ColabFold on Linux Cluster with HTCondor

Jean-Yves Sgro

Table of contents

Introduction

This tutorial is providing information on using the Container (Docker) ColabFold version of AlphaFold2 on local Linux Clusters. The example scripts use the currently latest version: v1.5.5. The links below jump to the designated cluster:

- BCC - Biochemistry cluster .
- CHTC - Center for High Throughput Computing

This tutorial is the hands-on expansion of the Blog entry AlphaFold2 with ColabFold in Container.

> Note: Some of the code was converted from the Colab repository (see below) to the scripts provided with the help of MS Copilot.

What is AlphaFold2?

> AlphaFold2, developed by DeepMind, uses deep learning to predict protein structures from amino acid sequences with high accuracy thanks to its use of neural networks trained on known protein structures. The network is trained with a specific deep learning method: the *attention mechanism*. This technique allows the model to focus on different parts of the input data (in this case, amino acid sequences) to understand the relationships and interactions between them in 3D. By doing so, the model can more accurately predict the spatial arrangement of amino acids in a protein, leading to precise 3D structure predictions. Essentially, it helps the model to "pay attention" to the most relevant parts of the sequence when making predictions. This allows AlphaFold2 to often match experimental results.

Running the "native" AlphaFold2 software requires the installation of a large (2.5 TeraBytes) database of known sequences and 3D structures. Therefore this version can only be run on large clusters. The "Colab" version (see below) is more flexible as it accesses "pre-made" multiple sequence alignments online, avoiding the need for the large database.

ColabFold

AlphaFold2 Colab is a Google Colab notebook that allows users to predict protein structures using the AlphaFold2 model. While free there are usage limits to using this Jupyter notebook version that make this not suitable for all projects. Fortunately, the software is available as a Container image that can run on multiple computers, from laptops to large servers. The presence of a GPU from Nvidia makes the computation faster than CPU-only.

The software is available from github.com/sokrypton/ColabFold/

ColabFold can run directly on a laptop or desktop computer by implementing the specific installation for a Macintosh, Windows, or Linux computer. These computers would typically be only equipped with a CPU and lack a GPU.

However, ColabFold can be run within a "Container". Using this method it can run on laptop and desktop without complex installation, but also on large Linux Clusters that typically are equipped with powerful GPU units.

The purpose of this tutorial is to help running the containerized version of ColabFold on a laptop/desktop and on large clusters.

ColabFold on Linux Cluster

Linux clusters offer large systems equipped with GPUs that accelerate the computations of structure prediction. Using a Container method allows the user to use the software without the complex software installation process.

Biochemistry personnel can use the Biochemistry cluster (BCC) or may have access to the Center for High Throughput Computing (CHTC.)

On a cluster we cannot use a command like `docker run...` as we need to use a *non-interactive* method through the HTCondor *scheduler* and a shell script to depict the job we want to accomplish *i.e.* running ColabFold with a protein sequence inside the container.

The method is similar on both clusters but a few details are different and explored in separate sections.

In both cases we need to create the following:

1. A shell script containing the commands we want to run
2. A submit file in the HTCondor style defining the parameters of the job

ColabFold on BCC

> NOTE: BCC does not currently recognize the GPU and will only work with the slow CPU method. This problem is being addressed.

- BCC account for Biochemistry personnel: see contact info on BCC
- Reminder: users should work from the `/scrath` directory.

The shell script

The container will be created from a preexisting image which has pre-defined parameters pertaining to the use. Some of these settings need to be changed for the process to work on the cluster.

One specific example is that the ColabFold scripts expect a "cache" directory located by default in `/cache` which is part of the "root" Administrator user that cannot be writen to by other process. This is why at the top of the script we define the "home" directory as the "current directory" (`$PWD`) and a few lines below we define the cache to the local directory created.

The command `python -m colabfold.download` dowloads the AlphaFold2 weights. The actual computation is done with the `colab_batch` commnand, in this case requesting an energy minimization with `--amber`. Many more commands exist by requesting help with `colab_batch --help`.

The results are written to a directory we call `output` which is then archived and compressed. This is the only file that will be given back to the user.

Here is the script altogether later called `runaf.sh` in the submit file below:

```
echo $HOSTNAME
export HOME=$PWD
mkdir output
mkdir cache
export XDG_CACHE_HOME=./cache
export MPLCONFIGDIR=./cache
# download AlphaFold2 weights
python -m colabfold.download

echo run test
colabfold_batch --amber --templates --num-models 1 \
--num-recycle 1 --jobname--prefix glucagon --zip  test.fa output

mv output/*.zip .
echo DONE
```

The protein sequence tested is `test.fa` taken from PDB 1gcn. A very short sequence such as that of glucagon can make testing faster. For example:

```
>test_peptide 1gcn
HSQGTFTSDYSKYLDSRRAQDFVQWLMNT
```

The submit file script

HTCondor is a program that can schedule the process of running the job described in the shell script. HTCondor will advertise the job and its requirements (for example requiring a GPU) to the pool of computers (called compute nodes) until a "macth" can be found between the required parameters (GPU, disk space, RAM memory, etc.) HTCondor then transfers the files to the compute node that executes the job, and sends back the resulting files.

By using the options contained in **`--jobname--prefix glucagon --zip`** all files are zipped in a single archive starting with the chosen job name prefix. However, the archive needs to be moved from the `output` directory into the current directory (`.`) to be recognized by HTCondor as a file to transfer back when the job is done (*i.e.* instruction `should_transfer_files = YES` in submit file below.)

Since we want to run a container, we need to specify **`universe = docker`"**. The next line specifies the name of the image, in this case it is pulled from the github repository for ColabFold.

We then specify the files to transfer: the shell script and the protein sequence to run. The remaining lines specify hardware requirements and a naming convention for the system output files. In this case we also use a `batch_name` variable that can help keep records on different runs.

- The `*.out` file contains text that *would* appear on a terminal screen if the job was interactive.
- The `*.err` file will list any error encountered.
- The `*.log` file is a usage report from HTCondor.

We can call this file `runaf.sub`

```
Universe   = docker
docker_image = ghcr.io/sokrypton/colabfold:1.5.5-cuda11.8.0

Executable = runaf.sh
transfer_input_files = runaf.sh, test.fa
should_transfer_files = YES
```

```
when_to_transfer_output = ON_EXIT

request_GPUs = 1
request_memory = 40 GB
request_disk = 20 GB
request_cpus = 4

batch_name = sokrypton
output = colabfold_$(batch_name).out
error = colabfold_$(batch_name).err
log = colabfold_$(batch_name).log

Queue
```

> NOTE: BCC can only support GPU drivers up to `cuda 11.8` this is why we chose this container image. CHTC will use a different image.

Running the job

When all 3 files are in place: `runaf.sub`, `runaf.sh`, and `test.fa` we can submit the job with:

```
condor_submit runaf.sub
```

We can monitor the running of the job in the queue with:

```
condor_q
```

If there is a problem the job can be removed with the job number provided by `condor_q` shown below as `123456.0`.

```
condor_rm 123456.0
```

When the job is done, the user will receive the `*.zip` file(s) as well as the job report files.

The output can be extracted with the command:

```
unzip *.zip
```

ColabFold on CHTC

The method on CHTC is very similar, but the files created above for BCC are not completely exchangeable. Therefore, if you are on CHTC you should follow the instructions below.

> CHTC account: All UW personnel can request a CHTC account, usually with the approval of a PI supervisor. Use this link to request an account.

On CHTC we use the **`Universe = container`** which can accommodate other container formats, including docker images. However, **`Universe = docker`** would also work with the examples below as they we are using docker images (as opposed to singularity `.sif` images.)

CHTC shell scripts

The BCC shell script above can be used "as is" for users that do not make a lot of predictions as the AlphaFold2 weights will be downloaded with each job.

CHTC users that will compute a large number of structures may prefer to save the AlphaFold2 weights in a **`/staging`** directory (to be requested from CHTC: chtc@cs.wisc.edu) and copy the weights back to a running container rather than download them each time a job is run. The scripts below detail that process.

The scripts were crafted with help of Copilot from the information located in the ColabFold Wiki.

CHTC GPUs support a higher level of "Cuda drivers" than BCC and we'll use the version 12.2.2 below.

Download script

This script is meant to download the AlphaFold2 weights into the `/staging/` directory. I called it `dl.sh`. The `colabfold/params` directory is automatically created within the defined `cache` and the files are then moved (command `mv`) to the `/staging/...` directory. (*Note*: Your directory will have a different name so edit before using!

The `ls` command is to simply verify that all the files were present before transfer.

```
#!/bin/bash
export HOME=$PWD
mkdir output
mkdir cache
export XDG_CACHE_HOME=./cache
export MPLCONFIGDIR=./cache

python -m colabfold.download

ls -lhat ./cache/colabfold/params/*

mv ./cache/colabfold/params/*.txt /staging/jsgro/af2_weights
mv ./cache/colabfold/params/*.npz /staging/jsgro/af2_weights
mv ./cache/colabfold/params/LICENSE /staging/jsgro/af2_weights

echo DONE
```

In the next script, we'll use the pre-downloaded weights by copying them within the running container.

Download submit file.

The following submit file does not require a GPU. I called it `download_weights.sub`.

```
Universe   = container
container_image = docker://ghcr.io/sokrypton/colabfold:1.5.5-cuda12.2.2
Executable = dl.sh
transfer_input_files = dl.sh
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
RequestCpus = 4
RequestMemory = 8GB
RequestDisk = 10GB
Log        = job.log
Output     = job.out
Error      = job.err
Queue
```

Submit the job with

```
condor_submit download_weights.sub
```

On CHTC the following command is best to monitor the queue rather than the command `condor_q`.

```
condor_watch_q
```

The terminal will be refreshed every 10 seconds, and display the jobs as *Idle* in yellow, *Running* in cyan blue. Jobs that finished without error are shown in green, and red if on hold or running with error.

The result of these steps will be the download and transfer of the AlphaFold2 weights within a directory on your `/staging` directory.

Note: saving on your regular user area would work but may use a lot of local space and might exacerbate the system. Check with CHTC to verify how to use if in doubt.

Structure prediction script

The main modification to this script is that we copy the AlphaFold2 weight files from the `/staging` directory (your directory name will be different.) For this to work we also pre-create the `colabfold/params/` directory within the predefined `cache` (this is happening within the container context).

This script compared to the BCC script has less limitations in the number of models or recycle.

I called this script (also) `runaf.sh`

```
#!/bin/bash
echo $HOSTNAME

export HOME=$PWD
mkdir ./output
mkdir -p ./cache/colabfold/params/
export XDG_CACHE_HOME=./cache
export MPLCONFIGDIR=./cache

# python -m colabfold.download
cp /staging/jsgro/af2_weights/* ./cache/colabfold/params/
```

```
echo run test
colabfold_batch --amber --templates --num-recycle 3 \
--use-gpu-relax  --jobname--prefix glucagon --zip  test.fa output

mv output/*.zip .

rm *64 # remove binary temp files

echo DONE
```

The protein sequence tested is `test.fa` taken from PDB 1gcn. A very short sequence such as that of glucagon can make testing faster. For example:

```
>test_peptide 1gcn
HSQGTFTSDYSKYLDSRRAQDFVQWLMNT
```

Structure prediction submit file

I called this file `runaf.sub`:

```
Universe   = container
container_image = docker://ghcr.io/sokrypton/colabfold:1.5.5-cuda12.2.2
Executable = runaf.sh
transfer_input_files = runaf.sh, test.fa
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
# CHTC requirements for GPU
requirements = (HasGpulabData == true)
request_GPUs = 1
+WantGPULab = true
request_memory = 40 GB
request_disk = 20 GB
request_cpus = 4

batch_name = sokrypton
output = colabfold_$(batch_name).out
error = colabfold_$(batch_name).err
log = colabfold_$(batch_name).log
```

```
Queue
```

Running the prediction job:

```
condor_submit runaf.sub
```

On CHTC the following command is best to monitor the queue:

```
condor_watch_q
```

The output will be archived as a `.zip` file.

Multimer prediction

The current ColabFold can automatically decipher that a protein sequence was provided with more than one sequence and will adjust accordingly. In this way the user does not have to specify specific parameters pertaining to multimer run.

However, the file format for multiple sequences is slightly different than the standard FastA format: in a standard format, in a multiple sequence file each sequence starts with a **>** symbol followed by a sequence name.

The colabfold multiple sequence format has only one **>** symbol and the various sequences are separated by a colon **:**.

For example, a colabfold version of hemoglobin is shown below, I called this file `hemoglobin-colab.fa`:

```
>hemoglobineA+B
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNA
VAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSK
YR:
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH:
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNA
VAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSK
```

```
YR:
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
```

To use this protein sequence the commands would be exactly the same:

```
colabfold_batch --amber --templates --num-recycle 3 --use-gpu-relax  --jobname--prefix
```

Note that multiple lines starting with `colabfold_batch` could be included within the shell script to run multiple, independent predictions.

ColabFold on laptop/desktop

The instructions to running on a personal computer such as a laptop or desktop are detailed in this Wiki document:

Running ColabFold in Docker

The local computer needs to have a supporting software (*e.g.* Docker or Podman) that can open an existing "image" to create a "container" that will run the "bottled" operating system, libraries, and software needed. (Singularity is typically only available on Linux clusters.)

The provided commands start with `docker` but the same command could be replaced with one starting with `podman` if that is what is installed.

> *Note*: The container used for these exercises *do not* work on Silicon-based Macintoshes (*e.g.* M1, M2, M3,etc.) but function on Intel-based Macintoshes. However, the ColabFold software can be installed directly on Silicon-based Macintoshes following the instructions for local installation also detailed in AlphaFold2 on Macintosh M1.