# 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.

# Tutorial

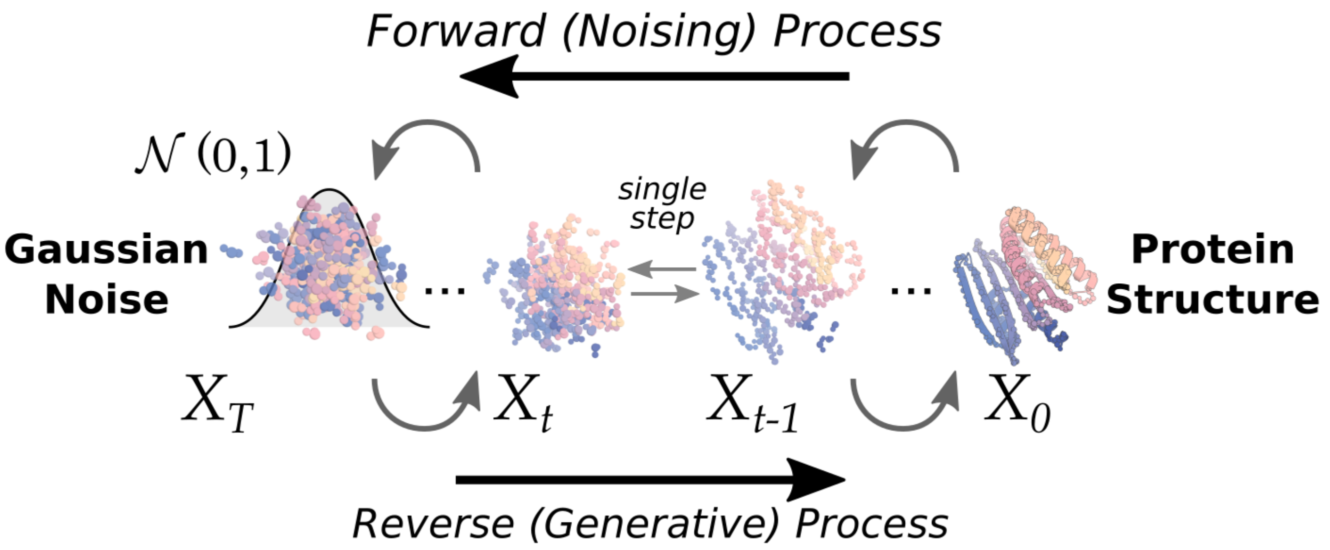
This tutorial will cover the absolute basics of the following topics:

- Unconditional monomer generation
- Binder generation with two differently trained models
- Remodeling of structural elements
- Motif scaffolding

A detailed explanation of the full capabilities of RFDiffusion can be found on the GitHub and the original publication Watson et al. (2023), Nature.

## Introduction

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.



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.

RFDiffusion is a **backbone-generation model** and can not generate sequences onto these backbones. That means all RFDiffussion generated structures are just **poly-Glycine** sequences! 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 wether the generated sequence is capable of folding back into the generated structure. Due to complexity and time-limit, this tutorial exclusively focuses on RFDiffusion.

## Setup

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. If the following command doesn't work, ask the workshop assistant how the correct environment is called.

```
conda activate SE3nv
```

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` directory, we will work from `my_files` for the rest of the tutorial. We also create some additional directories to store the results in an organized form.

```
cd ~/rosetta_workshop/tutorials/rfdiffusion/
mkdir my_files && cd my_files
mkdir out_monomer out_binder out_remodeling out_scaffold
```

## Basic execution - an unconditional monomer

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. **Length of the protein (150aa)**: Here we want to diffuse a 150aa protein and specify it with `'contigmap.contigs=[150-150]'`. For software specific 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 specify the range to only be 150. If we want to generate, e.g. a random length between 100 and 150aa, we would specify that with `'contigmap.contigs=[100-150]'`.
2. The **location** where the output should be written to: Location including a prefix for all generated models can be specified using `inference.output_prefix=path/to/output_folder/prefix`. **The output folder must exists**, otherwise the script will start running and only fail at the very end!
3. **Number of different backbones** to create: This can be done using "`inference.num_designs=5`", which in this case would create 5 different output backbones.

Change to the desired directory

```
cd out_monomer
```

The full command should look like the following:

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

**Side note:** *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.*

After completion of the script, you can go to your output folder (“`cd out_monomer`”) to take a closer look at the different generated output files. The output folder now should contain three types of files:

- “`.pdb`” files with the generated backbones in standard PDB format
- “`.trb`” files with all metadata associated with the specific run, including all input options
- A folder called “`traj`”, which contains all trajectory files. These files are **multi-step** pdbs, which show the full denoising process in action. There are two types: “`pX0`” predictions, containing the predicted model at the end of each timestep and the `Xt-1` files containing the structure that was given to the diffusion model as input for each timestep.

Let’s first take a look at the generated monomer models using PyMOL. For time reasons we will just pick one generated pdb file (the first one). Switch to the output folder and open the first model.

```
pymol monomer_gen_0.pdb
```

In the PyMOL window we can see one of the generated protein backbones with a length of 150aa. If we now look at the sequence (“`set seq_view, 1`” or just in the menu *Display/Sequence*) we can see that RFDiffusion only generates poly-glycine sequences.

Now lets take a look at the corresponding trajectory file. In your opened PyMOL window type the following commands into the commandline (1. load the `traj` file, 2. reverse the order, 3. make the movie slower, **only once**)

```
load traj/monomer_gen_0_pX0_traj.pdb
cmd.set_state_order('monomer_gen_0_pX0_traj', range(50, 0, -1))
set movie_fps, 10
mplay
```

To loop over the all 50 timesteps you can type in the PyMOL command line:

```
mplay       # <- starts the loop movie
mstop       # <- end the loop movie
```

To take a closer look, when can stop the video (`mstop`) and then take a look at a few different states. First we take a look a the first state (1), 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 by steps of 10, we can see how it slowly diffuses into a defined protein backbone. State 50 is the final backbone that is saved in the pdb file `monomer_gen_0.pdb`. In PyMOL:

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

You can also examine the structures provided to the diffusion model for each timestep which are stored in the “`*_Xt-1_traj.pdb`” files. At the first timestep, the input is a large, disordered ball due to the random noise used to initiate the diffusion process. As the steps progress, the bundle gradually spreads out, starting to resemble a protein. In state 1, a protein backbone with some secondary structure is clearly identifiable.

## Binder Design

One of the possibilities using RFDiffusion is the *de novo* design of a protein binder. RFdiffusion shows excellent *in silico* and experimental ability to design *de novo* binders. Because diffusion is computationally intensive, it is often a good idea to crop the protein target around the desired interface location. (*we won't do that in this tutorial, but you should keep that in mind for your own projects*). Additionaly, the model can be provided with so called "hotspot residues". During training, the model has learned to create an interface that includes these hotspot residues.

Here we want to show a workflow to generate *de novo* binder to the target "`insulin_target.pdb`", provided in the "`input_files`" directory. This target is part of a receptor tyrosine kinase responsible for mediating the pleiotropic actions of insulin. Pre-screening experiments identified the hydrophobic critical interaction residues Phe59, Phe83, and Phe91, which we will use as hotspots for our binder.
Change to the "`out_binder`" directory and create two new folders (they will be used in this section).

```
cd ~/rosetta_workshop/tutorials/rfdiffusion/my_files/out_binder
mkdir -p base beta
```

For the binder design, three parameters in the inference script are important:

1. "`inference.input_pdb`" for the protein target.
2. "`contigmap.contigs`" to specify which parts of the target protein will be fixed during the diffusion process as well as the desired length of the binder. Here we want to keep the full target fixed by marking it with a preciding letter of the chain "`A1-150/0`" (the symbol "`\0`" is necessary to symbolize a chain break), followed by the binder defined by the desired range of length ("`50-80`").
3. "`ppi.hotspot_res`" to specify the hotspots that define the interface region. The hotspots are refered to as the chain ID and position of the hotspot residue in the sequence. For residues Phe59, Phe83, and Phe91 of chain A that means "`A59,A83,A91`".

Additionally we add the flag "`inference.deterministic=True`" to make the results reproducible for a subsequent section. The full command (generating 2 designs, stored in the "`base`" folder) is the following:

```
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    inference.input_pdb=../../input_files/insulin_target.pdb \
    inference.deterministic=True \
    'contigmap.contigs=[A1-150/0 50-80]' \
    'ppi.hotspot_res=[A59,A83,A91]' \
    inference.num_designs=2 \
    inference.output_prefix=./base/binder_base
```

**Side note**: *The space in "`[A1-150/0 50-80]`" is important. This tells the diffusion model to add a big residue jump (200aa) to the input, so that the model sees the first chain as being on a separate chain to the second.*

Visualize the generated structures with PyMOL.

```
pymol base/binder_base_0.pdb base/binder_base_1.pdb
```

What do you notice? Exactly, the two generated models don't look like they would bind to the target correctly or cover the hotspots properly. Also, they both are just long helices. This was expected, due to the default RFdiffusion model often generating mostly helical binders.

Even if helical proteins often have high computational and experimental success rates, there may be cases where other kinds of topologies are desired. For this, the developer included a "beta" model, which generates a greater diversity of topologies, but has not been extensively experimentally validated. You can use it by modifying the default model path with "`inference.ckpt_override_path`". If you look in the RFDiffusion main directory you can find a folder called `models`, which also contains a number of other trained models for various tasks. For the binder design we can use the "beta" model with the following command (the path should be provided in a single line, only cropped to fit on page):

```
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    inference.input_pdb=../../input_files/insulin_target.pdb \
    inference.deterministic=True \
    'contigmap.contigs=[A1-150/0 50-80]' \
    'ppi.hotspot_res=[A59,A83,A91]' \
    inference.num_designs=2 \
    inference.output_prefix=./beta/binder_beta \
    inference.ckpt_override_path=~/rosetta_workshop/RFdiffusion/\
        models/Complex_beta_ckpt.pt
```

**Troubleshooting**: If RFDiffusion cant find the "beta" model, even when it is in the `RFdiffusion/model` directory, try to replace the path starting from the home directory (indicated by "`~/`") with the full absolute path. Example: `/home/user/rosetta_workshop/[...]`.

If we now look at the new models,

```
pymol beta/binder_beta_0.pdb beta/binder_beta_1.pdb
```

the generated structures look different to the first ones, especially the fist model (should) contain beta sheets as secondary structure elements. In real world application, it is always important to decide which trained model to use. Also, it should be noted that usually far more than just two models are generated. Often thousands of backbones are subsequently examined for their properties and filtered in order to identify the best binders.

## Loop remodeling

RFDiffusion can also be used to change specific regions of existing proteins. When *remodeling* specific regions, we need a method to specify that we want to remodel particular protein inputs (one or more segments from a `.pdb` file). Additionally, we need to specify how these segments should be connected and by how many residues in the new protein. Moreover, we want the ability to sample different lengths of the remodeled regions, as we generally don't know in advance the precise number of residues required to be optimal. First of all change to the directory "`my_files`" and create a new directory if it does not already exist. Change to the new directory

```
cd ~/rosetta_workshop/tutorials/rfdiffusion/my_files/
mkdir -p out_remodeling & cd out_remodeling
```

Here, we want to use the remodeling workflow to take our best binder **from the last section** "`binder_beta_0.pdb`" as input and refine it. We do this by remodeling a loop that is close to the interface, which is likely to be a significant part of the interaction. The loop comprises residues "`63-69`" of chain A and we want to remodel this region with a loop of length "`6-12`". So we need to tell RFDiffusion to keep all residues before ("`A1-62`") and after ("`A70-77`") this region of interest. Also we have to add a chainbreak ("`/0`") and the target protein ("`B78-227`"). We don't need new parameters, because this is handled with the parameter "`contigmap.contigs`". We will create 5 designs for the new loop.

```
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    inference.input_pdb=../out_binder/beta/binder_beta_0.pdb \
    'contigmap.contigs=[A1-62/6-12/A70-77/0 B78-227]' \
    inference.output_prefix=./remodel_loop \
    inference.num_designs=5
```

**Side note**: *The output should look like the following, where we can see clearly a new section inside the glycin chain. RFDiffusion marks all positons that are newly diffused or generally "newly" generated by the network with an "-".*

```
[...][rfdiffusion.inference.model_runners][INFO] - Using contig: [...]
[...][rfdiffusion.inference.model_runners][INFO] - Sequence init:
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG------------GGGGGGGG\
EVCPGMDIRNNLTRLHELENCSVIEGHLQILLMFKTRPEDFRDLSFPKLIMITDYLLLFRVYGLESLKDLFPNLTVIRGSRL\
FFNYALVIFEMVHLKELGLYNLMNITRGSVRIEKNNELCYLATIDWSRILDSVEDNHIVLNKDDNEEC
```

We now compare the created models and the input structure:

```
pymol ../out_binder/beta/binder_beta_0.pdb remodeling_0.pdb remodeling_1.pdb \
    remodeling_2.pdb remodeling_3.pdb remodeling_4.pdb
```

To make the differences more visible, we superimpose all structures using the PyMOL command (just type it in the PyMOL console)

```
alignto
```

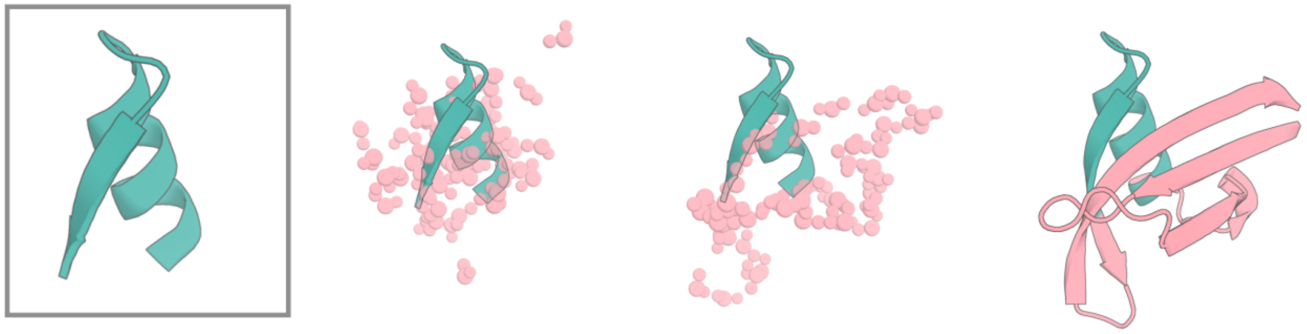
Additionally, by displaying the sequence with the command "`set seq_view, 1`" (or through the menu "`Display/Sequence`"), you can compare the differences in loop length and their impact on the three-dimensional structure.

## Motif scaffolding

We can also generate a protein scaffold for a motif that is known to fulfill a specific purpose. To scaffold protein motifs effectively, we require: 1. A method to specify the desired protein input (one or more segments from a .pdb file) 2. A way to define the connections between these segments in the new scaffolded protein 3. The ability to specify the number of residues for these connections

Additionally, we need to sample various connection lengths, as the optimal number of residues for scaffolding a motif is often not known in advance. Change to the directory "`my_files`" and - if it does not already exist - create the last result directory.

```
cd ~/rosetta_workshop/tutorials/rfdiffusion/my_files/
mkdir -p out_scaffold & cd out_scaffold
```



In this example we look at the interaction of membrane-proximal external region (MPER) of the HIV Envelope protein Env and antibody 10E8 provided in "`input_files/8U08.pdb`". We want to extract MPER and design an epitope scaffold that stabilizes MPER in the 10E8-bound conformation. The scaffold needs to be generated in a way that RFDiffusion builds exactly 39 residues ("`39-39`") N-terminally of "`C96-107`" from the input pdb, followed by "`20-50`" residues to its C-terminus. All new parts are randomly sampled at each inference cyle. Also we want to inpaint the presence of the Fv fragments for the heavy and light chain of the 10E8 antibody with "`H1-131/0 L2-119/0`".

```
~/rosetta_workshop/RFdiffusion/scripts/run_inference.py \
    inference.input_pdb=../../input_files/8U0B.pdb \
    'contigmap.contigs=[39-39/C96-107/20-50/0 H1-131/0 L2-119/0]' \
    inference.output_prefix=./out_scaffold/scaffold \
    inference.num_designs=2
```

Use PyMOL to investigate your results. Tomorrow you will need one of the models in the immunogen design workshop.