Machine Learning in Rosetta

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`)! Copy & pasting can sometimes lead to weird errors, so when in doubt try to type the commands instead.

Tutorial

In this tutorial we will use three different machine learning (ML) based methods embedded in Rosetta to predict stabilizing mutations of a plastic degrading enzyme called Polyester Hydrolase Leipzig 7 (PHL7). If you want to read more about our target protein, take a look at this paper (click here). Important for us will be that the crystallized structure is an inactive variant because of a point mutation (S131A). We'll use ProteinMPNN (Dauparas et al. 2022, this and all other citations are clickable links), the Evolutionary Scale Modeling (ESM) protein language model family (Lin et al. 2022, Rives et al. 2019) and the masked inverse folding with sequence transfer (MIF-ST) model (Yang et al. 2023). As a very short and simple comparison, ProteinMPNN was trained on protein structures including complexes, ESM was trained on sequences only, while MIF-ST was pre-trained on sequences and then conditioned on monomeric structures, which results in differences in their predictions which we will discuss. If you are running this on your own workstation you will need to compile Rosetta with TensorFlow/LibTorch which is described here (click me).

1. Create a directory in the ml_in_rosetta directory called my_files and switch to that directory. We will work from this directory for the rest of the tutorial.

   ```
   mkdir my_files
   cd my_files
   ```

2. In order to run all subsequent commands, set an alias to your Rosetta folder (which for this workshop is: `~/rosetta_workshop/rosetta`) by replacing `/PATH/TO/YOUR/ROSETTA/FOLDER/` with your actual path:

   ```
   export ROSETTA=/PATH/TO/YOUR/ROSETTA/FOLDER/
   ```

3. Prepare the input structure.

   1. Download the PDB file. (**The 8BRB.pdb file is also provided in the input_files directory.**)
      1. Download 8BRB from the Protein Data Bank.
         1. Go to rcsb.org and type '8BRB' in the search bar.
         2. Click on 'Download Files' on the right side of the page, then 'Legacy PDB Format'.
         3. Save the PDB file in the my_files directory as `8BRB.pdb`.
            - You may need to download the file into the ~/Downloads/ directory, then `mv` into the my_files directory.
      2. Open the structure in pymol, and check the number of chain for the structures. Which other non-protein objects can you find in the PDB?

         ```
         pymol 8BRB.pdb
         ```

         Type following commands in the PyMol command line:

         ```
         set seq_view, 1
         ```

   2. Clean the PDB from everything that's not protein. Be careful with the new numbering which is off by one, as the first residue in the PDB has no coordinates and gets removed by the clean_pdb script. Keep this in mind when comparing positions to literature later. Let's get chain A from 8BRB:

      ```
      python $ROSETTA/main/tools/protein_tools/scripts/clean_pdb.py 8BRB A
      ```

   3. Next it's time to repack the input structure. Repacking is often necessary to remove small clashes identified by the score function as present in the crystal structure. When the repacking runs are done, select the best scoring PHL7 structure. (For brevity, we only generated a single structure. For actual production runs, we recommend generating a number of output structures, by increasing the `-nstruct 1` option to e.g. `-nstruct 25`). This run will take approximatively 10 minutes.

      ```
      $ROSETTA/main/source/bin/relax.tensorflowtorch.linuxgccrelease -s 8BRB_A.pdb  \
        -nstruct 1 -ex1 -ex2 -constrain_relax_to_start_coords -out:suffix _relax -beta
      ```

      It can also be useful to pre-generate more backbone conformational diversity prior to design. However, backbone conformational diversity will not be explored in this tutorial due to computational time constraints.

4. Next, we predict amino acid probabilities with ProteinMPNN, MIF-ST and ESM. While ProteinMPNN is a quite fast calculation, ESM inference takes longer depending on which parameter size model is used. Here we use a slightly smaller model with 150M parameters `esm2_t30_150M_UR50D`. If your setup can handle it feel free to substitute this with the larger model `esm2_t33_650M_UR50D`. The reason for the longer calculation time is A) the larger parameter size and B) that for ESM prediction each residue gets masked and predicted one by one. The `-auto_download` flag is used to download the specified ESM model (if not already present) which can be 1-3 GB large and therefore is not included in Rosetta by default.

   1. Copy the xml file and inspect it:

      ```
      cp ../input_files/predict_probs.xml .
      gedit predict_probs.xml
      ```

   2. Run the script: The following command and XML runs predictions for ProteinMPNN & ESM & MIF-ST and then saves them. It will take approximatively 15 minutes.

      ```
      $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
        -s 8BRB_A_relax_0001.pdb -parser:protocol predict_probs.xml -auto_download \
        -out:file:score_only score.sc
      ```

   If things are taking a particularly long time to run, you can continue with the probabilies which have been written so far.

   3. Analyze the outputs: **The saved probabilities are also provided in the output_files directory as esm_probs.weights, mifst_probs.weights and mpnn_probs.weights.** Each files contains three columns: the residue position, the residue type and the calculated probabilities.

      ```
      gedit mpnn_probs.weights
      gedit esm_probs.weights
      gedit mifst_probs.weights
      ```

5. Now let's analyze and compare the predictions made by ProteinMPNN, ESM and MIF-ST. First, we will analyze the probabilities of the wild type amino acids, which will be between 0 (0%) and 1 (100%). To do so we load either prediction, take out just the probability for the current amino acid and then store it in the b factor column of the output PDB (the first column from the right). This allows us to easily visualize the scores in PyMol/ChimeraX.

   1. Copy the xml file and inspect it:

      ```
      cp ../input_files/current_probs.xml .
      gedit current_probs.xml
      ```

   2. Get the current probabilities for ProteinMPNN:

      ```
      $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
        -s 8BRB_A_relax_0001.pdb -parser:protocol current_probs.xml \
        -out:suffix _mpnn_probs -parser:script_vars filename=mpnn_probs.weights
      ```

   3. Get the current probabilities for ESM:

      ```
      $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
         -s 8BRB_A_relax_0001.pdb -parser:protocol current_probs.xml \
         -out:suffix _esm_probs -parser:script_vars filename=esm_probs.weights
      ```

   4. Get the current probabilities for MIF-ST:

```
$ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
  -s 8BRB_A_relax_0001.pdb -parser:protocol current_probs.xml \
  -out:suffix _mifst_probs -parser:script_vars filename=mifst_probs.weights
```

   5. Let's open PyMol and look at the differences, we will first color the residues by their probabilities going from red (low) to blue (high). Let's also load the original PDB which still has the bound product of PHL7 and take a look at the surrounding residues.

      ```
      pymol
      ```

      Type following commands in the PyMol command line.

      ```
      load 8BRB_A_relax_0001_mpnn_probs_0001.pdb
      load 8BRB_A_relax_0001_esm_probs_0001.pdb
      load 8BRB_A_relax_0001_mifst_probs_0001.pdb
      spectrum b, red_white_blue, minimum=0, maximum=1
      select residues, resi 63+64+68+131+155+176+208+131+130
      show sticks, residues
      load ../input_files/8BRB.pdb
      remove solvent
      remove 8BRB and chain B
      set seq_view, 1
      ```

   ProteinMPNN, ESM and MIF-ST don't have any direct information on the ligand, the product binding site or the function of PHL7, however, you will realize that ESM and MIF-ST predict the involved binding site residues as more likely as ProteinMPNN. This comes down to the way these two models were trained, while ESM relies heavily on evolutionary information from sequences, ProteinMPNN relies more on the actual provided input structure. Interestingly, all models predict that the mutation that makes PHL7 inactive (S130A in our numbering) is unlikely. This shows that while ESM/MIF-ST might have captured more evolutionary information, ProteinMPNN definitely has learned how a catalytic triad should look. The limitations with ProteinMPNN have been at least partially addressed by the recently released LigandMPNN which takes into consideration non-protein atoms, leading to increased prediction accuracy on residues interacting with ligands.

6. So now lets use the information from the prediction to find some mutations. First, we will predict the most impactful single-point mutations (useful if you are interested in just doing site-directed mutagenesis and don't want to change too much of your protein). For simplicity, we will first average the predictions made by ProteinMPNN, ESM and MIF-ST, but you could also run this for each separately. Run the BestMutationsFromProbabilitiesMover (which does exactly what the name suggests, by calculating the difference in probability between the wild type and the amino acid predicted as most likely).

   1. Copy the xml file and inspect it:

      ```
      cp ../input_files/best_mutations.xml .
      gedit best_mutations.xml
      ```

   2. Run the command:

      ```
      $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
        -s 8BRB_A_relax_0001.pdb -parser:protocol best_mutations.xml -beta     \
        -out:file:score_only best_mutations.sc
      ```

   3. Analysis: Check the top mutations printed at the end of the run. The mutation with the highest delta_probability is A130S, aka the mutation which restores the activity of PHL7, highlighting the evolutionary/structural information learned by ESM/ProteinMPNN. Another interesting proposed mutation is E147K. E147 and D232 are part of an metal ion binding site in PHL7 and Richter et al. replaced the sodium-mediated interaction by an electrostatic one through a D232K mutation. This resulted in a melting temperature increase of 0.9 °C and the PET-hydrolytic activity also increased slightly. It would be interesting to see if the reverse direction proposed by ProteinMPNN/ESM with E147K instead of D232K has the same effect.

7. Now we will use the information stored in the probabilities to create some new enzyme sequences using the `SampleSequenceFromProbabilities` mover. For time reasons we will just create 3 designs, feel free to create more depending on your constraints. You can control the diversity with the `pos_temp` and `aa_temp` option, where values higher than 1.0 lead to more diversity and lower than 1.0 to less. Additionally, you can choose the amount of mutations you'd like to have through the `max_mutations` option. Lastly, we score the protein using the Rosetta

energy function and another round of ProteinMPNN/ESM/MIF-ST predictions via the PseudoPerplexityMetric. The pseudo-perplexity calculates a single score (where lower is better, min=1) from the predicted probabilities for each ProteinMPNN, MIF-ST and ESM. For time reasons the MIF-ST pseudo-perplexity calculation is commented out in the XML, feel free to enable it depending on your time/computational constraints.

1. Copy the xml file and inspect it:

   ```
   cp ../input_files/sample_sequences.xml .
   gedit sample_sequences.xml
   ```

2. Run the command. It will take aproximatively 60 minutes, so feel free to copy the results from the output_files folder and proceed to the next step:

   ```
   $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
     -s 8BRB_A_relax_0001.pdb -parser:protocol sample_sequences.xml -beta   \
     -out:suffix _design -nstruct 3
   ```

3. Analysis: Take a look at the output structures (e.g. in PyMol), which mutations got introduced and where are they? How do their Rosetta total_scores compare to the wildtype (take a look at the score_design.sc scorefile). Are better total scores also reflected in a better ESM or ProteinMPNN pseudo-perplexity score? The following commands can help you to filter the score file.

   ```
   awk '{print $NF, $2, $27,$4,$26}' score_design.sc
   ```

8. Last but not least, we can restrain (aka “constrain” in Rosetta jargon) the energy function of Rosetta with our predictions and then run any protocol (like design) using the modified energy function. Additionally, we can turn off amino acids below a custom probability threshold for design, which drastically limits the sequence space we have to search. We use the saved probabilities (in PSSM formatting) together with the FavorSequenceProfileMover to constrain the energy function. Besides that, we use the RestrictAAsFromProbabilities task operation to turn off any amino acids that are less likely than the wild type, feel free to play around with the probability cutoffs. For the design part we are using the PackRotamersMover (aka fixBB) with a resfile allowing all amino acids except cysteine, but you could also modify this to keep the active site residues fixed.

   1. Copy the xml file and resfile, and inspect them:

      ```
      cp ../input_files/constrain_energy.xml .
      gedit constrain_energy.xml
      cp ../input_files/resfile.resfile .
      gedit resfile.resfile
      ```

   2. Run the command (it will take approximately 10 minutes):

      ```
      $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
        -s 8BRB_A_relax_0001.pdb -parser:protocol constrain_energy.xml -beta -ex1  \
        -ex2aro -nstruct 20 -out:suffix _constraint_design
      ```

   3. Analysis: The results in the scorefile score_constraint_design.sc include the Rosetta scores, the sequence recovery (% of positions that are identical to the wild type) and again the pseudo-perplexity score using ProteinMPNN. How much of the sequence got changed?

      ```
      awk '{print $NF,$2,$31,$25}' score_constraint_design.sc
      ```

      For the constraint designs we used the PackRotamersMover and therefore didn't change the backbone of our protein. If you have time try to modify the XML file to either introduce a FastRelaxMover after the design or directly replace the PackRotamersMover with a FastRelaxMover that has the same task operations (and therefore runs design). Take a look at the example from our run before input_files/sample_sequences.xml and the documentation here to get started. Besides that, play around with the probability thresholds of the RestrictAAsFromProbabilities, how does that influence the sequence recovery of the designed sequences?

9. Switching gears for the last part of this tutorial, let's dive further into the differences of the ML models used here. We already saw that models like MIF-ST and ESM, which are trained on large sequence datasets, capture evolutionary properties even in the absence of direct information. So in our example of enzyme design, we might prefer to use ProteinMPNN to sample mutations for stability/solubility objectives and ESM/MIF-ST to sample mutations that affect enzymatic activity. But what happens if we have a design objective that is not in line with the natural function of a protein? Another common design objective is to create/increase binding affinity of one protein to another protein

target. Let's take a look at how ProteinMPNN predicts the likelihood of residues laying in the interface of a heterodimer compared to MIF-ST. To do so we will use a structure of the nucleosome core particle (Davey et al. 2002), **provided as 1KX5_chAB.pdb**. A common way to score interfaces in Rosetta is to calculate the interface score ("dG_separated") by substracting the total_score of the separated state from the total_score of the complexed state. This can be done with the `InterfaceAnalyzerMover` (documented here), which also can calculate the delta for an arbitrary `RealMetric`. We will use this mover to not only calculate the Rosetta interface score but also the difference in ProteinMPNN/MIF-ST pseudo-perplexity between the complexed and separated states. This way we can analyze if the predicted probabilities change depending on whether we provide a protein complex or single proteins to the models.

1. Copy the pdb file and the xml file, and inspect them:

   ```
   cp ../input_files/1KX5_chAB.pdb .
   cp ../input_files/IFA_perplexity.xml .
   gedit IFA_perplexity.xml
   ```

2. Run the command (it will take approximately 10 minutes):

   ```
   $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
     -s 1KX5_chAB.pdb -parser:protocol IFA_perplexity.xml -beta \
     -out:file:score_only score_interface.sc
   ```

3. Analysis: Check the scorefile for the Rosetta interface score ("dG_separated") and the delta ProteinMPNN/MIF-ST pseudo-perplexity.

   ```
   awk '{print $NF,$2,$6,$12,$11}' score_interface.sc
   ```

   You'll see that the Rosetta score of the complexed state is lower (better) than of the separated states, and that this is also the case for the ProteinMPNN pseudo-perplexity. That means that the sequence of our proteins is seen as more likely by ProteinMPNN in their complexed state than separated by themselves, which makes sense as the inferace residues that drive the formation of the complex come with a cost to the stability of the monomeric proteins. Compare this to the delta pseudo-perplexity of MIF-ST which is basically zero, meaning that MIF-ST does not predict different probabilities between the complexed and separated states. One plausible sounding reason might be that MIF-ST was only trained on monomeric protein structures. However, while the delta is zero, that should still mean that the interface residues should be predicted as unlikely, as they are not ideal for the monomeric state.

10. Let's check whether the interface residues are predicted as unlikely by MIF-ST. We do this by using the `CurrentProbabilitiesMetric` which we have used before to easily color by probability in PyMol.

    1. Copy the xml file and inspect it:

       ```
       cp ../input_files/current_probs_mifst_dimer.xml .
       gedit current_probs_mifst_dimer.xml
       ```

    2. Run the command (it will take approximately 5 minutes):

       ```
       $ROSETTA/main/source/bin/rosetta_scripts.tensorflowtorch.linuxgccrelease \
         -s 1KX5_chAB.pdb -parser:protocol  current_probs_mifst_dimer.xml \
         -out:suffix _mifst_probs
       ```

    3. Analysis: Again open PyMol, load the PDB and color by probabilities:

       ```
       pymol
       ```

       Type in the pymol commandline:

       ```
       load 1KX5_chAB_mifst_probs_0001.pdb
       spectrum b, red_white_blue, minimum=0, maximum=1
       ```

       Are the interface residues predicted as unlikely by MIF-ST? As it turns out they are predicted as highly probable. That's because MIF-ST was trained on a large sequence dataset, has therefore learned more evolutionary information and predicts the interface residues as likely independent of the structural input. This difference between models like ESM or MIF-ST and ProteinMPNN is both a bug and a feature. Basically the evolutionary informed models are likely to conserve the natural function of your design target even without you providing explicit information about it. At the same time, if you'd like to move away from the natural function, e.g. prevent the binding and stabilize the monomer in our case, MIF-ST/ESM are probably not effective tools while ProteinMPNN allows you to achieve your design objective.

That's it for now, if you are further interested take a look at the documentation here (click me) for the RosettaScripts elements we worked with today.