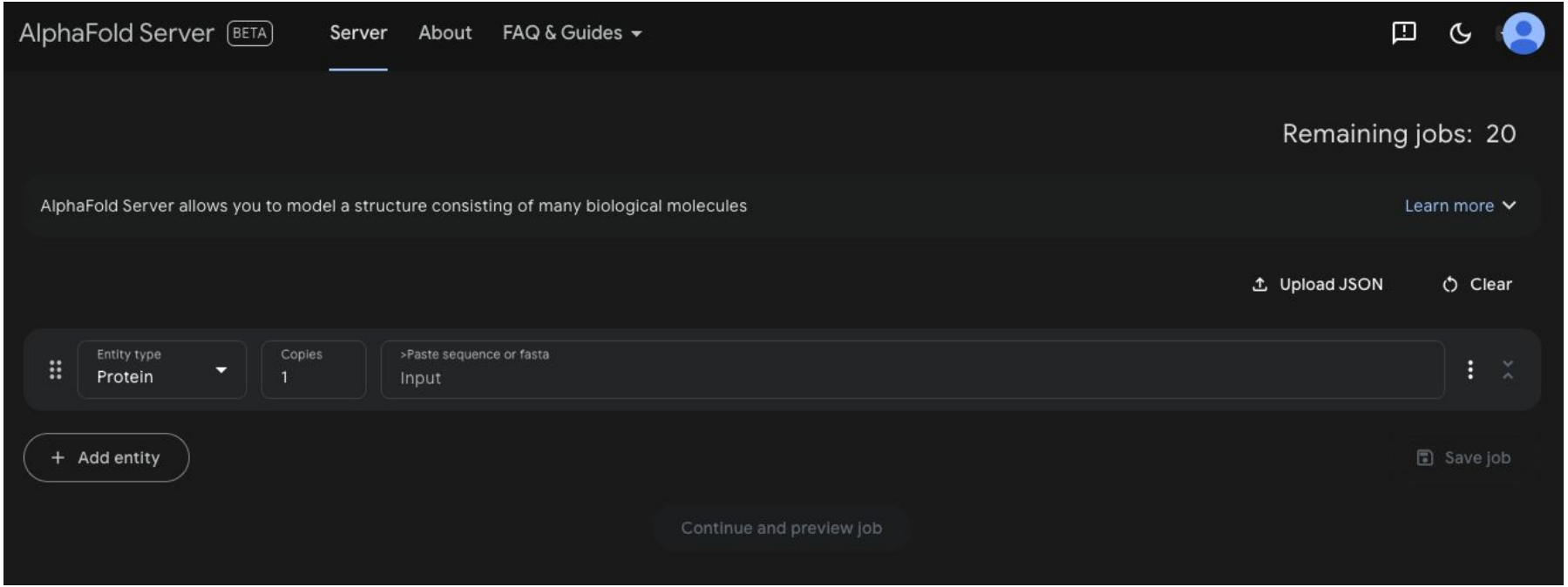
Run AlphaFold3 on BioHPC

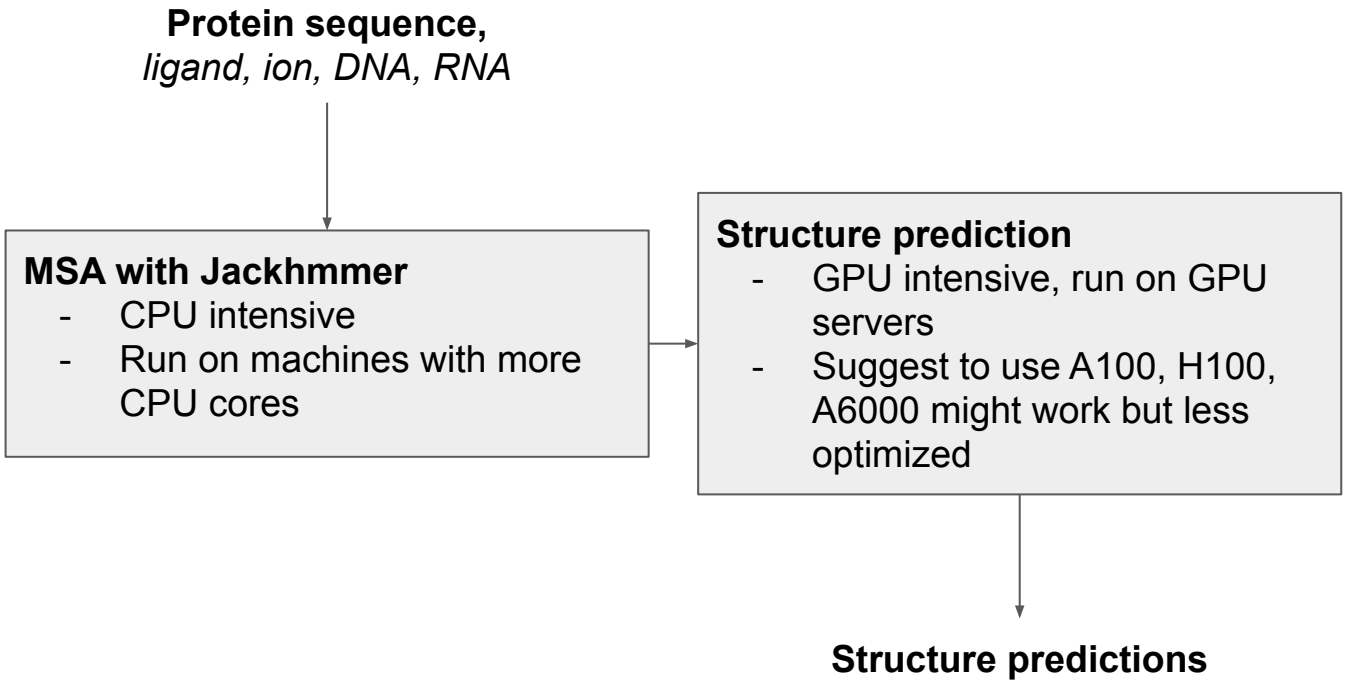
Yi-Yuan (Ian) Lee
Brito lab | Computational Biology | Cornell University

THE AlphaFold Server



https://alphafoldserver.com/

Brief introduction - AlphaFold3 pipeline



CPU and GPU servers on BioHPC

Machine Class	Machines available / total	Compute units per hour	Price per hour*	Price per core-hour*
○ general: 12-16 cores, 48GB RAM	0 / 3	0.33	$0.28	2.00¢
○ medium memory gen1: 24 cores, 128GB RAM	11 / 23	0.39	$0.33	1.39¢
○ medium memory gen2: 40 cores, 256GB RAM	6 / 11	0.58	$0.49	1.24¢
○ medium memory gen3: 48-64 cores, 256GB RAM	7 / 9	0.77	$0.66	1.13¢
○ large memory gen1: 64 cores, 512GB RAM	9 / 10	0.76	$0.65	1.01¢
○ large memory gen2: 80-112 cores, 512GB RAM	9 / 17	1.00	$0.85	0.93¢
○ large memory gen3: 104-112 cores, 512GB RAM	3 / 7	1.15	$0.98	0.89¢
○ extra large memory: 64-256 cores, 1-3TB memory**	3 / 4	0.94 - 2.00	$0.80 - 1.70	0.61 - 1.31¢
○ GPU gen1 servers**	3 / 3	0.73 - 0.94	$0.62 - 0.80	NA
○ GPU gen2 servers**	3 / 6	1.52 - 3.08	$1.29 - 2.61	NA
○ restricted	NA	NA	NA	NA
○ software	2 / 2	0.10	$0.09	NA

Server	cbsugpu05	cbsugpu06	cbsugpu07	cbsugpu08	cbsugpu09	cbsugpu10
System Info	Linux (Rocky 9.4) 112 cores; 512GB RAM AVX AVX2 supported /workdir 7.0TB SSD	Linux (Rocky 9.4) 112 cores; 512GB RAM AVX AVX2 supported /workdir 7.0TB SSD	Linux (Rocky 9.4) 128 cores; 512GB RAM AVX AVX2 supported /workdir 15.0TB NVMe SSD	Linux (Rocky 9.4) 112 cores; 512GB RAM AVX AVX2 supported /workdir 14.0TB NVMe SSD	Linux (Rocky 9.4) 256 cores; 512GB RAM AVX AVX2 supported /workdir 7.0TB NVMe SSD	Linux (Rocky 9.4) 256 cores; 512GB RAM AVX AVX2 supported /workdir 7.0TB NVMe SSD
GPU CUDA Compatibility	2 x NVIDIA A40 11.1+	2 x NVIDIA A40 11.1+	1 x Nvidia A100 80Gb 11.0+	2 x H100 11.8+	A6000 Lovelace 12.6	A6000 Lovelace 12.6
Cost	1.77 compute units per hour	1.77 compute units per hour	1.86 compute units per hour	3.08 compute units per hour	1.52 compute units per hour	1.52 compute units per hour
Core & CPU Speed Mem Speed & Latency	2.6 & 130.4 kEvents/s 29.5 Gb/s & 0.03 ms	2.6 & 130.5 kEvents/s 30.6 Gb/s & 0.03 ms	2.9 & 147 kEvents/s 38.5 Gb/s & 0.03 ms	2.8 & 168 kEvents/s 37.2 Gb/s & 0.03 ms	4.6 & 600.2 kEvents/s 52.6 Gb/s & 0.02 ms	4.6 & 618.4 kEvents/s 51.1 Gb/s & 0.02 ms
Access Notes						

Run AlphaFold3 with a Docker image

```
# run one prediction
docker1 run -it \
    --volume $INPUT:/root/af_input \
    --volume $OUTDIR:/root/af_output \
    --volume $WEIGHTS:/root/models \
    --volume $DATABASE:/root/public_databases \
    --gpus all \
    biohpc_netid/alphafold3 \
    python run_alphafold.py \
    --json_path=/root/af_input/fold_input.json \
    --model_dir=/root/models \
    --db_dir=/root/public_databases \
    --output_dir=/root/af_output
```

Binds the path on server to the docker image.

Example:

```
/home/users/username/input_folder:/root/af_input
```

```
/home/users/username/input_folder/my_input.json
```

→

```
/root/af_input/my_input.json
```

*Only do this when you do **not** have too many predictions and you can afford running the whole process on a GPU server.*

[optional] Create AF3 Singularity images for HPC

- g3.xl with a A100 GPU
- Ubuntu 24.04
- Commit: 2b8e912

```
sudo apt install singularity-container
docker build -t alphafold3 -f docker/Dockerfile .

docker run -d -p 5000:5000 --restart=always --name registry registry:2
docker tag alphafold3 localhost:5000/alphafold3
docker push localhost:5000/alphafold3

SINGULARITY_NOHTTPS=1 singularity build alphafold3.sif
docker://localhost:5000/alphafold3:latest

# test the singularity file
singularity exec --nv alphafold3.sif sh -c 'nvidia-smi'
```

[source]

Setup AlphaFold3

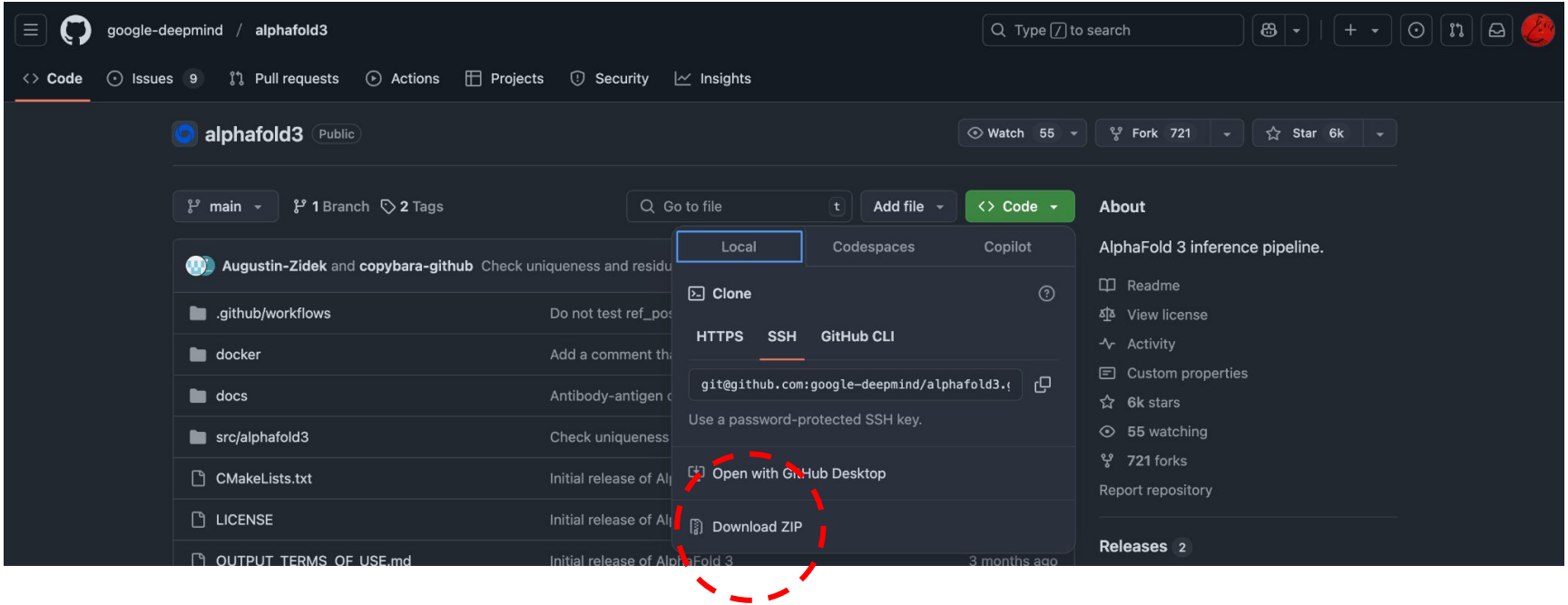
1. AlphaFold3 codebase
2. Model weights
3. Public databases

** This setup section assumes you do not have a lab server and are doing this on your laptop/desktop.*

Clone the repository from GitHub

local

- Go to https://github.com/google-deepmind/alphafold3?tab=readme-ov-file
- Download or git clone the repo if you have the sshkey set. Upload the folder to the server.



Obtain model parameters and download databases

local

- AlphaFold 3 GitHub
- This repository contains all necessary code for AlphaFold 3 inference. To request access to the AlphaFold 3 model parameters, please complete this form. Access will be granted at Google DeepMind's sole discretion. We will aim to respond to requests within 2–3 business days. You may only use AlphaFold 3 model parameters if received directly from Google. Use is subject to these terms of use.
- 626GB data will be downloaded. With fetch_databases.sh
- **Recommend to have a copy on your local machine, use wired network while transferring to BioHPC.**

```
(base) [18:14:25] ~/Downloads/alphafold3-main$ tree -L 1
.
├── CMakeLists.txt
├── LICENSE
├── OUTPUT_TERMS_OF_USE.md
├── README.md
├── WEIGHTS_PROHIBITED_USE_POLICY.md
├── WEIGHTS_TERMS_OF_USE.md
├── dev-requirements.txt
├── docker
├── docs
├── fetch_databases.sh
├── pyproject.toml
├── requirements.txt
├── run_alphafold.py
├── run_alphafold_data_test.py
├── run_alphafold_test.py
└── src
```

Create Docker image and transfer data

server

```
# build docker image, this step takes a while
cd your_alphafold3_folder/
mv docker/dockerfile .
docker1 build -t alphafold3 $PWD
```

local

```
# move public_databases and weight to server
# I suggest tar your mmcif_files/ for faster transfer
tar -cf mmcif_files.tar mmcif_files/
rsync -av path_to_public_databases
netid@server_name.biohpc.corenll.edu:/workdir/alphafold3
rsync -av path_to_af3.bin.zst
netid@server_name.biohpc.corenll.edu:/workdir/alphafold3/model
```

Setup ssh-key on BioHPC

```
# on your local machine
> ssh-keygen
# and press enter until the end
> less $HOME/.ssh/id_rsa.pub  # copy the string in the file
# on remote machine, paste your key in
> $HOME/.ssh/authorized_keys
```

Prepare MSA input

Input data format

```
{
  "name": "Job name goes here",
  "modelSeeds": [1, 2],
  "sequences": [
    {"protein": {...}},
    {"rna": {...}},
    {"dna": {...}},
    {"ligand": {...}}
  ],
  "bondedAtomPairs": [...],  # Optional
  "userCCD": "...",  # Optional
  "dialect": "alphafold3",  # Required
  "version": 2  # Required
}
```

```
{
  "protein": {
    "id": "A",
    "sequence": "PVLSCGEWQL",
    "modifications": [
      {"ptmType": "HY3", "ptmPosition": 1},
      {"ptmType": "P1L", "ptmPosition": 5}
    ],
    "unpairedMsa": ...,  # Mutually exclusive with unpairedMsaPath.
    "unpairedMsaPath": ...,  # Mutually exclusive with unpairedMsa.
    "pairedMsa": ...,  # Mutually exclusive with pairedMsaPath.
    "pairedMsaPath": ...,  # Mutually exclusive with pairedMsa.
    "templates": [...]
  }
}
```

MSA step

1. Prepare a fasta file for all of the sequences.
2. Prepare a pair file.
3. Make sure your fasta file has the same header as the ones in pair.txt. Ideally, use only UniProtKB id.
4. Run the prepare_json.py
5. Upload the output folder to the server
6. Run run_MSA_LM.sh
7. If failed meaning the htop not showing CPU usage but not getting all results. Kill the process and run fix_unfinished_jobs.sh
8. Then run run_MSA_LM.sh again, until all jobs are done.
9. On your local computer, run get_results.sh periodically. Expecting large data so prepare the space befor running.

Explain prepare_json.py

```python
from Bio import SeqIO
import os

prot2seqs = {}
for record in SeqIO.parse("path_to_your_fasta", "fasta"):
   prot2seqs[record.id.strip().replace(">", "")] = str(record.seq)


# make json for msa
outdir = "path_to_output_folder"
if not os.path.exists(outdir):
   os.makedirs(outdir)
for prot, seq in prot2seqs.items():
   with open(f"{outdir}/single_{prot}.json", "w") as file:
       json.dump(
           {
               "name": f"{prot}",
               "modelSeeds": [
                   1,
               ],  #
               "sequences": [
                   {
                       "protein": {
                           "id": "A",
                           "sequence": f"{seq}",
                       }
                   }
               ],
               "dialect": "alphafold3",  # Required
               "version": 1,  # Required
           },
           file,
           indent=4,
       )
```

- Make sure your fasta file has the same header as the ones in pair.txt. Ideally, use only UniProtKB id.
- This script takes your fasta file and generate json files for MSA step.

Explain the run_MSA_LM.sh [part 1]

```
# assume you transfer alphafold3/ with model/ and public_databases/ to
/workdir on remote server

# untar the mmcif_files.tar
cd /local/workdir/alphafold3/public_databases
if [ -d "mmcif_files" ]; then
        echo "[date] mmcif_files already extracted!"
else
        tar -xf mmcif_files.tar
        echo "[date] mmcif_files extracted!"
fi

# build the docker image
cd /local/workdir/alphafold3/ && docker1 build -t alphafold3 -f
dockerfile && echo "[date] Docker image built!"

# transfer your json_dir/ to /workdir
rsync -av your_json_dir netid@server.biohpc.cornell.edu:/workdir

# split json files into multiple jobs
ncpu=$(nproc --all) && njob=$(( ncpu/ 8 ))
INPUT="/local/workdir/"$(basename $json_dir_name)
ninput=$(ls $INPUT | wc -l) && ninput=$(( ninput/$njob )) &&
ninput=$(( ninput+1 ))
echo "[date] ncpu: $ncpu, njob: $njob"
for i in $(seq 1 $njob); do
  mkdir input_${i};
  export i=$i;
  export INPUT=$INPUT;
  ls $INPUT | head -n $ninput | xargs -I @@ bash -c 'mv $INPUT/@@
input_${i}';
done  && echo "[date] Data copied and split!"
```

Extraction takes around 1 hour

Build docker image on server, this takes around 1 hour. If this command complain docker file not found, remember to move the dockerfile from docker/

Explain the run_MSA_LM.sh [part 2]

```
MDIR="/local/workdir/alphafold3"
DATABASE="${MDIR}/public_databases"
WEIGHTS="${MDIR}/model"

OUTDIR=$(date +'%y%m%d%H'_$(hostname | awk -F '.' '{print
$1}'))
OUTDIR="/local/workdir/output_${OUTDIR}"
mkdir -p $OUTDIR

INPUT_PREF="/local/workdir/input_"
NPROC=$njob

cd /workdir/alphafold3
for i in $(seq 1 $NPROC); do
    INPUT="${INPUT_PREF}${i}"

    docker1 run \
        --volume $INPUT:/root/af_input \
        --volume $OUTDIR:/root/af_output \
        --volume $WEIGHTS:/root/models \
        --volume $DATABASE:/root/public_databases \
        --cpus 8 \
        biohpc_netid/alphafold3 \
        python run_alphafold.py \
         --input_dir=/root/af_input \
         --model_dir=/root/models \
         --db_dir=/root/public_databases \
         --output_dir=/root/af_output \
         --norun_inference &
done
wait
```

MDIR: where you clone the alphafold3 GitHub repo
DATABASE: the public_databases location
WEIGHTS: the model location

8 is the most optimal number for cost and compute efficiency

Update this with your Docker image, check with docker1 images

MSA only, & (background)

Email notification

```
# Email notification
if [ $? -eq 0 ]; then
  echo "[`date`] Completed successfully." | mutt -s "${hostname} Finished" netid@cornell.edu
else
  echo "[`date`] Encountered an error."| mutt -s "${hostname} Error" netid@cornell.edu
fi
```

Explain fix_unfinished_jobs.sh

```
# get data list
for file in $(find input*/*json -maxdepth 1); do jq '.name' $file | tr '[:upper:]' '[:lower:]'
| sed 's/"//g'  >> uniprotkb.txt; done

data=() # list
while IFS= read -r line; do
    data+=("$line")
done < "uniprotkb.txt"

# Define an array of output directories
outdirs=("output_dir1" "output_dir2")

# Initialize an array to store finished jobs
finished=()

# Iterate over each output directory
for outdir in "${outdirs[@]}"; do
    # Check if the directory exists
    if [ -d "$outdir" ]; then
        # Read the contents of the directory and add to finished array
        while read -r folder; do
            finished+=("$folder")
        done < <(ls "$outdir")
    else
        echo "Directory $outdir does not exist."
    fi
done

for i in "${data[@]}"; do echo $i; done | sort > tmp_data.txt
for i in "${finished[@]}"; do echo $i; done | sort > tmp_finished.txt

for i in "${finished[@]}"; do echo $i; done | sort > tmp_finished.txt
comm -23 tmp_data.txt tmp_finished.txt > unfinished.txt

# load the unfinished.txt into a hash table
unset hash_table
declare -A hash_table
while IFS= read -r line; do
  hash_table["$line"]=1
done < "unfinished.txt"

mkdir -p /workdir/unfinished_data

find input*/*json -maxdepth 1 | while read file; do
          name=$(jq '.name' $file | tr '[:upper:]' '[:lower:]' | sed 's/"//g' )
          if [[ -v hash_table[$name] ]]; then
              cp $file  /workdir/unfinished_data
          fi
done
```

Get all UniProtKB ids in input

Get finished jobs

Get unfinished jobs

Explain fix_unfinished_jobs.sh

```
cd /workdir
mkdir archive_finished_jobs
mv input* archive_finished_jobs/

ncpu=$(nproc --all) && njob=$(( ncpu/ 8 ))
INPUT="unfinished_data"

ninput=$(ls $INPUT | wc -l) && ninput=$(( ninput/$njob )) && ninput=$(( ninput+1 ))
echo "[date] ncpu: $ncpu, njob: $njob"
for i in $(seq 1 $njob); do
  mkdir input_${i};
  export i=$i;
  export INPUT=$INPUT;
  ls $INPUT | head -n $ninput | xargs -I @@ bash -c 'mv $INPUT/@@ input_${i}';
done  && echo "[date] Data copied and split!"

# run MSA
echo "[date] Running MSA..."
OUTDIR=$(date +'%y%m%d%H'_$(hostname | awk -F '.' '{print $1}'))
OUTDIR="/local/workdir/output_${OUTDIR}"
mkdir -p $OUTDIR

MDIR="/local/workdir/alphafold3"
DATABASE="${MDIR}/public_databases"
WEIGHTS="${MDIR}/model"

INPUT_PREF="/local/workdir/input_"
NPROC=$njob

cd /workdir/alphafold3
for i in $(seq 1 $NPROC); do
    INPUT="${INPUT_PREF}${i}"
    docker1 run \
        --volume $INPUT:/root/af_input \
        --volume $OUTDIR:/root/af_output \
        --volume $WEIGHTS:/root/models \
        --volume $DATABASE:/root/public_databases \
        --cpus 8 \
        biohpc_netid/alphafold3 \
        python run_alphafold.py \
         --input_dir=/root/af_input \
         --model_dir=/root/models \
         --db_dir=/root/public_databases \
         --output_dir=/root/af_output \
         --norun_inference &
done
wait
```

Run MSA again with json files in unfinised_data

If you need to do this multiple times, remember to remove the temporary files generated in this process to avoid file conflicts

Explain get_results.sh

```
# run this periodically on your local machine

rsync -av netid@server.biohpc.cornell.edu:/workdir/output* .
```

Prepare structure inference input

Prepare structure inference input

1. Prepare the output from the previous step with prepare_structure_inference_input_json.py
2. Then you will have a folder with a msas_templates/ and a list of json files.
3. Run structure_inference.sh to do prediction
4. If the machine you rent have multiple GPUs, I will need to try on your machine to use both of them for prediction.

Prepare structure inference input

```python
# create comb msa
def prep_msa(
    hp,
    bp,
    hp_paired_msa_file,
    hp_unpaired_msa_file,
    bp_paired_msa_file,
    bp_unpaired_msa_file,
):
    """
    Prepares and writes paired and unpaired multiple sequence alignments (MSA) to specified
files for both hp and bp.

    Parameters:
    hp (dict): Dictionary containing the hp sequences and their associated MSA data.
    bp (dict): Dictionary containing the bp sequences and their associated MSA data.
    hp_paired_msa_file (str): File path to write the hp paired MSA.
    hp_unpaired_msa_file (str): File path to write the hp unpaired MSA.
    bp_paired_msa_file (str): File path to write the bp paired MSA.
    bp_unpaired_msa_file (str): File path to write the bp unpaired MSA.

    Notes:
    - If any of the specified files already exist, they will not be overwritten.
    - The MSA data is expected to be in the "sequences" key of the hp and bp dictionaries,
under the "protein" key.
    """
    # if any of the files does not exist, then create the files
    # get the pairedMsa and unpairedMsa from loaded json
    if not os.path.exists(hp_paired_msa_file):
        hp_paired_msa = hp["sequences"][0]["protein"]["pairedMsa"]
        hp_unpaired_msa = hp["sequences"][0]["protein"]["unpairedMsa"]
        # write to a a3m file
        with open(hp_paired_msa_file, "w") as f:
            f.write(hp_paired_msa)
        with open(hp_unpaired_msa_file, "w") as f:
            f.write(hp_unpaired_msa)
    if not os.path.exists(bp_paired_msa_file):
        bp_paired_msa = bp["sequences"][0]["protein"]["pairedMsa"]
        bp_unpaired_msa = bp["sequences"][0]["protein"]["unpairedMsa"]
        # write to a a3m file
        with open(bp_paired_msa_file, "w") as f:
            f.write(bp_paired_msa)
        with open(bp_unpaired_msa_file, "w") as f:
            f.write(bp_unpaired_msa)
```

```python
def prep_mmccif(
    hp,
    bp,
    hp_mmcif_file,
    bp_mmcif_file,
):
    if len(glob(f"{hp_mmcif_file}*cif")) == 0:
        for idx, template in
enumerate(hp["sequences"][0]["protein"]["templates"]):
            hp_mmcif = template["mmcif"]
            with open(f"{hp_mmcif_file}_{idx}.cif", "w") as f:
                f.write(hp_mmcif)
    if len(glob(f"{bp_mmcif_file}*cif")) == 0:
        for idx, template in
enumerate(bp["sequences"][0]["protein"]["templates"]):
            bp_mmcif = template["mmcif"]
            with open(f"{bp_mmcif_file}_{idx}.cif", "w") as f:
                f.write(bp_mmcif)
```

Prepare structure inference input

```python
def update_path(hp, bp, hp_paired_msa_file, bp_paired_msa_file,
hp_unpaired_msa_file, bp_unpaired_msa_file, hp_mmcif_file, bp_mmcif_file):
    del hp["sequences"][0]["protein"]["pairedMsa"]
    del hp["sequences"][0]["protein"]["unpairedMsa"]
    del bp["sequences"][0]["protein"]["pairedMsa"]
    del bp["sequences"][0]["protein"]["unpairedMsa"]
    hp["sequences"][0]["protein"]["pairedMsaPath"] = hp_paired_msa_file.replace(
        comb_msa_ref, msa_ref_ncsa
    )
    hp["sequences"][0]["protein"]["unpairedMsaPath"] =
hp_unpaired_msa_file.replace(
        comb_msa_ref, msa_ref_ncsa
    )

    bp["sequences"][0]["protein"]["pairedMsaPath"] = bp_paired_msa_file.replace(
        comb_msa_ref, msa_ref_ncsa
    )
    bp["sequences"][0]["protein"]["unpairedMsaPath"] =
bp_unpaired_msa_file.replace(
        comb_msa_ref, msa_ref_ncsa
    )

    for idx in range(len(hp["sequences"][0]["protein"]["templates"])):
        del hp["sequences"][0]["protein"]["templates"][idx]["mmcif"]
        hp["sequences"][0]["protein"]["templates"][idx]["mmcifPath"] =
hp_mmcif_file.replace(
            comb_msa_ref, msa_ref_ncsa
        ) + "_" + str(idx) + ".cif"
    for idx in range(len(bp["sequences"][0]["protein"]["templates"])):
        del bp["sequences"][0]["protein"]["templates"][idx]["mmcif"]
        bp["sequences"][0]["protein"]["templates"][idx]["mmcifPath"] =
bp_mmcif_file.replace(
            comb_msa_ref, msa_ref_ncsa
        ) + "_" + str(idx) + ".cif"
    return hp, bp
```

```python
def is_standard_aa(sequence):
    # Define the set of standard amino acids
    standard_amino_acids = set("ARNDCEQGHILKMFPSTWYV")

    # Convert the sequence to uppercase to ensure case insensitivity
    sequence = sequence.upper()
    # Check for non-standard amino acids
    for aa in sequence:
        if aa not in standard_amino_acids:
            return True
    return False
```

Prepare structure inference input

```python
def create_comb_msa(
    hprot_msa,
    bprot_msa,
    outname,
    token_thres=4352,
    random_seed=[1],
    comb_msa_ref=None,
    msa_ref_ncsa=None,
    logfile=None,
):
    """
    Creates a combined MSA (Multiple Sequence Alignment) from two input MSAs and saves the
result to a specified output file.

    Args:
        hprot_msa (str): Path to the JSON file containing the MSA for the first protein.
        bprot_msa (str): Path to the JSON file containing the MSA for the second protein.
        outname (str): Output filename for the combined MSA.
        token_thres (int, optional): Threshold for the total number of tokens (sequence
length). Defaults to 4352.
        random_seed (list, optional): List of random seeds for the model. Defaults to [1].
        comb_msa_ref (str, optional): Reference path for combined MSA files. Defaults to
None.
        msa_ref_ncsa (str, optional): Reference path for NCSA MSA files. Defaults to None.
        logfile (str, optional): Path to a logfile for logging. Defaults to None.

    Returns:
        None
    """
    hp = json.load(open(hprot_msa))
    bp = json.load(open(bprot_msa))
    hp_paired_msa_file = f"{comb_msa_ref}/{hp['name']}_paired.a3m"
    hp_unpaired_msa_file = f"{comb_msa_ref}/{hp['name']}_unpaired.a3m"
    bp_paired_msa_file = f"{comb_msa_ref}/{bp['name']}_paired.a3m"
    bp_unpaired_msa_file = f"{comb_msa_ref}/{bp['name']}_unpaired.a3m"

    prep_msa(
        hp,
        bp,
        hp_paired_msa_file,
        hp_unpaired_msa_file,
        bp_paired_msa_file,
        bp_unpaired_msa_file,
    )

    hp_mmcif_file = f"{comb_msa_ref}/{hp['name']}"
    bp_mmcif_file = f"{comb_msa_ref}/{bp['name']}"
```

```python
    prep_mmccif(
        hp,
        bp,
        hp_mmcif_file,
        bp_mmcif_file,
    )

    hp["sequences"][0]["protein"]["id"] = "A"
    bp["sequences"][0]["protein"]["id"] = "B"

# check the size of sequences. I think we have to try the limits, this time I have
unified memory enabled but not sure how many tokens are allowed
    if (
        len(hp["sequences"][0]["protein"]["sequence"])
        + len(bp["sequences"][0]["protein"]["sequence"])
        > token_thres
    ):
        print(
            f"Skipping {outname} because the total tokens exceed the limit
{token_thres}",
            file=open(logfile, "a"),
        )
    if not is_standard_aa(hp["sequences"][0]["protein"]["sequence"]) or not
is_standard_aa(bp["sequences"][0]["protein"]["sequence"]):
        print(
            f"Skipping {outname} because of non standard aa.",
            file=open(logfile, "a"),
        )
        return

    hp, bp = update_path(hp, bp, hp_paired_msa_file, bp_paired_msa_file,
hp_unpaired_msa_file, bp_unpaired_msa_file, hp_mmcif_file, bp_mmcif_file)

    json.dump(
        {
            "dialect": hp["dialect"],
            "version": 2,
            "name": hp["name"] + "_" + bp["name"],
            "sequences": [hp["sequences"][0], bp["sequences"][0]],
            "modelSeeds": random_seed,
        },
        open(f"{outname}", "w"),
    )
```

Prepare structure inference input

```python
import random
"""
This script processes multiple sequence alignments (MSA) for human and bacterial
proteins associated with diseases. It combines MSAs for pairs of human and bacterial
proteins and saves the results to specified output files.

Functions:
    prep_msa(hp, bp, hp_paired_msa_file, hp_unpaired_msa_file, bp_paired_msa_file,
bp_unpaired_msa_file):
        Prepares and writes paired and unpaired MSAs to specified files for both human
and bacterial proteins.

    create_comb_msa(hprot_msa, bprot_msa, outname, token_thres=4352, random_seed=[1],
comb_msa_ref=None, msa_ref_ncsa=None, logfile=None):
        Creates a combined MSA from two input MSAs and saves the result to a specified
output file.

Variables:
    msa_ref_ncsa (str): Path to the reference directory for NCSA MSA files.
    hprot_msadir (str): Directory path containing MSA files for human proteins.
    bprot_msadir (str): Directory path containing MSA files for bacterial proteins.
    comb_msa_outdir (str): Output directory for combined MSA files.
    comb_msa_ref (str): Reference directory for combined MSA files.
    logfile (str): Path to the logfile for logging.

Processing:
    Iterates over disease-associated human and bacterial proteins, combines their
MSAs, and saves the results to output files. Logs progress and errors to a specified
logfile.
"""
msa_ref_ncsa = "/root/af_input/msa_ref"

# get uniprotkb map to msa.json file path
hprot_msadir = "path_to_your_hp/msa_outputs/"
bprot_msadir = "path_to_your_bp_msa_outputs/"

comb_msa_outdir = "path_to_where_you_want_to_save_the_combined_msa/"
comb_msa_ref = f"{comb_msa_outdir}/msa_ref"

logfile = (
    f"{comb_msa_outdir}/comb_msa.log"
)
```

```python
# load the pair file
with open("path_to_pair_file", "r") as file:
    for line in file:
        line = line.strip().split(' ')
        hprot, bprot = line[0], line[1]
        outname = f"{comb_msa_outdir}/{hprot}_{bprot}.json"
        hprot_msa =
f"{hprot_msadir}/{hprot.lower()}/{hprot.lower()}_data.json"
        bprot_msa =
f"{bprot_msadir}/{bprot.lower()}/{bprot.lower()}_data.json"

        # print error message if the file does not exist, but
continue running
        if not os.path.exists(hprot_msa) or not
os.path.exists(bprot_msa):
            print(f"Missing {hprot_msa} or {bprot_msa}",
file=open(logfile, "a"))
            continue
        k = 2 # number of predictions
        random_seeds = [random.randint(1, 1000) for _ in
range(k)]
        create_comb_msa(
            hprot_msa,
            bprot_msa,
            outname,
            token_thres=4352,
            random_seed=random_seeds,
            comb_msa_ref=comb_msa_ref,
            msa_ref_ncsa=msa_ref_ncsa,
            logfile=logfile,
        )
```

Explain structure_inference.sh

```
MDIR="/local/workdir/alphafold3"
DATABASE="${MDIR}/public_databases"
WEIGHTS="${MDIR}/model"

OUTDIR=$(date +'%y%m%d%H'_$(hostname | awk -F '.' '{print
$1}'))
OUTDIR="/local/workdir/output_${OUTDIR}"
mkdir -p $OUTDIR

INPUT_PREF="path_to_your_json_dir"
NPROC=$njob

cd /workdir/alphafold3
for i in $(seq 1 $NPROC); do
    INPUT="${INPUT_PREF}${i}"

    docker1 run \
        --volume $INPUT:/root/af_input \
        --volume $OUTDIR:/root/af_output \
        --volume $WEIGHTS:/root/models \
        --volume $DATABASE:/root/public_databases \
        --gpus all \
        biohpc_netid/alphafold3 \
        python run_alphafold.py \
         --input_dir=/root/af_input \
         --model_dir=/root/models \
         --db_dir=/root/public_databases \
         --output_dir=/root/af_output \
         --norun_data_pipeline &
done
wait
```

ACCESS

ACCESS

Estimate 10 seconds per pair for structural prediction.

Adjustment for A100 40G

Adjusting pair_transition_shard_spec in model_config.py:

```
pair_transition_shard_spec: Sequence[_Shape2DType] = (
    (2048, None),
    (3072, 1024),
    (None, 512),
)
```