Software & Tools

Software, tools, and applications available on the TAU HPC cluster.

- IDEs & Notebooks
  - Jupyter on Slurm
  - PowerIDE User Guide
- Scientific Software
  - AlphaFold
  - AlphaFold3
  - RELION
- Environment & Packages
  - Conda & Mamba Environments

IDEs & Notebooks

Development environments and notebook servers available on the cluster.

IDEs & Notebooks

Jupyter on Slurm

Run JupyterLab interactively on a compute node via SSH port forwarding. For a simpler browser-based experience, see **PowerIDE**.

Step 1: Start an Interactive Job

```
srun -p <partition-name> -A <account-name> --pty bash
```

Replace `<partition-name>` and `<account-name>` with your values from `check_my_partitions`. Note the compute node name you land on (e.g. `compute-0-62`) — you'll need it later.

Step 2: Load the Jupyter Module

```
module load mamba-env158/jupyter
```

Step 3: Start JupyterLab

```
jupyter lab --ip=* --port=8892 --no-browser
```

Once started, you'll see output containing a URL with a token:

```
http://localhost:8892/lab?token=<token>
```

Copy the full URL including the token.

Step 4: SSH Port Forwarding

Open a new terminal on your local machine and run:

```
ssh -N -L 8892:<compute-node-name>:8892 <username>@slurmlogin.tau.ac.il
```

Replace `<compute-node-name>` with the node from Step 1 (e.g. `compute-0-62`) and `<username>` with your TAU username. Keep this terminal open.

Step 5: Open in Browser

Paste the URL from Step 3 into your browser:

```
http://localhost:8892/lab?token=<token>
```

Step 6: Closing

When finished, press `Ctrl+C` in the interactive session to stop JupyterLab, then:

```
exit
```

Also close the SSH tunnel terminal.

Notes

- If port 8892 is in use, pick any other free port — use the same number in both the `jupyter lab` command and the SSH tunnel
- The SSH tunnel must stay open while you work
- For a simpler experience without SSH tunneling, use **PowerIDE** instead

PowerIDE User Guide

PowerIDE provides interactive access to the HPC cluster through a web browser. Run Jupyter notebooks, VS Code, and RStudio directly on compute nodes without needing SSH access.

Access PowerIDE at: https://poweride.tau.ac.il/jupyter

Getting Started

Log in with your TAU university credentials (same as email and other university services).

After logging in, click **Start My Server**. You'll see a Server Options form to configure your compute resources.

When you start your server, PowerIDE submits a Slurm job to the PowerSlurm cluster. Your session runs on a compute node — not on the PowerIDE server itself. This means:

- You get dedicated resources (CPUs, memory, GPUs) on a compute node
- Your job runs through the same Slurm scheduler as other HPC jobs
- Your session will queue if the cluster is busy

Configuring Resources

Partition

Select which partition to run on. The dropdown shows only partitions you have access to. Common options:

- `power-general-shared-pool` — general purpose computing
- `gpu-general-pool` — GPU-enabled nodes

Check with your PI or HPC admin if unsure which partition to use.

QOS

Controls priority and resource limits. **Default (owner)** is usually the right choice. Only valid QOS options for your selected partition are shown.

GPUs

Appears only when a GPU partition is selected. Specify how many GPUs you need (0 if none).

Time (D-HH:MM:SS)

How long your session should run. Default: `04:00:00`. Your session is terminated when time runs out — save your work regularly.

- `02:30:00` — 2.5 hours
- `1-12:00:00` — 1 day and 12 hours

CPUs per task

Default: 1. Increase for multi-threaded code.

Memory

Default: 1G. Examples: `2G`, `8G`, `500M`. Start small and increase if needed — over-requesting delays job start.

Working Directory

Default: your home directory. Change to your project directory to save navigation time after launch.

Stdout / Stderr Directory

Where job logs are written. Default (home directory) is fine for most users.

Starting Your Session

Click the orange **Start** button. PowerIDE submits a Slurm job and shows a progress page. Once a compute node is available (usually 10–60 seconds), you're automatically redirected to JupyterLab.

If the cluster is busy, you can close the browser and come back — your session will start when resources are available.

Using JupyterLab

- **Left sidebar** — file browser, running kernels, extensions
- **Main area** — notebooks, text files, terminals
- **+** button — opens the launcher for new tools

Common tasks:

- **New notebook** — click **+** → choose a Python kernel
- **Terminal** — click **+** → click Terminal (bash shell on the compute node)
- **Upload files** — drag and drop into the file browser
- **Download files** — right-click file → Download

Using VS Code

PowerIDE includes VS Code running in your browser:

1. Click **+** to open the launcher
2. Click the **VS Code** icon
3. VS Code opens in a new tab with access to all your files and the same resources as JupyterLab

Using RStudio

RStudio runs as a separate service on a dedicated compute node. It has its own launch form with R-specific options:

1. Click **+** to open the launcher
2. Click the **RStudio** icon
3. Fill in the resource form and select your R environment
4. Click **Start** — RStudio opens in a new tab once the job is running

R Environment

Select the R environment to load. Each environment is a named module (e.g. `webR-genomics-2024`) with R and a pre-installed set of packages. Contact HPC support if you need a package that isn't available.

R Library Path (optional)

If you have a personal R package library installed in a directory on the cluster, enter its full path here (e.g. `/home/user/R/library`). R will search this directory first, before the environment's default library.

Stopping RStudio

Use the **Stop** button in the PowerIDE topbar to terminate your RStudio job. **Do not use File → Quit Session** — that ends the R session but leaves the Slurm job running, continuing to consume resources.

Python Environments

PowerIDE provides one default kernel: **Python 3.12 (Base)**.

You can register your own conda/mamba environments as kernels:

```
module load mamba/mamba-2.1.1
mamba create -n my-project python=3.11 pandas matplotlib
mamba activate my-project
mamba install ipykernel

# Register as kernel (only visible to you)
python -m ipykernel install --user --name my-project --display-name "My Project (Python 3.11)"
```

Refresh your browser — the new kernel appears in the launcher. To remove a kernel:

```
jupyter kernelspec uninstall kernel-name
```

Stopping Your Server

Always stop your server when done to free resources for others.

- **From JupyterLab** — File → Hub Control Panel → Stop My Server
- **From PowerIDE home** — navigate to https://poweride.tau.ac.il/jupyter/hub/home → Stop My Server
- **VS Code / RStudio** — use the **Stop** button in the PowerIDE topbar

Best Practices

- Request only what you need — over-requesting delays your job and others
- Set a realistic time limit; restart if you need more
- Only request GPUs if your code actually uses them
- Store large datasets in scratch space, not your home directory
- Use Git for code — not for large data files

Scientific Software

Scientific applications available on the cluster.

Scientific Software

AlphaFold

AlphaFold is an AI program developed by DeepMind that predicts protein structures from amino acid sequences.

Databases

The necessary databases are pre-mounted on GPU nodes at `/alphafold_storage/alphafold_db` — no download needed.

Required Parameters

Parameter	Description
`-d <data_dir>`	Path to the supporting data directory
`-o <output_dir>`	Path to store results
`-f <fasta_paths>`	Path to FASTA file(s). Multiple sequences in one file = multimer. Multiple files comma-separated = fold sequentially
`-t <max_template_date>`	Maximum template release date (YYYY-MM-DD)

Optional Parameters

Parameter	Default	Description
`-g`	true	Enable NVIDIA GPU runtime
`-r`	true	Run final relaxation step
`-e`	true	Run relax on GPU
`-n`	all cores	OpenMM threads
`-a`	0	CUDA_VISIBLE_DEVICES — comma-separated GPU list

Parameter	Default	Description
`-m`	monomer	Model preset: `monomer`, `monomer_casp14`, `monomer_ptm`, `multimer`
`-c`	full_dbs	MSA database preset: `reduced_dbs` or `full_dbs`
`-p`	false	Use precomputed MSAs from disk
`-l`	5	Predictions per model (multimer only)
`-b`	false	Benchmark mode — excludes compilation time

Example Job Script

```
#!/bin/bash
#SBATCH --job-name=AlphaFold-Multimer
#SBATCH --partition=gpu2
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=32G
#SBATCH --gres=gpu:1
#SBATCH --output=alphafold_%j.out
#SBATCH --error=alphafold_%j.err

module load alphafold/alphafold_non_docker_2.3.1

bash $ALPHAFOLD_SCRIPT_PATH/run_alphafold.sh \
  -d $ALPHAFOLD_DB_PATH \
  -o ~/output_dir \
  -f $ALPHAFOLD_SCRIPT_PATH/examples/query.fasta \
  -t $(date +%Y-%m-%d)
```

Memory Guidelines

- **Monomer** — at least 32GB RAM
- **Multimer** — at least 64GB RAM; large/complex structures may need 128GB+

Additional Resources

- Sample FASTA: `/home/alphafold_folder/alphafold_multimer_non_docker/example/query.fasta`
- alphafold_non_docker GitHub repository

Scientific Software

AlphaFold3

AlphaFold3 runs via Singularity/Apptainer container on GPU nodes with the `af3` GRES constraint.

Loading the Module

```
module load alphafold3
```

This automatically loads Apptainer and sets the following environment variables:

Variable	Description
`$AF3_CONTAINER`	Path to the Singularity container
`$AF3_MODELS`	Path to model parameters
`$AF3_DB`	Path to the database
`$AF3_SRC`	Path to AlphaFold3 source directory

Bind Mounts

Singularity requires bind mounts to expose host directories inside the container:

```
--bind /path/on/host:/path/inside/container
```

Your input folder can be bound to any path inside the container — `/root/af_input` is not required, it's just an example.

Running AlphaFold3

```
singularity exec --nv \
    --bind $AF3_SRC:/root/custom_folder \
    --bind /tmp:/root/af_output \
    --bind $AF3_MODELS:/root/models \
```

```
    --bind $AF3_DB:/root/public_databases \
    --bind /home/user/alphafold_inputs:/root/custom_folder \
    $AF3_CONTAINER \
    python /root/custom_folder/run_alphafold.py \
        --json_path=/root/custom_folder/fold_input.json \
        --model_dir=/root/models \
        --db_dir=/root/public_databases \
        --output_dir=/root/af_output
```

Replace `/home/user/alphafold_inputs` with the actual path to your input folder.

For input file format, see the AlphaFold3 Input File Guide.

Listing All Available Flags

```
singularity exec --nv \
    --bind $AF3_SRC:/root/custom_folder \
    --bind /tmp:/root/af_output \
    --bind $AF3_MODELS:/root/models \
    --bind $AF3_DB:/root/public_databases \
    $AF3_CONTAINER \
    python /root/custom_folder/run_alphafold.py --helpfull
```

Example Job Script

```
#!/bin/bash
#SBATCH --job-name=alphafold3
#SBATCH --partition=gpu-general
#SBATCH --gres=gpu:1,af3
#SBATCH --cpus-per-task=8
#SBATCH --mem=64G
#SBATCH --time=1-00:00:00
#SBATCH --output=alphafold3_%j.out
#SBATCH --error=alphafold3_%j.err

module load alphafold3
```

```
singularity exec --nv \
    --bind $AF3_SRC:/root/custom_folder \
    --bind /tmp:/root/af_output \
    --bind $AF3_MODELS:/root/models \
    --bind $AF3_DB:/root/public_databases \
    --bind /home/user/alphafold_inputs:/root/custom_folder \
    $AF3_CONTAINER \
    python /root/custom_folder/run_alphafold.py \
        --json_path=/root/custom_folder/fold_input.json \
        --model_dir=/root/models \
        --db_dir=/root/public_databases \
        --output_dir=/root/af_output
```

Unloading

```
module unload alphafold3
```

This also unloads the Apptainer module.

Troubleshooting

- Module not found: `module avail alphafold3` to check the exact name
- No nodes available: `sinfo -o "%N %G"` to check GPU node availability
- Container fails: verify `$AF3_CONTAINER`, `$AF3_MODELS`, `$AF3_DB` are set correctly after module load
- Input files not found: confirm the correct host directory is bound and paths are referenced from inside the container

RELION

RELION is a cryo-EM structure determination package. On the TAU HPC cluster it runs on the dedicated `gpu-relion` partition with X11 forwarding.

Requirements

- Access to the `gpu-relion-users_v2` account — contact HPC support if you don't have it
- SSH connection with X11 forwarding enabled: `ssh -X username@slurmlogin.tau.ac.il`

Starting a RELION Session

Start an interactive job on the GPU RELION partition with X11:

```
srun --ntasks=1 -p gpu-relion-pool -A gpu-relion-users_v2 --qos=owner --x11 --pty bash
```

Load the RELION module:

```
module load relion/relion-4.0.1
```

Launch RELION:

```
relion
```

Notes

- RELION requires X11 — make sure your SSH connection was made with `-X` or `-Y`
- For batch processing pipelines, RELION can submit its own Slurm jobs from within the GUI
- For access or issues contact hpc@tauex.tau.ac.il

Environment & Packages

Managing software environments and package managers.

Conda & Mamba Environments

Conda and Mamba let you create isolated software environments with specific package versions. Use Mamba for faster dependency resolution.

Loading the Module

```
module load mamba/mamba-2.1.1
```

Listing Available Environments

Before creating your own, check if an environment already exists for your needs:

```
conda env list
```

Activating an Existing Environment

```
conda activate /path/to/envs/ENVIRONMENT_NAME
```

Creating Your Own Environment

Create a personal environment in your home directory:

```
conda create --prefix ~/envs/my_env
```

Then activate it:

```
conda activate ~/envs/my_env
```

Installing Packages

Before installing, set your cache directories to a writable location:

```
export CONDA_PKGS_DIRS=$HOME/.conda/pkgs
export CONDA_ENVS_DIRS=$HOME/.conda/envs
export MAMBA_ROOT_PREFIX=$HOME/.mamba
```

Then install packages:

```
conda install <package_name>
# or faster with mamba:
mamba install <package_name>
```

Using Your Environment in a Job Script

```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --account=public-users_v2
#SBATCH --partition=power-general-shared-pool
#SBATCH --qos=public
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=4G
#SBATCH --output=my_job_%j.out

module load mamba/mamba-2.1.1
conda activate ~/envs/my_env

python my_script.py
```

Deactivating & Unloading

```
conda deactivate
module unload mamba/mamba-2.1.1
```