# Running machine learning experiments on DAS-5, DAS-6, Ada (Bazis) and Snellius

At the VU, we have several sources of compute that you can use for your experiments. For machine learning work the most useful are Snellius, DAS-5, DAS-6 and Ada (formerly named Bazis).

Most of these are well-documented, but to use them effectively for modern machine learning requires a few tricks. The instructions below are for DAS-5, DAS-6, and Snellius, but will translate well to any SLURM-based cluster like Ada.

## The rules

### DAS5, DAS6 ([documentation](https://www.cs.vu.nl/das/))

DAS5 and 6 are lovely clusters for ML work, but remember that their *primary* purpose is for **short**, multi-node jobs. Not the long single-node jobs that ML researchers tend to run. We are guests, so be sure to stick to the following rules.

* During working days (mon-fri), **no jobs lasting longer than 15 mins between 8:00 and 20:00 on the regular nodes.** There are some machines available for longer jobs (see below).
* During weekends, holidays, and outside working hours you are free to schedule long jobs (but make sure they are finished before the next workday starts).
  + Use sbatch --begin=20:00 job.sh to queue an overnight job.
  + Make sure to check in the morning that your jobs have finished. If the cluster is busy they may have started later than the scheduled time.
* **Remember to acknowledge/cite DAS5 or 6 when the experiment leads to a paper.** This helps us to fund future clusters.

On the UvA site, the rules are a bit different, but the UvA site is also much busier as a result.

### Ada ([documentation](https://bazis.readthedocs.io/en/latest/))

Ada (formerly named Bazis) is a production cluster, available only to VU employees or students. Most of the compute nodes in Ada are dedicated to departments that invested hardware money in the system, but there is also a common part that is available for general purpose use by our CS department. At the time of writing it has a small number of A30 GPUs (24GB) available.

Ada is administered by ITvO (a.k.a. IT-for-Research, a subsection of VU/IT dedicated to research support).

### Snellius ([documentation](https://userinfo.surfsara.nl/systems/lisa/getting-started))

Snellius is a production cluster, which means you can schedule long jobs whenever you like. Unlike DAS-5/6, you may need to wait a while for a job to start. You can schedule debug runs that will start immediately by making them shorter than 15 mins.

Note that your Snellius account comes with a fixed budget, so be sure to start carefully, with short runs. On Snellius all GPU nodes have multiple GPUs, but you can run jobs that share the node with other jobs.

Your total budget for a project will be between 16 and 32 “GPU days”, depending on your situation. You can make this last longer by requesting half-GPUs (this will still give you 20Gb of GPU memory). If you also have DAS-6 access, make sure to do test runs and proofs-of-concept there before scaling up to Snellius.

Snellius also hosts [several models and datasets](https://servicedesk.surf.nl/wiki/display/WIKI/Available+datasets+and+models+on+Snellius#AvailabledatasetsandmodelsonSnellius-Datasetormodelnotlisted?) locally, so you don't have to download them.

## Getting an account

Accounts are only available for those working at the CS department of the VU or PhD/MSc students supervised by someone at the CS department.

### What do I need access to?

If you are a **researcher** in the department, you should request access to DAS-6, Snellius and Ada. They offer complementary things, so you may well end up using all three.

If you are a **student**, what you need depends on the project:

* If you are doing an educational project, where the main aim is for you to learn how to use a cluster and a GPU, you should request access to DAS-5. The GPUs there are a bit older, and no longer suitable for most production work, but they are plenty powerful, and you won't need to worry about getting in the way of researchers.
* If you are doing serious research that may turn into a paper, you should request DAS-6 and/or Snellius access. Consult with your supervisor first. Note that if you have DAS-6 access, you should follow the office hours rule, even if you see other users not following it (they may have special permission).
  + Requesting access to both Snellius and DAS is usually a good idea. You will initially get a limited budget on Snellius, so it's good to use DAS to practise a little with the use of a cluster. If you're worried about bothering the researchers, you can request access to DAS-5 to practise.

The following are good rules of thumb:

* Any work that doesn't require a GPU -> DAS-5
* Any project by a BSc student, including the thesis -> DAS-5
* Any non-thesis project by an MSc student -> DAS-5
* A simple MSc thesis, with limited demands. A case where neither the student nor the supervisor know what's needed. A student who is not yet very familiar with Deep Learning. -> DAS-5
* A more ambitious MSc project where at least the supervisor has a clear idea of what's required. A student who has taken the course *Deep Learning*, and possibly has some experience on the cluster. -> Snellius (and in special cases DAS-6).

### DAS5/6

Get permission from your supervisor. Then send an email to Kees Verstoep, [c.verstoep@vu.nl](mailto:c.verstoep@vu.nl), asking for an account (include your VUNet ID if you have one). Put your supervisor in cc. If you're at the UvA, you can ask Dennis Koelma, [koelma@uva.nl](mailto:koelma@uva.nl).

**If you have trouble logging with the account you are given do not contact the cluster administrator. Ask your supervisor for help first (also read the instructions under *Accessing the Servers* carefully).**

### Ada

A form for obtaining an account can be found on the VU IT service portal under

[Home > IT >My work field > Research > HPC Cluster Computing](https://services.vu.nl/esc?id=emp_taxonomy_topic&topic_id=68b9e61f97cf09d0e553359fe153af51)

### Snellius

For **employees**, make a request in [the Snellius system](https://userinfo.surfsara.nl/systems/lisa/account) and include your supervisor. If this doesn't work for any reason, our VU contact for Snellius is Peter Vos. If you need GPUs, make sure to request it specifically and include Peter Vos on the ticket as below.

There isn't a field in the ticket form to tag Peter Vos directly. Just make the ticket and add him (share) afterwards. If you log in through your VU account, we will probably be added automatically to any budget requests.

For **students**, the following instructions work for now:

Make a [request for an account](https://servicedesk.surf.nl/jira/plugins/servlet/samlsso?redirectTo=%2Fservicedesk%2Fcustomer%2Fplugins%2Fservlet%2Fdesk%2Fportal%2F13%2Fcreate%2F51), specifying that you need GPU access.

The request should go directly to the ITvO sysadmins. For a small/default number of SBU (see below), the request should be granted. For larger amounts, make sure to include the name of your supervisor so they can confirm (and where applicable assign the use to the correct budget).

Make sure to add the name **and email** of your supervisor. Ideally attached to the ticket, but if that doesn't work, you can add them in the text of the ticket.

#### How much SBU do I need?

SBU is the unit in which your budget is expressed (only on Snellius, the other clusters are not budgeted). 1 SBU is equivalent to one hour on one CPU core. An hour on half an A100 GPU (the cheapest available on Snellius) costs 64 SBU.

To estimate the required budget consider how long a single run of your experiment might take on a GPU like this (high-end with 20Gb of memory). For proper research, where you develop a model from scratch project you will likely run about 400 experiments (debugging, trying different datasets, and different hyperparameters), so if one run takes 2 hours, you'd need about 50 000 SBU to get started. If you don't want to develop a model from scratch, but just run something that already exists, you'll need fewer experiments, maybe as low as 40, to get the results you need.

Note that this grows quickly if you need to do multi-GPU training. Running a full 4-GPU node (160 Gb of memory) for 5 days will cost you 61 000 SBU.

*We have plenty of budget at the moment, but do consider that 100K SBU corresponds to 100E of electricity, and an 800E bill when we have to pay outside of our budget, so you shouldn't use your budget carelessly.*

If your budget is higher than 500 000 SBU, it may be that you can [request a compute grant through NWO](https://userinfo.surfsara.nl/systems/shared/iris-ncf) for 1M SBU (these are usually easy to obtain for staff members).

**Make sure to perform a careful estimation of what you need and to add it to the ticket. It should be a budget for an initial run of experiments: you can always ask for more if you find that you need it.**

See also:

* <https://servicedesk.surf.nl/wiki/display/WIKI/Snellius+partitions+and+accounting>
* <https://servicedesk.surf.nl/wiki/display/WIKI/Estimating+SBUs>

### Where do I get help?

If something doesn't work, first of all make sure to reread the instructions carefully. For many issues that are generic to all clusters of this type, you can google for a solution. In that case, it helps to know that these clusters run an operating system called [SLURM](https://stackoverflow.com/questions/tagged/slurm).

**For students** If you still cannot get something to work, the first person to ask for help is your supervisor. They will most likely use the cluster themselves, and can help with basic problems like connecting and downloading files.

If you are a student, you should not contact the cluster administrator directly. If you think there is a problem that only they can solve, discuss it with your supervisor and they will tell you what to do.

**For researchers** If these instructions are insufficient, please check first if there is somebody within your group who can help. In extreme cases you can contact the cluster administrator, but try to avoid this, since it isn't very scalable.

## Accessing the servers

### External access

If you want to use DAS-5, DAS-6 or Ada from outside the university, access is either by means of a suitable VPN network or via an ssh “stepping stone” machine, which allows remote access, but is independent of the clusters themselves.

* **VU users** the ssh stepping stone ssh.data.vu.nl should be available with your VUnetID (this is three letters and three numbers, e.g. pbm370).
* **UvA users** You can use a VPN, and use ssh directly. Instructions can be found here: <http://www.uva.nl/vpn>

To use the VU ssh stepping stone, the simplest approach makes use of the “jumphost” functionality in ssh. The following command for DAS-5 will be asking two passwords, namely your VU password for ssh.data.vu.nl, and then your DAS-5 password:

ssh -J <your vunet id>@ssh.data.vu.nl <das id>@fs0.das5.cs.vu.nl

For DAS6, Ada and Snellius replace the part in red by the relevant hostname:

* DAS-6: fs0.das6.cs.vu.nl
* Snellius: snellius.surf.nl
* Ada: ada.labs.vu.nl

Note that, although your DAS login id may be the same as your VUnetID, the passwords are different.

If this doesn't work try connecting to the data server first with just   
 ssh <your vunet id>[@ssh.data.vu.nl](mailto:VUNETID@ssh.data.vu.nl)

If this works, you will be given a shell, from which you can then ssh directly to the cluster with  
 ssh <das id>@fs0.das5.cs.vu.nl

the command above just combines these steps into one. Separating them can help you figure out which server you are unable to connect to.

Once you have this working you can also try tunneling, which allows you to access a local ip like it's the server (this may be useful for file access or things like debugging). You can also set up ssh aliases to log in more quickly. Both are described at the bottom of the document under "Advanced Tricks".

## Installing anaconda

*If you don't like anaconda, scroll down for instructions on how to set up python environments without it.*

SLURM (the cluster OS) comes with a packaging system, but you will often need obscure python packages and nightly releases of tensorflow or pytorch. The easiest option is to install *Anaconda*.

### DAS-5 and DAS-6

On DAS you have a limited amount of space in your home directory, not enough to install anaconda and the packages you need. You can install anaconda, but have to make sure you install it on your scratch space.

On the server:

cd /var/scratch/<username>

wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

Make the installer executable

chmod +x <the filename of the downloaded anaconda installer>

Execute the installer

./<the filename of the downloaded anaconda installer>

After accepting the licence agreement, you will be asked for the path where to install. **Specify /var/scratch/<username>/anaconda3**

*That is the important bit. Don't miss it or it will install in your homedir and you will run out of space quickly.*

The installation will take a while, and at the end it will ask whether you want to run conda init; answer yes.

Now, logout and back in again. Now, if you execute

conda activate

You should be in your fresh conda environment. If you run which python, it should now point to your scratch directory.

### Snellius

On Snellius the amount of space in the homedir is sufficient to install any packages you want. Install anaconda or miniconda by downloading the command line linux installer with wget and, and running it:

wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

Make the installer executable

chmod +x <the filename of the downloaded anaconda installer>

Execute the installer

./<the filename of the downloaded anaconda installer>

## Installing Pytorch for the GPU

Pytorch can be installed through anaconda with a pip command, but we need to make sure that we are installing for the correct version of CUDA that is available. Pytorch offers a little widget to generate the correct install command. Make sure to select Linux as the OS, the right CUDA version (see below) and **pip** as the installer.

<https://pytorch.org/get-started/locally/>

At the moment, pytorch can be installed for CUDA versions 11.7, 12.3, and 12.6. You should check which is available on the cluster. On DAS5/6, you can directly type

module load cuda (tab to autocomplete)

to see the available modules. Currently, 11.7 to 12.6 should be available. (see also the example script below).

**Be sure to test whether pytorch can actually *see* the GPU (see below for an example script). If it can't, it will often fail silently, and run your experiment on the CPU without telling you.**

On Snellius, a working version of cuda appears to be loaded by default, and module statements usually aren't necessary. If you run into problems, there are modules available for different versions.

The latest version may not work with any modules available on the cluster, in which case you'll have to go back to an earlier version. CUDA is backwards compatible, which means that if your pytorch is compiled for a newer version of CUDA than the one you've loaded, it will (probably) work too.

## Running jobs

Jobs are queued on the SLURM through the **sbatch** command. DAS, Snel lius and Ada allwork more or less the same. To define a job, you create a shell script that will be executed on the node. What you need to do is create a job script (see below) and give it a name (here example\_script.sh). Then, you submit the script to the job queue using

sbatch example\_script.sh

To check what the status is of you job, run

squeue -u $USER

To cancel your job, get the JOBID from the previous command and run

scancel <YOUR-JOB-ID-HERE>

**Please note the following carefully before starting your job:**

* **Unless you know why you need it, do not use srun to queue your jobs. The difference between that and sbatch is explained** [**here**](https://stackoverflow.com/questions/43767866/slurm-srun-vs-sbatch-and-their-parameters)**.**
* **On DAS-6 do *not* schedule jobs on the fatq unless you need a lot of CPU memory (i.e,. hundreds of gigabytes).**

### Jobscript example

Here is an example of a script requesting a GPU node and executing a python experiment:

#!/bin/bash

#SBATCH --job-name=test

#SBATCH --time=72:00:00

#SBATCH -N 1

#SBATCH --ntasks-per-node=1

#SBATCH --partition=<partition name, see below>

#SBATCH -C <resource name, see below> # this line is usually not needed

#SBATCH --gres=gpu:1

## in the list above, the partition name depends on where you are running your job.

## On DAS5/6 the default would be `defq` on Snellius the default would be `gpu` or `gpu\_shared`

## Typing `sinfo` on the server command line gives a column called PARTITION. There, you can find the name of a specific node, the state (down, alloc, idle etc), the availability and how long is the time limit . Ask your supervisor before running jobs on queues you do not know.

# Load GPU drivers

## Enable the following lines for **DAS5 or DAS6**

# . /etc/bashrc

# . /etc/profile.d/lmod.sh

# module load cuda12.3/toolkit

# module load cuDNN/cuda12.3

## For **Snellius**, modules are usually not needed

## <https://userinfo.surfsara.nl/systems/shared/modules>

# This loads the anaconda virtual environment with our packages

source $HOME/.bashrc

conda activate

# Base directory for the experiment

mkdir $HOME/experiments

cd $HOME/experiments

# Simple trick to create a unique directory for each run of the script

echo $$

mkdir o`echo $$`

cd o`echo $$`

# Run the actual experiment.

python /home/username/git/project/experiment.py --parameter

For a simple script to check if everything is loaded correctly (including the GPU drivers) replace the python command above by

python <<EOF

import torch

print(torch.cuda.is\_available())

EOF

### Notes on running jobs

* When you have scheduled a job, it does not start running immediately. Hence, it is not safe to start changing your code (e.g. by pulling it from git). Also, when the job has started running, you have to be careful since changes to your code might affect the program which is active anytime something is loaded in dynamically.
  + In python, it is often fine to pull after a job has started, since the code is loaded into memory when it is imported at the start of the job.
  + In java, changing the code while the script is running will almost always cause a crash.
  + To be absolutely sure and still allow you to pull while jobs are running, you can copy your code to the node before you run it.
* If you are running a single-node job, you can just put whatever commands need to be run inside the job script. If you are provisioning multiple nodes and you want the same script to run on each, put srun -N8 in front of the call to the script (where 8 is the number of nodes).

### DAS 5 partitions

* defq: default GPU partition. 15 minute limit during working hours. Default GPU (resource name) TitanX.
* proq: long GPU jobs (three nodes only). Default GPU (resource name) RTX2080Ti.
* longq: long CPU jobs, no GPUs
* knlq: Knight's landing CPUs, long jobs allowed, no GPU

More details <https://www.cs.vu.nl/das5/special.shtml>

### DAS 6 partitions

DAS 6 is the most heterogeneous of our clusters: it contains many different nodes with different kinds of GPUs. You most likely want to use the defq partition, but with a specified resource. The most common GPU type is A4000 with 16 GB memory, but there are also A5000 and A6000 nodes which are faster and that have more memory. See the documentation for what's available: <https://www.cs.vu.nl/das/special.shtml>

### Snellius partitions

* gpu is most likely the one you want. All nodes are shared nodes, so with the configuration above, you will get one quarter of a node. This is equivalent to gpu\_a100. If you need bigger GPUs, you can run on the gpu\_h100 partition, which has Hopper GPUs with 80Gb of memory. These cost more per GPU hour, but the cost per Gb of GPU memory is lower.
* Other partitions: <https://servicedesk.surf.nl/wiki/display/WIKI/Snellius+usage+and+accounting>

## Running long jobs

On Snellius, you can schedule jobs up to 5 days (120 hours) at any moment you like. Long jobs will get low priority, so you'll have to wait until they start. Make sure to test everything carefully in shorter jobs (<10 minutes will usually start immediately).

### DAS5

On DAS5, on the default partition (defq) long jobs may only be run at night or in the weekend.

If you have an experiment that really needs to run a long time, and you can't run at night or in the weekend, there are some nodes available for longer experiments. If you don't need a GPU, you can use the longq partition:

#SBATCH -p longq

(remove the --gres and -C parameters)

If you do need a GPU, there are three nodes with RTX2080Ti's in the proq partition. Use these nodes with the following sbatch parameters:

#!/bin/bash

#SBATCH --time=72:00:00

#SBATCH -N 1

#SBATCH --ntasks-per-node=1

#SBATCH -C RTX2080Ti

#SBATCH -p proq

#SBATCH --gres=gpu:1

Make sure you load the appropriate version of the CUDA toolkit consistent with your pytorch version. There are not many of these nodes, so please only use them if you can't use the regular nodes.

Finally, if you have a good reason to run long jobs in the main partition (conference deadline, large experiment etc.), you can always ask Kees for permission. You will usually get it, so long as you explain the reason and a clear timeframe during which you will have to monopolize the cluster.

## Monitoring and interactive jobs

If you have a job running, you can log in to the node its running on simply with

ssh node067

Where node67 is the name of the node (the rightmost column in the squeue output). This can be helpful, for instance to monitor your job. Be careful that you get the right node, because you can also log in to nodes that aren't running your jobs, potentially disturbing other people's experiments.

You'll have to reload the NVIDIA drivers again if you want to do anything GPU-related. For instance, you might do

module load cuda12.3/toolkit

module load cuDNN/cuda12.3

followed by

nvidia-smi

To check that the GPU is being used, and to see how much of its memory is used.

If you only want to use a node for interaction, you can run an srun command followed by all the parameters you would normally add to the top of your job script, and the switch --pty-bash -i. This will requisition a node, and give you a bash shell on that node (as soon as the job runs). For instance:

srun -N 1 --ntasks-per-node=1 --time=02:00:00 -p defq --gres=gpu:1 --pty bash -i

Don't forget to load the CUDA modules if you want to use the GPU.

## Common problems

### Out of space

The amount of space we get on DAS in the home directory is very limited (4 GB). We get a bit more in the scratch drive (40 GB), which can be increased further by mailing das-account@cs.vu.nl. On Snellius, you get 200GB in the home dir, but sometimes even that isn't enough.

A simple workaround is to put your data somewhere publicly accessible, and to *download it in the job script every time*. Each node has a local drive with at least a terabyte of space. On DAS, this is under /local/<userid>.

If you are an employee, you get 500GB on surfdrive. Any file on surfdrive can be made publicly accessible. This is through a webpage, but that webpage gives you a direct download link.

Similarly, you can put your data on dropbox. There, you can turn your public link from a webpage to a direct download link by changing download=0 to download=1 in the url.

*Downloading gigabytes of data for every single run of an experiment may seem excessive, but remember that the university internet is very fast. It will likely only take an hour or so, even for very big data. Especially if you host your data on Surfdrive, which should be well connected with Snellius.*

On DAS, remember to delete the data at the end of the script. On Snellius, this should happen automatically.

### Wrong python being used

Sometimes you can activate the right virtual environment in your jobscript with something like

source /home/myname/.bashrc

conda activate myenv

and sometimes this doesn't work (for some reason). You can troubleshoot this by adding

which python

python --version

to your script and seeing which python executable is being used when it runs. If you can't get the right one activated, you can always replace the line

python -u my\_experiment.py

with

/var/scratch/myname/miniconda3/bin/python -u my\_experiment.py

or whatever the full path or the correct python executable is. This will automatically ensure that all the packages from that environment are also used.

### Snellius: connection refused

When you enter the wrong password 5 times, the intrusion detection system will lock you out for 24 hours. This is very easy to do when you are, for instance, setting up your sFTP client. If this does happen, you can still ssh in from a different IP address, so if you have access to a different server, you can still work from there.

## Tools and applications

### File transfer

**Code** You should keep your code in git (e.g. through github or Codeberg). You can then simply push locally, and pull on the cluster. As a side benefit, your code is very regularly checked in.

**Files** For transferring files, like the results of an experiment, you need an FPT client.

The following clients work well:

* macOS: Cyberduck, FileZilla (Cyberduck is a little nicer, but doesn't always work on Snellius)
* Windows: FileZilla
* Linux: You can usually add the sftp url (sftp://username@localhost:3333 with a tunnel) directly to the location bar of Nautilus (or other file explorers) to mount the sFTP connection as a file system. From the command line, the easiest is to use scp.

You can either set a jump host (ssh.data.vu.nl) in the connection settings, or make a tunnel (see below) and connect the client to, for example, 127.0.0.1:444 . If you have an SSH alias set up, Cyberduck can import the connection settings automatically.

### Logging experiments

**Wandb** No changes are required for weights and biases. Just run as normal, and the plots show up in the online dashboard.

**Optuna** Log locally to an sqlite file. Copy it over from the server and run optuna-dashboard locally.

**Tensorboard** Log to tensorboard as normal and to transfer the log files to your local machine over sFTP. You can then run the tensorboard server locally to plot your progress.

### Git LFS

Git is installed on DAS, as is Git LFS (for large files).

Depending on the current version of git, you may need to clone your repository with

git lfs clone <url>

### Pytorch geometric

#### On DAS5/Snellius

To install pytorch geometric, you can run

pip install torch-scatter -f <https://pytorch-geometric.com/whl/torch-1.9.0+cu102.html>

pip install torch-sparse -f <https://pytorch-geometric.com/whl/torch-1.9.0+cu102.html>

pip install torch-geometric

#### On DAS6

With cuda 12.3 and conda installed, run

conda install pyg -c pyg

### Tensorflow

This has not been tested yet, but the principle is the same as above. Check which CUDA version is available on your cluster, and install the appropriate version of tensorflow through pip or conda.

*Tensorflow seems to be a little more particular about getting the exact right version of CUDA.*

## Advanced tricks

### External access via an SSH tunnel

An alternative approach is to make an SSH tunnel through the ssh stepstone.

You create a tunnel with:

ssh VUNETID@ssh.data.vu.nl -L 3333:fs0.das5.cs.vu.nl:22 -N

This logs in on ssh.data.vu.nl and tunnels all traffic on port 3333 on localhost to port 22 on the VU DAS-5 login node. **NB: Use your normal VU password here, not the one you were sent for DAS or Snellius.**

You can now open a new terminal and log in to DAS-5 with

ssh DASID@localhost -p 3333

See below for instructions on how to transfer files from and to the server.

If you have trouble getting this to work, try logging in to the VU server normally with ssh VUNETID@ssh.data.vu.nl. Then, from that server, log in to the head node of the cluster with ssh DASID@fs0.das5.cs.vu.nl. This should tell which part of the login process doesn't work.

*Windows users may have better results with* [*Putty*](https://putty.org/) *than with the prompt/command line.*

*If you get tired of setting up the tunnel every time, there is a faster set up detailed under "*[*Advanced Tricks*](#_gkeb2qvmzixt)*"*

For the other clusters, replace the DAS5 hostname with the following:

* DAS-6: fs0.das6.cs.vu.nl
* Snellius: snellius.surf.nl (BTW: Snellius also allows direct access)
* Ada: ada.labs.vu.nl

### *If you use multiple clusters, it can be helpful to give each its own port to tunnel over. E.g. 4444 for DAS-6, 5555 for Snellius, and so on.*

### Logging in more quickly (mac and linux)

If you're tired of setting up a tunnel every time you want to log in, here's how to streamline the process. This is for Snellius, but it should also work for DAS.

First, [generate a key pair for Snellius](https://servicedesk.surf.nl/wiki/display/WIKI/SSH+public-key+authentication) called Snellius, and [register it with the CUA portal](https://portal.cua.surf.nl/). This will allow you to log into Snellius without typing your password (for ssh.data.vu.nl a password is the only way to log in, sadly).

Then, open .ssh/config locally and add the following:

Host vu-data

HostName ssh.data.vu.nl

User <vu user name>

Host snellius

HostName snellius.surf.nl

User <snellius user name>

ProxyJump vu-data

IdentityFile ~/.ssh/snellius

Then, you can log into Snellius with a simple ssh snellius.

CyberDuck can read your ssh config file, and use these profiles directly. Just click open connection and add "snellius" as the name. The rest of the connection details will be filled in automatically.

*You can skip the identity file if you don't mind giving your password twice.*

### Using multiple GPUs

There are several ways to run your code in parallel over several devices, either on a single node or distributed over several nodes. See [this lecture](https://dlvu.github.io/sa/#video-072) for an overview of the options.

The quick and dirty way to do this is the pytorch [DataParallel wrapper](https://pytorch.org/docs/stable/generated/torch.nn.DataParallel.html). This requires just a single change in your code, wrapping your model with this custom object. The downsides are that this only works if all the available GPUs are on one device, and that this is not using all GPUs maximally.

For more efficiency, and flexibility, use the [distributed data parallel approach](https://pytorch.org/tutorials/intermediate/ddp_tutorial.html). This requires a little bit more manual work, but allows you to distributed more efficiently, and over multiple nodes. You can make your life easier by using a framework built on top of pytorch like [pytorch lightning](https://lightning.ai/docs/pytorch/stable/).

If your model is too big to fit into memory, you'll need a different strategy. Note that if this is the case, you'll likely need substantial data and compute as well, so make sure you have a reasonable team. In this case fully sharded parallelism (FSDA) is currently the most popular approach. This is available in pytorch and in many other frameworks.

### Running hyperparameter sweeps (task arrays)

If you need to test multiple values of some parameter quickly, a simple way to do that is to fire sbatch with an *array* parameter. That looks like this

sbatch -a 2,3,4 myscript.sh

This schedules three copies of the same job. Each copy receives a task id (2, 3, or 4), which is an integer that you can use to make sure each copy behaves slightly differently. The task id is stored in the environment variable SLURM\_ARRAY\_TASK\_ID. The jobs will run in parallel in different nodes.

For instance, if you want to try three different batch sizes, you could run your experiment (inside myscript.sh) with

python -u experiment.py --batch-size $SLURM\_ARRAY\_TASK\_ID

and schedule the job with

sbatch -a 16,32,64 myscript.sh

As you can see, the task ids can be any sequence of integer values. With a little creativity you can do quite a lot with this. For instance, to test different learning rates, you can use

python -u experiment.py --learning-rate 1e-$SLURM\_ARRAY\_TASK\_ID

and schedule the job with

sbatch -a 2,3,4 myscript.sh

To test learning rates 0.01, 0.001 and 0.0001.

If you are testing a large range of values you can use a command like

sbatch -a 7-16 myscript.sh

The range 7-16 is inclusive on both sides, so this runs jobs for array values 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16.

### Mapping task ids to parameter values

When you cannot represent your parameter values as integers, you'll need to map the current SLURM\_ARRAY\_TASK\_ID value to the value you need. Here's how to do that in bash.

#!/bin/bash

#SBATCH --job-name=midigpt

#SBATCH --time 00:30:00

#SBATCH -N 1

#SBATCH --ntasks-per-node=1

#SBATCH --partition=gpu\_titanrtx\_shared

#SBATCH --gres=gpu:1

#SBATCH --output=job\_%A\_%a.out

source /home/bloemgpu/.bashrc

lrs=(1e-6 3e-6 1e-5 3e-5 1e-4 3e-4) # the array of learning rates

lr=${lrs[SLURM\_ARRAY\_TASK\_ID]} # select the current learning rate

echo $$ $lr # print the current value to the log file

mkdir o`echo $$`\_$lr # create a separate directory for the run

cd o`echo $$`\_$lr

# this is all one line

python -u /home/bloemgpu/git/my-project/experiments/experiment.py --lr $lr --other-parm 1

And then schedule with sbatch -a 0-5 myscript.sh.

For more elaborate hyperparameter sweeps, you can use Weights and Biases or Optuna.

### Without Anaconda (plain virtual envs)

An alternative to Anaconda is to create Python virtual environments manually. These are custom Python environments that are separated from those of the host, and in which you can install packages without interfering with those on the host.

*Anaconda is built on top of venv's, so if you already have anaconda installed, you can use that to create your virtual environments.* [*https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/20/conda/*](https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/20/conda/)

As explained above, the home directory size is limited, which makes the scratch directory the best place to setup your Python VE.

1. Create virtual environment

$ python[<python\_version>] -m venv <name>

2. Load local variables

$ source bin/activate

3. Install packages (optional)

$ bin/pip install <package>[==<package\_version>]

4. Exit environment

$ deactivate

You can also link to the Python executable from your job file by using:

PYTHONENV=`realpath $HOME/scratch`/<name of VE>

### Running a Wikidata instance for SPARQL queries.

Using qEndpoint and HDT, it's possible to run a lightweight SPARQL endpoint for wikidata or any other knowledge graph for which you have an HDT dump. [You can find instructions here](https://docs.google.com/document/d/1NPiHqk5bwiUw5jVHiQnUfJ2NvuO1QTOHRgKm1jyFFzY/edit?usp=sharing).

## Tunneling to the compute node/Debugging on a server using an IDE

We do not recommend connecting your IDE directly to the cluster unless you need it for specific reasons (like complex debugging). Pushing and pulling over github is the safest way to transfer code. A laggy update can easily cause you to run old code and waste your budget. The tunneling described here may be useful for other tricks like running notebook servers (again, not the recommended way to develop).

*This section is new, please let* [*m.cochez@vu.nl*](mailto:m.cochez@vu.nl) *know if there are issues. These were only tested on linux, may the OS warriors be with you on your journey.*

*It is possible to debug using an IDE on a node of the HPC cluster. But, it requires ssh tunnelling over the login node (possible on top of tunnelling you already do to connect to the login node.*

The first thing you need is a free node on the cluster. If there is no free node, there is no place for you to debug.

To allocate this node for you, you can use the salloc command. This command will try to acquire the resources requested and immediately run the command specified. If no command is specified, it seems to depend on the cluster what happens. On snellius, you ssh to the node allocated, on DAS you stay on the login node (but can then use squeue to figure out what node was allocated and ssh to the node if you want).

On snellius, the following seems to work to allocate one GPU on a GPU node (There are 4 GPUs/node).

salloc --partition=gpu --gpus-per-node=1 -t 00:10:00

On DAS, the following would give you an allocation on a A6000 GPU:

salloc --partition=defq --gres=gpu:A6000:1 -t 00:10:00

Now, if the allocation is granted, get the name of the node ( node[0-9][0-9][0-9] on das, [tgfh]cn[0-9][0-9][0-9] on snellius )

Now, we need to setup the tunnelling. What we want is something like this:



The IDE needs to have ssh access to the node, but in order to get there, we need to tunnel it, first through an access node (if there is one), and next through the login node.

To make this easier on the command line, we can use an ssh config, which is a text file stored in ~/.ssh/config In this file, you can specify the hosts you use and their configuration. That way you do not need to type long ssh commands.

Your setup could looks like this

# only needed for DAS access from outside the university network

Host vudata

Hostname ssh.data.vu.nl

User <your vunet ID here>

PubkeyAuthentication yes

Host das6FromHome

Hostname fs0.das6.cs.vu.nl

User <your VUNET ID here>

ProxyJump vudata

Host das6

Hostname fs0.das6.cs.vu.nl

User <your VUNET ID here>

With this setting, typing ssh das6FromHome instruct ssh to use vudata as a proxy. In turn, vudata is an alias for ssh.data.vu.nl

This gets us to the login node. We still need to go one step further to get to the compute node. So, let's do that. Assume we are on das and we got node003 assigned. Now, we can add the following to our config.

Host node003

HostName %h

ProxyJump das6FromHome

User <your VUNET ID here>

Now, you could type ssh node003 on your local terminal to connect all the way to the compute node. It is this ability we will use to attach our IDE.

For snellius, the config could be like this:

Host snellius

Hostname snellius.surf.nl

User <your surf username here>

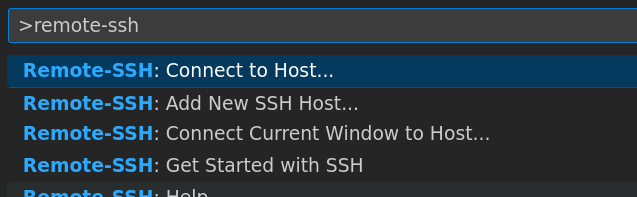
Host gcn\*

HostName %h

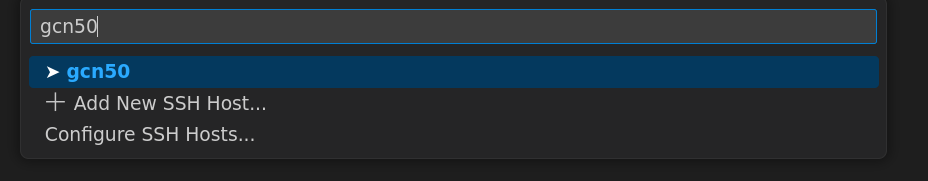
ProxyJump snellius

User <your surf username here>

In your IDE (I use vscode, it might work in others as well, see <https://servicedesk.surf.nl/wiki/display/WIKI/Using+PyCharm+and+other+JetBrains+IDEs+for+remote+development> for instructions with pycharm), install the Remote - SSH extension (ms-vscode-remote.remote-ssh). Once that is done, press ctrl-shift-p to open the command palette and type remote-ssh, select connect to host…

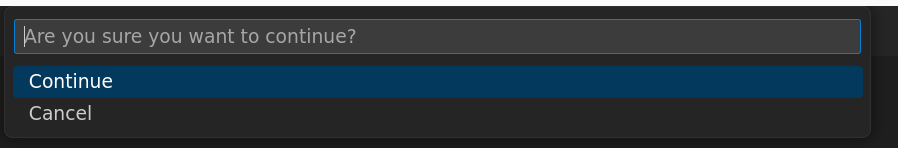


Form the popup list, either select the right configuration (e.g., node003) or type it in the field.

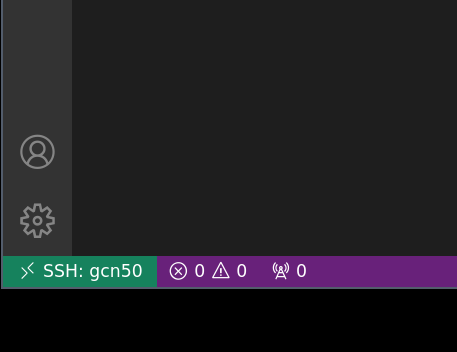


A new editor window will open, with this dialog, choose continue.

(If interested, you can check in the terminal what it is asking about)



After waiting for a moment (vscode instals some parts in your home directory so it can execute things on the server), you are connected. You should see this indicated on the left bottom corner. After the first this will go a lot faster since all parts are already installed.



What is still needed is that you install the extensions you want to use in the remote vscode setup. This is independent of the local installation (I think, at least, you have to reinstall them.)

Once that is done, you can think of any action as if it was done on the server. You can open a server directory and work there, you can open a terminal, and work there, etc. If you debug, you are actually debugging on the server itself!

To load modules (e.g., on DAS) for your debugging session, I ended up adding

module load cuda12.3/toolkit

To the .bashrc file in the home directory.

I also tried these stackoverflow approaches, but did **not** get them to work:

<https://stackoverflow.com/q/63501859> , <https://stackoverflow.com/a/75151719>

There is also a method using RemoteCommand described in <https://servicedesk.surf.nl/wiki/display/WIKI/Visual+Studio+Code+for+remote+development> , but also that I could not get to work immediately, if you get this to work successfully, let us know.

## Other random tips, tricks etc.

* [Joblib](https://joblib.readthedocs.io/en/stable/) is a set of tools to provide **lightweight pipelining in Python**. In particular: (1) transparent disk-caching of functions and lazy re-evaluation (memoize pattern), (2) easy simple parallel computing.