

AI on High Performance Computing in a Nutshell

HPC Services, NHR@FAU

hpc-support@fau.de

<https://doc.nhr.fau.de>

Agenda

This presentation is a follow-up to "**HPC in a Nutshell**" and assumes familiarity with foundational HPC concepts discussed there. It is highly recommended to review the **HPC in a Nutshell** presentation before proceeding with this material.

1. Introduction

- Why efficient data formats matter in HPC.
- Challenges with many small files.

2. Working with Data for AI

- Storage Systems
- Overview of formats like HDF5, NetCDF, Parquet, webdataset

3. When to Use Python venvs, Conda, or Containers

- Guidelines for choosing the right tool for environment management.

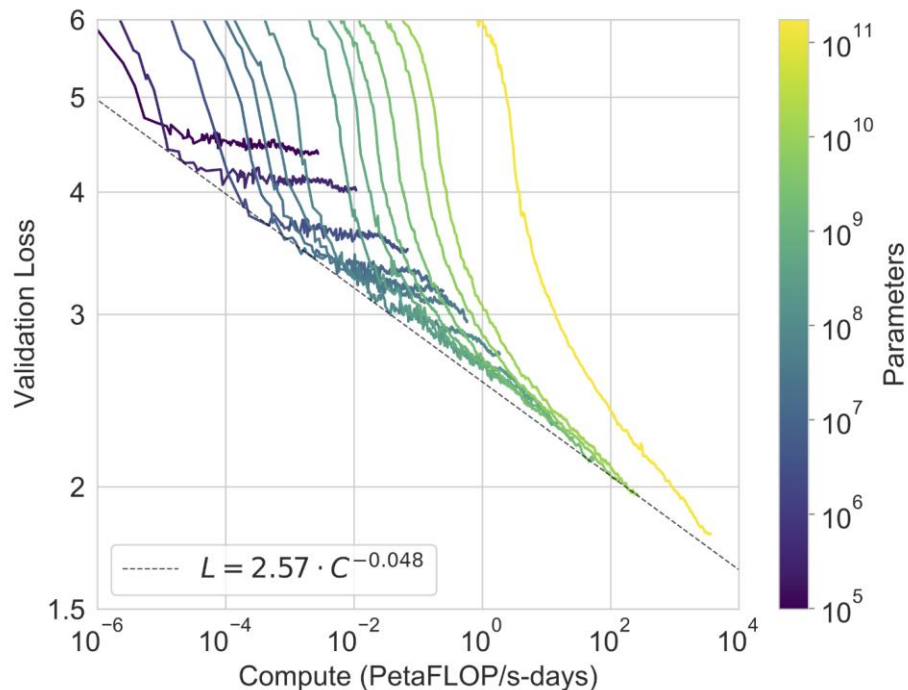
4. Conclusion and Best Practices

- Recap of efficient HPC usage, monitoring Jobs on HPC systems

Introduction to AI and HPC

Why is HPC essential for AI workloads?

- AI scales with massive parallel processing
- GPUs can handle thousands of operations in parallel, greatly speeding up tasks like matrix multiplication in neural networks
- HPC GPU Clusters are needed to scale AI-Models and reduce training time

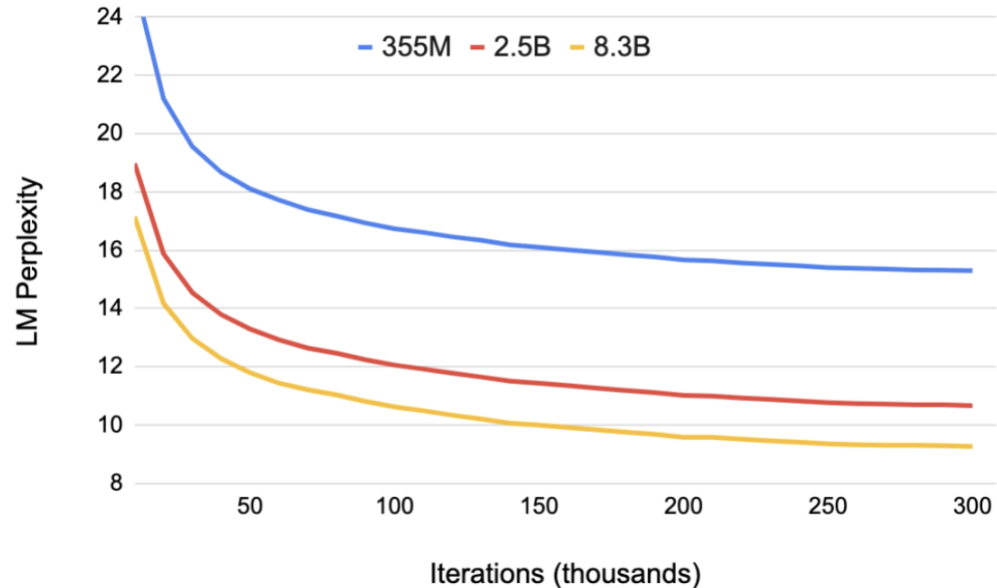


Source: <https://arxiv.org/pdf/2005.14165>

Why is scaling AI important?

- “As the model size increases, the validation perplexity decreases and reaches a validation perplexity of 9.27 for the 8.3B model”
- “We observe the trend that increasing model size also leads to lower perplexity on WikiText103 and higher cloze accuracy on LAMBADA”

* Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism
<https://arxiv.org/pdf/1909.08053>



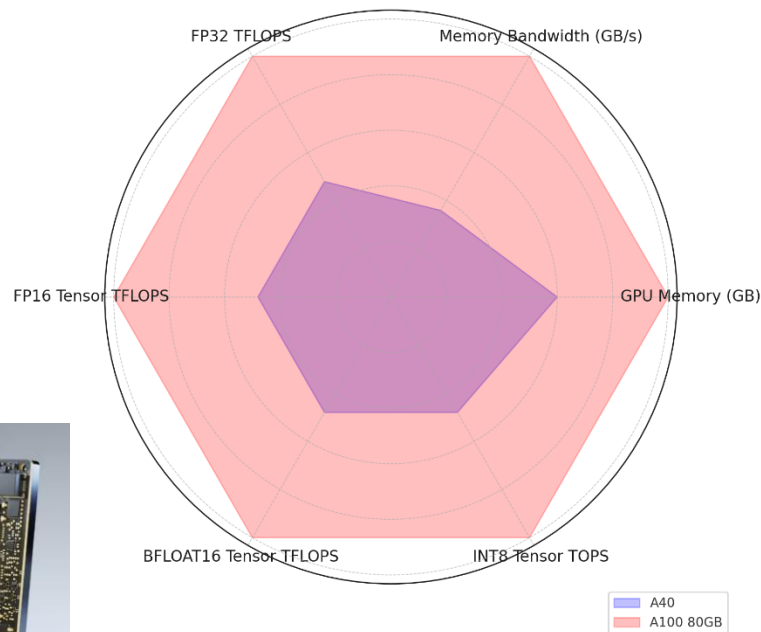
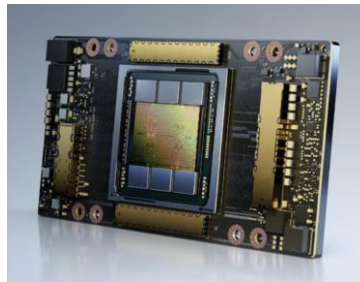
HPC systems at NHR@FAU

<https://doc.nhr.fau.de/clusters/overview/>

“Alex” cluster

NHR GPGPU cluster, open for Tier3 users after application
Application through Principal Investigator (PI)

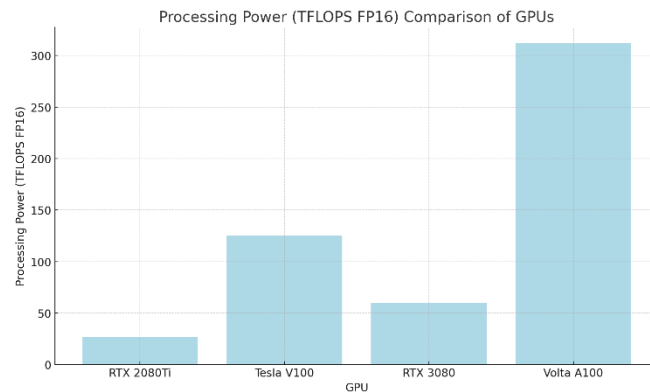
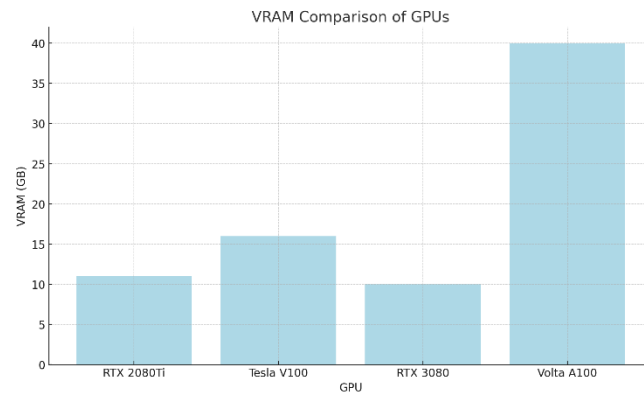
- 44 nodes with
 - 8x **NVIDIA A100** (each 40 GB / 80GB HBM2)
 - 1024 GB / 2048 GB of main memory
 - 14TB local NVMe SSD
 - HDR200 Infiniband network
- 38 nodes with
 - 8x **NVIDIA A40** (each with 48 GB DDR6)
 - 512 GB of main memory
 - 7 TB local NVMe SSD



“TinyGPU” cluster

for GPU workloads – not all nodes always generally available (Tier3)

- 12 nodes with 4x RTX 2080Ti
- 4 nodes with 4x Tesla V100
- 7 nodes with 8x RTX3080
- 8 nodes with 4x Volta A100



Which GPUs should I use?

Does your training need multiple GPUs and your code is supporting it?

Yes

(e.g. by using fsdp or deepspeed)

Does your Model fit into the VRAM?

(for example Large Scale AI Models)

Yes

Look at **A100 40GB**
or **A40**

No

Add GPUs or choose
A100 80GB

No

Only a single GPU

Do you need very large amounts of VRAM

Yes

A100 80GB

No

Is your GPU fully utilized
and the training still too
slow?

No

A40, TinyGPU

Yes

A100 40GB

Which cluster(s) are you planning to use?



Accessing HPC systems

<https://doc.nhr.fau.de/access/overview/>

SSH – Troubleshooting

- Troubleshooting guide: <https://doc.nhr.fau.de/access/ssh-command-line/#troubleshooting>
- FAQs for most frequent SSH problems: <https://doc.nhr.fau.de/faq/#ssh>
- In case of problems with login, send output of the following command to hpc-support@fau.de: `ssh -vv hpcaccount@csnhr.nhr.fau.de`

Working with data for AI

<https://doc.nhr.fau.de/data/filesystems/>

File systems overview

Available file systems differ in size, redundancy and how they should be used

Mount point	Access	Purpose	Technology	Backup	Snapshots	Data lifetime	Quota
/home/hpc	\$HOME	Source, input, important results	NFS	YES	YES	Account lifetime	50 GB
/home/vault	\$HPCVAULT	Mid-/long-term storage	NFS	YES	YES	Account lifetime	500 GB
/home/{woody, saturn, titan, janus, atuin}	\$WORK	General-purpose, log files	NFS	NO	NO	Account lifetime	NHR: 10 TB TIER3: 1 TB
/???	\$TMPDIR	Node-local job-specific dir	SSD/ ramdisk	NO	NO	Job runtime	NO
/anvme/???	\$ws_find <name>)	General-purpose	anvme	NO	NO	Upto 90 days, extendable 10x	-

Working with workspaces on ALEX

Store large models and datasets on workspaces and save time while loading the model/data

- Create workspace with name <name> for the duration of <days> days:
 - After <days> the workspace will be deleted.
 - <days> must be in the range of 1 to 90 days.
 - If <days> is omitted, duration is 1 day.
 - Duration can be changed and extended multiple times (max 10x) later
- `ws_allocate <name> [<days>]`
- `ws_find <name>`

More details: <https://doc.nhr.fau.de/data/workspaces/>

Datasets Preprocessing

Use non GPU-Machines if you don't need GPU-acceleration for preprocessing

- Use standard dataset file formats and integrations (e.g. .tar)
- Preprocess the Dataset separately before the training and not during the training

- Example of CPU only machines for preprocessing:
 - Memoryhog
 - TinyFat

More details: <https://doc.nhr.fau.de/clusters>

Working with large datasets containing small files

- In a job, avoid *accessing large numbers of files*
`$HOME`, `$HPCVAULT`, `$WORK`, `$SATURNHOME`
- **Expensive** operations on NFS (and also parallel file systems):
 - Access file stats like creation/modification time, permissions...
 - Opening/closing files
- These cause high load on servers
 - This slows down your job and impacts all other users
- Use instead
 - **pack files into an archive** (e.g. tar, preferably w/o compression) and use node-local SSDs (huge amounts of file opens are no problem there)
 - if supported by application: **HDF5, file-based databases**

Efficient Dataset Formats for HPC Clusters

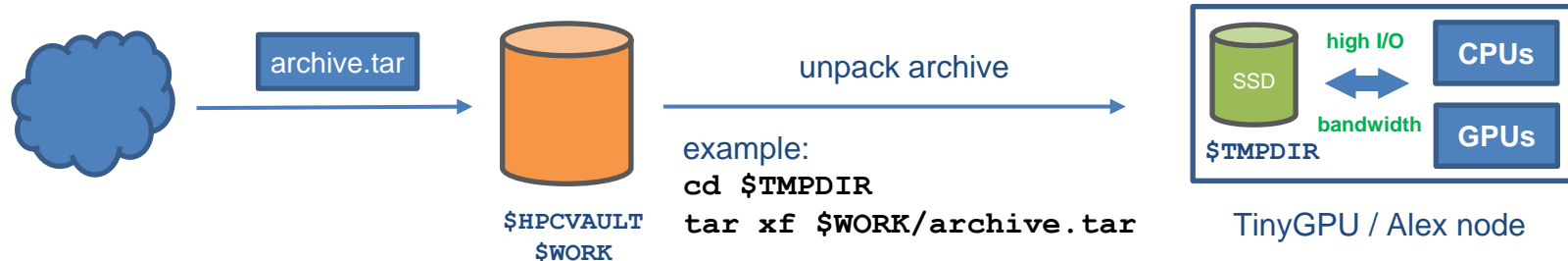
- HPC systems handle large-scale data efficiently.
- Avoid many small files: reduces metadata overhead and optimizes I/O.
- Use formats designed for parallel processing and scalability.
- Example for formats:
 - HDF5: Hierarchical data, parallel I/O, compression.
 - NetCDF: Multidimensional scientific data, metadata-rich.
 - Parquet: Columnar storage, efficient for analytics.

Efficient Dataset Formats for HPC Clusters

- Webdatasets <https://huggingface.co/docs/hub/datasets-webdataset>
- Efficient data pipeline for large-scale datasets.
- Dataset stored as sharded TAR archives.
- Benefits:
 - Reduces I/O overhead with fewer file operations.
 - Optimized for streaming and parallel loading.
- Use Case: Ideal for training ML models on distributed systems.
- Integration: Compatible with Hugging Face and PyTorch DataLoaders.

Working with large datasets containing small files

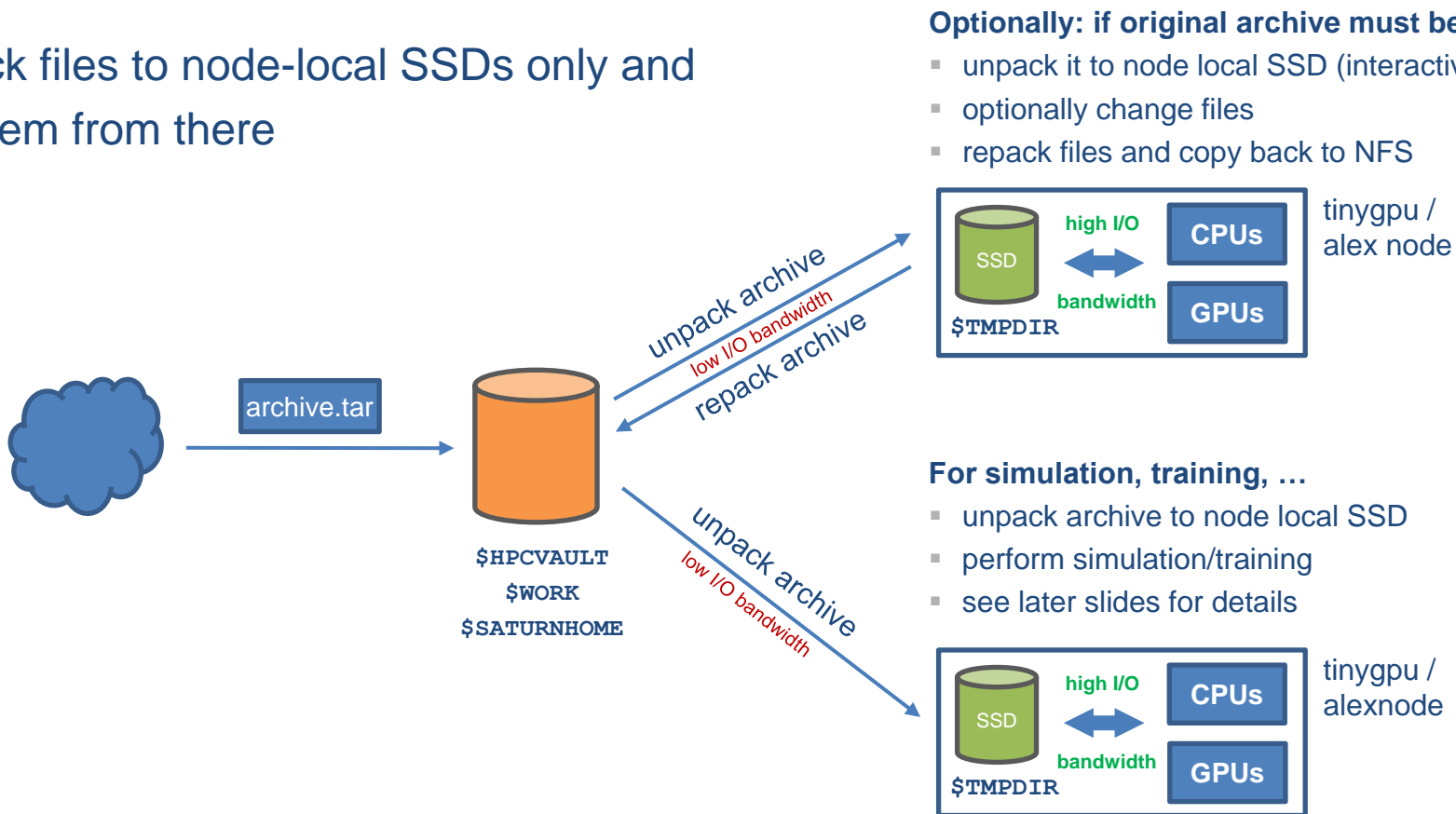
- Pack small files into archive. Do **not unpack** archive to `$HOME/$HPCVAULT/$WORK`
- Unpack files to node-local SSDs only and use them from there



More details: <https://doc.nhr.fau.de/data/staging/>

Working with large datasets containing small files

Unpack files to node-local SSDs only and use them from there



Pre-downloaded Datasets

- Ask your project supervisor if they already have a working directory of their saved datasets
- Growing dataset library: We provide an expanding collection of ML/DL datasets, all available in archive format (e.g. tar)
 - No downloading required
 - Save disk space

Environments

Environment modules: <https://doc.nhr.fau.de/environment/modules/>
Development and Tools: <https://doc.nhr.fau.de/sdt/overview/>
Applications: <https://doc.nhr.fau.de/apps/overview/>

What type of frameworks are you using?



One thing up front...

- The clusters at the computing center are not like your own PC
 - **You are not root**, even if the “How-To” in the github repo you found assumes that
 - **Do not blindly copy&paste** commands you do not understand
 - **Your home directory is not a local hard disk** but a shared volume that is mounted over the network
- Things that do not work:
 - `sudo apt install python-3.9`
 - `yum install`
 - `make && sudo make install`
 - `sudo <anything>`
 - `pip install pandas`

JupyterNotebook

- Start your Jupyternotebook from <https://portal.hpc.fau.de>
 1. Login at the [HPC Portal](#).
 2. Go to the **User** page.
 3. Under **Your accounts**, select the **Account** you want to use for JupyterHub. You might have more than one account.
 4. Click on the button **Go to JupyterHub**.
 5. A new window opens where you have to accept our Terms of Service and then get redirected to the actual JupyterHub.

The screenshot displays the HPC-Portal interface. At the top, there is a navigation bar with the NHR@FAU logo and links for 'HPC-Portal', 'Startseite', 'User', 'Management', 'Advisor', and 'Support'. Below this, a section titled 'Externe Tools' contains two prominent blue buttons: 'Zu ClusterCockpit wechseln' and 'Zu JupyterHub wechseln'. The main content area is titled 'Nutzung der Ressourcen' and features a dropdown menu set to '2024' with the instruction 'Wählen sie das Jahr der Verbrauchsdaten'. Below the dropdown, two resource usage summaries are shown:

- Resource: alex-a100**
 - Diesen Monat: 0 GPU h
 - Jahr Gesamt: 7534 GPU h
 - Monate mit Nutzung: 9
- Resource: alex-a40**
 - Diesen Monat: 0 GPU h
 - Jahr Gesamt: 1193 GPU h
 - Monate mit Nutzung: 7

At the bottom of the resource usage section, there is a blue button labeled 'Verlauf anzeigen' with an external link icon.

JupyterNotebook

- Available resources NHR:
 - 2 cores/4GB on a shared node,
 - one A40 or A100 GPU in [Alex](#),
 - one node of [Fritz](#)
- Available resources Tier-3:
 - 2 cores/4GB on a shared node
 - 1 – 4 dedicated GTX1080Ti GPUs
 - 1 – 4 cores and 8 – 32 GB on TinyFat

When you are done with your work, **stop** your jupyter instance manually. Closing the browser tab or only logging out from Jupyterhub does NOT free resources.

- File -> Hub Control Panel
- Stop My Server

Server Options

This Jupyterhub is for interactive (development) work and **not for production runs**. Production runs have to be done in the traditional HPC way by manually submitting batch jobs from the cluster frontends.

Please be patient after pressing the **Start** button!

- Starting **locally on jupyterhub** typically will take a couple of seconds (~10s). That's the mode you typically should use although you will end up on a shared node (without GPUs).
- The **other job profiles** will submit a batch job in the background and you have to wait for your dedicated resources to become available. As there are no reserved resources for Jupyter jobs, **you may have to wait quite long (some or even many hours)** for your job to start. There is also no way to request specific node types.

When you are done with your work, stop your instance manually. Closing the browser tab or only logging out from Jupyterhub does NOT free resources.

Select a job profile:

Alex 1x A40, 4 hours

show advanced options (only for Slurm job profiles)

Start

The module command

Show all available modules: `module avail`

```
$ module avail
----- /apps/modules/data/applications -----
amber/20p12-at21p11-mpi-gnu          gromacs/2021.5-gcc11.2.0-mpi-mkl
amber/20p12-at21p11-mpi-intel       gromacs/2022.1-gcc11.2.0-mpi-mkl
amber/20p12-at21p11-openmpi-gnu-cuda11.5  gromacs/2022.1-gcc11.2.0-mkl-cuda
----- /apps/modules/data/compiler -----
gcc/10.3.0 gcc/11.2.0 gcc/12.1.0 intel/2021.4.0 intel/2022.1.0 nvhpc/22.1 nvhpc/22.2
----- /apps/modules/data/development -----
cuda/11.3.1          intelmpi/2021.4.0          openmpi/4.1.2-gcc11.2.0-cuda
cuda/11.4.2          intelmpi/2021.6.0          openmpi/4.1.2-intel2021.4.0-cuda
cuda/11.5.0          openmpi/4.1.2-gcc10.3.0-cuda  openmpi/4.1.2-oneapi2021.4.0-cuda
```

Load a module: `module load <modulename>`

Display loaded modules: `module list`

Module command summary

Command	What it does
module avail	List available modules
module whatis	Shows verbose listing of all modules
module list	Shows which modules are currently loaded
module load <pkg>/<version>	Loads specific version of module package, i.e. adjusts environment
module unload <pkg>	Undoes what the load command did
module help <pkg>	Shows a detailed description of package
module show <pkg>	Shows which environment variables are modified and how

<https://doc.nhr.fau.de/environment/modules/>

Using Python

- Use anaconda modules instead of system installation

```
$ module avail python
----- /apps/modules/modulefiles/tools -----
python/3.6-anaconda  python/3.7-anaconda (default)  python/3.8-anaconda
```

- Install packages via conda/pip with `--user` option
- Change default package installation path from `$HOME` to `$WORK`
- It might be necessary to configure a proxy to access external repositories
- Build packages in an interactive job on the target cluster (especially for GPUs)

- More details:
 - <https://doc.nhr.fau.de/sdt/python/>
 - <https://doc.nhr.fau.de/environment/python-env/>

Setting up a python environment

1. If not already exists, create the file `~/.bash_profile` (located in your `$HOME`) with the following content:

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```
2. Ensure you have a Python module loaded:

```
module list
```

output should contain a Python module:
Currently Loaded Modulefiles:
1) python/3.9-conda
3. Store newly installed conda packages and conda environments under `$WORK` to save space in `$HOME` by executing:

```
conda config --add pkgs_dirs $WORK/software/private/conda/pkgs
conda config --add envs_dirs $WORK/software/private/conda/envs
```
4. Check the configuration is used, note that the variable `$WORK` will be expanded to the real path:

```
conda info
```

output should contain:
package cache : /apps/python/...
<real path of \$WORK>/software/private/conda/pkgs
envs directories : /apps/python/...
<real path of \$WORK>/software/private/conda/envs

Setting up a python environment

- Not all compute nodes have direct internet access. Configure a proxy to enable access, either in the shell:

```
export http_proxy=http://proxy:80
export https_proxy=http://proxy:80
```
- `conda create -n <env. name> python=<py. version>`
This creates a conda environment named `<env. name>`
The new environment uses Python of the specified version `<py. version>`
- `conda activate <env. name>`

Source: <https://doc.nhr.fau.de/environment/python-env/>

Installing LitGPT example

- **Claim GPU:**

```
srun --gres=gpu:1 --partition=a40 -t 0-2 --pty /bin/bash -l
```

- **Load Module**

```
module avail ..  
module load git/2.X.X  
module load python/3.X-anaconda
```

- **Get internet connection**

```
export http_proxy=http://proxy:80  
export https_proxy=http://proxy:80
```

- **Create and activate Conda Environment**

```
conda create -n litgpt python=3.10  
conda activate litgpt
```

- **Create or use workspace**

```
ws_allocate litgpt 90  
ws_find <name>
```

Installing LitGPT example

- Clone litgpt into workspace directory:

```
cd /anvme/workspace...
```

```
git clone https://github.com/Lightning-AI/litgpt
```

```
cd litgpt
```

```
pip install -e './[all]'
```

Example conda module installation

- [93m [WARNING] [0m async_io: please install the libaio-devel package with yum
- Package is not available and we have no rights to do yum install

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda install libaio
```

Running jobs

https://doc.nhr.fau.de/batch-processing/batch_system_slurm/

Batch System

- Users can interact with the resources of the cluster via the “Batch system”
- “Batch jobs” encapsulate:
 - Resource requirements (number of nodes, number of GPUs, ...)
 - Job runtime (usually max. 24 hours)
 - Setup of runtime environment
 - Commands for application run
- Batch system will handle queuing of jobs, resource distribution and allocation
- Job will run when resources become available



GPU Jobs on TinyGPU / Alex

- Nodes are shared, GPUs are always exclusive
- Granularity is one GPU with a corresponding portion of CPU and main memory
- Request GPUs with **sbatch** option e.g.
 - `--gres=gpu:rtx3080:1` (to request a specific type)
 - `--gres=gpu:a100:1 --partition=a100` (necessary for V100 and A100 GPUs on TinyGPU)
- More details and examples:
<https://doc.nhr.fau.de/clusters/alex/>
<https://doc.nhr.fau.de/clusters/tinygpu/>

Example: Batch script for Alex

```
#!/bin/bash -l
```

```
#SBATCH --gres=gpu:a40:1
```

Resource requirements

```
#SBATCH --time=06:00:00
```

Max. runtime

```
#SBATCH --job-name=testjob_gpu
```

```
#SBATCH --export=NONE
```

Other job options (name, notifications,...)

```
unset SLURM_EXPORT_ENV
```

Prevent export of environment to job

```
module load python
```

```
conda activate test-environment
```

Set up job environment

```
srun python train.py
```

Actual run of your binary

Example: Batch script for Alex

```
#!/bin/bash -l
```

```
#SBATCH --gres=gpu:a40:1
```

Resource requirements

```
#SBATCH --time=06:00:00
```

Max. runtime

```
#SBATCH --job-name=testjob_gpu
```

```
#SBATCH --export=NONE
```

Other job options (name, notifications,...)

```
unset SLURM_EXPORT_ENV
```

Prevent export of environment to job

```
module load python
```

```
conda activate test-environment
```

```
cd $TMPDIR
```

```
tar xf $WORK/large-archive-with-small-files.tar
```

Set up job environment

```
srunch python train.py datamodule.root="$TMPDIR/"
```

Actual run of your binary

Slurm documentation

- NHR@FAU
 - General: https://doc.nhr.fau.de/batch-processing/batch_system_slurm/
 - Cluster-specific: <https://doc.nhr.fau.de/clusters/overview/>
 - HPC Café on “Slurm - basics, best practices and advanced usage”:
<https://hpc.fau.de/files/2022/04/2022-04-12-hpc-cafe-slurm.pdf>,
<https://www.fau.tv/clip/id/41306>
- Official Slurm documentation
 - Separate documentation for every command and the available options:
https://slurm.schedmd.com/man_index.html
 - Slurm commands and their counterparts in different batch systems:
<https://slurm.schedmd.com/rosetta.pdf>
 - Slurm tutorials: <https://slurm.schedmd.com/tutorials.html>

Containers

<https://doc.nhr.fau.de/environment/apptainer>

Using Containers



- Use pre-built containers or build them yourself
- Build containers from scratch (interactively or via definition file)
- Run/shell/import of a (pre-built) container is possible for all users
- Generally: you are the same user inside the container than outside!
- Container images are build immutable to preserve reproducibility

Using Containers

At NHR@FAU, [Apptainer](#) (formerly known as Singularity) is the standard container solution. It is specifically designed for HPC systems and causes no performance penalties.

- Using existing containers:
 - Download / pull a container from a container repository (DockerHub) and it will be automatically converted into the Apptainer (.sif) format:
 - Online container: `apptainer pull docker://<repository>`
 - Local container: `apptainer build <outfile>.sif docker-archive://<filepath_docker>`
 - Enter container with a shell: `apptainer shell <container_name>`
 - Execute commands inside a container: `apptainer exec <container_name> <command>`
 - Run pre-defined runsript of container: `apptainer run <container_name>` or `./<container name>`
- For Multi-Node: make sure to include `rdma-core`, `libibverbs1`, etc. in your image.
- Check (debug) the output (see [Debugging NCCL](#). For NCCL as used by Pytorch, etc. check for an error such as `NCCL INFO Failed to open libibverbs.so[.1]`.

More details:

- <https://doc.nhr.fau.de/environment/apptainer>

Using containers

Hints:

```
export https_proxy="http://proxy.rrze.uni-erlangen.de:80"  
export https_proxy="http://proxy.rrze.uni-erlangen.de:80"
```

If disk out of space:

```
export APPTAINER_CACHEDIR=$TMPDIR or $WORK  
export APPTAINER_TMPDIR=$TMPDIR or $WORK
```

Nvidia:

```
apptainer remote login --username \${oauthtoken} docker://nvcr.io  
docker login nvcr.io
```

```
Username: \${oauthtoken}  
Password: <token>
```

```
apptainer instance list  
apptainer pull docker://nvcr.io/nvidia/nemo:24.09
```

Docker:

```
apptainer pull docker://lmsysorg/sglang:v0.4.0.post1-cu124-srt
```

- .sif containers will be saved to the current path of the terminal

When to Use Python venvs, Conda, or Containers

- **Containers (Apptainer):**

- Portable, reproducible environments.
- Encapsulate entire software stack.
- Problems with multi-node runs and often hard to setup properly.

- **Python venvs:**

- Lightweight environment management.
- Suitable for simple Python dependencies.
- Limited control over non-Python libraries.

- **Conda:**

- Manages Python and non-Python dependencies.
- Supports complex workflows.
- Ideal for multi-language (e.g., Python + C/C++) requirements.

Some troubleshooting

Good practices

- Check your jobs regularly
 - Are the results OK?
 - Does the job actually use the allocated nodes in the intended way? Does it run with the expected performance?
 - Check if your job makes use of the GPUs
 - **Attach to a running job** (https://doc.nhr.fau.de/batch-processing/batch_system_slurm/#attach-to-a-running-job)
 - Use e.g. “watch nvidia-smi” to check GPU utilization
- Job Monitoring
 - How to use it and what to look out for: <https://doc.nhr.fau.de/job-monitoring-with-clustercockpit/>

ClusterCockpit



HPC-Portal [Startseite](#) [User](#) [Management](#) [Advisor](#) [Support](#)

Externe Tools

[Zu ClusterCockpit wechseln](#)

[Zu JupyterHub wechseln](#)

Nutzung der Ressourcen

2024

Wählen sie das Jahr der Verbrauchsdaten

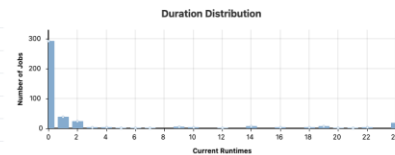
- Ressource: alex-a100
- Diesen Monat: 0 GPU h
- Jahr Gesamt: 7534 GPU h
- Monate mit Nutzung: 9

- Ressource: alex-a40
- Diesen Monat: 0 GPU h
- Jahr Gesamt: 1193 GPU h
- Monate mit Nutzung: 7

[Verlauf anzeigen](#)

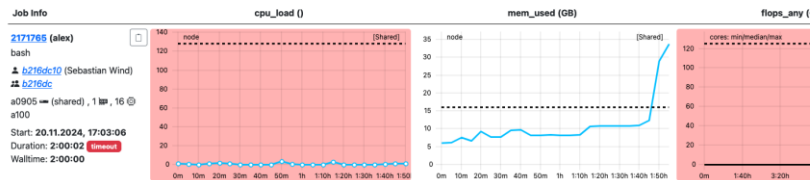
ClusterCockpit [My Jobs](#) [Jobs](#) [Tags](#) [Users](#) [Projects](#) [Analysis](#)

Username	b216dc10
Name	Sebastian Wind
Total Jobs	421
Short Jobs	165
Total Walltime	1207
Total Core Hours	90162



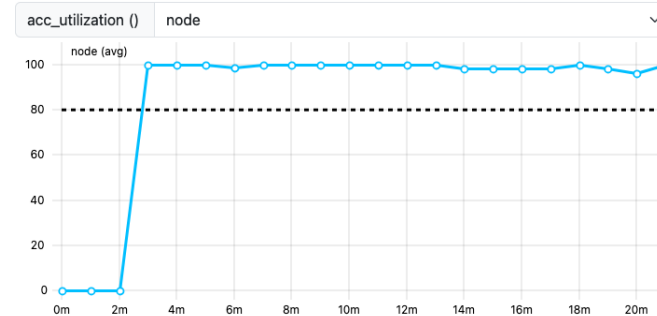
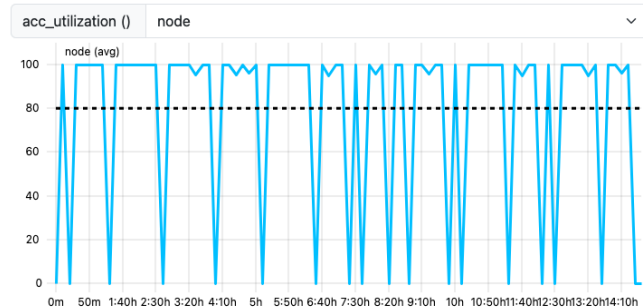
Select Histograms

No footprint histograms selected.



Resource underutilization

- Inefficient usage of allocated resources, like training on a single GPU instead of all available GPUs, often occurs due to improper configuration or unoptimized code
- This results in wasted resources and longer training times
- To address this, ensure the training script explicitly supports multi-GPU setups, and verify GPU utilization across all devices
- Claim the appropriate GPUs for you (not always highend GPU)



ClusterCockpit

The GPU out of memory problem

```
[rank7]: return Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
[rank7]: torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.53 GiB. GPU 0 has a total capacity of 79.26 GiB of which 1.28 GiB is free.
a0931:247812:247812 [7] NCCL INFO cudaDriverVersion 12050
```

- **Start Small:** Use low batch size, minimum model, monitor memory
- **Optimize Inputs/Model:** Resize inputs, adapt architecture, mixed precision
- **Consider Larger GPUs:** Use high-VRAM GPUs
- **Utilize Multi-GPU:** Distribute load with data/model parallelism
- **Consider CPU offloading:** Only for short finetunings

Data bottlenecks

- Low data loading and transfer rates, especially when handling large datasets, can create I/O bottlenecks, reducing GPU utilization
- Monitor your job on ClusterCockpit
 - More details: <https://hpc.fau.de/systems-services/documentation-instructions/job-monitoring-with-clustercockpit/>
- GPU Utilization should be close to 100% for optimized trainings

THANK YOU.

NHR@FAU

<https://doc.nhr.fau.de>

hpc-support@fau.de