



Figure: Comparison of OpenFold and AlphaFold2 predictions to the experimental structure of PDB 7KDX, chain B.

OpenFold

A faithful but trainable PyTorch reproduction of DeepMind's AlphaFold 2.

Contents

- OpenFold
 - Contents
 - Features
 - Installation (Linux)
 - Download Alignment Databases
 - Inference
 - ★ Monomer inference
 - ★ Multimer Inference
 - ★ Soloseq Inference
 - Training
 - Testing
 - Building and Using the Docker Container
 - Copyright Notice
 - Contributing

-
- Citing this Work

Features

OpenFold carefully reproduces (almost) all of the features of the original open source monomer (v2.0.1) and multimer (v2.3.2) inference code. The sole exception is model ensembling, which fared poorly in DeepMind’s own ablation testing and is being phased out in future DeepMind experiments. It is omitted here for the sake of reducing clutter. In cases where the *Nature* paper differs from the source, we always defer to the latter.

OpenFold is trainable in full precision, half precision, or `bfloat16` with or without DeepSpeed, and we’ve trained it from scratch, matching the performance of the original. In addition, we have trained new models for single sequence inference. We’ve publicly released model weights and our training data — some 400,000 MSAs and PDB70 template hit files — under a permissive license. Model weights, MSAs, and embeddings (for the single sequence model) are hosted by the Registry of Open Data on AWS (RODA) and are available for download via scripts in this repository. Try out running inference for yourself with our Colab notebook.

OpenFold also supports inference using AlphaFold’s official parameters, and vice versa (see [scripts/convert_of_weights_to_jax.py](#)).

OpenFold has the following advantages over the reference implementation:

- **Faster inference** on GPU, sometimes by as much as 2x. The greatest speedups are achieved on Ampere or higher architecture GPUs.
- **Inference on extremely long chains**, made possible by our implementation of low-memory attention (Rabe & Staats 2021). OpenFold can predict the structures of sequences with more than 4000 residues on a single A100, and even longer ones with CPU offloading.
- **Custom CUDA attention kernels** modified from FastFold’s kernels support in-place attention during inference and training. They use 4x and 5x less GPU memory than equivalent FastFold and stock PyTorch implementations, respectively.
- **Efficient alignment scripts** using the original AlphaFold HHblits/JackHMMER pipeline or ColabFold’s, which uses the faster MMseqs2 instead. We’ve used them to generate millions of alignments.
- **FlashAttention** support greatly speeds up MSA attention.
- **DeepSpeed DS4Sci_EvoformerAttention kernel** is a memory-efficient attention kernel developed as part of a collaboration between OpenFold and the DeepSpeed4Science initiative. The kernel provides substantial speedups for training and inference, and significantly reduces the model’s peak device memory requirement by 13X. The model is 15% faster during the initial

training and finetuning stages, and up to 4x faster during inference. To use this feature, simply set the `use_deepspeed_evo_attention` option in `openfold/config.py`.

Installation (Linux)

All Python dependencies are specified in `environment.yml`. For producing sequence alignments, you'll also need `kalign`, the HH-suite, and one of {`jackhmmer`, `MMseqs2` (nightly build)} installed on your system. You'll need `git-lfs` to download OpenFold parameters. Finally, some download scripts require `aria2c` and `aws`.

This package is currently supported for CUDA 11 and Pytorch 1.12

To install: 1. Clone the repository, e.g. `git clone https://github.com/aqlaboratory/openfold.git` 1. From the `openfold` repo: - Create a Mamba environment, e.g. `mamba env create -n openfold_env -f environment.yml` Mamba is recommended as the dependencies required by OpenFold are quite large and mamba can speed up the process. - Activate the environment, e.g. `conda activate openfold_env` 1. Run `scripts/install_third_party_dependencies.sh` to configure kernels and folding resources.

For some systems, it may help to append the Conda environment library path to `$LD_LIBRARY_PATH`. The `install_third_party_dependencies.sh` script does this once, but you may need this for each bash instance.

Download Alignment Databases

If you intend to generate your own alignments, e.g. for inference, you have two choices for downloading protein databases, depending on whether you want to use DeepMind's MSA generation pipeline (w/ HMMER & HHblits) or ColabFold's, which uses the faster MMseqs2 instead. For the former, run:

```
1 bash scripts/download_alphafold_dbs.sh data/
```

For the latter, run:

```
1 bash scripts/download_mmseqs_dbs.sh data/      # downloads .tar files
2 bash scripts/prep_mmseqs_dbs.sh data/          # unpacks and preps the
   databases
```

Make sure to run the latter command on the machine that will be used for MSA generation (the script estimates how the precomputed database index used by MMseqs2 should be split according to the memory available on the system).

If you're using your own precomputed MSAs or MSAs from the RODA repository, there's no need to download these alignment databases. Simply make sure that the `alignment_dir` contains one

directory per chain and that each of these contains alignments (.sto, .a3m, and .hhr) corresponding to that chain. You can use [scripts/flatten_roda.sh](#) to reformat RODA downloads in this way. Note that the RODA alignments are NOT compatible with the recent .cif ground truth files downloaded by [scripts/download_alphafold_dbs.sh](#). To fetch .cif files that match the RODA MSAs, once the alignments are flattened, use [scripts/download_roda_pdb.sh](#). That script outputs a list of alignment dirs for which matching .cif files could not be found. These should be removed from the alignment directory.

Alternatively, you can use raw MSAs from ProteinNet. After downloading that database, use [scripts/prep_proteinnet_msas.py](#) to convert the data into a format recognized by the OpenFold parser. The resulting directory becomes the [alignment_dir](#) used in subsequent steps. Use [scripts/unpack_proteinnet.py](#) to extract .core files from ProteinNet text files.

For both inference and training, the model's hyperparameters can be tuned from [openfold/config.py](#). Of course, if you plan to perform inference using DeepMind's pretrained parameters, you will only be able to make changes that do not affect the shapes of model parameters. For an example of initializing the model, consult [run_pretrained_openfold.py](#).

Download Embeddings for SoloSeq

ESM embeddings for the PDB set and the distillation set are available for download at RODA. These embeddings were used to train the single sequence SoloSeq model, and can be used to retrain the model, if desired. The dataset is composed of two separate directories of ESM-1b embeddings- one containing embeddings for the 120,450 unique PDB chains and the other for the 268,699 chains of the distillation set generated from UniClust30 clusters. The chains and sequences are the same as OpenProteinSet except for being capped at 1022 residues because of the limitations of the ESM-1b model. The chains filtered out were a small fraction of the total unique sequences derived from the OpenProteinSet PDB chains (<10%).

For more information, and for instructions on generating embeddings for more chains, see the README file on RODA.

Inference

OpenFold now supports three inference modes: - Monomer Inference: OpenFold reproduction of AlphaFold2. Inference available with either DeepMind's pretrained parameters or OpenFold trained parameters. - Multimer Inference: OpenFold reproduction of AlphaFold-Multimer. Inference available with DeepMind's pre-trained parameters. - Single Sequence Inference (SoloSeq): Language Model based structure prediction, using ESM-1b embeddings.

More instructions for each inference mode are provided below:

Monomer inference

To run inference on a sequence or multiple sequences using a set of DeepMind's pretrained parameters, first download the OpenFold weights e.g.:

```
1 bash scripts/download_openfold_params.sh openfold/resources
```

then run e.g.:

```
1 python3 run_pretrained_openfold.py \
2     fasta_dir \
3     data/pdb_mmcif/mmcif_files/ \
4     --uniref90_database_path data/uniref90/uniref90.fasta \
5     --mgnify_database_path data/mgnify/mgy_clusters_2018_12.fa \
6     --pdb70_database_path data/pdb70/pdb70 \
7     --uniclust30_database_path data/uniclust30/uniclust30_2018_08/
8     uniclust30_2018_08 \
9     --bfd_database_path data/bfd/
10    bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \
11    --jackhmmer_binary_path lib/conda/envs/openfold_venv/bin/jackhmmer
12    \
13    --hhblits_binary_path lib/conda/envs/openfold_venv/bin/hhblits \
14    --hhsearch_binary_path lib/conda/envs/openfold_venv/bin/hhsearch \
15    --kalign_binary_path lib/conda/envs/openfold_venv/bin/kalign \
16    --config_preset "model_1_ptm" \
17    --model_device "cuda:0" \
18    --output_dir ./ \
19    --openfold_checkpoint_path openfold/resources/openfold_params/
20    finetuning_ptm_2.pt
```

where `data` is the same directory as in the previous step. If `jackhmmer`, `hhblits`, `hhsearch` and `kalign` are available at the default path of `/usr/bin`, their `binary_path` command-line arguments can be dropped. If you've already computed alignments for the query, you have the option to skip the expensive alignment computation here with `--use_precomputed_alignments`.

`--openfold_checkpoint_path` or `--jax_param_path` accept comma-delineated lists of .pt/DeepSpeed OpenFold checkpoints and AlphaFold's .npz JAX parameter files, respectively. For a breakdown of the differences between the different parameter files, see the README downloaded to `openfold/resources/openfold_params/`. Since OpenFold was trained under a newer training schedule than the one from which the `model_n` config presets are derived, there is no clean correspondence between `config_preset` settings and OpenFold checkpoints; the only restraints are that `*_ptm` checkpoints must be run with `*_ptm` config presets and that `_no_template` checkpoints are only compatible with template-less presets (`model_3` and above).

Note that chunking (as defined in section 1.11.8 of the AlphaFold 2 supplement) is enabled by default in inference mode. To disable it, set `globals.chunk_size` to `None` in the config. If a value is specified, OpenFold will attempt to dynamically tune it, considering the chunk size specified in the config as a minimum. This tuning process automatically ensures consistently fast runtimes regardless of input sequence length, but it also introduces some runtime variability, which may be undesirable for certain users. It is also recommended to disable this feature for very long chains (see below). To do so, set the `tune_chunk_size` option in the config to `False`.

For large-scale batch inference, we offer an optional tracing mode, which massively improves runtimes at the cost of a lengthy model compilation process. To enable it, add `--trace_model` to the inference command.

To get a speedup during inference, enable FlashAttention in the config. Note that it appears to work best for sequences with < 1000 residues.

To minimize memory usage during inference on long sequences, consider the following changes:

- As noted in the AlphaFold-Multimer paper, the AlphaFold/OpenFold template stack is a major memory bottleneck for inference on long sequences. OpenFold supports two mutually exclusive inference modes to address this issue. One, `average_templates` in the `template` section of the config, is similar to the solution offered by AlphaFold-Multimer, which is simply to average individual template representations. Our version is modified slightly to accommodate weights trained using the standard template algorithm. Using said weights, we notice no significant difference in performance between our averaged template embeddings and the standard ones. The second, `offload_templates`, temporarily offloads individual template embeddings into CPU memory. The former is an approximation while the latter is slightly slower; both are memory-efficient and allow the model to utilize arbitrarily many templates across sequence lengths. Both are disabled by default, and it is up to the user to determine which best suits their needs, if either.
- Inference-time low-memory attention (LMA) can be enabled in the model config. This setting trades off speed for vastly improved memory usage. By default, LMA is run with query and key chunk sizes of 1024 and 4096, respectively. These represent a favorable tradeoff in most memory-constrained cases. Powerusers can choose to tweak these settings in `openfold/model/primitives.py`. For more information on the LMA algorithm, see the aforementioned Staats & Rabe preprint.
- Disable `tune_chunk_size` for long sequences. Past a certain point, it only wastes time.
- As a last resort, consider enabling `offload_inference`. This enables more extensive CPU offloading at various bottlenecks throughout the model.
- Disable FlashAttention, which seems unstable on long sequences.

Using the most conservative settings, we were able to run inference on a 4600-residue complex

with a single A100. Compared to AlphaFold's own memory offloading mode, ours is considerably faster; the same complex takes the more efficient AlphaFold-Multimer more than double the time. Use the `long_sequence_inference` config option to enable all of these interventions at once. The `run_pretrained_openfold.py` script can enable this config option with the `--long_sequence_inference` command line option

Input FASTA files containing multiple sequences are treated as complexes. In this case, the inference script runs AlphaFold-Gap, a hack proposed here, using the specified stock AlphaFold/OpenFold parameters (NOT AlphaFold-Multimer).

Multimer Inference

To run inference on a complex or multiple complexes using a set of DeepMind's pretrained parameters, run e.g.:

```
1 python3 run_pretrained_openfold.py \
2     fasta_dir \
3     data/pdb_mmcif/mmcif_files/ \
4     --uniref90_database_path data/uniref90/uniref90.fasta \
5     --mgnify_database_path data/mgnify/mgy_clusters_2022_05.fa \
6     --pdb_seqres_database_path data/pdb_seqres/pdb_seqres.txt \
7     --uniref30_database_path data/uniref30/UniRef30_2021_03 \
8     --uniprot_database_path data/uniprot/uniprot.fasta \
9     --bfd_database_path data/bfd/
10     bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \
11     --jackhmmer_binary_path lib/conda/envs/openfold_venv/bin/jackhmmer \
12     --hhblits_binary_path lib/conda/envs/openfold_venv/bin/hhblits \
13     --hmmsearch_binary_path lib/conda/envs/openfold_venv/bin/hmmsearch \
14     --hmmbuild_binary_path lib/conda/envs/openfold_venv/bin/hmmbuild \
15     --kalign_binary_path lib/conda/envs/openfold_venv/bin/kalign \
16     --config_preset "model_1_multimer_v3" \
17     --model_device "cuda:0" \
18     --output_dir ./
```

As with monomer inference, if you've already computed alignments for the query, you can use the `--use_precomputed_alignments` option. Note that template searching in the multimer pipeline uses HMMSearch with the PDB SeqRes database, replacing HHSearch and PDB70 used in the monomer pipeline.

Upgrade from an existing OpenFold installation

The above command requires several upgrades to existing openfold installations.

1. Re-download the alphafold parameters to get the latest AlphaFold-Multimer v3 weights:

```
1 bash scripts/download_alphafold_params.sh openfold/resources
```

2. Download the UniProt and PDB SeqRes databases:

```
1 bash scripts/download_uniprot.sh data/
```

The PDB SeqRes and PDB databases must be from the same date to avoid potential errors during template searching. Remove the existing `data/pdb_mmcif` directory and download both databases:

```
1 bash scripts/download_pdb_mmcif.sh data/  
2 bash scripts/download_pdb_seqres.sh data/
```

3. Additionally, AlphaFold-Multimer uses upgraded versions of the MGnify and UniRef30 (previously UniClust30) databases. To download the upgraded databases, run:

```
1 bash scripts/download_uniref30.sh data/  
2 bash scripts/download_mgnify.sh data/
```

Multimer inference can also run with the older database versions if desired.

Soloseq Inference

To run inference for a sequence using the SoloSeq single-sequence model, you can either precompute ESM-1b embeddings in bulk, or you can generate them during inference.

For generating ESM-1b embeddings in bulk, use the provided script: `scripts/precompute_embeddings.py`. The script takes a directory of FASTA files (one sequence per file) and generates ESM-1b embeddings in the same format and directory structure as required by SoloSeq. Following is an example command to use the script:

```
1 python scripts/precompute_embeddings.py fasta_dir/  
   embeddings_output_dir/
```

In the same per-label subdirectories inside `embeddings_output_dir`, you can also place `*.hhr` files (outputs from HHSearch), which can contain the details about the structures that you want to use as templates. If you do not place any such file, templates will not be used and only the ESM-1b embeddings will be used to predict the structure. If you want to use templates, you need to pass the PDB MMCIF dataset to the command.

Then download the SoloSeq model weights, e.g.:

```
1 bash scripts/download_openfold_soloseq_params.sh openfold/resources
```

Now, you are ready to run inference:

```

1 python run_pretrained_openfold.py \
2     fasta_dir \
3     data/pdb_mmcif/mmcif_files/ \
4     --use_precomputed_alignments embeddings_output_dir \
5     --output_dir ./ \
6     --model_device "cuda:0" \
7     --config_preset "seq_model_esm1b_ptm" \
8     --openfold_checkpoint_path openfold/resources/
    openfold_soloseq_params/seq_model_esm1b_ptm.pt

```

For generating the embeddings during inference, skip the `--use_precomputed_alignments` argument. The `*.hhr` files will be generated as well if you pass the paths to the relevant databases and tools, as specified in the command below. If you skip the database and tool arguments, HHSearch will not be used to find templates and only generated ESM-1b embeddings will be used to predict the structure.

```

1 python3 run_pretrained_openfold.py \
2     fasta_dir \
3     data/pdb_mmcif/mmcif_files/ \
4     --output_dir ./ \
5     --model_device "cuda:0" \
6     --config_preset "seq_model_esm1b_ptm" \
7     --openfold_checkpoint_path openfold/resources/
    openfold_soloseq_params/seq_model_esm1b_ptm.pt \
8     --uniref90_database_path data/uniref90/uniref90.fasta \
9     --pdb70_database_path data/pdb70/pdb70 \
10    --jackhmmer_binary_path lib/conda/envs/openfold_venv/bin/jackhmmer
    \
11    --hhsearch_binary_path lib/conda/envs/openfold_venv/bin/hhsearch \
12    --kalign_binary_path lib/conda/envs/openfold_venv/bin/kalign \

```

For generating template information, you will need the UniRef90 and PDB70 databases and the JackHmmer and HHSearch binaries.

SoloSeq allows you to use the same flags and optimizations as the MSA-based OpenFold. For example, you can skip relaxation using `--skip_relaxation`, save all model outputs using `--save_outputs`, and generate output files in MMCIF format using `--cif_output`.

NOTE: Due to the nature of the ESM-1b embeddings, the sequence length for inference using the SoloSeq model is limited to 1022 residues. Sequences longer than that will be truncated.

Training

To train the model, you will first need to precompute protein alignments.

You have two options. You can use the same procedure DeepMind used by running the following:

```

1 python3 scripts/precompute_alignments.py mmcif_dir/ alignment_dir/ \
2   --uniref90_database_path data/uniref90/uniref90.fasta \
3   --mgnify_database_path data/mgnify/mgy_clusters_2018_12.fa \
4   --pdb70_database_path data/pdb70/pdb70 \
5   --uniclust30_database_path data/uniclust30/uniclust30_2018_08/
6   uniclust30_2018_08 \
7   --bfd_database_path data/bfd/
8   bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \
9   --cpus_per_task 16 \
10  --jackhmmer_binary_path lib/conda/envs/openfold_venv/bin/jackhmmer
11  \
12  --hhblits_binary_path lib/conda/envs/openfold_venv/bin/hhblits \
13  --hhsearch_binary_path lib/conda/envs/openfold_venv/bin/hhsearch \
14  --kalign_binary_path lib/conda/envs/openfold_venv/bin/kalign

```

As noted before, you can skip the `binary_path` arguments if these binaries are at `/usr/bin`. Expect this step to take a very long time, even for small numbers of proteins.

Alternatively, you can generate MSAs with the ColabFold pipeline (and templates with HHsearch) with:

```

1 python3 scripts/precompute_alignments_mmseqs.py input.fasta \
2   data/mmseqs_dbs \
3   uniref30_2103_db \
4   alignment_dir \
5   ~/MMseqs2/build/bin/mmseqs \
6   /usr/bin/hhsearch \
7   --env_db colabfold_envdb_202108_db
8   --pdb70 data/pdb70/pdb70

```

where `input.fasta` is a FASTA file containing one or more query sequences. To generate an input FASTA from a directory of mmCIF and/or ProteinNet .core files, we provide `scripts/data_dir_to_fasta.py`.

Next, generate a cache of certain datapoints in the template mmCIF files:

```

1 python3 scripts/generate_mmcif_cache.py \
2   mmcif_dir/ \
3   mmcif_cache.json \
4   --no_workers 16

```

This cache is used to pre-filter templates.

Next, generate a separate chain-level cache with data used for training-time data filtering:

```

1 python3 scripts/generate_chain_data_cache.py \
2   mmcif_dir/ \
3   chain_data_cache.json \
4   --cluster_file clusters-by-entity-40.txt \

```

where the `cluster_file` argument is a file of chain clusters, one cluster per line.

Optionally, download an AlphaFold-style validation set from CAMEO using `scripts/download_cameo.py`. Use the resulting FASTA files to generate validation alignments and then specify the validation set's location using the `--val_...` family of training script flags.

Finally, call the training script:

```
1 python3 train_openfold.py mmcif_dir/ alignment_dir/ template_mmcif_dir/
  output_dir/ \
2 2021-10-10 \
3 --template_release_dates_cache_path mmcif_cache.json \
4 --precision bf16 \
5 --gpus 8 --replace_sampler_ddp=True \
6 --seed 4242022 \ # in multi-gpu settings, the seed must be
  specified
7 --deepspeed_config_path deepspeed_config.json \
8 --checkpoint_every_epoch \
9 --resume_from_ckpt ckpt_dir/ \
10 --train_chain_data_cache_path chain_data_cache.json \
11 --obsolete_pdb_file_path obsolete.dat
```

where `--template_release_dates_cache_path` is a path to the mmCIF cache. Note that `template_mmcif_dir` can be the same as `mmcif_dir` which contains training targets. A suitable DeepSpeed configuration file can be generated with `scripts/build_deepspeed_config.py`. The training script is written with PyTorch Lightning and supports the full range of training options that entails, including multi-node distributed training, validation, and so on. For more information, consult PyTorch Lightning documentation and the `--help` flag of the training script.

Note that, despite its variable name, `mmcif_dir` can also contain PDB files or even ProteinNet .core files.

To emulate the AlphaFold training procedure, which uses a self-distillation set subject to special pre-processing steps, use the family of `--distillation` flags.

In cases where it may be burdensome to create separate files for each chain's alignments, alignment directories can be consolidated using the scripts in `scripts/alignment_db_scripts/`. First, run `create_alignment_db.py` to consolidate an alignment directory into a pair of database and index files. Once all alignment directories (or shards of a single alignment directory) have been compiled, unify the indices with `unify_alignment_db_indices.py`. The resulting index, `super.index`, can be passed to the training script flags containing the phrase `alignment_index`. In this scenario, the `alignment_dir` flags instead represent the directory containing the compiled alignment databases. Both the training and distillation datasets can be compiled in this way. Anecdotally, this can speed up training in I/O-bottlenecked environments.

Testing

To run unit tests, use

```
1 scripts/run_unit_tests.sh
```

The script is a thin wrapper around Python's `unittest` suite, and recognizes `unittest` arguments. E.g., to run a specific test verbosely:

```
1 scripts/run_unit_tests.sh -v tests.test_model
```

Certain tests require that AlphaFold (v2.0.1) be installed in the same Python environment. These run components of AlphaFold and OpenFold side by side and ensure that output activations are adequately similar. For most modules, we target a maximum pointwise difference of $1e-4$.

Building and Using the Docker Container

Building the Docker Image

Openfold can be built as a docker container using the included dockerfile. To build it, run the following command from the root of this repository:

```
1 docker build -t openfold .
```

Running the Docker Container

The built container contains both `run_pretrained_openfold.py` and `train_openfold.py` as well as all necessary software dependencies. It does not contain the model parameters, sequence, or structural databases. These should be downloaded to the host machine following the instructions in the Usage section above.

The docker container installs all conda components to the base conda environment in `/opt/conda`, and installs openfold itself in `/opt/openfold`,

Before running the docker container, you can verify that your docker installation is able to properly communicate with your GPU by running the following command:

```
1 docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

Note the `--gpus all` option passed to `docker run`. This option is necessary in order for the container to use the GPUs on the host machine.

To run the inference code under docker, you can use a command like the one below. In this example, parameters and sequences from the alphafold dataset are being used and are located at `/mnt/alphafold_database` on the host machine, and the input files are located in the current work-

ing directory. You can adjust the volume mount locations as needed to reflect the locations of your data.

```
1 docker run \  
2 --gpus all \  
3 -v $PWD:/data \  
4 -v /mnt/alphafold_database:/database \  
5 -ti openfold:latest \  
6 python3 /opt/openfold/run_pretrained_openfold.py \  
7 /data.fasta_dir \  
8 /database/pdb_mmcif/mmcif_files/ \  
9 --uniref90_database_path /database/uniref90/uniref90.fasta \  
10 --mgnify_database_path /database/mgnify/mgy_clusters_2018_12.fa \  
11 --pdb70_database_path /database/pdb70/pdb70 \  
12 --uniclust30_database_path /database/uniclust30/uniclust30_2018_08/  
    uniclust30_2018_08 \  
13 --output_dir /data \  
14 --bfd_database_path /database/bfd/  
    bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \  
15 --model_device cuda:0 \  
16 --jackhmmer_binary_path /opt/conda/bin/jackhmmer \  
17 --hhblits_binary_path /opt/conda/bin/hhblits \  
18 --hhsearch_binary_path /opt/conda/bin/hhsearch \  
19 --kalign_binary_path /opt/conda/bin/kalign \  
20 --openfold_checkpoint_path /database/openfold_params/finetuning_ptm_2.  
    pt
```

Copyright Notice

While AlphaFold's and, by extension, OpenFold's source code is licensed under the permissive Apache Licence, Version 2.0, DeepMind's pretrained parameters fall under the CC BY 4.0 license, a copy of which is downloaded to [openfold/resources/params](#) by the installation script. Note that the latter replaces the original, more restrictive CC BY-NC 4.0 license as of January 2022.

Contributing

If you encounter problems using OpenFold, feel free to create an issue! We also welcome pull requests from the community.

Citing this Work

Please cite our paper:

```

1 @article {Ahdritz2022.11.20.517210,
2   author = {Ahdritz, Gustaf and Bouatta, Nazim and Floristean,
              Christina and Kadyan, Sachin and Xia, Qinghui and Gerecke,
              William and O{\textquoteright}Donnell, Timothy J and Berenberg,
              Daniel and Fisk, Ian and Zanichelli, Niccolò and Zhang, Bo and
              Nowaczynski, Arkadiusz and Wang, Bei and Stepniewska-Dziubinska,
              Marta M and Zhang, Shang and Ojewole, Adegoke and Guney, Murat
              Efe and Biderman, Stella and Watkins, Andrew M and Ra, Stephen
              and Lorenzo, Pablo Ribalta and Nivon, Lucas and Weitzner, Brian
              and Ban, Yih-En Andrew and Sorger, Peter K and Mostaque, Emad
              and Zhang, Zhao and Bonneau, Richard and AlQuraishi, Mohammed},
3   title = {{0}pen{F}old: {R}etraining {A}lpha{F}old2 yields new
              insights into its learning mechanisms and capacity for
              generalization},
4   elocation-id = {2022.11.20.517210},
5   year = {2022},
6   doi = {10.1101/2022.11.20.517210},
7   publisher = {Cold Spring Harbor Laboratory},
8   URL = {https://www.biorxiv.org/content/10.1101/2022.11.20.517210},
9   eprint = {https://www.biorxiv.org/content/early
              /2022/11/22/2022.11.20.517210.full.pdf},
10  journal = {bioRxiv}
11 }

```

If you use OpenProteinSet, please also cite:

```

1 @misc{ahdritz2023openproteinset,
2   title={{0}pen{P}rotein{S}et: {T}raining data for structural
              biology at scale},
3   author={Gustaf Ahdritz and Nazim Bouatta and Sachin Kadyan and
              Lukas Jarosch and Daniel Berenberg and Ian Fisk and Andrew M.
              Watkins and Stephen Ra and Richard Bonneau and Mohammed
              AlQuraishi},
4   year={2023},
5   eprint={2308.05326},
6   archivePrefix={arXiv},
7   primaryClass={q-bio.BM}
8 }

```

Any work that cites OpenFold should also cite AlphaFold and AlphaFold-Multimer if applicable.