HPC setup

This project heavily relies on GPU acceleration via JAX to increase optimisation speed. As we cannot expect that everyone has a GPU (that supports JAX) readily available, you can use UGent's high performance computing infrastructure (HPC).

Getting started

Read the following pages from the HPC docs:

1. https://docs.hpc.ugent.be/introduction/
2. https://docs.hpc.ugent.be/getting_started/
3. https://docs.hpc.ugent.be/account/
4. https://docs.hpc.ugent.be/connecting/
5. https://docs.hpc.ugent.be/web_portal/
6. https://docs.hpc.ugent.be/setting_up_python_virtual_environments/

Issues or questions?

Check the 'Known issues and Solutions' section below. If the issue persists, contact teaching assistants:

- Dries Marzougui via dries.marzougui@ugent.be
- Matthias Pex via matthias.pex@ugent.be

Start email subject line with [SEL3]. Email can be in Dutch or English.
Always send your questions to both teaching assistants.

Preparing your HPC Python environment

1. Connect to the HPC via the web portal (if you're not on the UGent network, use a VPN) (if an HPC firewall tab opens, don't close the tab until you are finished with your interactive session).
   If you already have an account with active access: navigate to HPC Login
   a. **If you have used the hpc before**, it is recommended to clear some storage on the $VSC_HOME and $VSC_DATA drives for this project. Your current storage and limits can be checked on the VSC Account Page. Some recommended checks (can be executed from any terminal, be it in an interactive session or via the login node via "Clusters > RHEL9 Login Node Shell Access"):
      i. Comprehensible overview of big files and folders in $VSC_HOME:
         du -h --max-depth 1 $VSC_HOME | egrep '[0-9]{3}M|[0-9]G'
      ii. Comprehensible overview of big files and folders in $VSC_DATA:
         du -h --max-depth 1 $VSC_DATA | egrep '[0-9]{3}M|[0-9]G'
      iii. Deleting $VSC_HOME/.cache:
         rm -r $VSC_HOME/.cache/*
2. Launch a shell on the cluster via the web portal
   a. On the top of the webpage, click on **'Interactive Apps > Shell (tmux)'**
   b. Set cluster to **donphan (interactive/debug)**
   c. After starting the shell using the **Launch button**, you will see it being added as 'Queued' and then 'Starting' in the overview of interactive sessions (see My Interactive Sessions menu item). When your session is ready, click on 'Connect
3. Python setup.
   a. We prepared an install script that does the necessary HPC module loading, Python virtual environment setup and link as a Jupyter Notebook kernel. In your shell execute the following command. While it is running, take a look at the content of the script yourself to see what exactly is happening.

      curl https://raw.githubusercontent.com/Co-Evolve/SEL3-tutorials-2026/refs/heads/main/hpc_install.sh | bash

   b. The created virtual environment is named 'sel3_${VSC_INSTITUTE_CLUSTER}'
      (the variable in this case equals 'donphan' as this is our selected cluster).
      To activate your virtual environment, execute the following command:

      source $VSC_HOME/venvs/sel3_${VSC_INSTITUTE_CLUSTER}/bin/activate

   c. Now a quick test (if it prints nothing, we're fine):

      python -c 'import biorobot'

   d. Python setup complete! As discussed in the HPC docs, **Python virtual environments are cluster specific.** If later in this project you decide to run your experiments on a different cluster (see the Cluster Selection section below), you will need to rerun this installation script while being logged in on the other cluster.
4. During the pip installs, we collected quite some cache and your $VSC_STORAGE might be getting full. Clear it with the following command.

   rm -rf $VSC_HOME/.cache/*

5. During week 2 and week 3 of the course, you will be asked to follow some Jupyter Notebook tutorials that introduce the basics of the project. You can go ahead and download them already via:

   git clone https://github.com/Co-Evolve/SEL3-tutorials-2026.git

Launching a Jupyter Notebook on the HPC

Create an interactive Jupyter Lab session via the web portal

1. On the top of the webpage, click on 'Interactive Apps'
2. Select 'Jupyter Lab' and use the following settings (note: you can increase the 'time' setting for later tutorials):
   a. cluster: donphan (interactive/debug)
   b. time: your choice
   c. Number of nodes: 1
   d. Number of cores: 4
   e. JupyterLab version: **4.2.5 GCCcore 13.3.0**
   f. custom code:

      ml load Python/3.12.3-GCCcore-**13.3.0**
      ml load FFmpeg/7.0.2-GCCcore-**13.3.0**

   g.
   h. Extra Jupyter Arguments: *Blank*
   i. Extra qsub arguments: *Blank*
   j. After starting the Jupyter notebook using the Launch button, you will see it being added as 'Queued' and then 'Starting' in the overview of interactive sessions (see My Interactive Sessions menu item). When your session is ready, click on 'Connect to Jupyter Lab'

Test your setup by creating a Jupyter Notebook (File > New > Notebook)

6. In the 'Select kernel' prompt, you should now see your newly created virtual environment kernel 'sel3_donphan_kernel'
7. Quick test that our JAX install has GPU access. Paste the following in a code cell and execute it (can take a while). This should print `gpu` and `CudaDevice(id=0)]:`
   a. import jax
   b. print(jax.default_backend())
   c. print(jax.devices())

8. Jupyter Notebook setup complete!
9. Warning: Jax can only be loaded by one kernel at a time: **shut down kernels before going to different notebooks**. The overview of kernels can be acquired via the "running terminals and kernels" tab, available in Jupyterlab in the tabs on the left, or in the status bar.

Development options

- Develop locally (running JAX on CPU if no GPU is available), then submit job to HPC.
- Directly on the HPC → interactive job (Donphan)
    - Can connect jupyter notebook or vscode

Cluster selection

- You will do the tutorials on the donphan cluster. This cluster is targeted towards interactive use for development and debugging. As they limit the amount of compute for each session, this cluster is almost always available for use.
  Information on Donphan user limits, processors, etc.: UGent HPC Infrastructure
- For actual experiments (which require stronger compute), you can choose between the Accelgor, Joltik and litleo clusters (these have much stronger GPUs, allowing you to parallelize even more). This stronger compute however comes with a rule: **per group only use 1 of these GPU's at the same time. Furthermore, please do not keep your session or job running idle or longer than needed! Queue times for these clusters can sometimes be quite long, so make sure to first do a test run on donphan (filter outbugs) and choose your experiments wisely!**
- Cluster load: https://shieldon.ugent.be:8083/pbsmon-web-users/
  Cluster load also visible when opening login node shell access
  (via hpc web portal > clusters)

Known issues and solutions

The physics simulator that we're using, MuJoCo XLA (MJX), is still quite new. Consequently, some strange issues might appear during this project. Here, we will maintain a list of possible issues that you might encounter during the project and some solutions to try out. If nothing works, contact Dries Marzougui (see contact info).

- RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so this will likely lead to a deadlock.
  → A warning that can safely be ignored.
- mujoco.FatalError: an OpenGL platform library has not been..
- Solution: set the environment variable MUJOCO_GL to "EGL" before starting your script

    - `export MUJOCO_GL=EGL`
- Whenever you encounter an unexpected error in a jupyter notebook (especially when related to JAX), always start by rebooting your kernel (kernel > shut down all kernels > restart kernel) (Jax can only be used by 1 kernel at a time)
- Issue: Environment returns nan's after a single env.step (and robot disappears in rendering):
    - Solution: recreate your virtual environment
- Mail "HPC HOME directory (almost) full" - Causes instant job submission failure without explanation: HPC Documentation
  Information about your account (including disk usage: VSC Account)
  Usual causes include:
    - Accidentally installing packages in home drive instead of environment (see also §4.d above under "Preparing your HPC Python environment"
    - Storing videos or images in $VSC_HOME directory instead of $VSC_DATA
    - Cache accumulation
  Useful commands:
    - Comprehensible overview of big files and folders in $VSC_HOME:
      du -h --max-depth 1 $VSC_HOME | egrep '[0-9]{3}M|[0-9]G'
    - Deleting $VSC_HOME/.cache:
      rm -r $VSC_HOME/.cache/*
- Error "[Errno 122] Disk quota exceeded" when installing conda packages
    - Make sure environments are stored in $VSC_DATA (cfr. virtual link in the install script in step 3.a above)
    - Make sure $VSC_DATA drive is not full (videos, images, unused environments)
    - Try: rm -r $VSC_HOME/.cache/*
    - Try: conda clean --all
    - Useful command to find big files/directories:
      > du -h --max-depth 1 $VSC_HOME | egrep '[0-9]{3}M|[0-9]G'
      > du -h --max-depth 1 $VSC_DATA | egrep '[0-9]{3}M|[0-9]G'
- Interactive session instantly completes after queuing/starting: Sometimes you need to explicitly put "JupyterLab version" to the same toolchain version as the loaded modules (e.g. GCCcore 13.3.0) (when launching interactive job)
- Interactive session stays in queue: Remember the Donphan user limits UGent HPC Infrastructure
- Error "WARNING:jax._src.xla_bridge:An NVIDIA GPU may be present on this machine, but a CUDA-enabled jaxlib is not installed. Falling back to cpu"
    - Make sure you are in the correct environment
    - pip install --upgrade pip
    - pip install --upgrade "jax[cuda13]"
- Error "Error NumPy 1.x cannot be run in NumPy 2.x.x"
    - Make sure you are in the correct python environment
    - pip install --upgrade numpy==1.26.4
- Error "INTERNAL: cuSolver internal error"
  You can only have one kernel at a time using jax/cuda. Closing a notebook does not kill the kernels. Killing all notebook kernels can be done by clicking in the left column on the tab "running terminals and kernels" on at the bottom left corner of the screen by clicking terminal/kernel icon. Next click on "kernel > shut down all"

Other causes include:
    - problems with the jaxlib installation (see setup earlier in this document)
    - jax-cuda mismatch: check cuda version with nvidia-smi and jax/jaxlib versions with pip freeze | grep -e jax
- The Jupyter Kernel does not work in a Jupyter Notebook: make sure that, when you launch JupyterLab, you use the GCCcore versions that are mentioned in the setup above.
- Terminals are not available in JupyterLab: make sure that, when you launch JupyterLab, you use the GCCcore versions that are mentioned in the setup above.