How to use Kaya?

About the UWA High Performance computational research platform.

Pascal Sun

Natural & Technical Language Processing Group

`https://nlp-tlp.org/`

The University of Western Australia

March 24, 2025

Table of Contents

HPC and Kaya

What is Kaya?

- **Kaya** means "Hello" in Nyoongar language[1]
- Nyoongar is the language of the Aboriginal people from the south-west region of Western Australia
- UWA's high-performance computing (HPC) research platform is named as Kaya

[1] Source:https://www.noongarculture.org.au/language/

What is High-Performance Computing (HPC)?

- **Definition:** HPC stands for High Performance Computing
- **Concept:** Using distributed computing resources to accelerate different workloads
- **Core Idea:**
  - Combining computational power of many computers
  - Used for intensive calculations
- Why you need this?
  - You computer do not have enough resources for your purpose.
  - Typically GPU Resources for us (AI/Machine Learning).
  - Some user cases (e.g: Biology) require large CPU memory or intensive CPU resources.

Hardware

Architecture of a Standard HPC

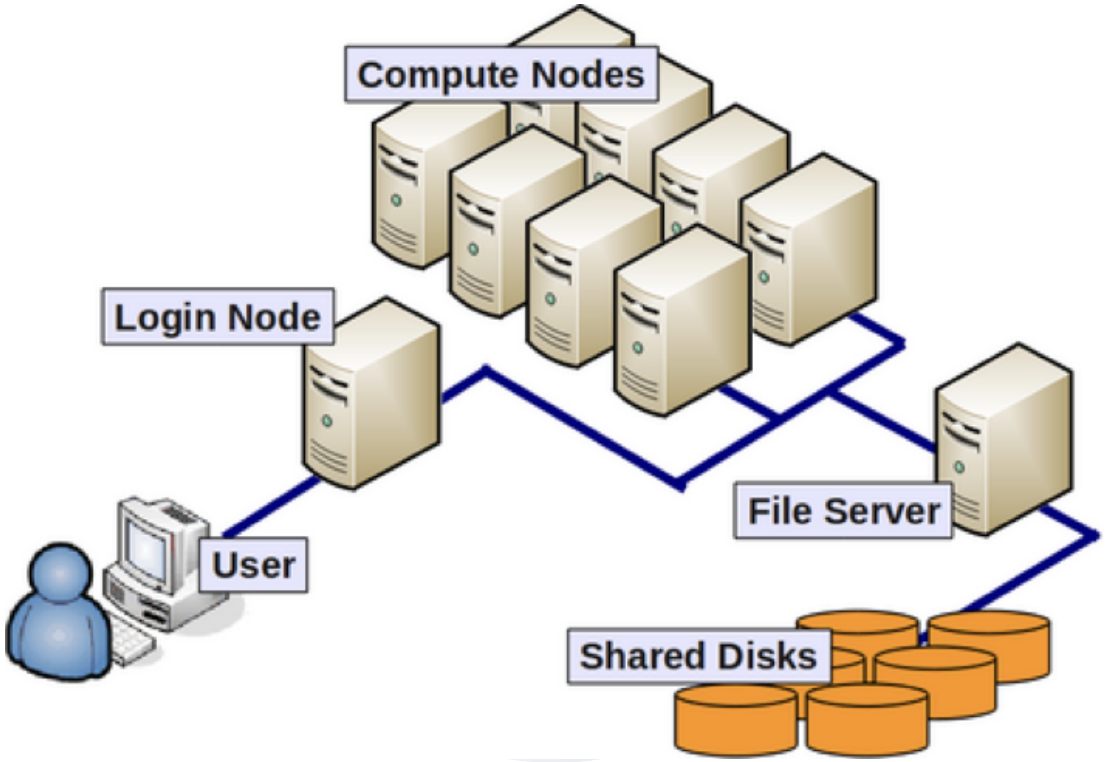


Figure: Standard Architecture of HPC System

Three key components in an HPC system:

Login Node File Server Compute Nodes

Architecture of a Standard HPC

- **Login Node:** Entry point for users via SSH, send commands, queue tasks, etc.
- **File Server:** Shared storage accessible by all nodes including login and compute nodes
- **Compute Nodes:** Perform heavy computational tasks in parallel

Software

How to Interact with HPC?

- Primarily via Linux shell
- Some GUI interfaces available
- CS students should know shell interaction[2][3]

What is a shell?

- Program that takes keyboard commands
- Passes commands to OS to perform
- In general word: This is the tool making you sounds like a hacker.
- Another name of it is: **command line**

Recommended learning:

- What is Linux?
- What is a shell?

For more info: `https://www.tutorialspoint.com/unix/unix-what-is-shell.htm`

[2] Take the unit: Open Source Tools and Scripting, Code: CITS4407

[3] `https://cits4407.github.io/resources/`

Some frequent used commands[6][7]

1. **ssh**: Login to the login node via *ssh* command
2. **cd**: change directory
3. **slurm**: interact with HPC System
   - *sbatch*: submit a job
   - *squeue*: check list of jobs
   - *scancel*: cancel a job
   - *salloc*: allocate a compute node, and access it interactively
4. **module**: Load modules like cuda, conda.[4]
5. **python**, **pip** or **conda**[5]

[4] You can not install any package on the HPC, you can only load modules. apt install is not possible

[5] You can use Conda to switch python versions and install requirements

[6] These commands will cover 90% of your user cases

[7] We will talk about `slurm` and `module` command later, but if you are not familiar with `ssh`, `cd`, `python` and `conda`, make yourself familiar with that.

Software Part in HPC: The Module System

- **Challenge:** Installing packages on HPC systems
  - No root access for users
  - Inefficient to install all possible packages
  - Different versions and conflicting dependencies
- **Solution:** The *module* concept
  - Pre-compiled modules available on the system
  - Load modules as needed
  - Unload when not in use
  - You can not run `aptget install` or `sudo` commands on HPC

Key Benefits

- Flexibility in software management
- Avoids version conflicts
- Efficient use of system resources

Software Part in HPC: The Module System

Frequently Used Commands:

- `module avail`: list all available modules on the system
- `module list`: list all loaded modules
- `module load xxx`: load a specific module
- `module unload xxx`: unload a specific module

These are the essential commands you'll need. **Common module load examples:**

- `module load cuda/12.4`
- `module load conda/xxx`
- `module load gcc/xxx`

Note: Replace 'xxx' with specific versions as needed.

Software Part in HPC: The Module System

After you login to the login node, you can run the commands to check

```
(base) psun@kaya1[TimelineKGE]$ module list
No Modulefiles Currently Loaded.
(base) psun@kaya1[TimelineKGE]$ module avail
------------------------------------------------------------------------------------------------------------ /usr/share/Modules/modulefiles ------------------------------------------------------------
dot  module-git  module-info  modules  null  use.own

------------------------------------------------------------------------------------------------------------ /uwahpc/centos8/modulefiles/apps ------------------------------------------------------------
abaqus/2023           comsol/5.6a                 gromacs-gpu/2020.4(default)  leptonica/1.83.1(default)  orca/5.0.2              r/4.0.5              sqlit
ansys/v195            comsol/6.0                  gromacs/2020.4(default)      matlab/R2023b              orca/6.0.0(default)     r/4.2.0              sqlit
ansys/v231            gaussian/g16                gsl/2.6                      metis/5.1.0(default)       parmetis/4.0.3(default) r/4.3.1              starc
ansys/v242            gaussian/g16-brdwl          gsl/2.7(default)             molpro/2015.1(default)     petsc/3.16.2(default)   r/4.4.0              starc
bzip2/1.0.8(default)  geos/3.10.2(default)        gsl/2.8                      mrcc/2020-02-22(default)   r/3.6.3                 rstudio/2023.12.1    starc
comsol/5.6            ghostscript/10.03.1(default)  lammps/29Oct20(default)    nwchem/7.2.0(default)      r/4.0.3(default)        scotch/7.0.2(default)  stata

------------------------------------------------------------------------------------------------------------ /uwahpc/centos8/modulefiles/bio-apps ------------------------------------------------------------
abyss/2.3.1(default)    blast+/2.13.0          cgmaptools/v0.1.2(default)  htslib/1.13(default)   maxquant/2.4.2.0        ncbi-vdb/2.11.2(default)  platanus/1.2
bbmap/39.06(default)    bowtie/1.3.0(default)  fastqc/0.12.1(default)      kat/2.4.1(default)     minimap2/2.22(default)  ngs/2.11.2(default)       platanus_all
bcftools/1.13(default)  bowtie2/2.4.4(default)  gatk/4.2.3.0(default)       libgff/2.0.0(default)  mothur/1.44.3           ngs/3.0.1                 platanus_tri
bcl2fastq/2.20.0.422    bwa/0.7.17(default)    Geneious/2024.0.3           maxquant/2.1.4.0       mothur/1.48.0           picard/3.1.1(default)     plink/1.07(d

------------------------------------------------------------------------------------------------------------ /uwahpc/centos8/modulefiles/devel ------------------------------------------------------------
boost/1.58.0           cmake/3.21.3(default)  dbcsr/2.2.0(default)          gcc/12.4.0           intel-tbb/2020.4(default)  kokkos/3.7.01(default)  netcdf-fort
boost/1.65.1           cmake/3.25.2           eigen/3.4.0(default)          go/1.22.2            intel/2015.6               lapack/3.9.1(default)   netcdf-fort
boost/1.68.0(default)  cmake/3.30.2           fftw-parallel/3.3.9(default)  hdf5/1.12.0(default)  intel/2020.4               lapack/3.10.1           netcdf/4.8.
boost/1.74.0           cuda/11.6              fftw/3.3.9(default)           hwloc/2.4.0(default)  java/1.8.0_311             lapack/3.12.0           nvhpc-hpcx-
boost/1.83.0           cuda/11.8              gcc/8.5.0                     hwloc/2.11.1          java/11.0.12               libxsmm/1.17(default)   nvhpc/22.3
boost/1.84.0           cuda/12.0(default)     gcc/9.4.0                     intel-mkl/2020.4      java/16.0.2(default)       netcdf-c/4.8.1(default)  nvhpc/23.11
cmake/3.14.5           cuda/12.4              gcc/11.5.0(default)           intel-tbb/2018.4      julia/1.10.0               netcdf-c/4.9.2          openjdk/17.

------------------------------------------------------------------------------------------------------------ /uwahpc/centos8/modulefiles/python ------------------------------------------------------------
Anaconda3/2023.07  Anaconda3/2023.09  Anaconda3/2024.06(default)  python/3.9.0(default)

------------------------------------------------------------------------------------------------------------ /uwahpc/centos8/modulefiles/tools ------------------------------------------------------------
awscli/2.10.3  ffmpeg/4.4(default)  getexample/1.0.0(default)  gnuplot/5.4.2(default)  maali/1.5h(default)  mono/6.12.0.122(default)  nasm/2.16.03(default)  osu-
(base) psun@kaya1[TimelineKGE]$ module load cuda/12.4
(base) psun@kaya1[TimelineKGE]$ module list
Currently Loaded Modulefiles:
 1) cuda/12.4
(base) psun@kaya1[TimelineKGE]$ module unload cuda/12.4
(base) psun@kaya1[TimelineKGE]$ module list
No Modulefiles Currently Loaded.
(base) psun@kaya1[TimelineKGE]$
```

Figure: Demonstration regarding the mentioned commands

Software Part in HPC: Slurm

Question

How can I interact and run my code on Compute nodes from the login node?

Answer: Use Slurm

What is Slurm?

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
- Purpose: Job scheduling in HPC environments

For more information: `https://en.wikipedia.org/wiki/Slurm_Workload_Manager`

Frequently Used Slurm Commands

`salloc`	Obtain a compute node, and run it interactively
`sbatch`	Submit a batch script as a job into the queue system
`squeue`	Check the queue status
`scancel`	Cancel a job

Software Part in HPC: Slurm

```
(base) psun@kaya1[~]$ salloc -p weics --gres=gpu --mem=32G
salloc: Pending job allocation 508493
salloc: job 508493 queued and waiting for resources
salloc: job 508493 has been allocated resources
salloc: Granted job allocation 508493
(base) bash-4.4$ nvidia-smi
Thu Sep 26 09:28:52 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.14              Driver Version: 550.54.14      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A6000               Off |   00000000:3B:00.0 Off |                  Off |
| 58%   82C    P2           283W /  300W |   35016MiB /  49140MiB |    100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX A6000               Off |   00000000:AF:00.0 Off |                  Off |
| 30%   34C    P8            20W /  300W |       0MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A    944883      C   python                                    35010MiB |
+-----------------------------------------------------------------------------------------+
(base) bash-4.4$ exit
exit
salloc: Relinquishing job allocation 508493
salloc: Job allocation 508493 has been revoked.
(base) psun@kaya1[~]$ squeue -u psun
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            508044       gpu quantum_     psun PD       0:00      1 (Priority)
            508043       gpu quantum_     psun  R   12:24:31      1 n046
            508006     weics quantum_     psun  R    7:25:07      1 n014
(base) psun@kaya1[~]$ cd /group/pmc015/psun/TimelineKGE/
(base) psun@kaya1[TimelineKGE]$ ls
LICENSE      data          learner.py     process_gdelt.py  requirements.txt  src_data
README.md    datasets.py   models.py      process_icews.py  results           timer.py
__pycache__  job.slurm     optimizers.py  regularizers.py   slurm_jobs.py     venv
(base) psun@kaya1[TimelineKGE]$ sbatch job.slurm
Submitted batch job 508495
(base) psun@kaya1[TimelineKGE]$ squeue -u psun
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            508044       gpu quantum_     psun PD       0:00      1 (Priority)
            508495       gpu quantum_     psun PD       0:00      1 (Priority)
            508043       gpu quantum_     psun  R   12:24:55      1 n046
            508006     weics quantum_     psun  R    7:25:31      1 n014
(base) psun@kaya1[TimelineKGE]$ scancel 508495
(base) psun@kaya1[TimelineKGE]$ squeue -u psun
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            508044       gpu quantum_     psun PD       0:00      1 (Priority)
            508043       gpu quantum_     psun  R   12:25:09      1 n046
            508006     weics quantum_     psun  R    7:25:45      1 n014
(base) psun@kaya1[TimelineKGE]$
```

Figure: Demonstration with above commands

Software Part in HPC: Slurm batch scripts

- Command to submit: `sbatch job.slurm`
- `job.slurm` is a bash script with specific declaratives

```
#!/bin/bash
#SBATCH --job-name=job_name  # name your job
#SBATCH --output=output.txt # running logs
#SBATCH --nodes=1 # how many nodes you want?
#SBATCH --time=24:00:00 # how long your task will last
#SBATCH --partition=gpu # which queue you want to get in
#SBATCH --mem=32G # how many RAM you will need
#SBATCH --gres=gpu:v100:1 # specific filtering
module load cuda/12.4
module load gcc/12.4.0
source venv/bin/activate
# your python script here
python xxx
```

assets_kaya/job.slurm

Software Part in HPC: Slurm batch scripts

- After submitting your task:
  - You will receive a job ID
  - View task status: `squeue -u username`
  - Refer to previous slides for examples
- View specific job logs:
  - Check `output.txt` file specified in the job script
- If you want to know more about the HPC, can check this: `https://hpc-wiki.info/hpc/Modules`
- If you want to know more about the slurm, can check this: `https://slurm.schedmd.com/pdfs/summary.pdf`

Software Part in HPC: Slurm Summary

The Basic Unit of Work

A job script is a shell script that:

- Defines required resources (e.g., number of cores, RAM)
- Sets up the environment for the application
- Prepares data for the job
- Executes the application/code
- Performs cleanup (input data, saving output files)

Admin

Accessing Kaya HPC Service

Overview

The Kaya HPC service is free for Research and Education purposes at UWA. Access is project-based.

Project Requirements

- Principal Investigator (PI): senior researcher or academic staff
- Project description with duration and user estimates
- GitHub repository link
- ORCIDs for all project members
- IRDS share for long-term data storage

PI Responsibilities

- Accountable for project resource usage
- Responsible for project data and resources

How to Request Access?

Process

Log a ticket with the University IT Helpdesk[a]

[a] https://uwa.service-now.com/sp?id=sc_cat_item&sys_id=c38afd311b397910025ada86b04bcbf6

Project Description Should Include

- Nature of the research (couple of paragraphs)
- Expected software to be used
- Rough estimate of data storage requirements

Note

The HPC team will contact you if further information is required.

How to Connect to the HPC System

Welcome to Kaya!

You should have received a welcome email with:

- Your system userID
- A temporary password
- Other information (e.g., project name/projectid)

Important Note

The Kaya password is separate from your Uni ID (formerly Pheme) password.

- Same complexity requirements
- 6-month expiry

Access with Unifi or VPN

You must within UWA Unifi or VPN to be able to connect to Kaya[a]

[a] `https://ipoint.uwa.edu.au/app/answers/detail/a_id/1572/~/uniconnect-explained`

First Login - Changing Your Password

Terminal Requirements

You will need a terminal to run the ssh:

- Windows: PowerShell, Terminal, or Windows Subsystem for Linux, or anything you familiar with
- Mac: iTerm or Terminal
- Linux: Built-in terminal

Password Change Process

You will be prompted 3 times during this process[8]:

1. Enter current (temporary) password
2. Enter new chosen password
3. Confirm new password

[8] When you type your password, you will not be able to see anything on the terminal. It looks like nothing is happening, but it is for security reasons.

Password Complexity Requirements

Kaya passwords must:

- Be a minimum of 10 characters long
- Include at least one character from each of:
  - Lower case characters
  - Upper case characters
  - Numbers
  - 'Special' characters (e.g., #$%^&)

Login to Kaya

- Login: `ssh your_username@kaya.hpc.uwa.edu.au`
- If you have done everything correct, you should be able to see this:

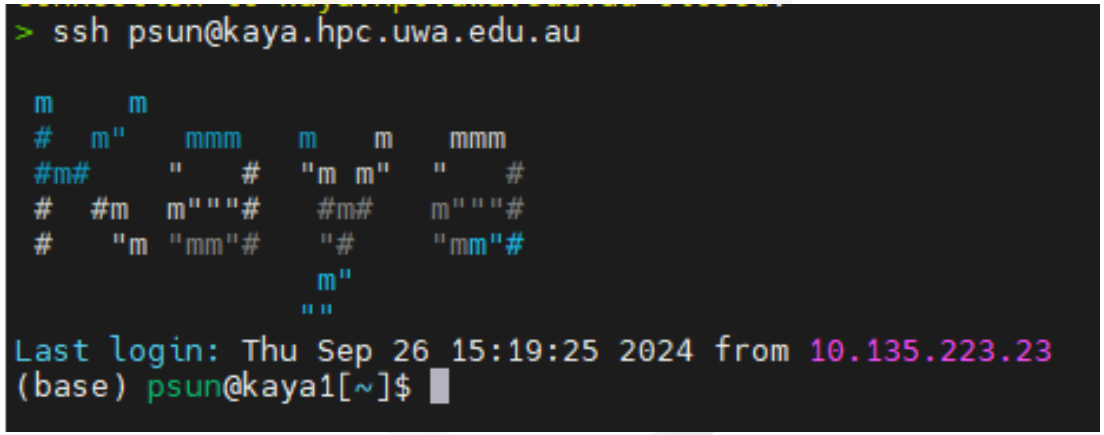


Figure: Login to Kaya

You are in your home directory on Kaya now.

Advanced: Simplifying SSH Access to Kaya

Using ssh-copy-id

To avoid entering your password each time:

1. Run: `ssh-copy-id your_username@kaya.hpc.uwa.edu.au`
2. Enter your password when prompted
3. For future logins, simply use without typing password:
   `ssh your_username@kaya.hpc.uwa.edu.au`

Further Simplification

Edit `~/.ssh/config` and add:

```
Host Kaya
    HostName kaya.hpc.uwa.edu.au
    User your_username
    Port 22
```

After this setup, log in with just `ssh Kaya`

How to Upload Data to Kaya

Upload Methods

scp: `scp file user@kaya.hpc.uwa.edu.au:/group/xxx`

FileZilla: `https://filezilla-project.org/`

Important Notice

Kaya does not provide persistent data storage. Always:

- Keep your data stored in a separate location
- Consider using iRDS[a] (similar to Google Drive/Dropbox/OneDrive)
- Download important data and back it up elsewhere

[a] UWA Provided Serivce

Capability

Resources

- GPU
  - 5 nodes, each node with two V100, GPU Memory is 32GB
  - 10 nodes, each node with four P100, GPU memeory is 16GB
  - Comes with the Dual Intel Xeon CPUs, 36 cores, in total 768 RAM[9]
- CPU
  - 15 medium nodes, each with Dual Xeon CPUs and 256GB RAM
  - 5 large nodes, each with Dual Xeon CPUs and 512GB-1.5TB RAM
- Storage
  - In total is 120TB
  - Four type folders
    - `/group`: where you data and results should stay
    - `/scratch`: intermediate results for running jobs, limited 30TB storage, you should move your results from here to your group folder after finished.
    - `/tmp`: Linux System Default for system to running
    - `/home`: which is where you login
  - DO NOT PUT YOUR PROJECT IN YOUR HOME DIRECTORY
  - ALWAYS PUT YOUR PROJECT IN group FOLDER!

[9] You will not be able to run GPU alone, you will still need CPU and Memeory to make it a computer

Resources

Some key points when you use Kaya:

- Always put your data, especially large ones, in your `/group` directory!
- Do not run any of your programs or experiments on the login node!
- Do not put your data in your `/home` directory, which is the default one when you login.
- Do not hold your nodes longer than you need, as they are shared resources!

Slurm Queues on Kaya

Queue Overview

Slurm uses queues (officially called "partitions") to target different machine types.

Kaya Queue Configuration

Queue	Max Walltime	Machines
test	10 min	All
work	3 days	Med/Large CPU
gpu	3 days	GPU
long	1 week	GPU
ondemand	4 hours	Med CPU

Monitor Tools for Kaya

Figure: Current Status Monitor

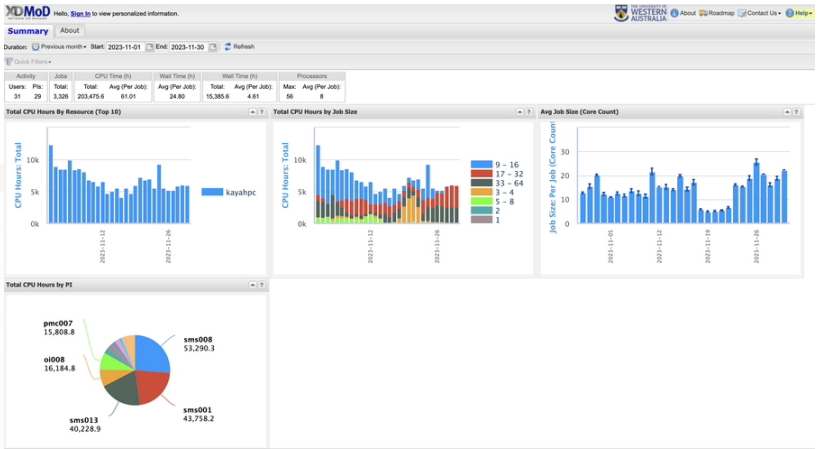


Figure: Historical Usage Summary

`https://monitor.hpc.uwa.edu.au/`

- Monitor Jobs queue
- View CPU/GPU usage
- Current system status

`https://metrics.hpc.uwa.edu.au/`

- Historical resource usage
- Project-wise utilization
- Monthly summaries

Both tools are only accessible via the campus network.

CLI

Typical process to run your Python project on Kaya

Steps to Run Python Projects on Kaya

1. Login to Kaya
2. Transfer your code and data to Kaya
3. Set up a virtual environment using conda or pip
4. Allocate a compute node
5. Load required modules on the compute node
6. Execute your program on the compute node
7. Retrieve your results from login node

Accessing Kaya via Command Line Interactively

1. Compress your code and data
2. Transfer files to your `/group` directory:
   - Use `scp` or FileZilla
3. Open your terminal
4. Login with the previously provided command
5. In the `/group` directory:
   - Extract your data and code
   - Organize files appropriately
6. Allocate a compute node using `salloc`:

```
salloc -p gpu --mem=16G -N 1 -n 8 --gres=gpu:v100:1
```

   Note: You will now be on a compute node, not the login node
7. Prepare your environment:
   - Load required modules
   - Activate your Python environment
8. Execute your program

Accessing Kaya via Command Line Interactively

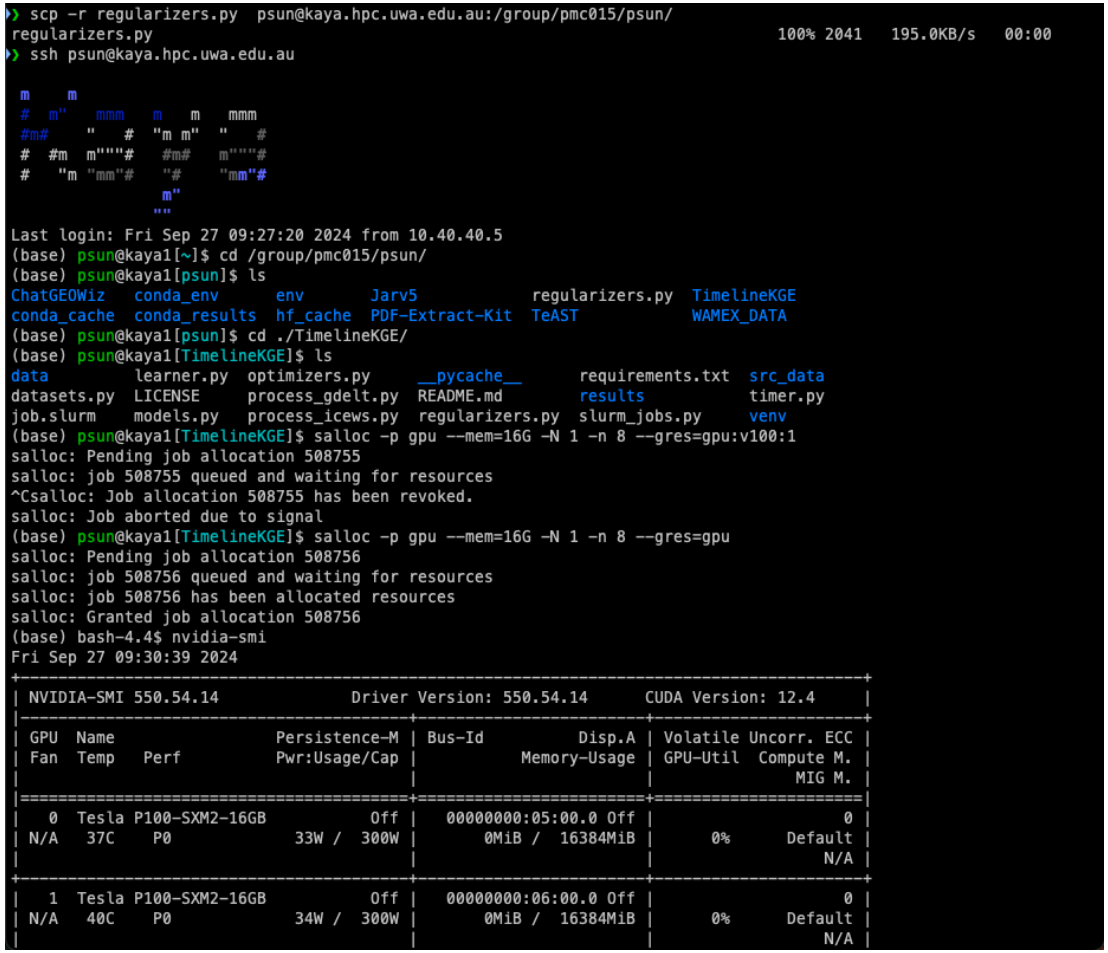


Figure: Example regarding: scp a file to Kaya, request and GPU Compute node

Accessing Kaya via Command Line Interactively

```
(base) bash-4.4$ ls
data  datasets.py  job.slurm  learner.py  LICENSE  models.py  optimizers.py  process_gdelt.py  process_icews.py  __
pycache__  README.md  regularizers.py  requirements.txt  results  slurm_jobs.py  src_data  timer.py  venv
(base) bash-4.4$ module load cuda/12.4
(base) bash-4.4$ module load gcc/12.4.0
(base) bash-4.4$ source venv/bin/activate
(venv) (base) bash-4.4$ ls
data  datasets.py  job.slurm  learner.py  LICENSE  models.py  optimizers.py  process_gdelt.py  process_icews.py  __
pycache__  README.md  regularizers.py  requirements.txt  results  slurm_jobs.py  src_data  timer.py  venv
(venv) (base) bash-4.4$ python learner.py --dataset ICEWS14 --emb_reg 0.002 --learning_rate 0.1 --model LearnableTr
ajectoryKGEModel --valid_freq 1 --max_epochs 200 --rank 50 --batch_size 200
2024-09-27 09:31:24.556 | INFO     | datasets:__init__:61 - Assume all timestamps are regularly spaced
2024-09-27 09:31:24.557 | INFO     | datasets:__init__:74 - Not using time intervals and events eval
2024-09-27 09:31:25.008 | INFO     | __main__:learn:117 - Dataset ICEWS14 loaded
2024-09-27 09:31:25.008 | INFO     | __main__:learn:118 - Dataset shape: (7128, 460, 7128, 365)
2024-09-27 09:31:25.008 | INFO     | __main__:learn:119 - including activation: False
2024-09-27 09:31:25.009 | INFO     | __main__:learn:120 - number of heads: 4
2024-09-27 09:31:27.473 | INFO     | __main__:learn:155 - Starting training process: LearnableTrajectoryKGEModel on
 ICEWS14 using rank = 50, lr = 0.1, emb_reg = 0.002, time_reg = 0.0
2024-09-27 09:31:27.473 | INFO     | __main__:learn:168 - [ Epoch: 0 ]
```

Figure: Load modules, activate Python environment and run Python program

Accessing Kaya via Job Slurm

Why Use Slurm?

Running jobs directly in the terminal has limitations:

- You need to keep the terminal open
- Exiting may interrupt your program

Slurm Workflow

1. Follow initial steps as before (login, file transfer, etc.)
2. Instead of manual node allocation and script execution:
   - Create a `job.slurm` script
   - Submit job using `sbatch` command
3. Slurm queues your job and executes it when resources are available

Attention!

To run your experiments on KAYA, you should/must use this way!
All other ways are to debug your codes!

Access Kaya via Job Slurm

- Command to submit: `sbatch job.slurm`
- `job.slurm` is a bash script with specific declaratives

```
#!/bin/bash
#SBATCH --job-name=job_name  # name your job
#SBATCH --output=output.txt # running logs
#SBATCH --nodes=1 # how many nodes you want?
#SBATCH --time=24:00:00 # how long your task will last
#SBATCH --partition=gpu # which queue you want to get in
#SBATCH --mem=32G # how many RAM you will need
#SBATCH --gres=gpu:v100:1 # specific filtering
module load cuda/12.4
module load gcc/12.4.0
source venv/bin/activate
# your python script here
python xxx
```

assets_kaya/job.slurm

Access Kaya via Job Slurm

```
(base) psun@kaya1[TimelineKGE]$ # I am in login node now
(base) psun@kaya1[TimelineKGE]$ ls
data         learner.py  optimizers.py     __pycache__       requirements.txt  src_data
datasets.py  LICENSE     process_gdelt.py  README.md         results           timer.py
job.slurm    models.py   process_icews.py  regularizers.py   slurm_jobs.py     venv
(base) psun@kaya1[TimelineKGE]$ sbatch job.slurm
Submitted batch job 508765
(base) psun@kaya1[TimelineKGE]$ squeue -u psun
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            508765       gpu quantum_     psun PD       0:00      1 (Priority)
            508044       gpu quantum_     psun  R   23:56:55      1 n001
(base) psun@kaya1[TimelineKGE]$
```

Figure: Replace the commands from `salloc` in previous screenshots

IDE

Edit code on Kaya

Limitations of Previous Methods

Previous approaches assume:

- Your code is complete and ready to run
- You only need to execute for results

Challenges in Development

However, you may need to:

- Debug and modify your code
- Explore and develop using GPU resources

Solution: VSCode Remote SSH or Jetbrains Gateway

VSCode Remote SSH allows for:

- Real-time code editing on Kaya
- Interactive debugging sessions
- Seamless integration with Kaya's GPU/CPU resources

Editing Code on Kaya

Workflow Overview

1. SSH into the login node
2. Edit your code on the login node
3. Execute the code on a compute node

How It Works

- Login and compute nodes share the same file server
- Changes made on the login node are reflected on compute nodes

Key Advantage

This setup allows you to:

- Immediately run the updated code on HPC compute nodes
- Fail Fast, Succeed Faster

Accessing Kaya via VSCode Remote SSH

Prerequisites

To get started, you'll need:

1. Visual Studio Code (VS Code)
2. Remote - SSH extension for VS Code

Installation Resources

VS Code

`https://code.visualstudio.com/`

Remote SSH Guide

`https://code.visualstudio.com/docs/remote/ssh`

Accessing Kaya via VSCode Remote SSH

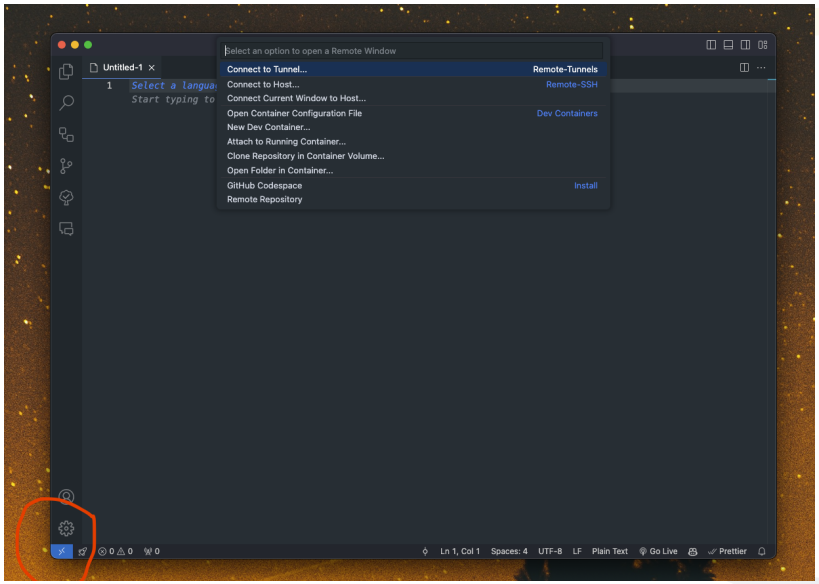


Figure: Left: Click "Connect to Host" and add new SSH host

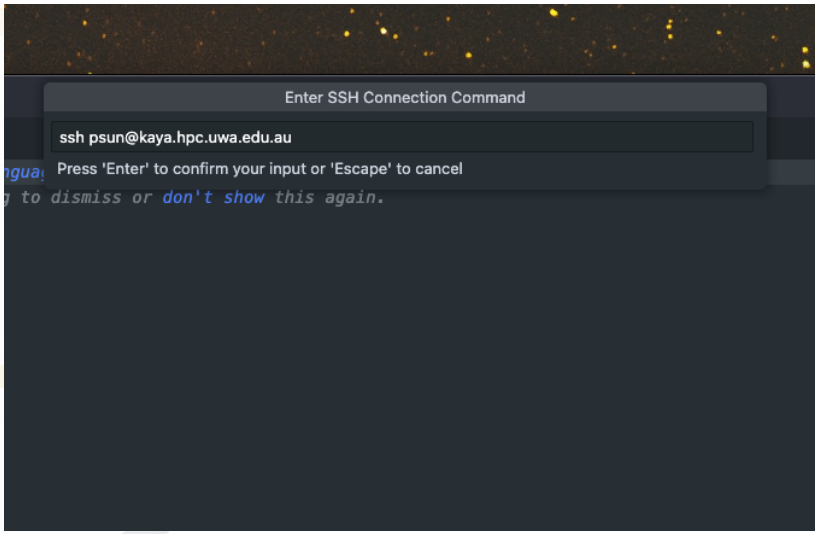


Figure: Right: Enter SSH connection details for Kaya

Accessing Kaya via VSCode Remote SSH

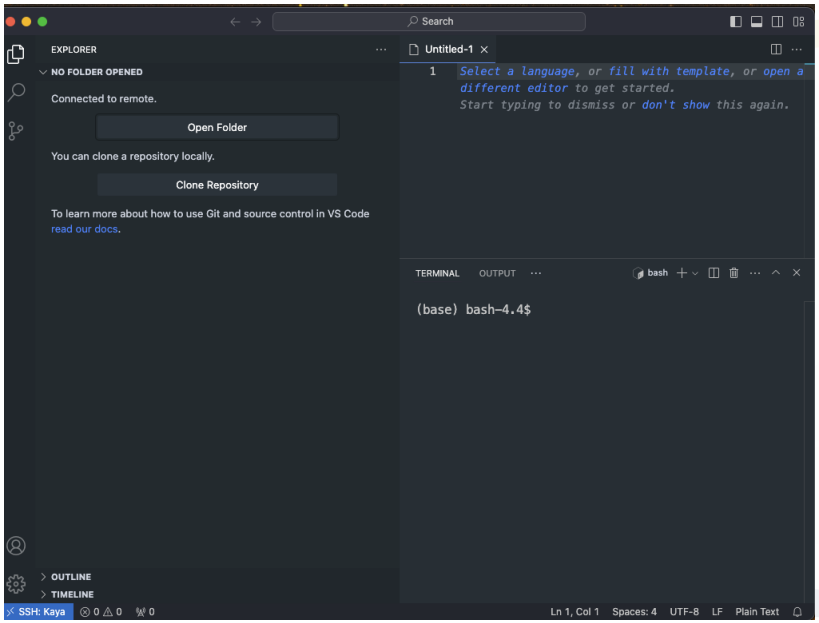


Figure: Open Folder

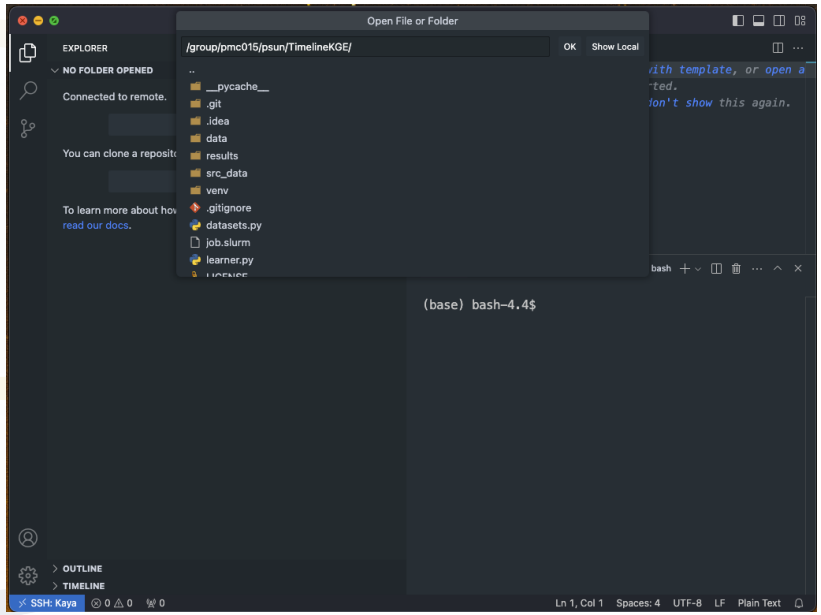


Figure: Select your project folder within /group directory

Accessing Kaya via VSCode Remote SSH

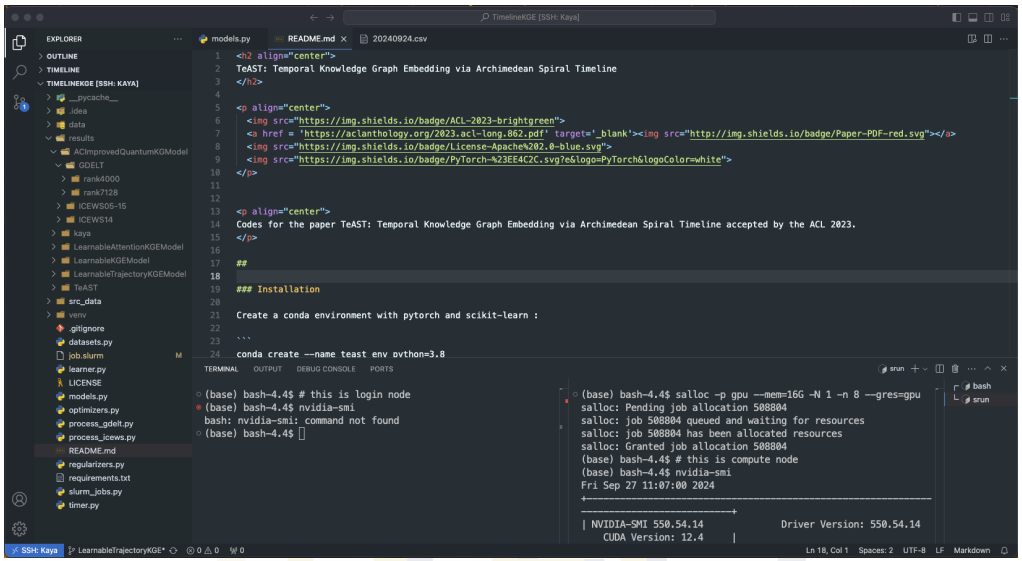


Figure: Ideal workspace setup in VSCode for Kaya

Workspace Overview

- Project opened via VSCode Remote SSH
- Left terminal: Login node
- Right terminal: GPU Compute Node (via `salloc`)

Workflow

1. Edit code in the VSCode editor
2. Execute code in the right terminal (Compute Node)
3. Seamless integration of development and HPC

Accessing Kaya via Jetbrain Gateway

About Jetbrain Gateway

- Similar solution to VSCode Remote SSH
- Requires a license (free for students)
- Python projects will actually open via PyCharm

Benefits for Students

- Entry-level developer friendly IDE (PyCharm)
- Promotes good coding practices

Installation Resources

Jetbrain Gateway `https://www.jetbrains.com/remote-development/gateway/`

Student License `https://www.jetbrains.com/community/education/#students`

Access Kaya via Jetbrain Gateway

It will follow the similar process and idea as the VSCode Remote SSH

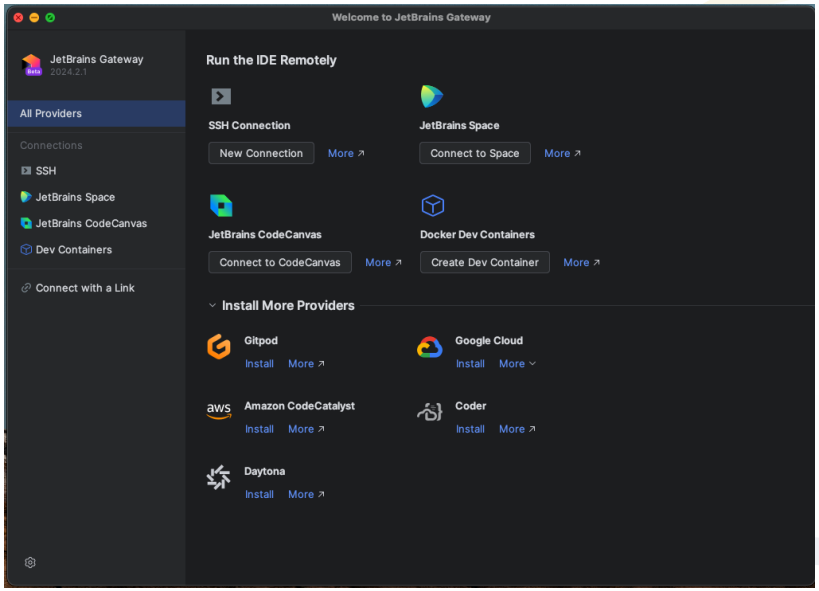


Figure: Select the SSH Connection

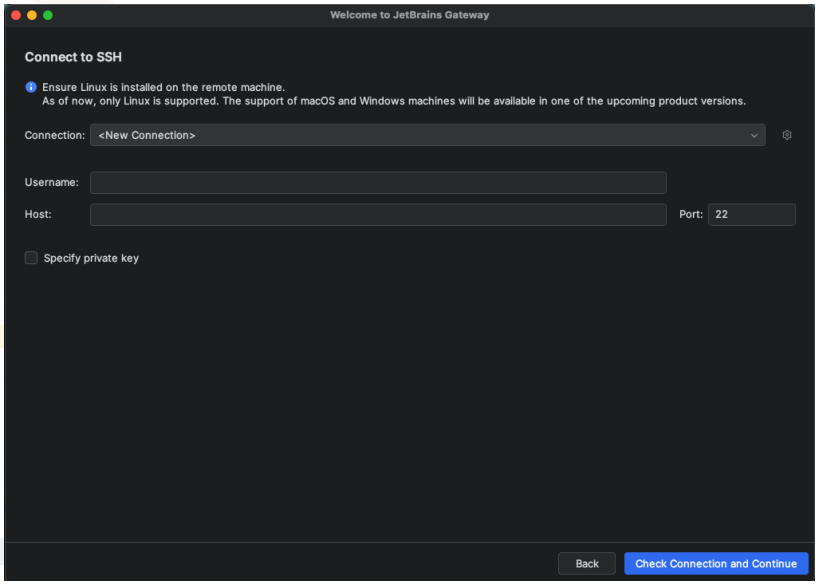


Figure: Put your username, host and then connect

Access Kaya via Jetbrain Gateway

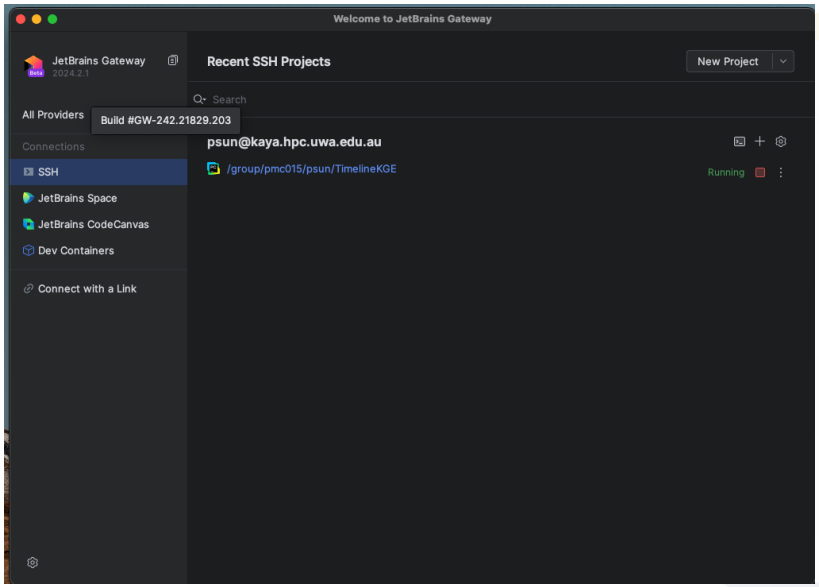


Figure: Click "New Project"

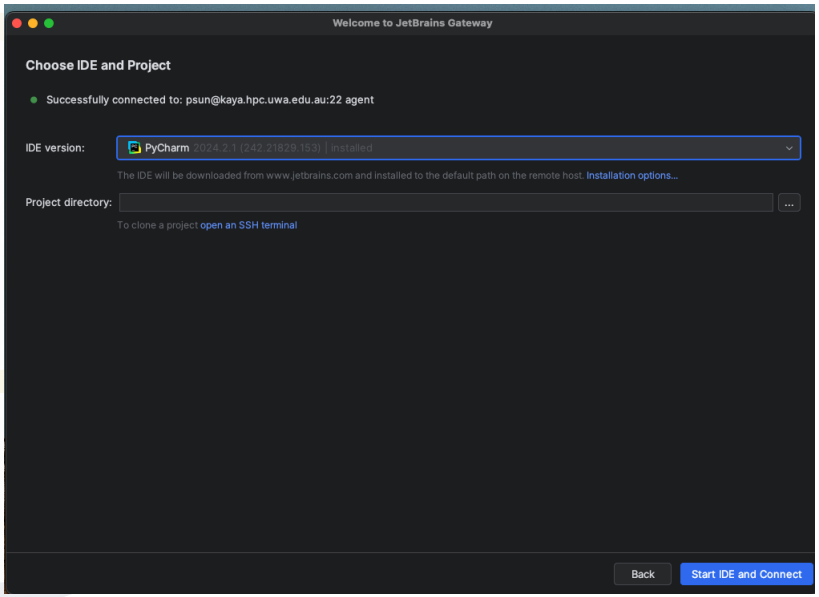


Figure: Select PyCharm and Project folder under /group

Access Kaya via Jetbrain Gateway

Figure: Setup in Jetbrain Gateway for Kaya

- Open the terminal
- Request a GPU compute node
- Edit code in IDE
- Run code within terminal: `python xx.py`

Web

Web-based Access

Web based GUI Solutions for Non-Hackers

While not as flexible as command-line options, these solutions work well in most circumstances, and you will access them via your browser.

Option 1

OnDemand

Provided by Kaya

Do NOT DO THIS Option

SSH Reverse Tunnel[a]

[a] Do not do it in KAYA

Access Kaya via OnDemand

- Web service: `https://ondemand.hpc.uwa.edu.au`
- Login with your username and password of Kaya
- You will need to connect to Unifi or UniVPN

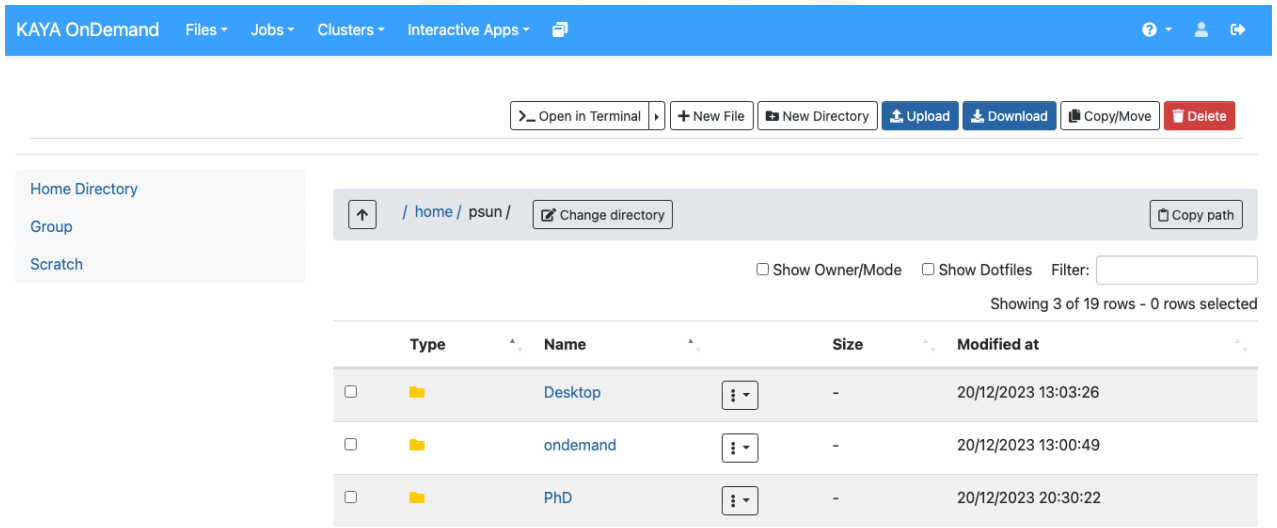


Figure: After you login, click the Files, what you will see is above, you can upload files here to Group folder

Access Kaya via OnDemand: Request CPU Resources

- Click: Interactive Apps ⇒ Kaya OnDemand (CPU)

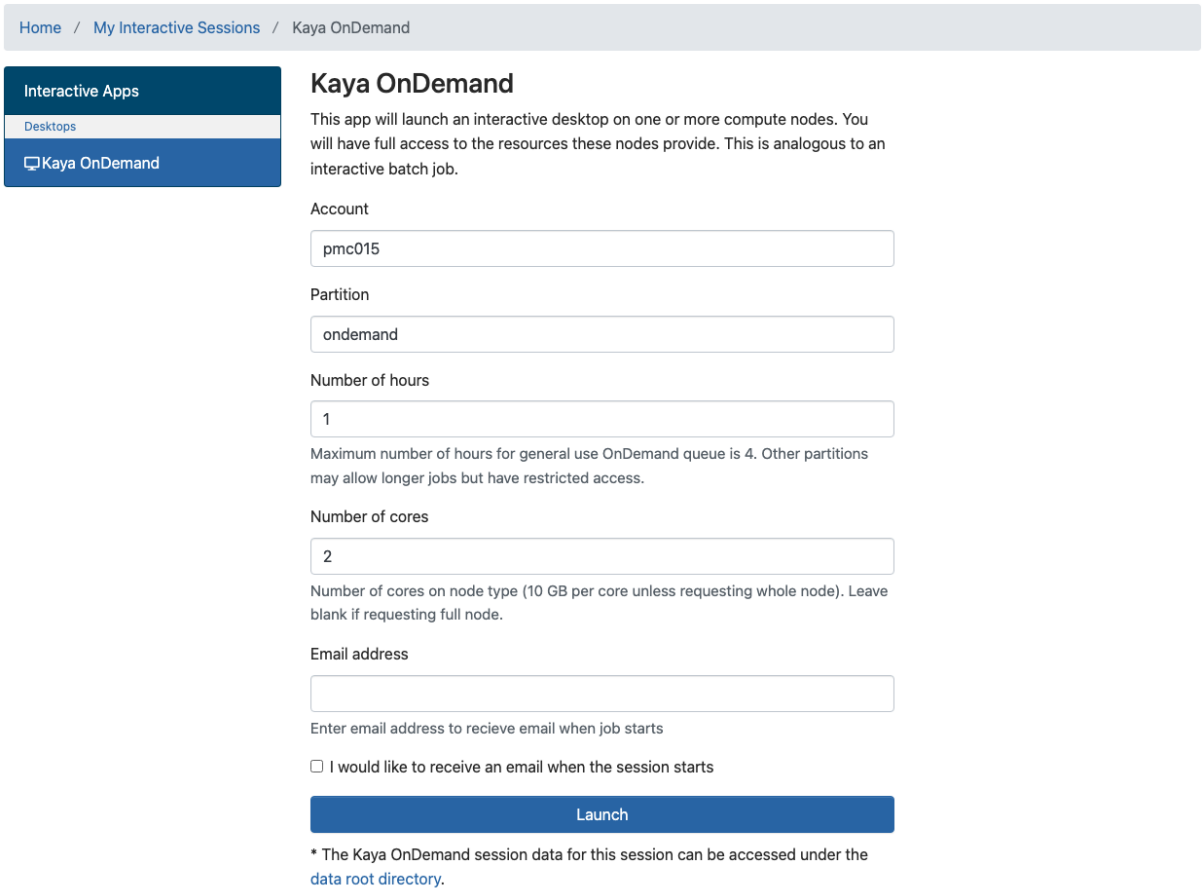


Figure: This is the interface to request the CPU resources

Access Kaya via GPU OnDemand: Request GPU Resources

- Account = your project ID
- Partition = currently only `ondemand`

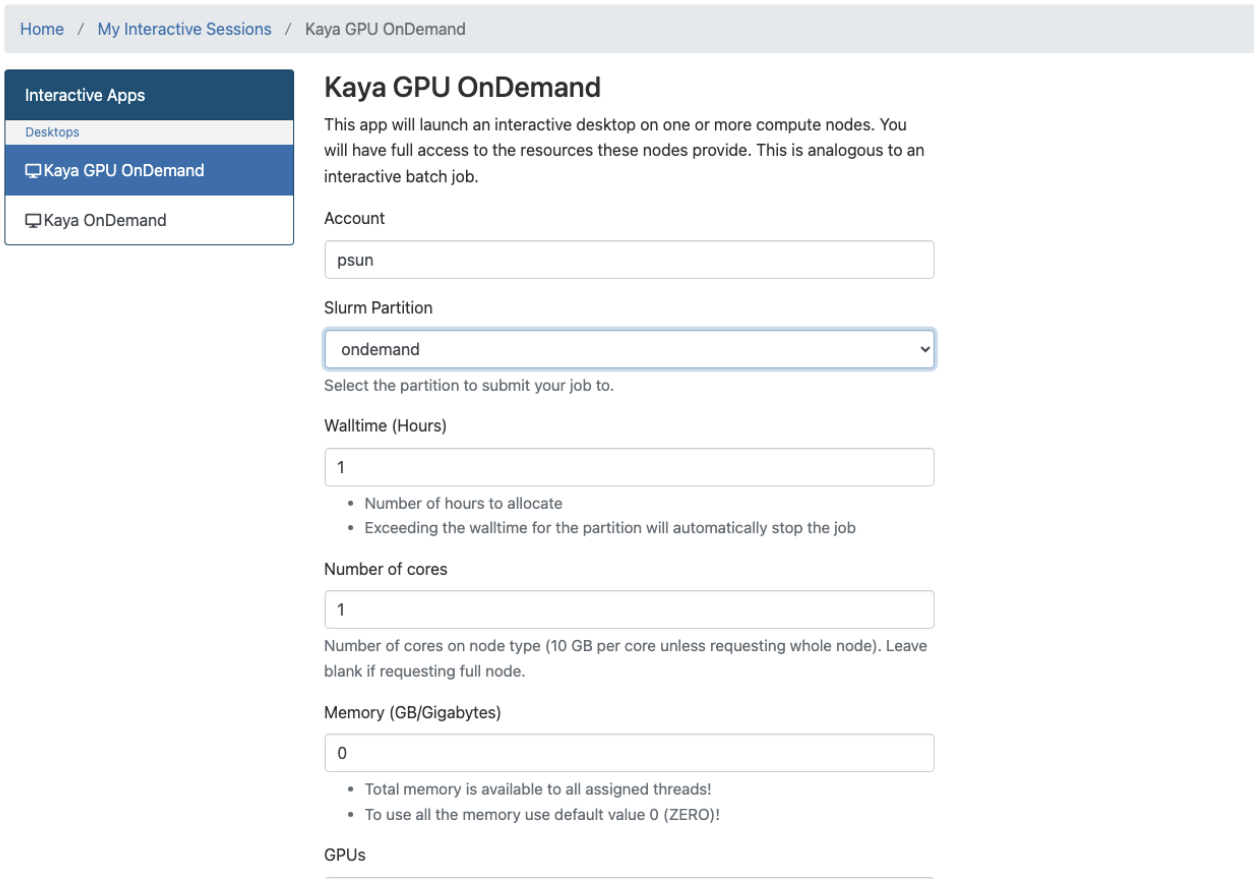


Figure: Launch GPU Resources

Access Kaya via OnDemand: Request GPU/CPU Resources

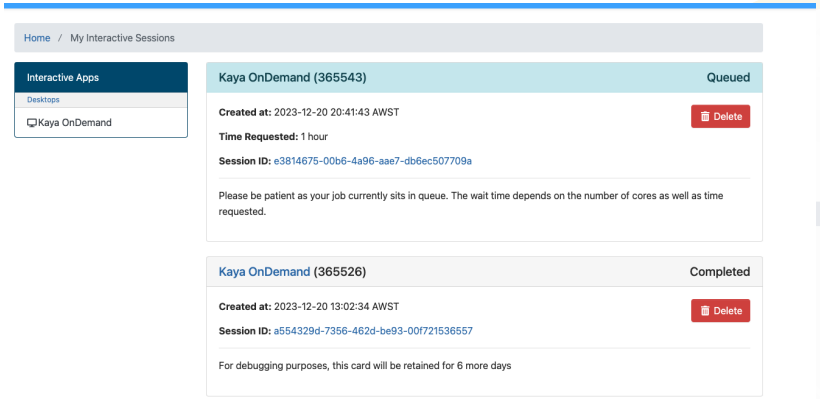


Figure: Waiting for allocating resources

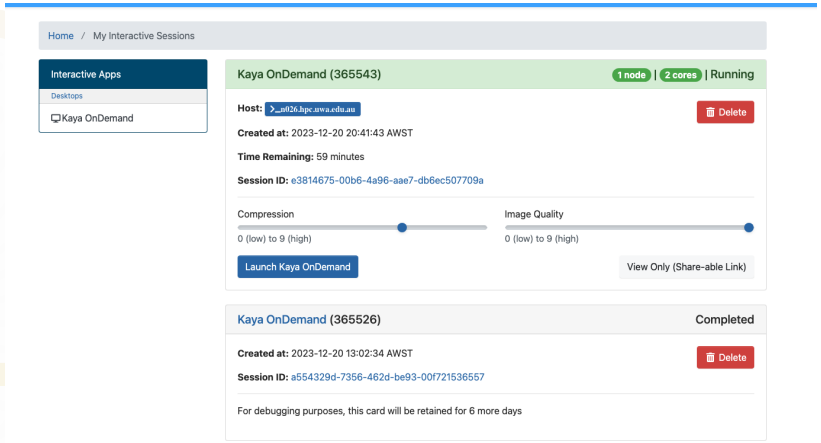


Figure: Resources allocated

Then you can click “Lanuch Kaya OnDemand” to open the virtual destkop

Access Kaya via OnDemand: Linux Desktop in Browser

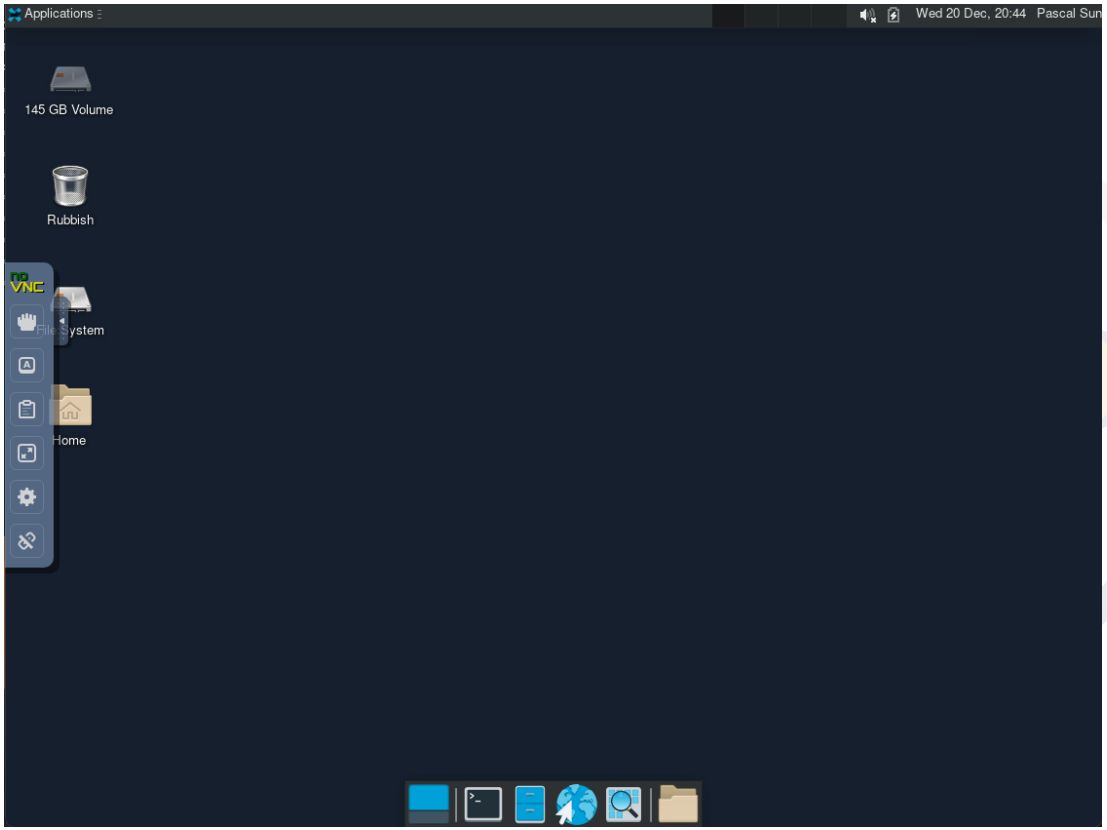


Figure: Linux Desktop Interface via OnDemand

- Access the Compute Node in your browser
- Use terminal in the browser to interact with the Compute Node
- You can try to use install jupyter-notebook
- And then open the browser to access it

Access Kaya via OnDemand: Linux Desktop in Browser

- Open the Terminal
- cd to your project directory
- Run desired commands:
  - Execute Python scripts
  - Launch Jupyter Notebook
  - Check GPU resources:

```
1 #!/bin/bash
2 nvidia-smi
```

assets_kaya/nvidia.slurm

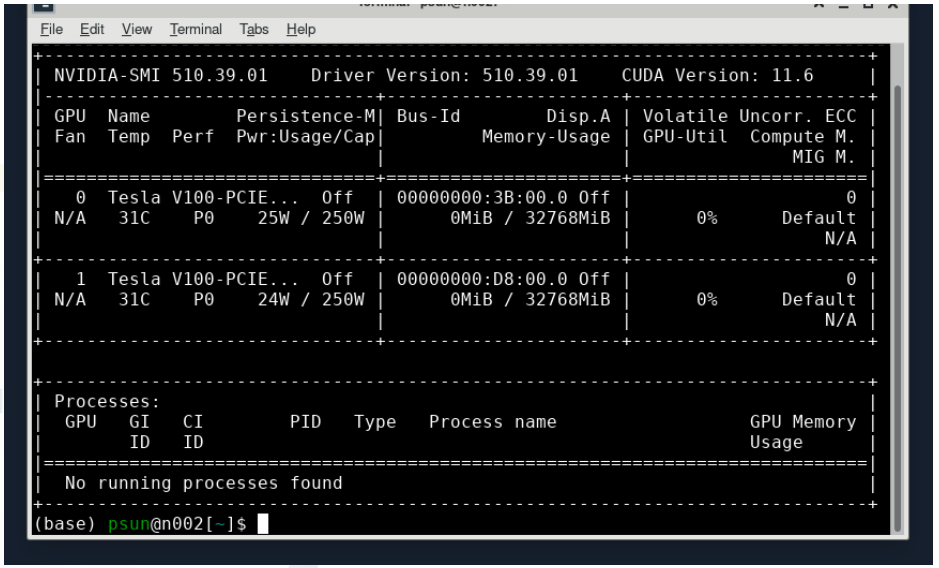


Figure: Terminal showing GPU resources

Tip

The browser within this Linux Desktop can be used to open and interact with Jupyter Notebooks.[a] All you need to do is operating on this computer within the browser.

[a] It is like browser in browser

Access Kaya via OnDemand: Sbatch Queue

You can also submit a job into the queue (like what `sbatch` for)

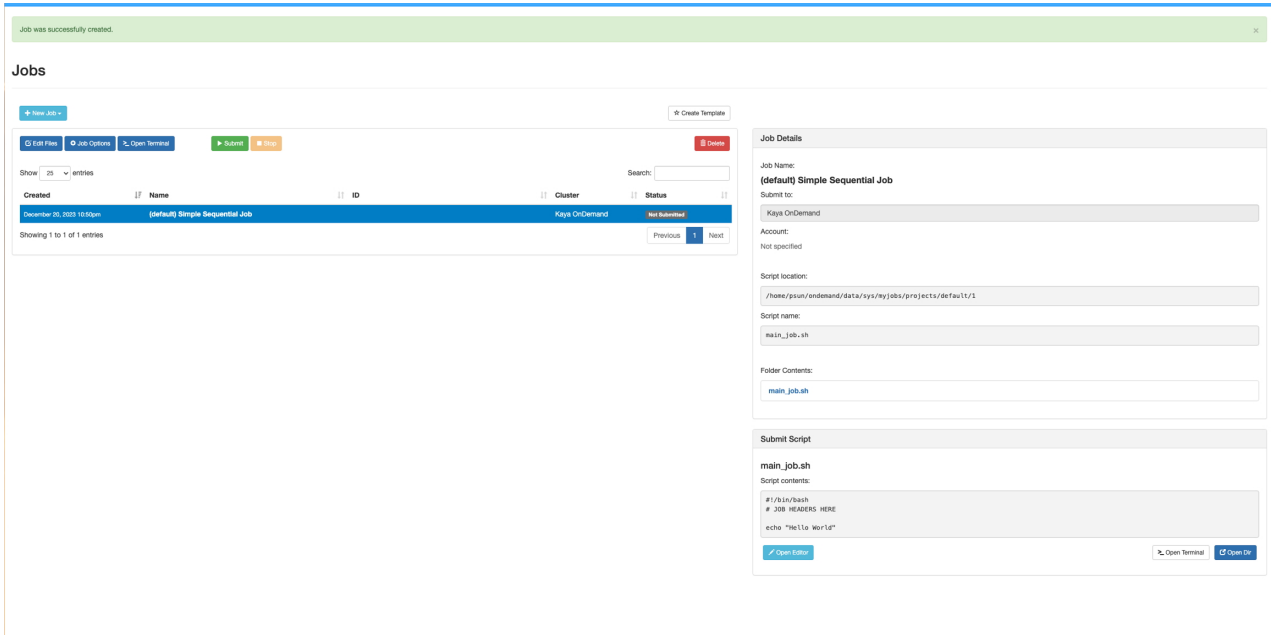


Figure: All you need to do is clicking the jobs, and put proper params.[10]

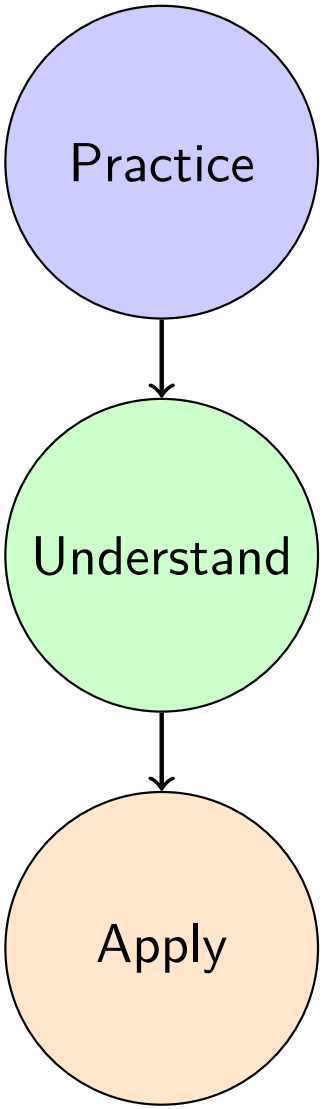
[10] ondemand will pop your login state out every 15 minutes, so you need to login back again to check the status, it is a known bug.

Demo

Demo Project: Putting It All Together

Next Steps

- Practice what you've learned
- Understand each detail thoroughly
- Try our detailed demo project



Demo Project URL

`https://tutorial.nlp-tlp.org/gpu-resources/kaya/demo-project`

Tips

Setup Conda Environment under Group Directory

Why Use Group Directory?

- Default: Conda stores environments in your home directory
- Home directory limit in Kaya: 30GB storage
- Solution: Use group directory for larger storage

Conda Commands

1. Create environment under specific directory:

```
conda create --prefix /group/your_group/your_username/envs/env_name
```

2. Activate the environment:

```
conda activate /group/your_group/your_username/envs/env_name
```

Another solution

Use `pip` to manage virtual environment, under group directory

```
python -m venv venv
source venv/bin/activate
```

HuggingFace Cache Directory

Problem

- HuggingFace downloads large models
- Default cache: `~/.cache/huggingface/hub`
- Home directory has limited size

Solution

Set `HF_HOME` to point to the group directory

Implementation

Add to `~/.bashrc`:

```
export HF_HOME=/group/your_group/your_username/hf_cache
```

OR include in your scripts:

```
import os
os.environ['HF_HOME']='/group/your_group/your_username/hf_cache'
```

Thank You for Your Attention!

Do you have any questions?

Contact Kaya Team:

`hpc-admin@uwa.edu.au`

- We are NLP-TLP Team from UWA.[11]
- We use Kaya a lot for Our Research, shout out for UWA HPC Team!
- Hopefully this can also help your research.
- Infrastructure is crucial for advancing research.
- **We encourage you to advocate for continued investment in computational resources at UWA and across WA.**

[11] A documentation version of this is available: `https://tutorial.nlp-tlp.org/gpu-resources`