refers to both the code for the model and test code. Traceability helps users identify the version used in the calculation and fetch copies of it if needed. Retrievability aid the user to justify earlier analyses. Here, maintainability does not refer to maintaining code but rather the ability for the user to organize and re-organize the validation components (model, test, and result). The ability to add further details can help clarify the analysis.

2.1.1 A Dynamic System.

The process of systematic validation should involve a dynamic, interactive database stored and accessible through the internet (Rizzi and Vos, 1996). The database performs the functions:

- Archives models, experimental data, tests, and results.
- Accessibility to scrutinize for validation exercises.
- Provides access and scrutiny to past validation data.
- Data visualization (plots).
- Allows ongoing discussion.

And, because it is dynamic it can

- be updated,
- be searched (via keywords),
- include reports and provide generated documents, and
- include expert evaluations and comments.

The interoperability between systems based on the Representative State Transfer (REST) style allows a uniform predefined set of stateless protocol; a communication protocol where no session information is retained by the receiver. A webservice based on RESTful architecture comes with six guiding principles: client-server architecture, statelessness, cacheability, layered system, code on demand and uniform interface (Leonard and Sam, 2007).

2.1.2 Validation of Model Instance.

In our validation framework the *model instances* gets validated. They refer to models with certain definite parameter values. Some refer to these as weak models as opposed to strong models based on whether it includes the parameters (Leijnse and Hassanizadeh, 1994). The grounds for choosing model instances (or weak models) to validate is purely practical. Roach (Roache, 1998b) says,

> [O]ne cannot validate a [model] but only a particular calculation or at least a range of "nearby" calculations defined over some [definite] parameter range.

Despite the fact that the validations are for model instances it is recommended that the validator does not abandon use of the term "model validation" (Leijnse and Hassanizadeh, 1994). However, the validation should carefully specify what the model constitute and what is being validated.

2.1.3 Analysis with Application in Mind.

Application plays an important role in validation. It helps determine the accuracy level of a model being validated from the perspective of its intended use (Mehta, 1996). Some contend that "the concept of a 'validated [model]', ascertained to be so independent of the intended variable measured, is a myth" (Roache, 1998b).

There are at least two characteristics of the accuracy level (Roache, 1998c):

- The model may be convincingly validated for one use but shown to be unacceptably inaccurate for another use.
- The standard of accuracy level is higher for purely scientific projects (than for an engineering project).

2.2 Framework Implementation

In this section we shall provide a brief overview of the overall architecture of the validation framework and the different components involved in its workflow. In the following section, through specific use case scenarios, we will demonstrate in more detail how to employ these various components to undertake a validation study.

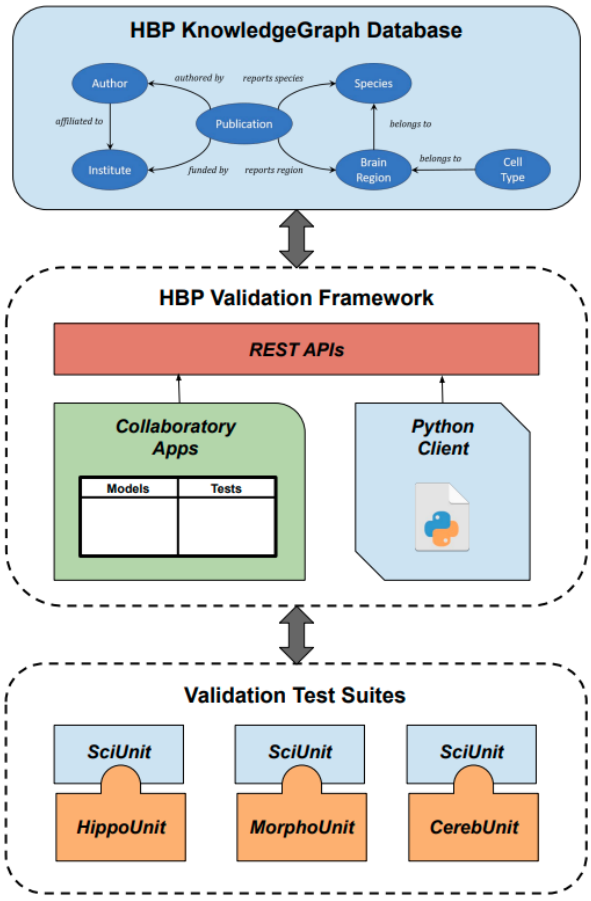

Figure 4: Representation of the various components involved in the validation workflow

The overall architecture of the validation system presented here can be broadly divided into three tiers as shown in fig. 4. The topmost tier consists of the Knowledge Graph, a system of storing information, whereby related data units are linked to one another by means of appropriate relations. It can be imagined as a graph where the nodes represent the individual units of data, and the edges signify the relation between them. The validation framework employs HBP's implementation of the KnowledgeGraph for storing data and recording their relations.

The implementation of the validation framework specific tools and services primarily spans the middle and bottom tiers. The middle tier comprises of the various components developed as part of the validation framework, and the lower tier represents the test libraries that will leverage the functionality of the middle tier to establish a complete validation workflow.

2.2.1 Validation Framework

The validation framework server has been implemented using the Django REpresentational State Transfer (REST) framework. This enables information to be retrieved via its REST APIs. These RESTful APIs can either be utilized directly to interact with the validation framework, or indirectly through the usage of the dedicated web applications and the Python client, as described below.

HBP provides an online platform, called the 'Collaboratory', for scientific collaboration. The Collaboratory consists of numerous collaborative workspaces, termed as Collabs. Collabs can host several kinds of resources

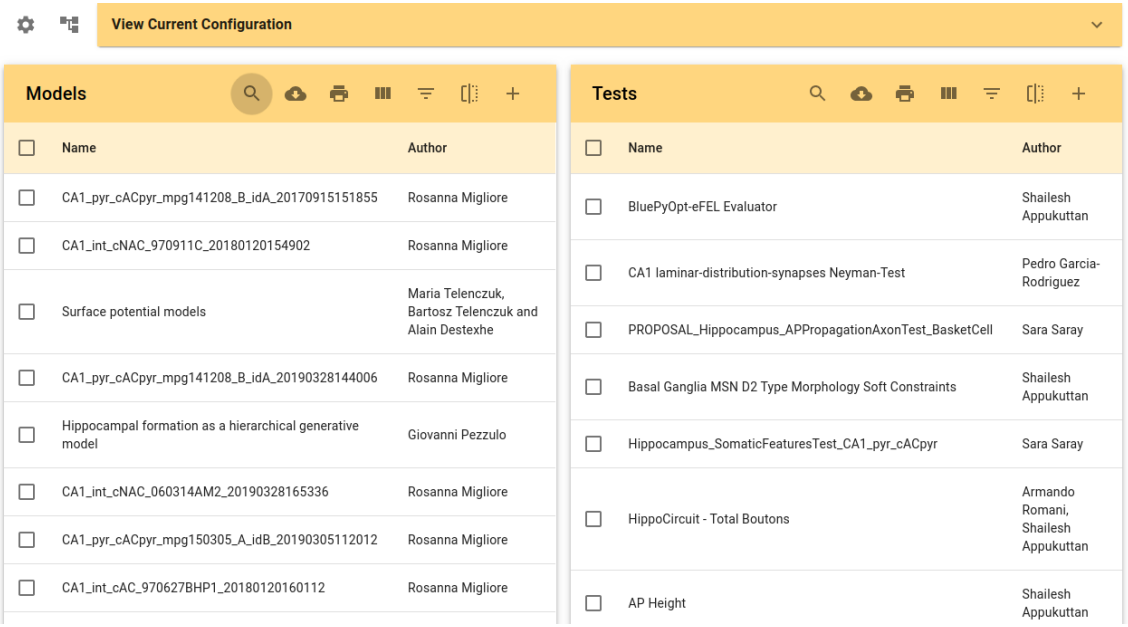

Figure 5: Screenshot: Listing of models and tests in the web app

such as web applications, Jupyter notebooks, documents, and also has its own dedicated storage space. The validation framework offers a web application for ease of access to its various features. The app provides a listing of all the models and tests registered on the validation framework, as shown in fig. 5. Clicking on a model or a test in this listing redirects the user to a page with detailed info about the model or test. Users can also add new models and tests, or edit existing ones. Additionally, it provides a section for displaying all the validation results associated with that model or test. Clicking any of the validation scores further redirects the user to a page with complete information of the validation that was undertaken. We shall elaborate on these features while presenting the use case scenarios.

The validation framework additionally provides a Python language based client for accessing the functionality of the validation framework. Like the web applications described above, the Python client employs the REST APIs offered by the validation framework to communicate with it. But unlike them, the Python client allows users to access the validation framework programmatically, thereby enabling scripting of the validation process and achieving desired levels of automation.

2.2.2 Validation Test Suites

The validation test suites constitute the library of validation tests that can be used to evaluate the models. Each test suite can comprise of a number of different tests, typically packaged into modules with respect to the brain region they address (e.g. Hippocampus), or the specific biophysical properties they evaluate (e.g. neuronal morphology). All these test suites are developed based on the SciUnit framework.

SciUnit is a test-driven framework for formally validating scientific models against data (Omar et al., 2014a). At its core, it manages the decoupling of the test implementation from the model implementation. It achieves this via the concept of 'Capabilities', which are basically interfaces through which a model and test can communicate. Fig. 6 illustrates this concept of capabilities as a layer of separation between models and tests. Each test defines the capabilities that are required of a model to undertake that particular test. Example of capabilities can be such as the ability to record the somatic membrane potential, or to inject a specifed stimulus at the soma or at a particular distance along the dendrites from the soma. Typically, capabilities are defined at a level of granularity such that they can be reused by other tests, thereby allowing the implementation of a set of capabilities by a model, to enable it to undertake a number of tests.

Once the model has implemented the required capabilities, the test will invoke the execution of the model under the desired simulation environment, to produce necessary data as output. This data is then compared

with experimental data tied to the test, to arrive at a goodness-of-fit between the model and the target reference data. The SciUnit framework allows tests to be model agnostic, and thereby enables the creation of a vast library of tests that can be employed for a number of models.

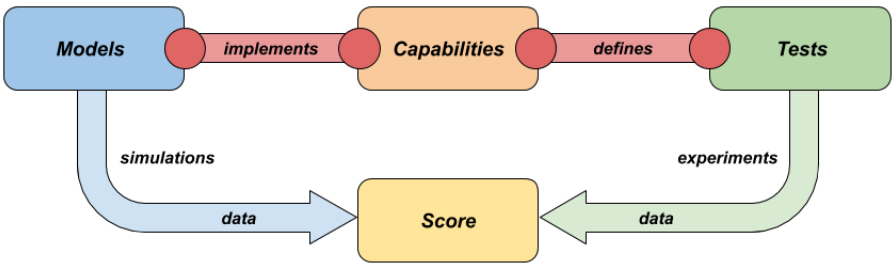

Figure 6: Overview of model and test interaction using the SciUnit framework

Each test within a test suite comprises a specification of the required model capabilities, the location of the reference dataset, and data analysis code to transform, wherever required, the recorded parameter, e.g. membrane potential, into a format, such as a histogram, that allows results to be statistically compared to reference data.

A number of validation test suites are currently at various stages of their development. HippoUnit (Sáray et al., 2020), CerebUnit, BasalUnit are examples of test suites targeted towards validating electrophysiological properties of single cell models of specific brain regions. HippoNetworkUnit and NetworkUnit (Gutzen et al., 2018) aims to test network level features of large-scale models, while SynapseUnit evaluates the subcellular properties underlying synaptic models. Another test suite, MorphoUnit, aims to automate the testing of morphologically-detailed cell models, according to neurite (axon or dendrite) and soma features and can be used in quantitative assessment of digital neuronal reconstructions for any brain region. In this present paper we shall employ HippoUnit as an example of a validation test suite to elucidate the working of our validation framework. An overview of Hippounit and its constituent validation tests is provided in section **??** of the supplementary document.

3 Validation Use Cases

3.1 Use Case 1: How good is the model?

3.1.1 The model

For this demonstration a model of the hippocampal CA1 pyramidal neuron is chosen for validation. This model is listed in the model catalog (Figure 7). Catalogued models are given a unique ID and are version tracked. Information provided for the model includes: brain region, cell type, model scope and species. This model shall be referred to as `CA1_Bianchi_2012`.

3.1.2 The tests

Available tests (validation) are listed in the test catalog which can be keyword searched and filtered for viewing list of tests that may be compatible for any desired model (Figure 8). Validation for `CA1_Bianchi_2012` will be based on: somatic feature test, depolarization block test, back-propagation test, post-synaptic potential attenuation test and oblique integration test. Details for each test can be viewed in a webpage (Figure 9).

The catalogued tests are version tracked; this allows the use of different test parameters. The test details includes information about brain region, cell type, species, experimental data modality and data type. The experimental data for any respective test can be of any file type. Figure 10 shows a data as a JavaScript Object Notation.

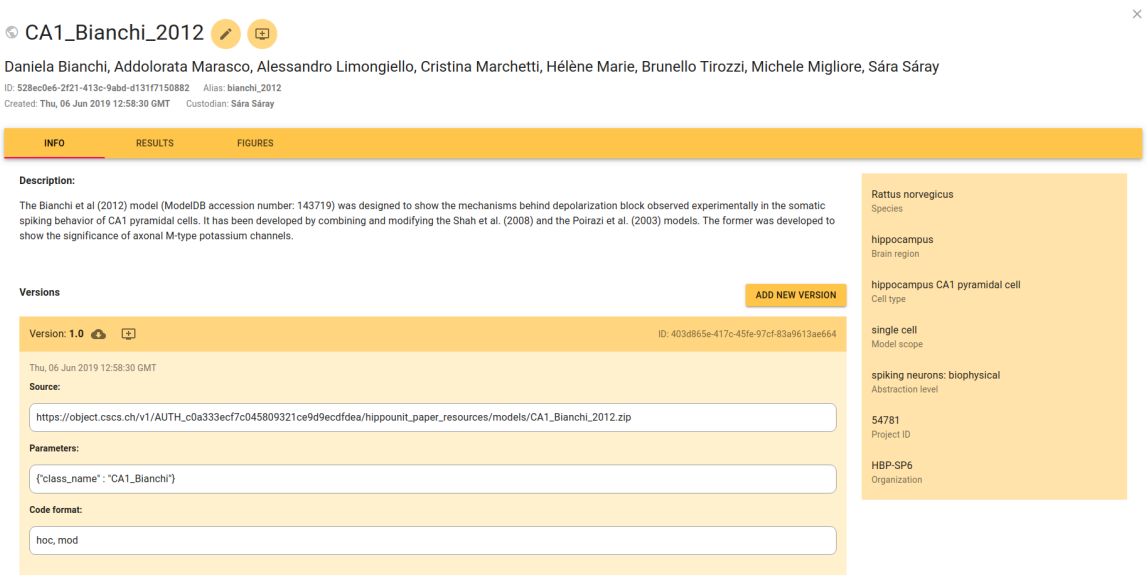

Figure 7: Detailed view of the `CA1_Bianchi_2012` in the model catalog.

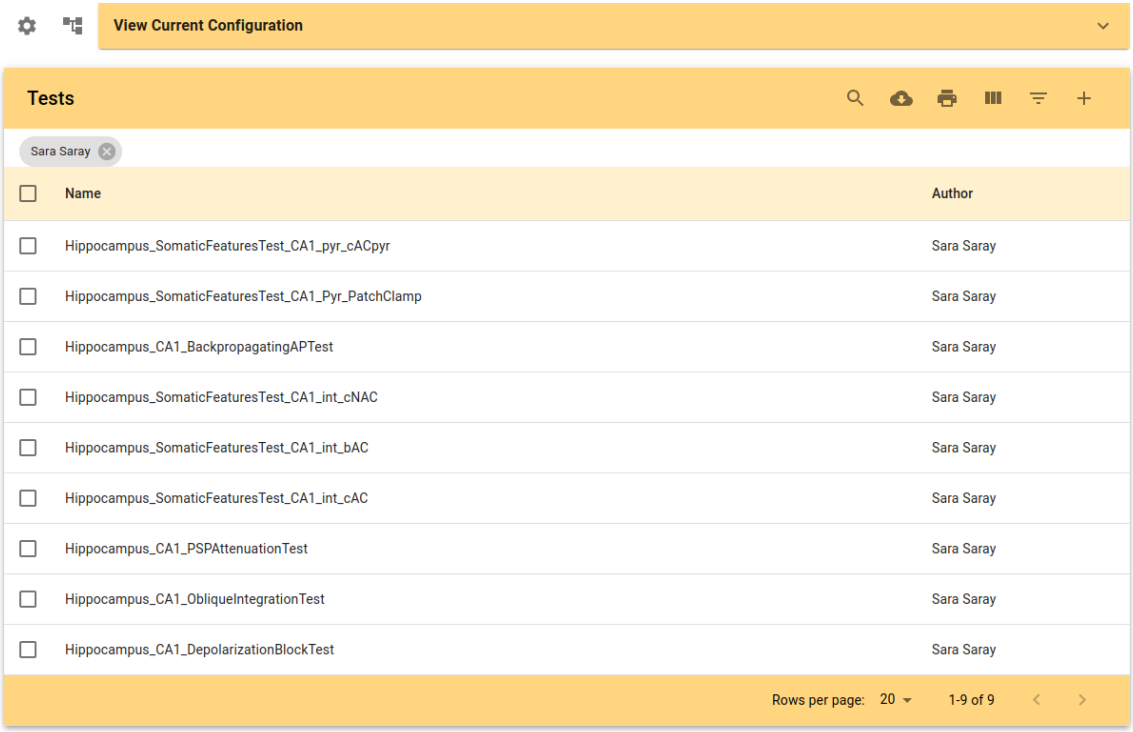

Figure 8: The filtered test catalog showing the list of potentially compatible tests.

3.1.3 Test result

Results for a completed test can be loaded on a webpage. Figure 11 shows the results for running the somatic feature test on `CA1_Bianchi_2012`. The statistics implemented in the testing code determines the type of the score. The figure shows the standardized z-score. If the testing code includes generating files like plots and a detailed statistical report they can be viewed in the files tab of the results page (Figure 12).

3.1.4 Results summary for multiple tests

If the model validation involves validation against a single test the user may look at the results page (Figure 11) and make validation analysis. Since the objective is to validate `CA1_Bianchi_2012` against multiple tests

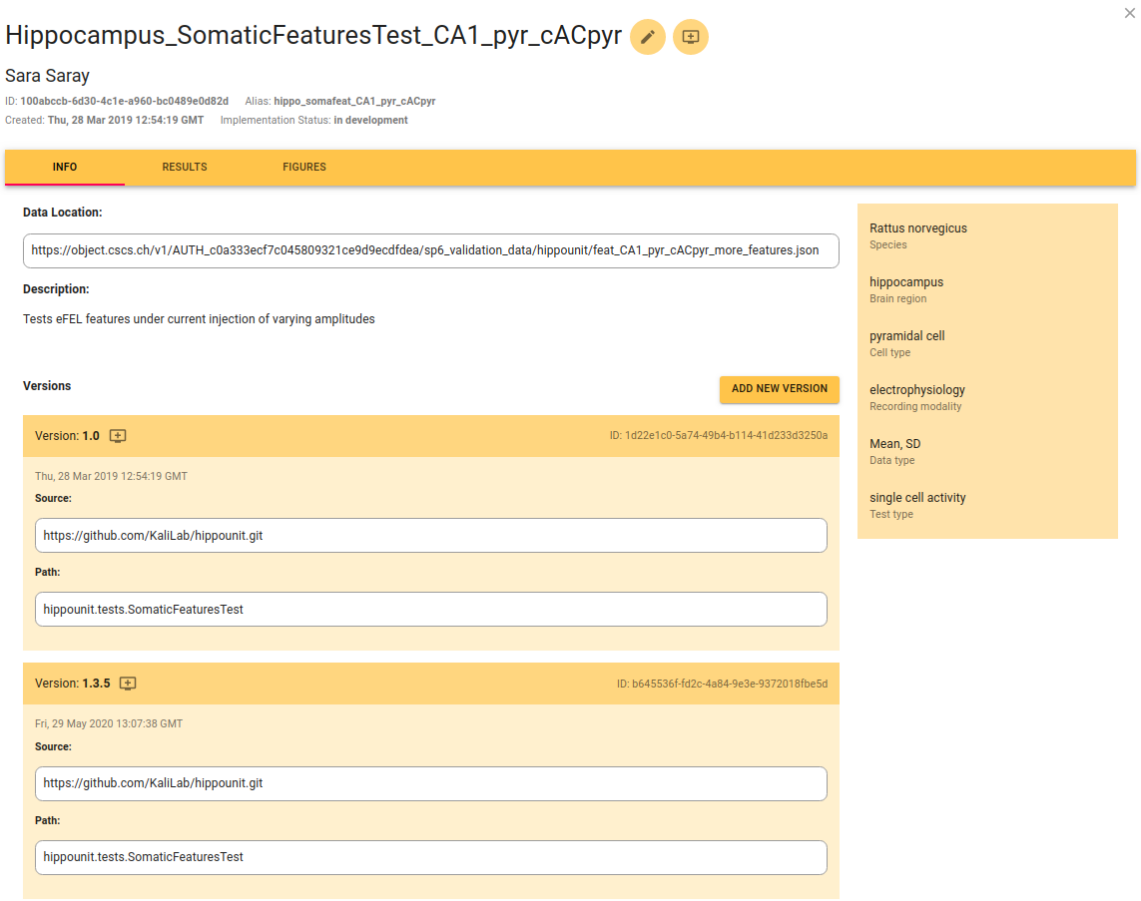

Figure 9: A detailed view of the somatic feature test.

each test is invoked. The resultant validation results are summarized on the web app, and selecting a particular result from the table returns the detailed results for that specific validation.

3.2 Use Case 2: Which of these models is the best?

A common situation consists in having several models of the same kind, *e.g.* digitally reconstructed morphologies or single cell models, that were built to reproduce anatomical or electrophysiological experimental data, respectively, though related with the same brain area. In such scenario, the model user faces often the challenge of making an informed decision about the best model candidate available in that set. Carrying out a rigorous model selection can be eased through a collective model assessment, which can be supported by a detailed analysis based on a set of validation tests.

In this section, it is shown how the information and tools available on the VF may be used towards that aim. Specifically, several published biophysically detailed multi-compartment models for Pyramidal cells (PC) of CA1 Hippocampus region in the rat brain, are compared. These include Golding et al. (2001), Poirazi et al. (2003b), Poirazi et al. (2003a), Katz et al. (2009), Migliore et al. (2011), González et al. (2011) and Bianchi et al. (2012). Two more recent collection of models currently used in the HBP (Migliore et al., 2018; Ecker A, 2020), are jointly considered along with these.

The corresponding validation results registered on the VF were previously obtained via the use of HippoUnit (Sáray et al., 2020). To facilitate comparison with the published models, the HBP model collections can be reduced to 'average' models derived from their available test results. The validation tools, both the web app and the Python client, can be queried to fetch the validation results for all these models, as shown in Fig. 14. The info thus retrieved is illustrated in Fig. 15. .

As discussed in detail in Sáray et al. (2020), it is evident that interpreting validation scores is not trivial.

```
"AHP1_depth_from_peak.Step1.0" : {
    "Mean" : "91.6409",
    "Std" : "4.9559",
    "Stimulus" : "Step1.0",
    "Type" : "AHP1_depth_from_peak",
    "Weight" : "1"
},
"AHP2_depth_from_peak.Step0.8" : {
    "Mean" : "82.8879",
    "Std" : "8.2237",
    "Stimulus" : "Step0.8",
    "Type" : "AHP2_depth_from_peak",
    "Weight" : "1"
},
"AHP2_depth_from_peak.Step1.0" : {
    "Mean" : "79.8599",
    "Std" : "8.0481",
    "Stimulus" : "Step1.0",
    "Type" : "AHP2_depth_from_peak",
    "Weight" : "1"
},
"AHP_depth.Step0.8" : {
    "Mean" : "11.673",
    "Std" : "4.0636",
    "Stimulus" : "Step0.8",
    "Type" : "AHP_depth",
    "Weight" : "1"
},
"AP_amplitude_change.Step1.0" : {
    "Mean" : "-0.2138",
    "Std" : "0.0925",
    "Stimulus" : "Step1.0",
    "Type" : "AP_amplitude_change",
    "Weight" : "1"
},
"AP_amplitude_from_voltagebase.Step0.8" : {
    "Mean" : "93.084",
    "Std" : "11.1059",
    "Stimulus" : "Step0.8",
    "Type" : "AP_amplitude_from_voltagebase",
    "Weight" : "1"
},
"AP_amplitude_from_voltagebase.Step1.0" : {
    "Mean" : "89.7448",
    "Std" : "11.789",
    "Stimulus" : "Step1.0",
    "Type" : "AP_amplitude_from_voltagebase",
    "Weight" : "1"
},
"AP_begin_voltage.Step0.8" : {
    "Mean" : "-49.3576",
    "Std" : "6.4407",
    "Stimulus" : "Step0.8",
    "Type" : "AP_begin_voltage",
    "Weight" : "1"
},
```


Figure 10: Experimental data as a JavaScript Object Notation (JSON).

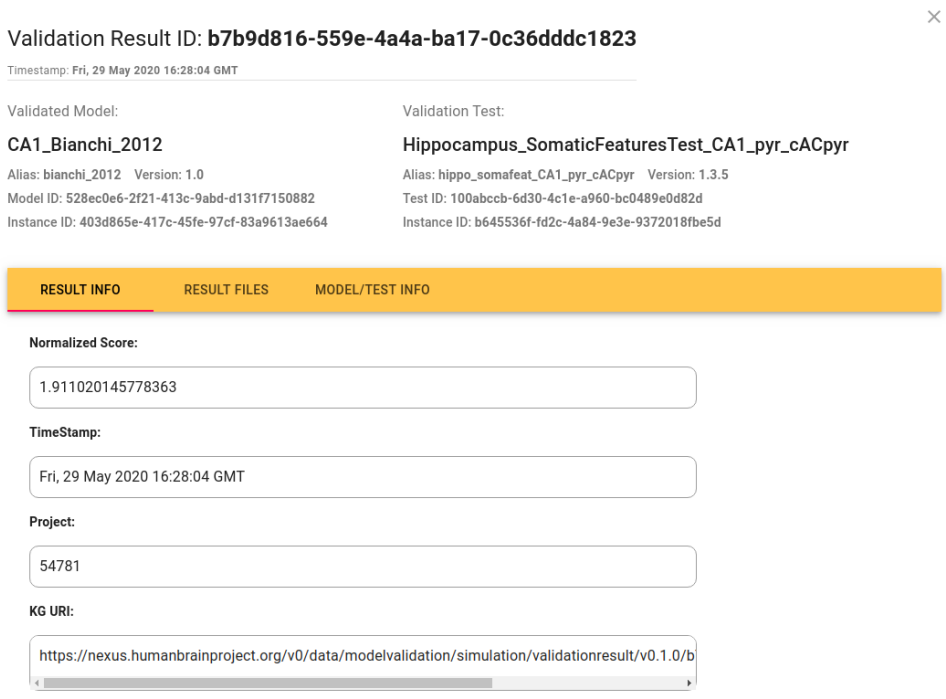

Figure 11: Results page for a single test.

The best way to combine and compare scores from different tests is a scientific question; it may depend for example on the level of confidence we have in the accuracy of the experimental data, inherent stochasticity of the underlying models, amongst several other parameters. For example, the validation results available for the *depolarization block* test of HippoUnit, poses some problems. Firstly, some of them pass the test while others

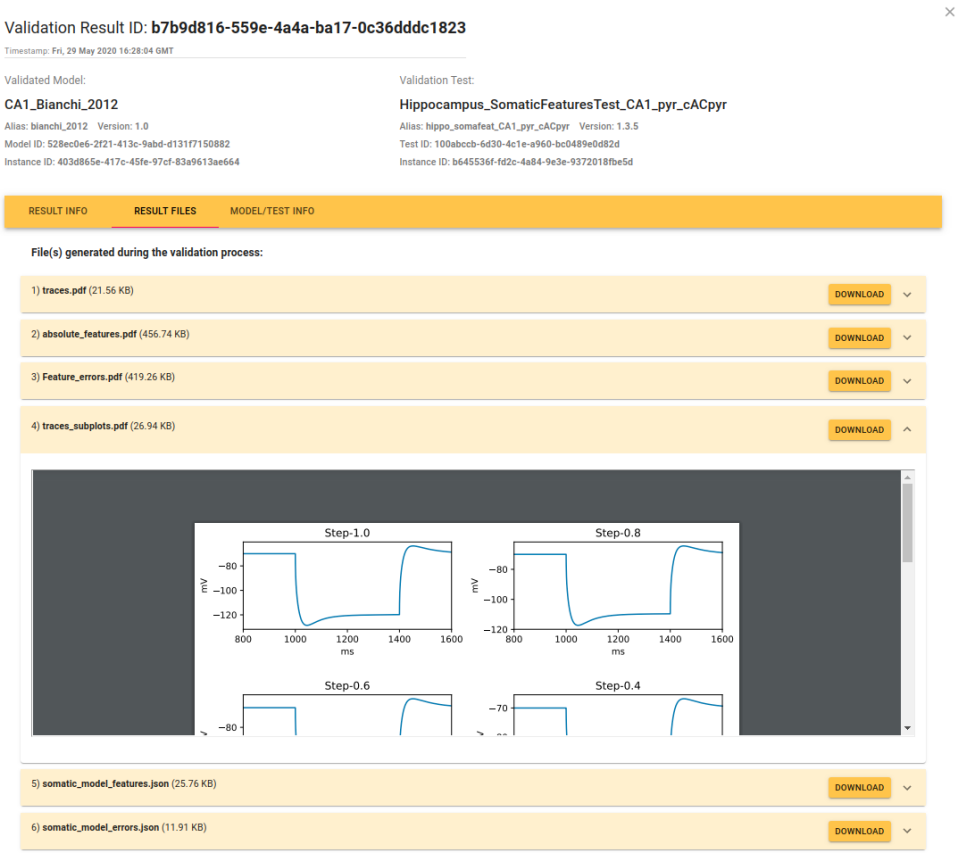

Figure 12: View of the result files tab within the results page.

do not. Those models passing the *depolarization block* test, are scored with a meaningful *Z-score*, as all other tests within HippoUnit. By contrast, any model that fails to enter a depolarization block, is assigned an ad-hoc penalty score of 100 (Sáray et al., 2020). Such values can not be combined with the other test scores, to obtain an idea of the model's overall performance.

3.3 Use Case 3: Registering models, tests

Models and tests can be registered on the validation framework in two ways: (i) using the web applications on the HBP Collaboratory, (ii) using the Python client. The former provides a user-friendly graphical user interface, while the latter provides a command line interface for more advanced users, enabling programmatic access to the validation framework. We present both these approaches below. For reasons of brevity, we shall limit this discussion to creating new models, but it should be noted that the process for adding new tests is very similar.

3.3.1 Via the web apps - GUI approach

Fig. 18 shows the top half of the model creation page of the model catalog web application. We begin by assigning a name to the model. This name does not have to be unique. Optionally, we can also assign an "alias" that can be used to uniquely identify the model on the validation framework. We also specify various metadata about the model, such as the species, brain region and cell type that the model corresponds to, the modeling scope and licence. Other info involves that such as the author and the description of the model itself.

Fig. 19 shows the bottom half of the model creation page corresponding to creation of model instances. Here we can register all the model variants for that particular model. This requires providing info on the location of the source code for the model instance, any relevant parameters that might be required for execution, a brief description, URL to any morphology that might have been employed, and a unique version name to identify

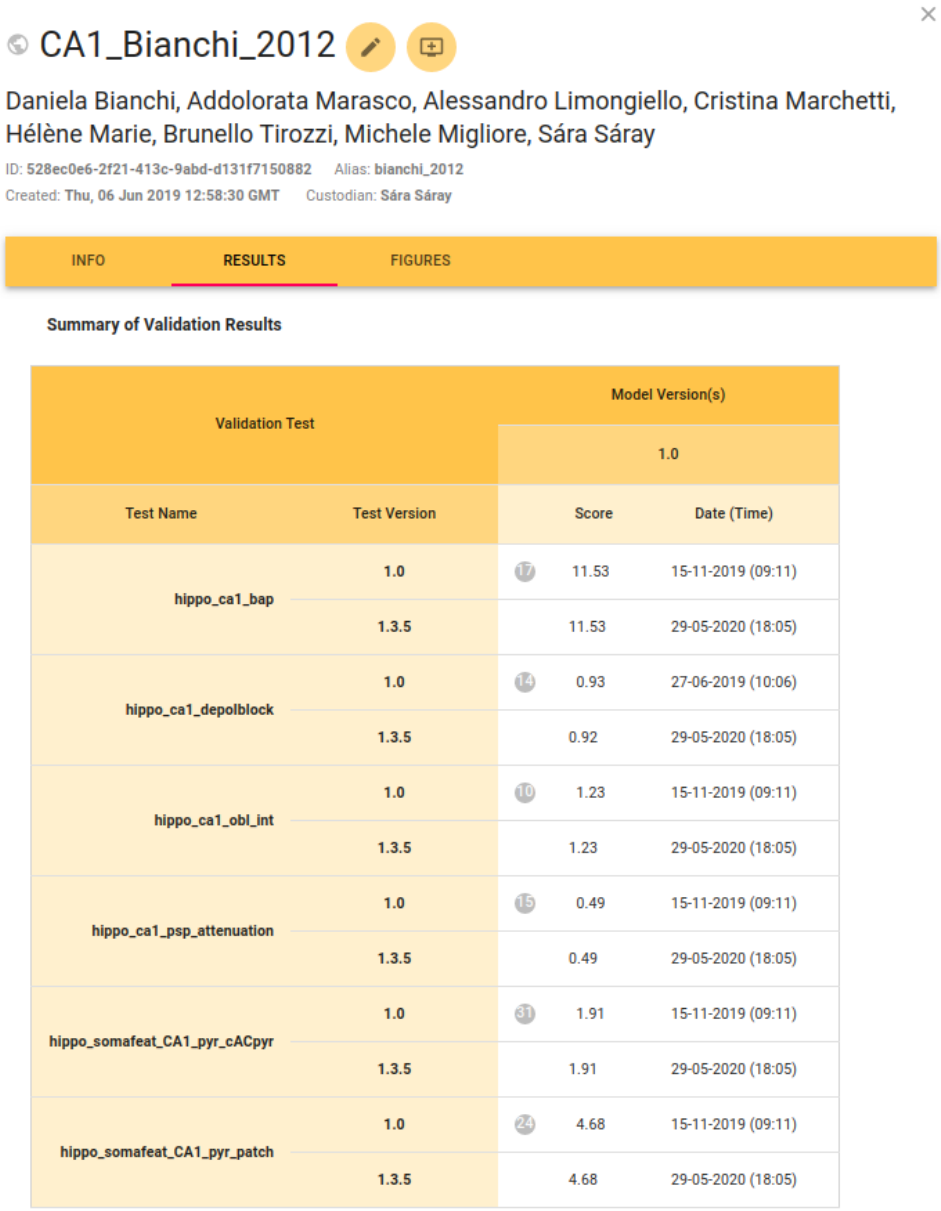

Figure 13: Results page summarizing the test results performed on a chosen model.

that particular model instance amongst other instances of that model.

On clicking the "Save" button, the model is officially registered on the model catalog and a separate unique identifier (UUID) is assigned to the model and each of its model instances. Once created, this can be viewed on the model's page on the model catalog, as seen in previous use cases. The model and its instances can henceforth be identified via these UUIDs, and also using the model's alias, if set, and the version name of the model instance.

3.3.2 Via the Python client - CLI approach

The Python client provides features for adding and editing both models and tests. The various features of the Python client are described in its online documentation. Listing 1 shows the equivalent model registration process via the command line interface:

```
from hbp_validation_framework import ModelCatalog
mc = ModelCatalog(username=HBP_USERNAME)

model_id = mc.register_model(app_id="538473",
                  name="Passive Model",
```

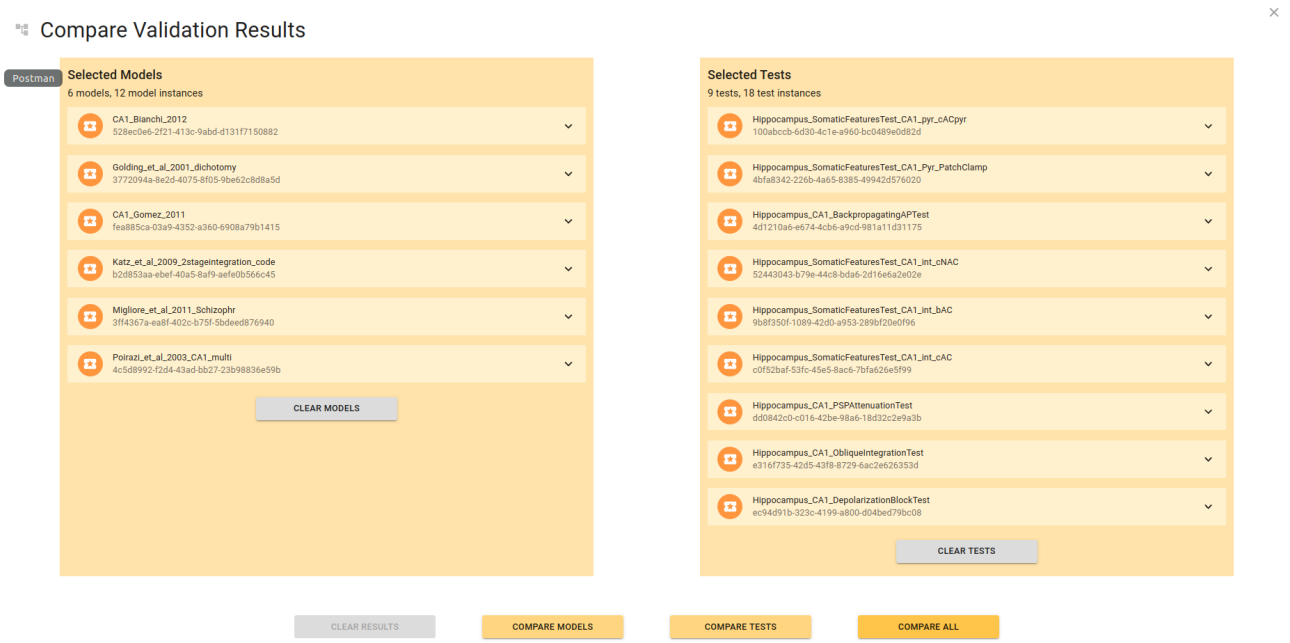

Figure 14: Screenshot: Validation results for multiple prevailing CA1 PC models against multiple tests

	hippo_ ca1_bap (1.0)	hippo_ca1_ depolblock (1.0)	hippo_ca1_ obl_int (1.0)	hippo_ca1_ psp_attenuation (1.0)	hippo_somafeat_ CA1_pyr_cACpyr (1.0)	hippo_somafeat_ CA1_pyr_patch (1.0)
bianchi_2012(1.0)	11.535	0.926	1.229	0.489	1.917	4.680
golding_2001(fig8B)	4.943	100.000	1.115	1.035	3.407	4.129
golding_2001(fig8A)	2.897	100.000	1.031	1.036	3.149	4.269
golding_2001(fig9B)	2.557	12.316	1.143	0.949	3.276	4.646
gomez_2011(n123_morph)	3.747	100.000	0.811	4.335	2.968	3.609
gomez_2011(n125_morph)	4.068	100.000	5.392	5.615	1.688	4.151
gomez_2011(n128_morph)	2.886	100.000	0.753	3.371	2.808	4.870
gomez_2011(n129_morph)	1.793	2.965	2.229	2.280	6.127	4.478
gomez_2011(n130_morph)	3.972	100.000	0.647	2.397	4.189	3.070
katz_2009(1.0)	4.577	1.756	0.867	0.906	3.951	4.607
migliore_2011(1.0)	3.366	100.000	1.252	2.327	1.238	4.398
poirazi_2003(1.0)	4.483	100.000	0.831	3.783	2.893	10.325

Figure 15: Screenshot: Validation results for multiple prevailing CA1 PC models against multiple tests

```
                alias="VF_passive_model",
                author="Shailesh Appukuttan",
                owner="Shailesh Appukuttan",
                private=False,
                species="Mus musculus",
                brain_region='hippocampus',
                cell_type='hippocampus CA1 pyramidal cell',
                model_scope='single cell',
                abstraction_level='spiking neurons: biophysical',
                organization="Other",
                description="Passive cell with only leak channels"
               )


model_inst_id = mc.add_model_instance(alias="VF_passive_model",
                version="g_pas_0.001",
                description="self.soma.e_pas = -73.9 #mV self.soma.g_pas = 0.001 #uS/cm2",
```

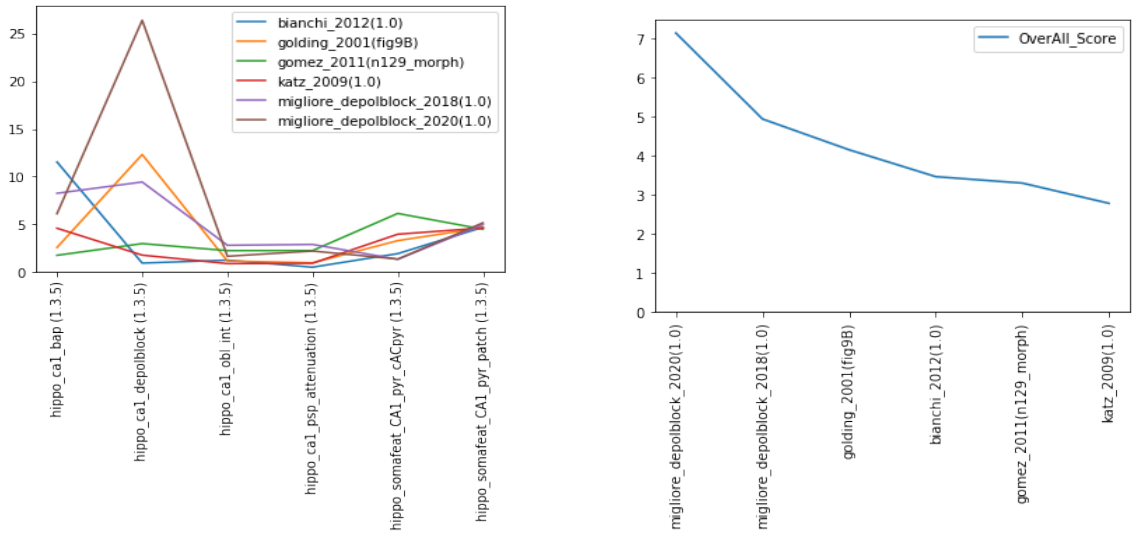

Figure 16: Validation scores for CA1 PC models that passed all HippoUnit tests

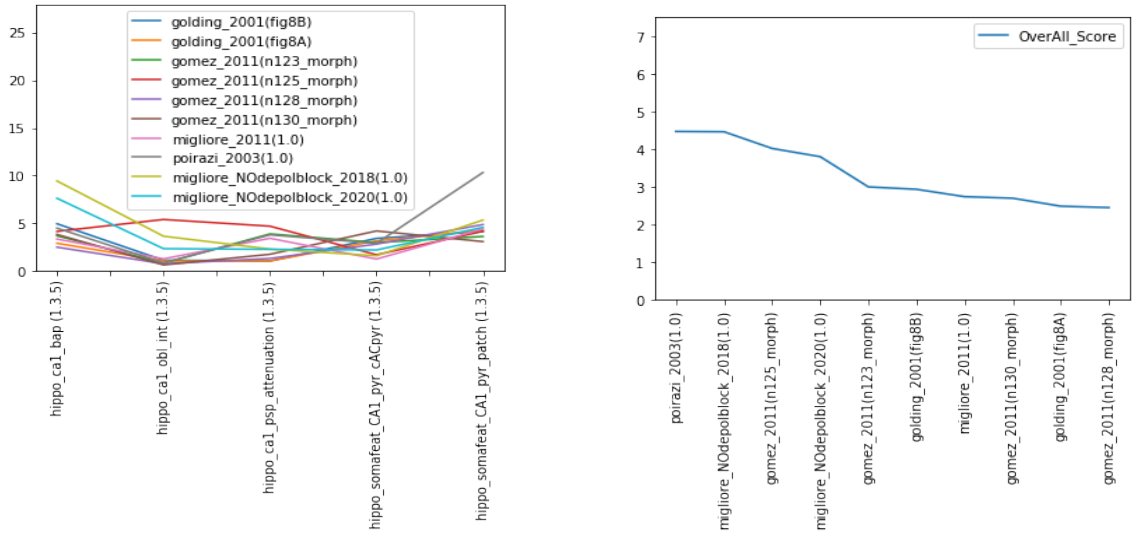

Figure 17: Validation scores for CA1 PC models that failed the *depolarization block* HippoUnit test

```
                parameters="{'class_name': 'PassiveCell'}",
                morphology="",
                source="https://object.cscs.ch/v1/AUTH_c0a333ecf7c045809321ce9d9ecdfdea/CA1_synaptic_plasticity/stdpmodel.zip",
                code_format="Py, NEURON")
```

Listing 1: Registering a model using the Python client

Though the usage of the Python client might involve a slight learning curve, it enables much faster interaction with the validation framework and its underlying features. This is particularly beneficial when handling multiple models and/or tests.

3.4 Use Case 4: Running validations and registering results

Typically users would run the validation tests locally on their workstations. But for more complex models, it might be essential to have simulations run on HPC machines. Below we explore how to undertake model validations in both these scenarios.

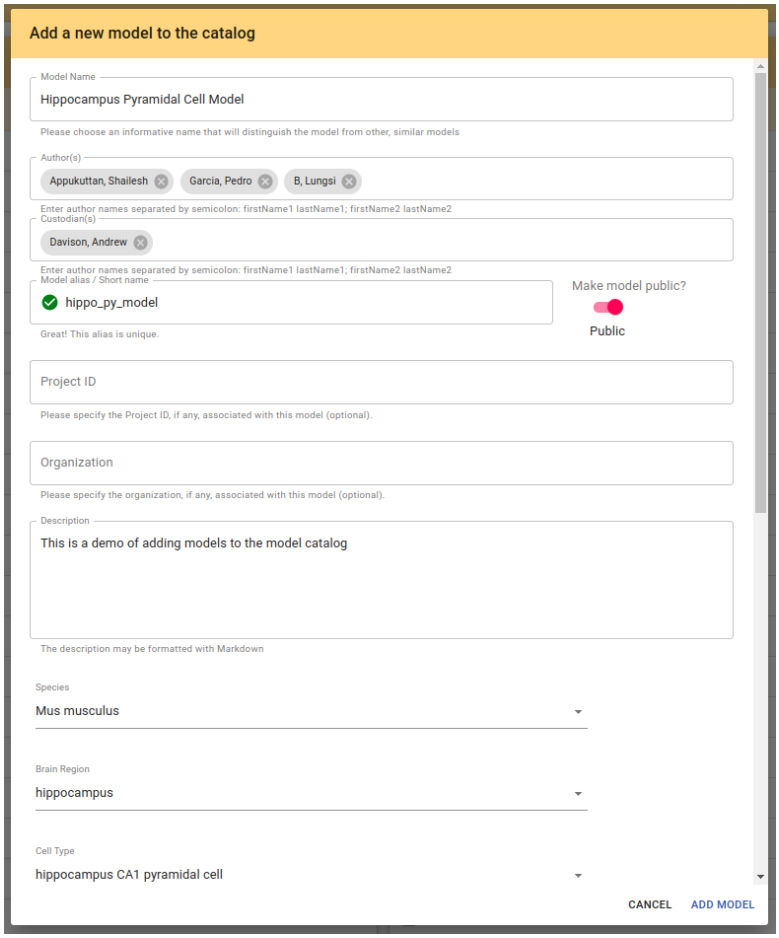

Figure 18: Creating a model entry on the model catalog web app

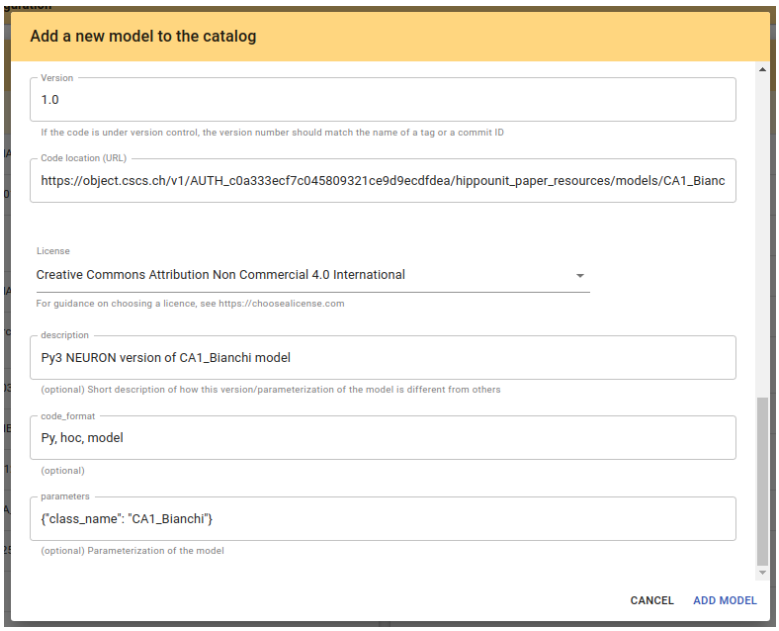

Figure 19: Defining an instance of a model on the model catalog web app

3.4.1 Running validations locally on workstation

The user is provided a 'utility' module within the Python client whereby they can access a variety of useful functionalities. The `run_test()` method of this module provides a ready-to-use validation launch script. The method is input with the model object and identifiers for the required test instance, along with certain parameters for managing storage and registration. The method will internally communicate with the validation framework to retrieve test related info, fetch the required file containing the target reference data, and then load and initialize the validation test from the locally installed test package. Using SciUnit's `judge()` method, it will then run the validation on the model object and return the result. This result can optionally be automatically registered on the validation framework and the files stored on the specified HBP Collaboratory. If registered, the method will also return a UUID that uniquely identifies that validation result on the validation framework. Listing **??** in the supplementary document provides an example of this for running the `Oblique Integration Test` test on `Bianchi_et_al_2012` model.

3.4.2 Running validations on HPCs

The `run_test()` method, described above, requires internet connectivity, as the Python client needs to connect to the validation framework to retrieve the model info and test related info. This works fine when running validations on local workstations, where internet connectivity is mostly guaranteed. But it becomes a concern when running tests on HPC resources where the compute nodes might not have access to external networks, or only have restricted access.

To resolve this situation, the functionality of `run_test()` is split into three sub-tasks, as described below and illustrated in fig. 20:

1. `prepare_run_test_offline()`: This method requires internet connectivity. It prepares for running a specified validation test by retrieving the test related info such as the module path, run time parameters, and saves the reference data file required by the test locally. This local path and other test metadata is stored in a local JSON file for further use.

2. `run_test_offline()`: This method can be entirely run offline. It uses the JSON file created above, along with the locally stored observation file, to instantiate the test class. The validation test is executed for the specified model, which is made available locally, and the results are stored as a pickled file. Together with any generated test result related files, these files are saved on the local file system.

3. `upload_test_result()`: This method requires internet connectivity. It loads the pickled file created above and uses the stored info to register the test result info onto the validation framework, along with all the result related files.

As depicted in fig. 20, when running on HPC resources the intended workflow is to run `prepare_run_test_offline()` on the master node, or locally if necessary and move the generated JSON file to the HPC server storage. `run_test_offline()` can then be executed to run the validation test on the supercomputers, without any need for internet connectivity. Finally, `upload_test_result()` can be invoked on the master node, or locally if required after copying over the files, to register the generated validation result on the validation framework. It should be noted that the model is made available locally, before launching jobs with `run_test_offline()`. Listing **??** in the supplementary document shows an example of launching a model validation on an HPC machine,

4 Discussion

The validation framework presented here offers a regime for the systematic validation and benchmarking of computational models in neuroscience. Being built on top of the existing SciUnit framework provides familiarity and ease of integration for users. The test units developed for use with the framework are extensible libraries, with each test being implemented in a model-agnostic manner. Every model can undertake any test

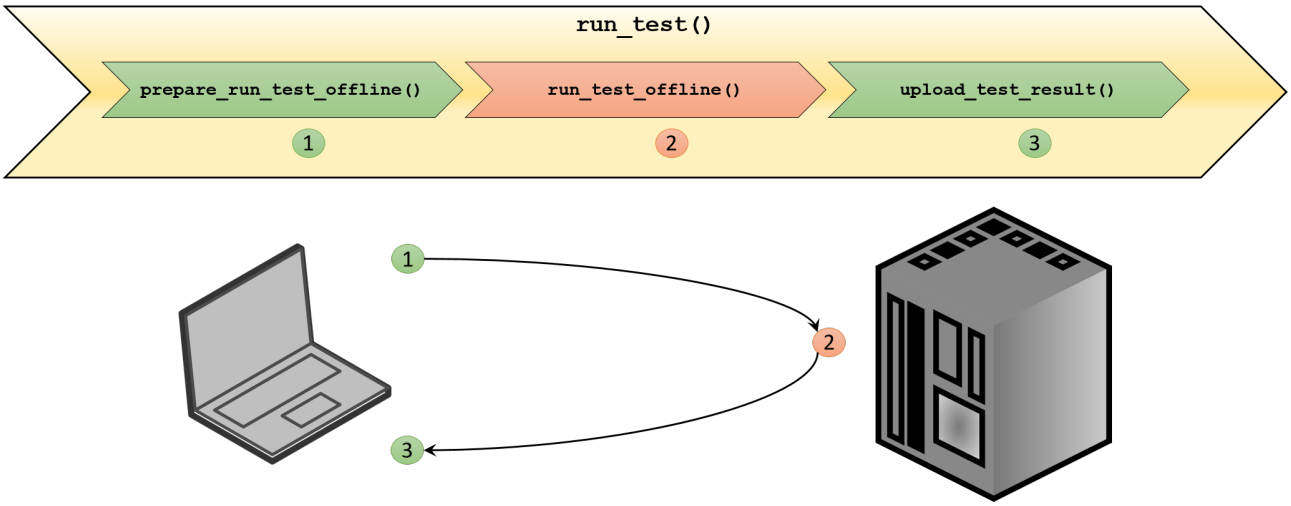

Figure 20: Block diagram illustrating the implementation of the `run_test()` method of the 'utility' module of the validation framework Python client. Tasks 1 and 3 require internet connectivity, while task 2 can be run offline.

by implementing the required interfaces, which are often short snippets of code as demonstrated in our example. This approach encourages community-driven development of these test packages as every test developed caters to the entire scientific community. The benefits of employing, and when required extending, these libraries outweigh the efforts and gains to build custom validations for every model. More importantly, these umbrella of tests packages provide standardized measures for the evaluation of computational model, that help identify the extent of biological realism.

The Validation Framework presented here will greatly ease the comparison between different models and track their performance over iterations during development. Our services facilitate in-depth exploration of individual validation outcomes, not just for the model owner but the entire scientific community. It holds potential to promote collaborative modelling communities for model development and validation.

Unlike ModelDB (Hines et al., 2004), the framework presented here is not purposed merely for published models. In fact, a core application of the framework lies in the process of model development itself. Models are seldom developed in a single step, but are the outcome of several iterations of fitting and tuning parameters. Every iteration produces a variant of the model, corresponding to our terminology of "model instance", that needs to be evaluated to determine if it is better than the previous iterations or not. This evaluation involves the testing of these model instances using a a set of validation tests. Currently, this process is undertaken ad-hoc with often no documented records for subsequent perusal. Employing the validation framework in the model development workflow offers a systematic and thorough assessment, along with complete documentation to track the development and improvement of a model over time.

It is pertinent to mention that model instances may also encompass other changes, and not necessarily merely re-tuning of parameters. For example, a particular model may be ported from one simulator to another, which might involve the re-write of source code between the simulator languages, but while expected to retain identical biophysical properties. The validation framework offers an ideal platform for testing this agreement, or to identify the lack of it. Similarly, a model may vary in the level of its abstraction. With an eye on large network level simulations, a multi-compartmental model might require to be reduced to fewer compartments, or even a point neuron model. This process naturally entails variation in biophysical properties and thereby in model responses. Here again the validation framework provides a tool to quantify the extent of such variations, and to help direct and/or restrict this process of simplification by offering comparative metrics for each model instance.

To provide users with a richer experience, we have enabled several neuroscience tools to be used from within or alongside the validation framework. Multiple test packages used with the validation framework, such as HippoUnit, already employ implicit capabilities that support models output by BluePyOpt, an extensible

framework for data-driven model parameter optimisation (Van Geit et al., 2016). This enables these models to undertake the corresponding validation tests without having to manually implement the capability requirements. Such implicit capabilities can also be implemented in future for other standardized model formats, such as the SONATA format for network models (Dai et al., 2019). The web app also integrates with BlueNaaS (Neuron as a Service), a web application that allows the simulation of NEURON models online from within the web browser. Model instances uploaded in a BlueNaaS compatible format, e.g. format produced by BluePyOpt, can directly be launched in BlueNaaS from the web app. For model instances with morphological data, the web app can launch the morphology inside a 3D viewer. For network models, users can specify a list of morphologies employed in the model, and each of these can be individually viewed. Our intention is to keep expanding on this front, and incorporate more tools based on community feedback and requirements.

The importance of systematic and comparable validations of computational models cannot be understated. With the field of computational neuroscience flourishing with increasing rapidity, it is essential to have a well-defined system in place that consolidates the validation efforts undertaken by the ever growing scientific community. The HBP validation framework described in the present work provides such a regime that offers standardization of the process and encourages community-driven development. The benefits apply not only to the modelers, who gain by having access to a vast library of ready-to-use validations for their models, but also to reviewers and other external users, who will be better equipped to gauge the novelty, accuracy, and applicability of any given model. If adopted at the community level, the framework has the potential to act as a constantly evolving benchmark for systematic and thorough validation of computational models in neuroscience.

4.1 Proposed Work Ahead

Tests in the validation framework are packaged into modules based on their applicability, e.g. the region of brain they validate (e.g. `HippoUnit` test module for single cell models of the Hippocampus), or the modeling aspect they help evaluate (e.g. `MorphoUnit` test module for model morphologies). Even with this organization, very often not all the tests within a single module are relevant for all models. For example, tests for dendritic features are not appropriate for point neuron models. Currently, the user is required to explicitly choose and specify every validation test they wish to undertake on a given model. In the future, we intend to allow the concept of "Test Suites" whereby a group of often employed tests can be packaged into an easily specifiable entity, thereby making the process of test selection simpler.

We have also procured HPC resources via CSCS (Swiss National Supercomputing Centre) for enabling execution of validations on the cloud. As opposed to the current requirement of running the validations on the user's own resources, we are in the process of piloting validations on a general service account linked to the framework itself, and made freely available to the scientific community. Initially, this feature is expected to be restricted to models with limited demand for resources, and can later be extended to larger models with the availability of greater resources. Alongside, we also intend to enable HPC users to employ their own alloted resources by interfacing with common HPC providers. Some of this work has already been undertaken within the HBP.

An outcome of cloud computing for the validation services will also allow auto-launch, wherever applicable, of relevant validation tests whenever a new model instance is registered to an existing model in the model catalog. The model owner would not need to explicitly specify and launch validations for each iteration of their model, and can simply initiate this process by registering the new variant.

In recent years, there has been an increasing trend of standardization in the field of computational neuroscience. This is also visible in the realm of model definition, with simulator simulator-independent languages (Davison et al., 2009) and standardized formats (Dai et al., 2019) being in vogue. This has permitted the development of tools and services that can be readily exploited by the models. The validation framework resolves to support this trend by offering implicit implementation of the test interfaces, whereby models adhering to established formats and structures can readily undertake the validation tests without necessitating the writing of any additional code. As described earlier, we have already implemented this feature for a single cell format,

and propose to extend this to a recently published network level specification (Dai et al., 2019).

In summary, the validation framework, by supporting model agnostic, quantitative validation tests and automated workflows, has the potential to enable community-based development of test suites and hence a much more robust and systematic approach to the validation of computational models of nervous systems.

Conflict of Interest Statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Author Contributions

The Author Contributions section is mandatory for all articles, including articles by sole authors. If an appropriate statement is not provided on submission, a standard one will be inserted during the production process. The Author Contributions statement must describe the contributions of individual authors referred to by their initials and, in doing so, all authors agree to be accountable for the content of the work. Please see here for full authorship criteria.

Funding

This project was developed in part or in whole in the Human Brain Project, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreements No. 720270 and No. 785907 (Human Brain Project SGA1 and SGA2).

Data Availability Statement

The source code, data and services presented in this study can be found in the following repositories:

- `https://github.com/HumanBrainProject/hbp-validation-framework`
- `https://github.com/HumanBrainProject/hbp-validation-client`
- `https://github.com/HumanBrainProject/fairgraph`
- `https://object.cscs.ch/v1/AUTH_c0a333ecf7c045809321ce9d9ecdfdea/sp6_validation_data`

References

Aeschliman, D. P. and Oberkampf, W. L. (1998). Experimental methodology for computational fluid dynamics code validation. *AIAA Journal* 36, 733. doi:10.2514/2.461

Barber, T. J. (1998). Role of code validation and certification in the design environment. *AIAA Journal* 36, 752–758. doi:10.2514/2.433

Bhattacharya, B. S. and Chowdhury, F. N. (eds.) (2015). *Validating Neuro-Computational Models of Neurological and Psychiatric Disorders* (London: Springer)

Bianchi, D., Marasco, A., Limongiello, A., Marchetti, C., Marie, H., Tirozzi, B., et al. (2012). On the mechanisms underlying the depolarization block in the spiking dynamics of ca1 pyramidal neurons. *Journal of computational neuroscience* 33, 207–225

Coleman, H. W. and Steele, W. G. (1995). Engineering application of experimental uncertainty analysis. *AIAA Journal* 33, 1888. doi:10.2514/3.12742

Coleman, H. W., Steele, W. G., and Taylor, R. P. (1995). Implications of correlated bias uncertainties in single and comparative tests. *ASME Journal of Fluids Engineering* 117, 552–556. doi:10.1115/1.2817300

Dai, K., Hernando, J., Billeh, Y. N., Gratiy, S. L., Planas, J., Davison, A., et al. (2019). The sonata data format for efficient description of large-scale network models. *NEURON-D-19-00705*

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics* 2, 11

Ecker A, S. S. K. S. M. M. M. A. e. a., Romani A (2020). Data-driven integration of hippocampal ca1 synapse physiology in silico. *Hippocampus*

Golding, N. L., Kath, W. L., and Spruston, N. (2001). Dichotomy of action-potential backpropagation in ca1 pyramidal neuron dendrites. *Journal of neurophysiology* 86, 2998–3010

González, J. F. G., Mel, B. W., and Poirazi, P. (2011). Distinguishing linear vs. non-linear integration in ca1 radial oblique dendrites: it's about time. *Frontiers of Computational Neuroscience* 5, 1–12. doi:10.3389/fncom.2011.00044

Gutzen, R., von Papen, M., Trensch, G., Quaglio, P., Grün, S., and Denker, M. (2018). Reproducible neural network simulations: statistical methods for model validation on the level of network activity data. *Frontiers in neuroinformatics* 12

Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. (2004). Modeldb: a database to support computational neuroscience. *Journal of computational neuroscience* 17, 7–11

Jay, F. (1984). *IEEE Standard Dictionary of Electrical and Electronic Terms* (Institute of Electrical and Electronic Engineers: ANSI/IEEE Std. 100-1984)

Katz, Y., Menon, V., Nicholson, D. A., Geinisman, Y., Kath, W. L., and Spruston, N. (2009). Synapse distribution suggests a two-stage model of dendritic integration in ca1 pyramidal neurons. *Neuron* 63, 171–177. doi:10.1016/j.neuron.2009.06.023

Leijnse, A. and Hassanizadeh, S. M. (1994). Model definition and model validation. *Advances in Water Resources* 17, 197–200. doi:10.1016/0309-1708(94)90041-8

Leonard, R. and Sam, R. (2007). *RESTful Web Services* (Sebastopol, California: O'Reilly Media)

Marvin, J. G. (1995). Perspective on computational fluid dynamics validation. *AIAA Journal* 33, 1778–1787. doi:10.2514/3.12727

Mehta, U. B. (1996). Guide to credible computer simulations of fluid flows. *Journal of Propulsion and Power* 12, 940–948. doi:10.2514/3.24126

Migliore, M., Blasi, I. D., Tegolo, D., and Migliore, R. (2011). A modeling study suggesting how a reduction in the context-dependent input on ca1 pyramidal neurons could generate schizophrenic behavior. *Neural Networks* 24, 552–559. doi:10.1016/j.neunet.2011.01.001

Migliore, R., Lupascu, C. A., Bologna, L. L., Romani, A., Courcol, J.-D., Antonel, S., et al. (2018). The physiological variability of channel density in hippocampal ca1 pyramidal cells and interneurons explored using a unified data-driven modeling workflow. *PLoS computational biology* 14, e1006423

Omar, C., Aldrich, J., and Gerkin, R. C. (2014a). Collaborative infrastructure for test-driven scientific model validation. In *Companion Proceedings of the 36th International Conference on Software Engineering* (ACM), 524–527

Omar, C., Aldrich, J., and Gerkin, R. C. (2014b). Sciunit: Collaborative infrastructure for test-driven scientific model validation. In *ICSE New Ideas and Emerging Results (NIER)*

Poirazi, P., Brannon, T., and Mel, B. W. (2003a). Arithmetic of subthreshold synaptic summation in a model ca1 pyramidal cell. *Neuron* 37, 977–987. doi:10.1016/S0896-6273(03)00148-X

Poirazi, P., Brannon, T., and Mel, B. W. (2003b). Pyramidal neuron as two-layer neural network. *Neuron* 37, 989–999. doi:10.1016/S0896-6273(03)00149-1

Rizzi, A. and Vos, J. (1996). Toward establishing credibility in computational fluid dynamics simulations. *AIAA Journal* 36, 668–675. doi:10.2514/2.442

Roache, P. J. (1998a). Code quality assurance and certification. In *Verification and Validation in Computational Science and Engineering* (Albuquerque: Hermosa). 351–383

Roache, P. J. (1998b). Difficulties with experiments and validation. In *Verification and Validation in Computational Science and Engineering* (Albuquerque: Hermosa). 273–297

Roache, P. J. (1998c). Methodologies and examples of validations, calibrations, and certifications. In *Verification and Validation in Computational Science and Engineering* (Albuquerque: Hermosa). 299–347

Roache, P. J. (1998d). Semantics: Terminology, taxanomies, and definitions. In *Verification and Validation in Computational Science and Engineering* (Albuquerque: Hermosa). 19–61

Sáray, S., Rössert, C. A., Appukuttan, S., Migliore, R., Vitale, P., Lupascu, C. A., et al. (2020). Systematic comparison and automated validation of detailed models of hippocampal neurons. *bioRxiv*

Sarma, G. P., Jacobs, T. W., Watts, M. D., Ghayoomie, S. V., Larson, S. D., and Gerkin, R. C. (2016). Unit testing, model validation, and biological simulation. *F1000Research* 5. doi:10.12688/f1000research.9315.1

Schlesinger, S. (1979). Terminology for model credibility. *Simulation* 32, 103–104. doi:10.1177/003754977903200304

Tsang, C. (1991). The modeling process and model validation. *Ground Water* 29, 825–831. doi:10.1111/j.1745-6584.1991.tb00568.x

Van Geit, W., Gevaert, M., Chindemi, G., Rössert, C., Courcol, J.-D., Muller, E. B., et al. (2016). Bluepyopt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Frontiers in neuroinformatics* 10, 17

Viswan, N. A., HarshaRani, G. V., Stefan, M. I., and Bhalla, U. S. (2018). Findsim: A framework for integrating neuronal data and signaling models. *Frontiers in Neuroinformatics* 12, 38. doi:10.3389/fninf.2018.00038

W. G. Steele, C. A. J. R. P. T., P. K. Maciejewski and Coleman, H. W. (1996). Asymmetric systematic uncertainties in the determination of experimental uncertainty. *AIAA Journal* 34, 1458–1463. doi:10.2514/3.13253