RFDiffusion

Bold text means that these files and/or this information is provided.

Italicized text means that this material will NOT be conducted during the workshop

```
fixed width text means you should type the command into your terminal
```

If you want to try making files that already exist (e.g., input files), write them to a different directory (`mkdir my_dir`)

Tutorial

RFDiffusion is a open-source structure generation method. This method can generate completely new (*de-novo*) protein backbones, with or without a given input structure, target or motif. This allows potential application in a broad variety of research fields, including *de-novo* protein design, enzyme design, motif scaffolding and binder design. To start, this tutorial should give a quick overview of a few applications which can be done using RFDiffusion, such as *de-novo* protein design and motif scaffolding.

In the past, this tutorial was taught in a classroom setting with RFDiffusion already installed. The intent of this tutorial is to provide a quick overview and some easy examples, to get familiar with the RFDiffusion method. Additionally, this tutorial refers to using PyMOL, a protein structure visualization tool, to manipulate files. Students and teachers can register for a limited-use license. Other users can purchase a license here or use UCSF Chimera, although instructions for using Chimera will not be included in the tutorials.

Information: All RFDiffussion generated structure are just poly-Glycine sequences. RFDiffusion is a backbone-generation model and can not generate sequences onto these backbones. To generate sequences for the created backbones other methods such as ProteinMPNN need to be used, which are then often followed by a folding step (AlphaFold2, OmegaFold, ESMFold,...) to validate if the generate sequence is capable of folding back into the generated structure. Due to complexity and time-limit, this tutorial exclusively focuses on RFDiffusion.

RFDiffusion is a Python program which needs a number of dependencies installed. Normally, these dependencies are installed in a Conda environment, which needs to be activated prior to running the RFDiffusion scripts.

```
conda activate SE3nv
```

In this tutorial we are going to use the co-crystal structure of the hepatitis C glycoprotein E2, the broadly neutralizing AR3A antibody and the non-neutralizing antibody E1.

1. Change your current directory to RFDiffusion and then create a directory called my_files and switch to that directory. Although many files you need for the tutorial are located in the `input_files` directory, we will work from `my_files` for the rest of the tutorial.

   ```
   cd ~/rosetta_workshop/tutorials/rfdiffusion/
   mkdir my_files
   cd my_files
   ```

2. In the first step, we will start simple, by just creating a unconditional *de-novo* protein completely from scratch. The protein we want to create should have a length of 150aa. For this, we need to specify three things:

   1. The length of the protein (150aa): The protein length can be specified using the flag `contigmap.contigs`. To understand what exactly this command does, we need to take a quick look how RFDiffusion is setup. RFDiffusion uses Hydra, which is a open-source Python framework which simplifies the guidance of complex machine learning software. In this case, the Hydra configs tell the inference script (Main RFDiffusion script) how it should run. This config has multiple sub-configs, one of the being `contigmap`, which

controls everything related to contig string (protein which should be build). Now how to exactly specifiy the protein length. If we want to diffuse a 150aa protein, we use `'contigmap.contigs=[150-150]'`. For hydra reasons the entire argument must be enclosed in `''`. The contig string allows to specify a length range of the protein which should be created. In this case we want a protein which is exactly 150aa long, so we just specific the range to only be 150.

   2. The location, where the output should be written too: The output location including a prefix for all generated models can be specified using `inference.output_prefix=path/to/output_folder/Prefix`.
   3. The number of different backbones we want to create: This can be done using `inference.num_designs=5`, which in this case would create 5 different output models

```
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    'contigmap.contigs=[150-150]' \
    inference.output_prefix=./output_monomer/monomer \
    inference.num_designs=5
```

The first time RFDiffusion is run, it will take some time while `Calculating IGSO3` (the diffusion model). This will be cached for future reference, so it only needs to be done once. While running, we can see that RFDiffusion, predicts backbones for multiple timesteps (default are 50 timesteps). *The console output is always saved in the* `outputs` *folder, under the date they were produced and then in the folder, with the time they were started.*

3. After completion of the script, we can go to the output folder to take a closer look at the different generated output files.

```
cd ./output_monomer
```

The output folder now should contain:

```
1. PDB files, containing the generated backbones
2. *`.trb` files, which contain all metadata associated with the specific run, including all input op
3. A folder called `traj` which contains all trajectory files. These files are multi-step pdbs, which
```

4. Let's first take a look at the generated pdb files using PyMOL. For time reasons we will just pick one generated pdb file (the first one).

```
pymol monomer_0.pdb
```

In the now opened PyMOL window we can see one of the generated protein backbones, with a length of 150aa. If we now look at the sequence we can see, as already described under information, that RFDiffusion only generates poly-glycine sequences. Enter the following command in the PyMOL command prompt.

```
set seq_view, 1
```

5. Now lets take a look at the trajectory files. For this we first go to the trajectory folder.

```
cd ./traj
```

In the trajectory folder, we can see that two files were generated for each model. The `pX0` predictions, contain the predicted model at the end of each timestep and the `Xt-1` files contain the model, that was given the diffusion model as input for each timestep. Let's first take at look at one `pX0` file. We again open the file using PyMOL and then can play the trajectory.

```
pymol monomer_0_pX0_traj.pdb
```

Then in PyMOL:

```
mplay
```

This will now loop over all 50 timesteps. To take a closer look, when can first stop the video and then take a look at a few different states. First we take a look a the last state (50), where we can see the generated backbone after the first timestep. At this time, the protein is mostly a bundle. If we check the states in steps of 10, we can see how it slowly diffuses into a defined protein backbone. State 0 then is the final backbone, which we already save in the pdb file. In PyMOL:

```
mstop
set state, 50
set state, 40
set state, 30
set state, 20
set state, 10
set state, 1
```

Now we will take a look at the `Xt-1` files. We again open them in pymol, first play all the trajectories and then stop, the loop, to take a look at some states. First we again have a look at the last state (first timestep). The input for the first timestep is just a big ball, due to the random noise used for the diffusion start. State 25 (Timestep 25), we still have a just a bundle but not as compressed as at the last state. In state 10 (timestep 40), we can see the bundle to space further apart and kinda starts to look like it could become a protein. In the first state (timestep 50) we can clearly identify a protein backbone, with some secondary structure. But as this is only the input for the last prediction step, it is of course not as defined as the final backbone.

```
pymol monomer_0_Xt-1_traj.pdb
```

Then in PyMOL:

```
mplay
mstop
set state, 50
set state, 25
set state, 10
set state, 1
```

6. Now lets do something more difficult using RFDiffusion, motif scaffolding. For that we are going to use the PDB ID 6UYG. **The 6UYG.pdb file is provided in the input_files directory**

    1. Go to rcsb.org and type '6UYG' in the search bar.
    2. Click on 'Download Files' on the right side of the page, then 'PDB Format'.
    3. Save the PDB file in the `my_files` directory as '6UYG.pdb'.

7. To scaffold protein motifs, we need to specify which regions of the protein we want to conserve and which regions should be generated as a new scaffold protein, and how these regions are connected. Additionally, we want to sample different lengths of the connecting protein between the motifs, as we don't know, what will be the best linker lengths. For this, we again use the `'contigmap.contigs'` argument.

    - Anything prefixed by a letter indicates, that this region is a motif, with the letter corresponding to the chain ID and the number corresponding to the residue numbers in the input pdb file. E.g. A2-10 corresponds to residues 2,3,4…10 from chain A. Every residue we don't specify, won't be given RFDiffusion as input, which means that we don't need to clean or truncate the protein structures.
    - Anything that is not prefixed by a letter indicates residues which will be built. If given a range, RFDiffusion will randomly sample a length of the given range in each iteration of RFDiffusion inference.

Let's say we want to use the motif corresponding to residues 422-440 of chain E from '6UYG.pdb' and create a scaffold for this motif, this can be done with `'contigmap.contigs=[10-20/E422-440/40-50]'`. This tells RFDiffusion to build 10-20 residues N-terminally of the motif (422-440 from chain A from the input pdb), followed by 40-50 residues built C-terminally. The input pdb file can be provided via it's path using `inference.input_pdb=path/to/file.pdb` The complete command to use for the scaffolding then is (make sure that you are again in the `my_files` directory when starting the command):

```
cd ../../
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    'contigmap.contigs=[20-30/E422-440/40-50]' \
    inference.input_pdb=6UYG.pdb \
    inference.output_prefix=./output_scaffold/scaffold \
    inference.num_designs=1
```

Due to time reasons, we will from now on only create one model.

While the script runs, we should now see something like:

```
[rfdiffusion.inference.utils][INFO] - Sampled motif RMSD: 0.43
[rfdiffusion.inference.model_runners][INFO] - Timestep 50, input to next step: -------------------IN
```

The first line shows, the structure similarity (RMSD) of the motif compared to the motif in the input pdb file. The second line now shows a '-' for each residue which is built by RFDiffusion and the corresponding residue for the different motif positions.

8. After the script finished we again can take a look at the output again, now written to `output_scaffold`. This is similar to what we already did, for the monomer design. Let's first open the pdb file again and then compare the motif to the motif from the input pdb file.

    ```
    cd ./output_scaffold
    pymol scaffold_0.pdb
    ```

    Then in PyMOL:

    ```
    load ../6UYG.pdb
    align scaffold_0, 6uyg
    set seq_view, 1
    ```

    In the sequence viewer we can now select the motif residues and should see, that the highlighted residues in the structure should overlay perfectly with the motif from the 6uyg pdb. Note: As we only gave RFDiffusion the motif as input, it is possible, that the predicted scaffold can clash with other chains from the 6uyg structure. Also take a closer look at the sequence length of the generated

9. If we now take a look at the `pX0` file in the trajectory folder, we can when playing the trajectories, that the motif is already present in the last state (Timestep 1), as it doesn't get changed by RFDiffusion. *The same can be seen in the `Xt-1` file.*

    ```
    cd ./traj
    pymol scaffold_0_pX0_traj.pdb
    ```

    Then in PyMOL:

    ```
    mplay
    ```

10. Now let's take the motif scaffolding one step further and allow RFDiffusion, to also see the chains of the AR3A antibody (H,L), so that the predicted backbones will not clash with these chains. The RFDiffusion command is mostly the same as for the normal motif scaffolding but we now also need to specify the other two chains, which will be given to RFDiffusion as an input. For this we just add these two chain (`H4-113` & `L2-109`) to the contigs. We also need to tell RFDiffusion, that the input contains separate chains and not one long chain. This can be done via `/0`, which specifies a chain break. The complete command now is:

```
cd ../../
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    'contigmap.contigs=[10-20/E422-440/40-50/0 H4-22/H29-71/H77-113/0 L2-29/L31-95/L97-109]' \
    inference.input_pdb=6UYG.pdb \
    inference.output_prefix=./output_scaffold/scaffold_Ab \
    inference.num_designs=1
```

Note: When an error occurs running the script, after copying it to the console, try to type it into the console on your own. The problem appears to be the line-break within the 'contigmap.contigs' string.

Please take a closer look at this command. The chain break indicator (`/0`) is followed by a white space, before we declare the residues of the next chain.

As the structures of the antibody chains contain a few gaps in the pdb file, we need to specify only the residues, actually present in the pdb file, to not get an error similar to this: `AssertionError: ('L', 96) is not in pdb file!` In this case, we told RFDiffusion to use residue 96 from chain L, but this residue does not exist in the input pdb file.

While running the command, we can now also see in the printed output, that RFDiffusion, sees the two chains of the antibody, with its sequence additionally to the specified motif. Due to the more residues we gave as an input, the running time will also increase (While testing: ~1 minute for only the motif scaffolding and ~ 4 minutes for motif scaffolding with the antibody)

11. Now we can again take a look at the output files (same as 8. & 9., but note that we changed the prefix of the output files from 'scaffold' to 'scaffold_Ab') and should now see the predicted backbone ,as well as the two chains from the antibody we also gave as an input.

12. Additional information about RFDiffusion:

    1. In the paper: Watson, J.L., Juergens, D., Bennett, N.R. *et al.* De novo design of protein structure and function with RFdiffusion. *Nature* **620**, 1089–1100 (2023).
    2. In the GitHub, which contains all the code as well as further examples: GitHub - RosettaCommons/RFdiffusion: Code for running RFdiffusion