

# Using Jupyter Notebooks to run jobs on SpiNNaker

## [For the current version see](#)

This document describes how to access and use Jupyter notebooks to run PyNN scripts and access the HBP/EBRAINS Neurorobotics Platform on SpiNNaker machines. It is based on the presentation at:

[https://docs.google.com/presentation/d/15XWZL-aAlJIN-lxa\\_uJuEjxqnn3yvarXBkWiHcWvE/edit#slide=id.g607786f23b\\_0\\_0](https://docs.google.com/presentation/d/15XWZL-aAlJIN-lxa_uJuEjxqnn3yvarXBkWiHcWvE/edit#slide=id.g607786f23b_0_0)

## Contents

[1. Getting Started](#)

[2. Running a PyNN script using a Jupyter Notebook / Lab](#)

[3 Using the Neurorobotics Platform \(NRP\) with Jupyter](#)

## 1. Getting Started

This section describes the current state of the SpiNNaker machine and the software stack's limitations.

### 1.1 SpiNNaker Machine

Before starting to run PyNN scripts on SpiNNaker machines via Jupyter, be aware that the total SpiNNaker machine capacity is a 10 cabinet machine, comprising 1,036,800 cores, as shown in Figure 1.



Figure 1: A 10 cabinet machine, wired together (Note: the 11th cabinet in the picture contains servers for handling jobs sent to the machine).

## 1.2 Software Limitations

To understand how this relates to a PyNN script, be aware that SpiNNaker stores all the data needed to execute a PyNN model within its on-board memory (SDRAM). This SDRAM is limited in size, as each SpiNNaker chip contains only 128MB of memory, split dynamically between the 18 cores (including the operating system core) that reside on the chip. On average, every core has available approximately 8MB to store the neuron parameters as well as the synaptic matrix of the atoms<sup>1</sup> it is executing. Most models, to date, are limited to 256 atoms per core<sup>2</sup>, and therefore the connectivity between these 256 atoms and the rest of the simulation must fit within these constraints.

Due to these limitations, the maximum possible number of atoms that can be simulated by a PyNN script on the largest SpiNNaker machine available is 250,675,200. This sounds rather large, but when it is factored in that each atom can only have a maximum incoming fixed number of connections of 8000, and that any more than this will result in a smaller number of neurons per core; therefore a smaller total neurons being simulated.

To make matters worse, currently, if a PyNN Projection between two Populations has delays greater than 16 machine time steps<sup>3</sup> then each core that simulates these neurons will have a delay model attached to it, which in practice reduces the maximum number of simulated neurons by half.

Finally, if the PyNN model includes plastic connections, these require more memory than fixed connections and therefore will further reduce the number of neurons per core. A summary of the rest of the SpiNNaker back end limitations can be found here:

<http://spinnakermanchester.github.io/spynaker/6.0.0/SPyNNakerModelsAndLimitations.html>

## 1.3 Script Limitations

This section describes some basic limitations that the PyNN script can experience when running using Jupyter:

1. A PyNN script running on the SpiNNaker backend needs to adhere to the subset of PyNN that is supported by sPyNNaker<sup>4</sup>, a list of what is not implemented can be found here:

<http://spinnakermanchester.github.io/spynaker/6.0.0/SPyNNakerModelsAndLimitations.html>

---

<sup>1</sup> Atoms here represents the atomic element which each core models. In the case of PyNN, these are neurons from a given population.

<sup>2</sup> This limitation may be lifted in the future, but to date, this is due the way that synaptic delays are implemented on SpiNNaker.

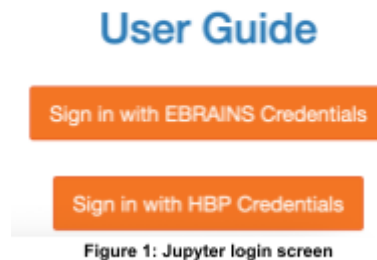
<sup>3</sup> This is 16 milliseconds in a simulation that runs at 1ms time steps, or 1.6 milliseconds in a simulation running at 0.1 ms time steps

<sup>4</sup> This is the SpiNNaker software stack's front end for simulating PyNN neuron models. See <https://www.frontiersin.org/articles/10.3389/fnins.2018.00816/full> for more details.

## 2. Running a PyNN script using a Jupyter Notebook / Lab

This section describes how an end user starts a PyNN simulation using a Jupyter Notebook.

First of all, access the Jupyter Notebook login at <https://spinn-20.cs.man.ac.uk>, or alternatively add the “SpiNNaker Jupyter” Community App to an EBRAINS collab (see [http://spinnakermanchester.github.io/common\\_pages/6.0.0/spinnaker\\_ebrains\\_portal\\_use.pdf](http://spinnakermanchester.github.io/common_pages/6.0.0/spinnaker_ebrains_portal_use.pdf) for more details on setting this up) and access it in a similar manner from there.



### 2.1 Logging in

You can either log in using your HBP or EBRAINS credentials (these can be obtained by following the instructions at <https://ebrains.eu/register>). Once you have logged in, you can choose between the Jupyter Notebook interface, and the Jupyter Lab interface. The Jupyter Notebook interface should look something like Figure 2, and the Lab interface like Figure 3.



Figure 2: Jupyter Notebook screen following login.

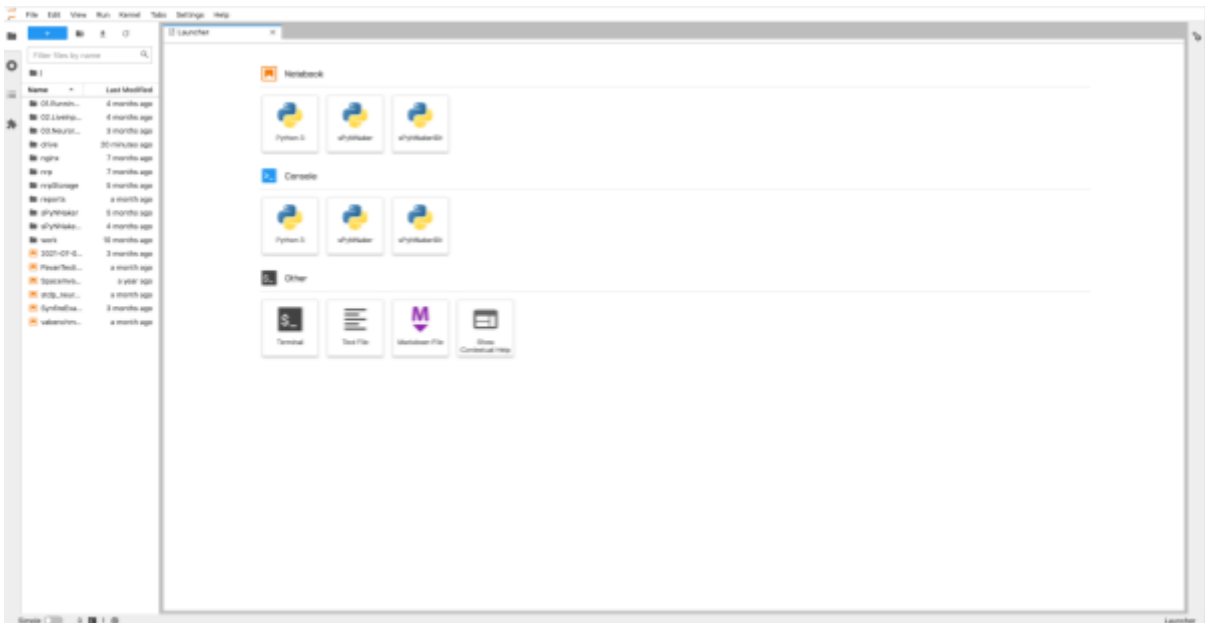


Figure 3: Jupyter Lab screen following login

## 2.2 Using a pre-prepared notebook

A pre-prepared synfire chain example is available: click on it to open it (Figure 4).



Figure 4: SpiNNaker Synfire Example

This can be run through in the usual manner for a Jupyter notebook via the control buttons at the top of the page. Be aware that running commands out of order may cause unexpected results; for example, performing `run(...)` before `setup(...)` will lead to an error. Running this particular example should give you the output shown in Figure 4.



Figure 4: Results from the synfire example

## 2.3 Running your own script

It is possible to create your own notebook and run PyNN scripts from within it. The simplest way to do this in the Notebook interface is to click the dropdown “New” menu, and under Notebook select “sPyNNaker”. This will give you an input box as in Figure 5 into which you can paste a PyNN script; then clicking e.g. “Run” will run this script.

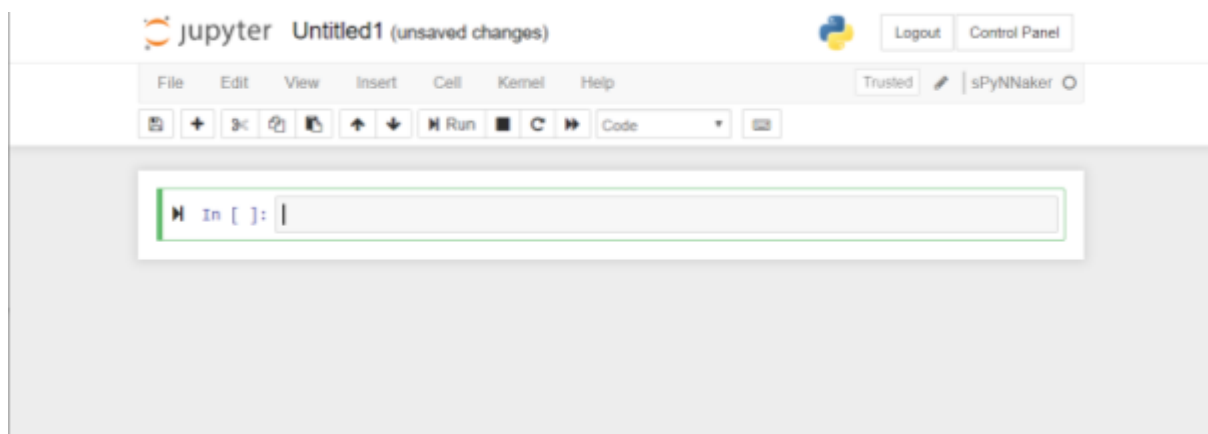


Figure 5: sPyNNaker kernel input for a new notebook

You can choose from the kernels “sPyNNaker” (release version 6.0) or “sPyNNakerGit” (the instantiation of the master branches of the GitHub libraries downloaded when your account was created). In the Lab interface, use the “Launcher” tab that was open when you first logged in (or, if you have closed the tab, it can be opened again using File - New Launcher) to select a Notebook using a particular kernel.

Be aware when using sPyNNakerGit that what you will be using are the master branches at the point at which you first logged in to the system, and that these will not change when you next log in; if you require any changes that have subsequently been made to master or new branches then you will need to manually get these yourself. This can be done from within the Notebook interface by clicking on the drop down menu “New” and selecting “Terminal”, or from the Lab interface by selecting Terminal from the Launcher tab, and then using the usual git commands from the command line within the “sPyNNakerGit” directory. You will need to be in the sPyNNakerGit environment when updating; to get into the environment, go into the sPyNNakerGit directory

```
cd sPyNNakerGit
```

and then use the command

```
source bin/activate
```

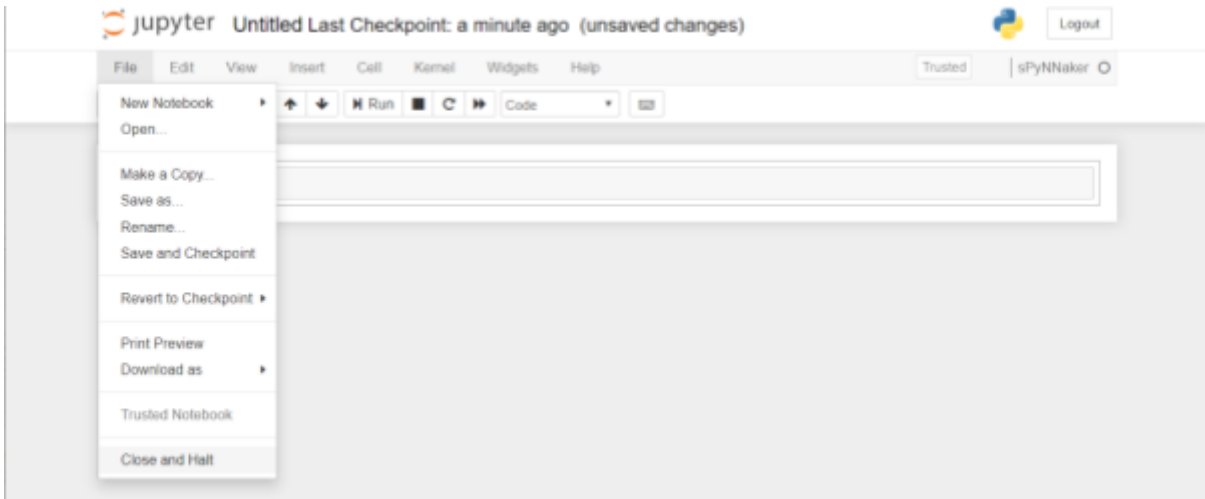
There are some automatic scripts provided to help you with this inside the sPyNNakerGit/support directory; in particular support/gitupdate.sh.

If you update master or switch branches, we also strongly recommend rebuilding the C code that runs on the SpiNNaker machine using the script support/automatic\_make.sh as it is likely that this will have changed during a recent update. It is also possible to change the kernel once you have started by choosing Kernel->Change kernel (in both the Notebook and Lab interfaces) and selecting what you wish to change to.

Within a notebook/lab it is possible to perform all the usual Python commands, including the ability to use “pip” to install anything that is not installed by default for SpiNNTools (see <http://spinnakermanchester.github.io/development/devenv6.0.html#PythonRequirements> for a non-exhaustive list of these). It is also possible to use pip etc. inside a Terminal window to install anything you need; simply make sure you are in the correct environment for the Kernel you wish to install something for and it should be possible to install it.

## 2.4 Closing notebooks and logging out

When you are finished, we request that you close any open notebooks (File->Close and Halt), and then logout using the button in the top right of the screen (see Figure 6).



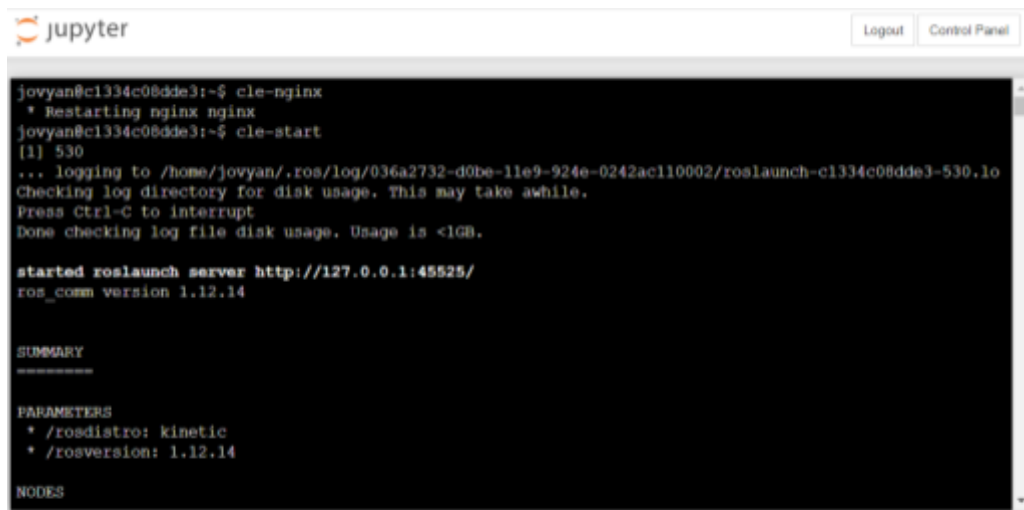
**Figure 6: Logging out / closing**

### 3 Using the Neurorobotics Platform (NRP) with Jupyter

It is also possible to run virtual robotic experiments within the Jupyter framework. To set this up you need to use the username and password you set earlier, and a few further steps.

#### 3.1 Setting up

Once you are logged in to the Jupyter system, create a new terminal and type in the command “cle-nginx”, and then “cle-start”. This will start up the NRP system in the background (see Figure 7).



```
jupyter
Logout Control Panel
jovyan@c1334c08dde3:~$ cle-nginx
* Restarting nginx nginx
jovyan@c1334c08dde3:~$ cle-start
[1] 530
... logging to /home/jovyan/.ros/log/036a2732-d0be-11e9-924e-0242ac110002/roslaunch-c1334c08dde3-530.1o
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://127.0.0.1:45525/
ros_comm version 1.12.14

SUMMARY
-----

PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.14

NODES
```

Figure 7: Terminal commands to start up NRP

Once you have done this, you can then log into the NRP system by going to the web address <https://spinn-20.cs.man.ac.uk/user/<username>/proxy/9000/#/esv-private>, where <username> is replaced with your username. This should give you the login screen in Figure 8.

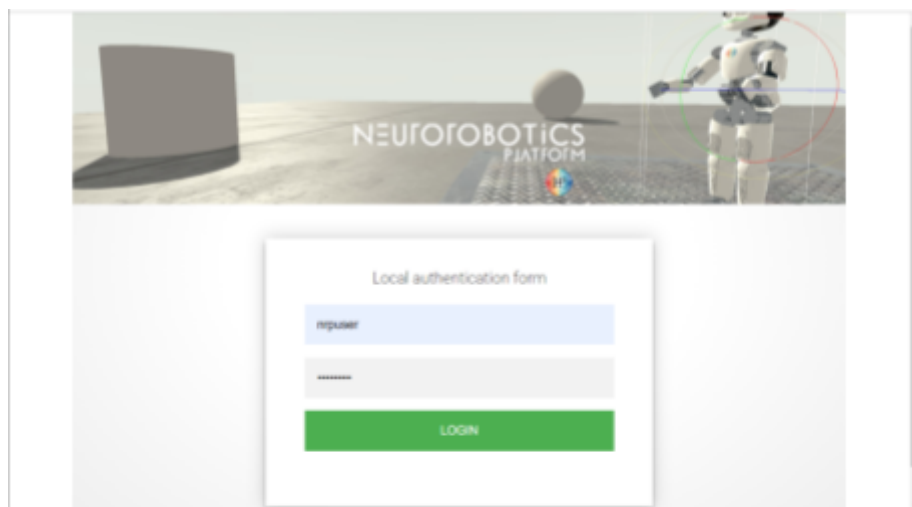


Figure 8: Login screen for NRP



On this screen, log in using the username “nrpuser” and the password “password”. This should ask you to accept the NRP’s terms of service (Figure 9),

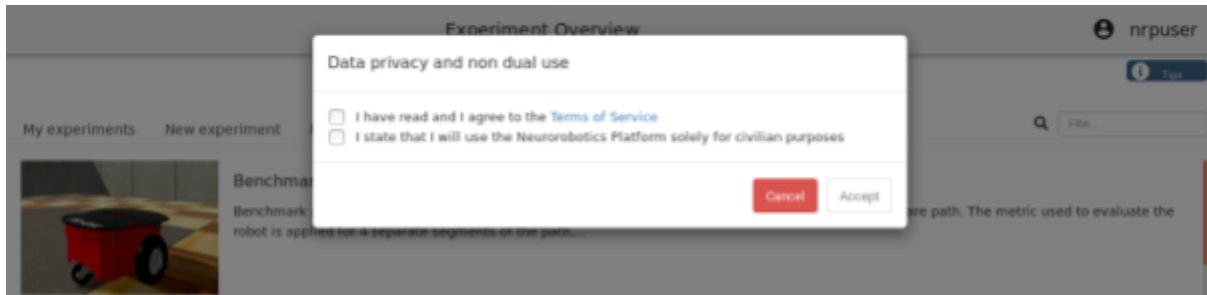


Figure 9: NRP terms of service

and then show you a list of experiment templates (Figure 10).

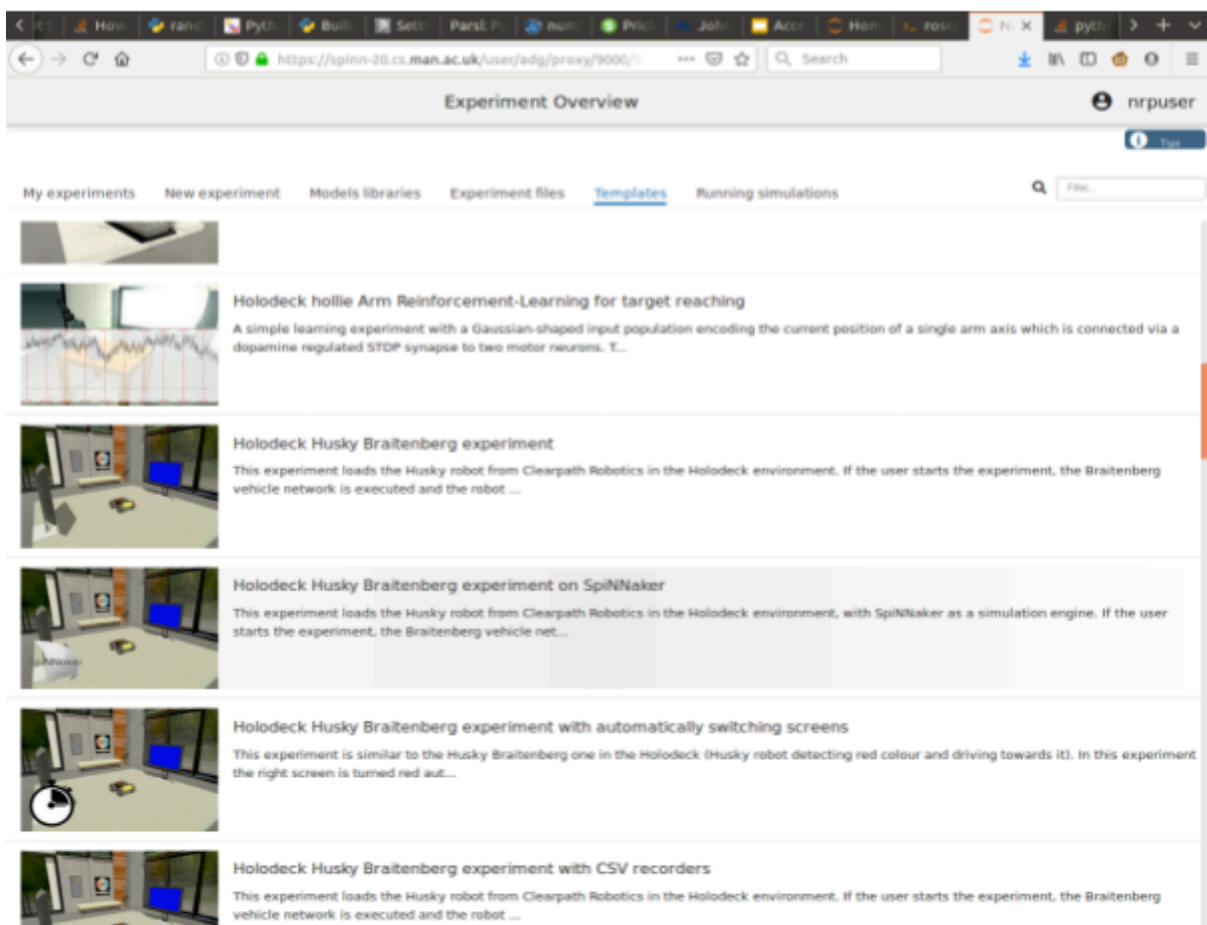


Figure 10: List of NRP experiment templates

### 3.2 Running a job

From this list of experiment templates, the only SpiNNaker experiment you can clone so far is “Holodeck Husky Braitenberg experiment on SpiNNaker”; select this, click on “Clone” and this will add it to your “My Experiments” tab. On this screen, click on the “+ Launch” button and this will load the experiment environment in your browser window. Once it is loaded, click on the Play button; after a minute or so, the robot should start turning. (At this point it is

loading the relevant data onto the SpiNNaker machine - if you switch back to the terminal window where you launched the NRP system, you should see the loading logging information). In this particular experiment, the robot turns to the left until it sees a red screen; you can right click on the screens and select a new colour.

For more details on how to use the Neurorobotics Platform, follow any of the help links once you have logged in to the system.