Scientific Software

Scientific applications available on the cluster.

- AlphaFold
- AlphaFold3
- RELION

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

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