CSCI-GA.3033-111: Protein Design HW4

Christine Wu (email:cw4459@nyu.edu)

Dec 7 2024

A. Make a copy of the RFDiffusion colab.

Solution: We made the copy of the colab page for RFDiffusion from RFDiffusion Colab.

B. Unconditional generation: generate 8 samples in the length range 80-100 amino-acids. Visualize 5 designs at random. NOTE: This takes a while, do not be alarmed by high run times.

Solution: We did the unconditional design by setting up the parameters as the following from the RFDiffusion Colab, and we changed the parameters as 1) name: unconditional_generation, 2) contigs: 80-100, 3) iternations: 50, and 4) num_designs: 8. We keep the rest of the parameters the same. Then, we visualized 5 different proteins at random:

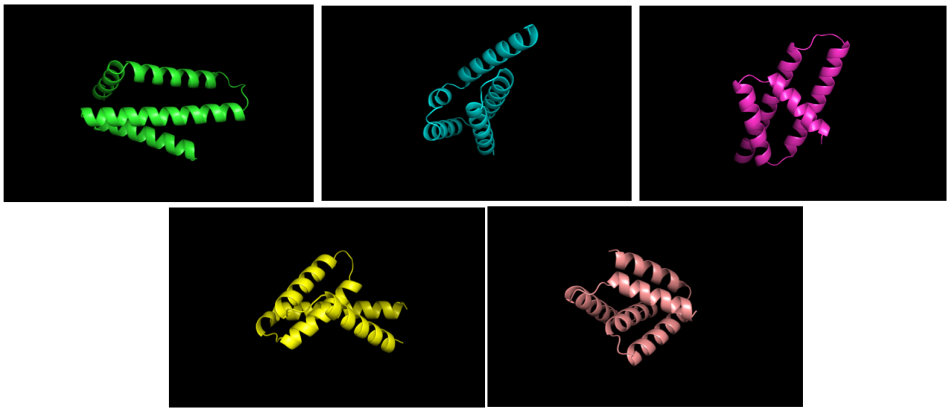

Figure 1: Protein Visualization at Random

C. Plot a histogram of the fraction of alpha helix per design.

Solution: We plotted the histogram of alpha helix fractions and a bar plot of alpha helix fractions per design, using the pdb files downloaded from RFdiffuison. The histogram shows the overall distribution of alpha-helix content across all designs, and the bar chart allows us to compare the alpha-helix fraction for each specific structure.

We 1) cleaned PDB files to remove disordered atoms, analyzing their secondary structures using DSSP (DSSP: Assign secondary structure to proteins), and calculated the fraction of alpha helices for each protein design. 2) Then, we visualized results by generating a histogram and bar plot of alpha-helix fractions, saving the outputs and results to an organized directory.

The plots are the following:

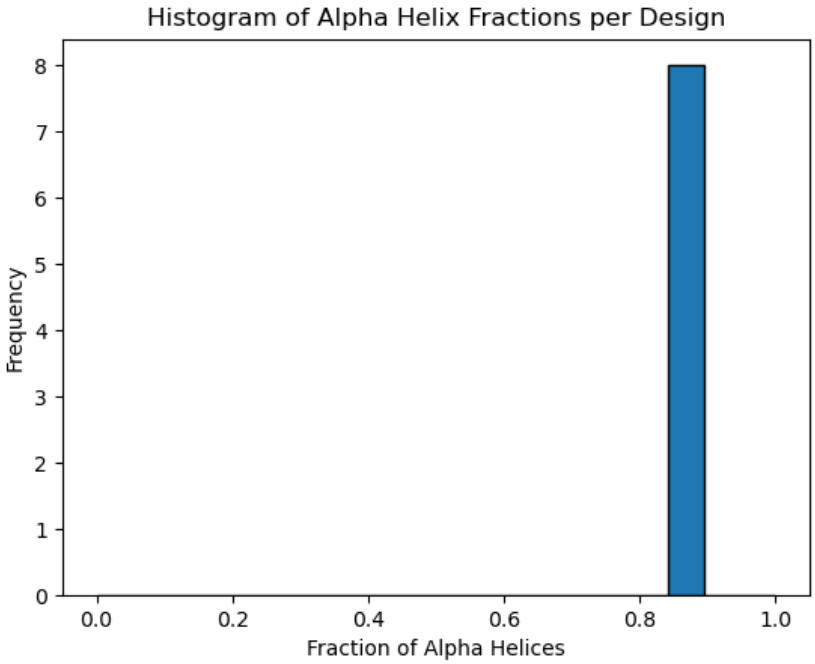

Figure 2: Histogram of Alpha Helix Fractions per Design

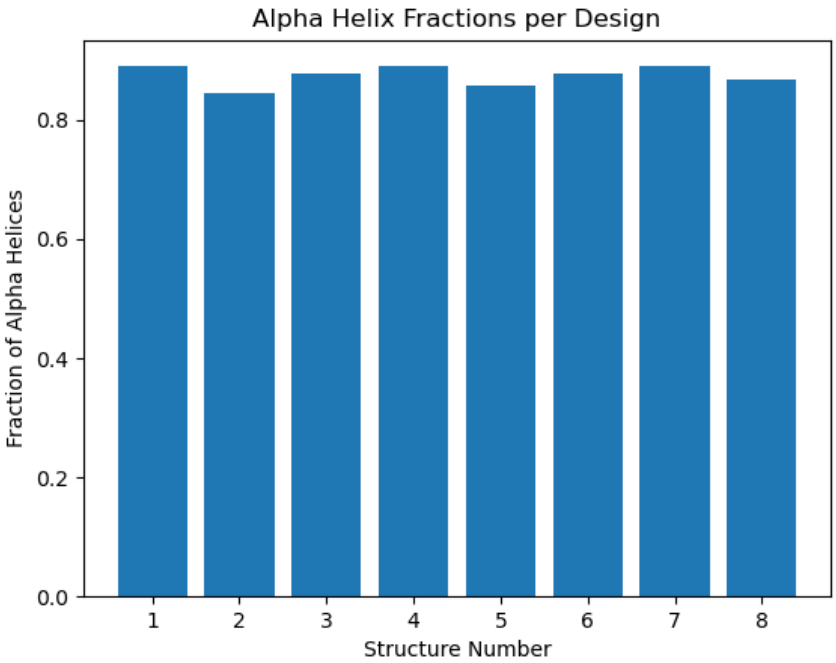

Figure 3: Alpha Helix Frations per Design

The output results from the code is:

```
Alpha Helix Fractions: [0.8901098901098901, 0.8461538461538461, 0.8791208791208791, 0.8901098901098901,
    0.8571428571428571, 0.8791208791208791, 0.8901098901098901, 0.8681318681318682]
Rounded Alpha Helix Fractions: [0.8901, 0.8462, 0.8791, 0.8901, 0.8571, 0.8791, 0.8901, 0.8681]
Results saved to: ./output_results
```

Summary: As we can tell from the two plots, our 8 designs are mostly alpha-helix liked. We have

a narrow peak at 0.8 means most designs are highly alpha-helical from our first plot, and the bar plot made us recognize that our 8 designs are all close to 0.8 percent alpha-helical uniformly. This conclusion actually matches our visualization from part B. The code we used is:

```python
import os
from Bio.PDB import PDBParser, DSSP, PDBIO, Select
import matplotlib.pyplot as plt
import numpy as np

# Class to clean a PDB file
class CleanATOM(Select):
    def accept_atom(self, atom):
        return atom.is_disordered() == 0 # Remove disordered atoms

def clean_pdb(input_pdb, output_pdb):
    parser = PDBParser(QUIET=True)
    structure = parser.get_structure("protein", input_pdb)
    io = PDBIO()
    io.set_structure(structure)
    io.save(output_pdb, select=CleanATOM())

def calculate_alpha_fraction(pdb_file):
    parser = PDBParser(QUIET=True)
    structure = parser.get_structure("protein", pdb_file)
    model = structure[0] # Assuming first model
    dssp = DSSP(model, pdb_file)

    alpha_helix_count = sum(1 for residue in dssp if residue[2] == 'H')
    total_residues = len(dssp)
    return alpha_helix_count / total_residues if total_residues > 0 else 0

# Input and output directories
pdb_dir = "./" # Input PDB files directory
output_dir = "./output_results" # Output directory for results

# Create output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Process PDB files
pdb_files = [f for f in os.listdir(pdb_dir) if f.endswith(".pdb")]
alpha_fractions = []

# File to save results
results_file = os.path.join(output_dir, "alpha_helix_fractions.txt")

with open(results_file, "w") as f:
    for pdb_file in pdb_files:
        try:
            # Clean the PDB file
            clean_file = os.path.join(output_dir, pdb_file.replace(".pdb", "_clean.pdb"))
            clean_pdb(os.path.join(pdb_dir, pdb_file), clean_file)

            # Run DSSP on the cleaned file
            alpha_fraction = calculate_alpha_fraction(clean_file)
            alpha_fractions.append(alpha_fraction)

            # Write results to file
            f.write(f"{pdb_file}: {alpha_fraction:.4f}\n")
        except Exception as e:
            print(f"Error processing {pdb_file}: {e}")

print("Alpha Helix Fractions:", alpha_fractions)

# Save the histogram plot
plt.hist(alpha_fractions, bins=np.linspace(0, 1, 20), edgecolor="black")
plt.title("Histogram of Alpha Helix Fractions per Design")
plt.xlabel("Fraction of Alpha Helices")
plt.ylabel("Frequency")
histogram_path = os.path.join(output_dir, "alpha_helix_histogram.png")
plt.savefig(histogram_path)
plt.show()
```

D. What does "iterations" setting relate to in the colab?

Solution: The "iterations" setting controls the number of denoising steps in the RFDiffusion model, balancing between accuracy and computational time.

RFDiffusion generates protein structures through a step-by-step process where noise is iteratively removed to refine the structure toward a valid protein design. Each "iteration" represents one step of this refinement process. More iterations allow the model to gradually improve the structure by adjusting it at each step.

A higher iterations leads to more refined and realistic protein structures but increases runtime. A lower iterations means faster results but might lead to incomplete or less accurate designs.

E. Generate some symmetric homo-trimers and visualize a few of them.

Solution: In molecular biology, a homotrimer is a protein which is composed of three identical subunits of polypeptide, arranged symmetrically. This indicates that we need to tune the symmetry parameter from the Colab page. So we set it to symmetry to "cyclic" and the order as 1 for a cyclic 3-fold symmetric homo-trimer. We keep the general diffusion setting the same, and we have the following results.

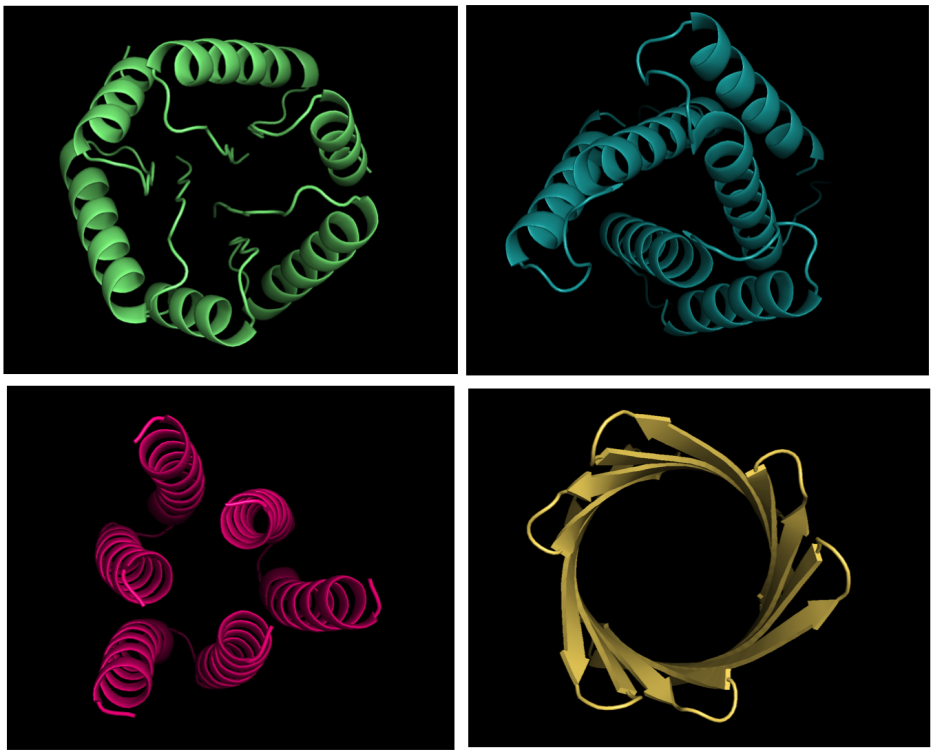

Figure 4: Symmetric Homo-Trimers

F. Generate binders for the target IL-7Rα (PDB: 3DI2). Use length range 70-100 AA. Use hotspots B58, B80, B139. Visualize 3 of them.

Solution: We did what happened was asked in the question:

1. We first downloaded the target IL-7R from the PDB.
2. Then we generated the binders with contigs setting as 70-100, and the hotsppot as B58, B80, B139.

Then results are the following:

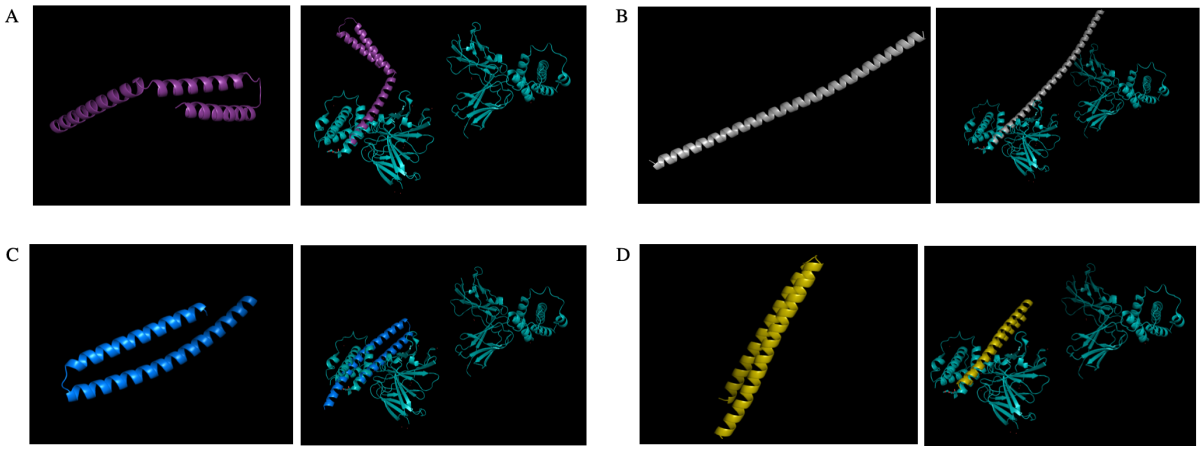

Figure 5: A pair of binder designs: Left: the binder itself and Right: when the binder aligns with the target. We have 4 pairs, which are A, B, C and D.

G. Generate some scaffolds for the motif present in this PDB file (Japanese firefly luciferase: 2d1s).

Solution: We first downloaded the PDB file from the PDB bank for 2d1s (Japanese firefly luciferase: 2d1s), then we followed the guideline indicated by the assignment. We were able to find two different scaffolds. One is 5 $\mathring{A}$ and the other one is 8 $\mathring{A}$. The results are the following:

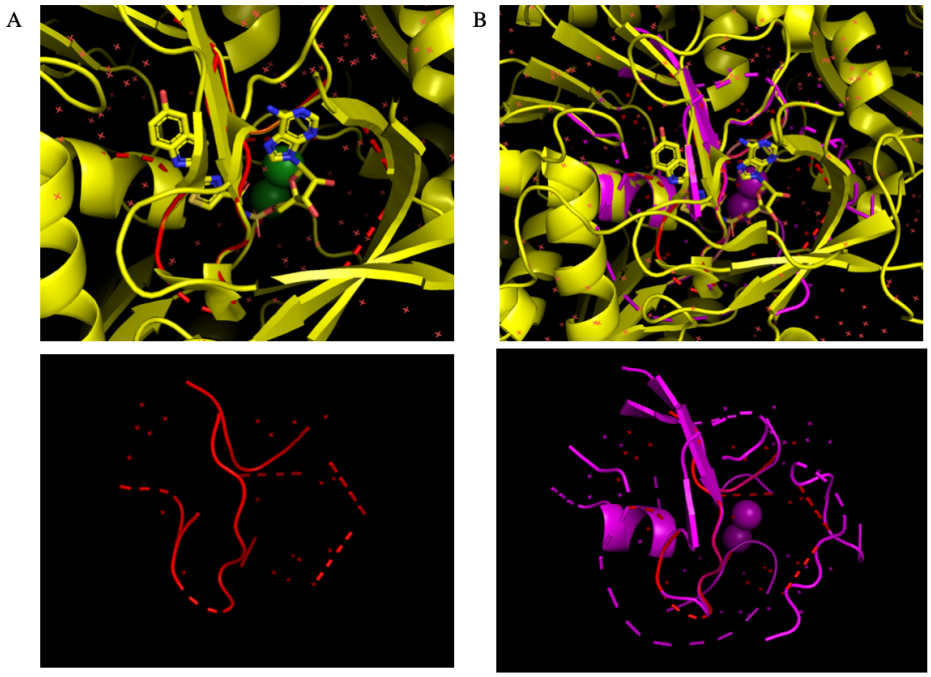

Figure 6: Scaffolds with 5 $\mathring{A}$ and 8 $\mathring{A}$. The red one is 5 $\mathring{A}$ and the purple one is 8 $\mathring{A}$. The top row shows how they are from the motif and the bottom row shows how they are independently.

H. Run partial diffusion on 2d1s. Visualize the output, overlaid the original.

Solution: We ran three partial diffusion runs from the RFDiffusion colab page (Updated RFDiffuison Colab page).

1. For the first run, we used the original target protein 2d1s and all settings as default (contigs: 100, iterations: 50; for the advanced setting for partial diffusion, we kept everything as default).
2. For the second run, we used the same parameters as the first one but changed the advanced option to `use_beta_model` to generate a more balanced and realistic structure.
3. For the third run, we modified the original 2d1s structure in PyMOL by removing residues around the binding target area (since the colab page didn't work for me with the hotspot setting, I'm trying to mimic the effect of setting partial diffusion with the whole protein and froze the segments that were in the reaction sites). This modification increased the flexibility of the model during partial diffusion, allowing it to explore and refine the binding region more effectively.

The below are the results:

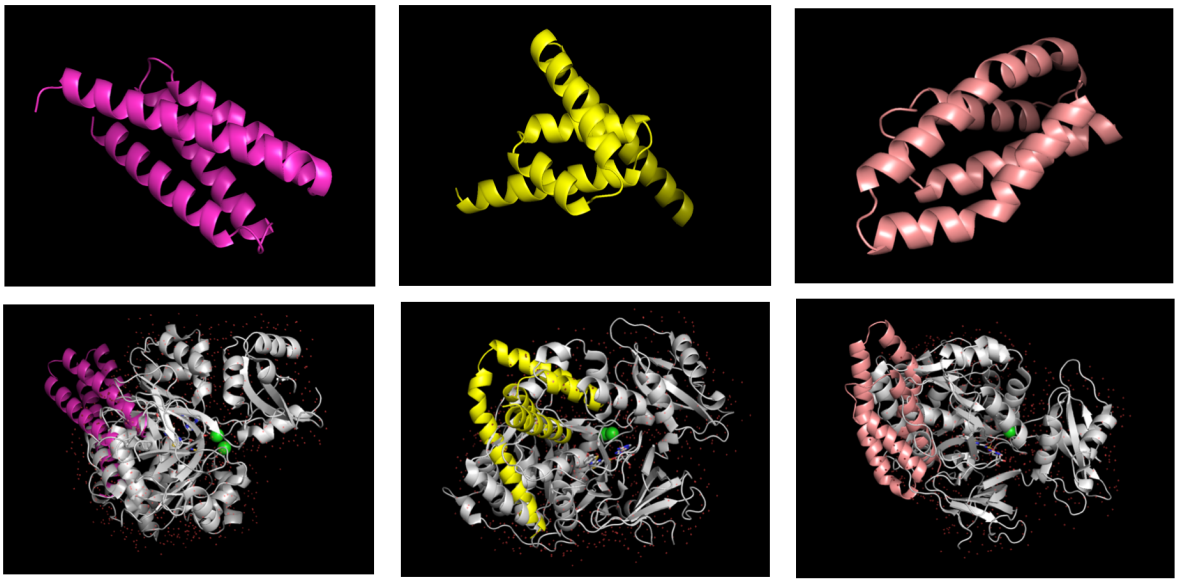

Figure 7: Partial diffusion results on 2d1s. Binding and without binding (overlaid) to the original.

I. Compare the 2d1s structures generated using motif scaffolding and partial diffusion. Overlay them in PyMol.

Solution: We used the previous 3 structures generated, and compared with the 2 scaffolds (one with $5\mathring{A}$ and the other with $8\mathring{A}$). We have the results as below:

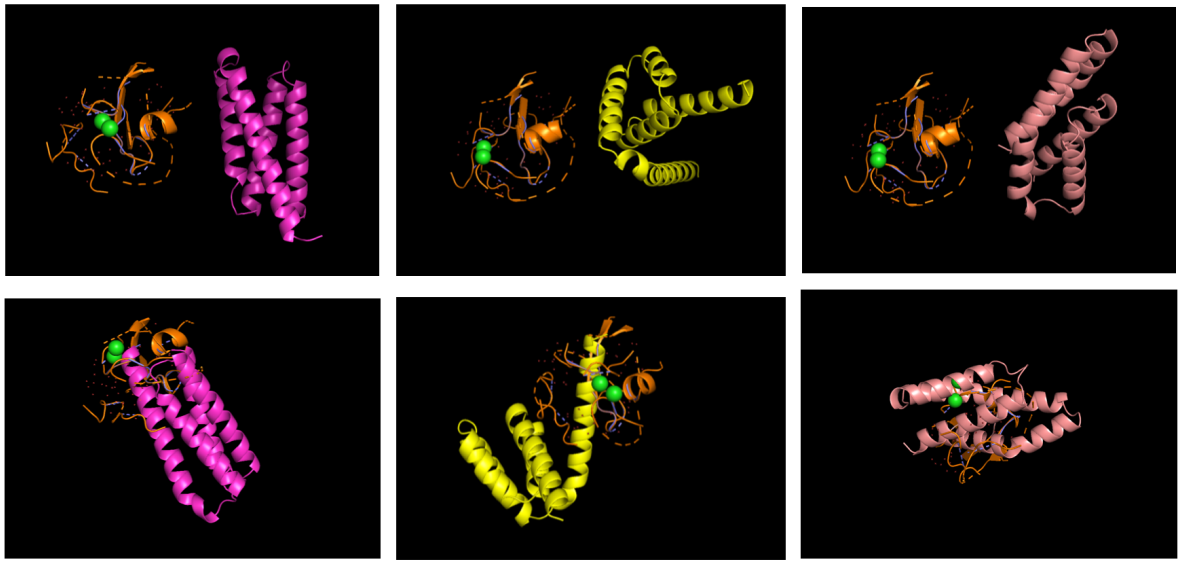

Figure 8: Top: structure comparisons; Bottom: overlaid view

Summary: As we can see, the last run we performed with RFDiffusion with the modified 2d1s structure gives the best percentage overlaid, whereas the other two (left and middle) don't perform too well, and have large derivation.

J. Generate protein complexes for the partial diffusion and scaffolded proteins using Chai-1. You should use the lowest PTM sequence present in the MPNN step with the lowest RMSD score, and the ligand present in the PDB file. Calculate rosetta energy scores for these complexes. Compare the partial diffused structure, the scaffolded structure, and the original 2d1s file.

Solution: We aimed to have three pairs of the complexes:

1 The sequence with the lowest RMSD score from partial diffused structure - ligand.

2 The scaffolded structure - ligand.

3 The original 2d1s strutcure - ligand.

For item 1, we selected the lowest RMSD score as from the mpnn_results.csv file, and we picked the sequence with an RMSD score of 0.47883141040802, which is:

ADPAAARAALRAALAEELAALEAFLARHRASLSPEELALLEQAVAAARAALTPDLAERAAALEALLALLRELAARLPEHA
AELEARAARVAALLAEVRAL.

For item 2, we converted the pdb file into a sequence using the below code:

```
from Bio.PDB import PDBParser, PPBuilder

# Load the PDB file
pdb_file = "/users/chessbunny/desktop/2d1s_sa_5.pdb"
parser = PDBParser()
structure = parser.get_structure("protein", pdb_file)

# Extract the amino acid sequence
ppb = PPBuilder()
```

```python
for i, pp in enumerate(ppb.build_peptides(structure)):
    print(f"Sequence {i + 1}: {pp.get_sequence()}")
```

This code gave the output of:

```
Sequence 1: SS
Sequence 2: HGF
Sequence 3: ASGGAP
Sequence 4: QGYGLTE
Sequence 5: TSA
```

Then, we combined those pieces together for a whole chai-1 input which is:

`SSHGFASGGAPQGYGLTETSA.`

For item 3, we used the same code above to output the sequence which is:

`Sequence 1: DENIVVGPKPFYPIEEGSAGTQLRKYMERYAKLGAIAFTNAVTGVDYSYAEYLEKSC`
`Sequence 2: LGKALQNYGLVVDGRIALCSENCEEFFIPVIAGLFIGVGVAPTNEIYTLRELVHSLGISKPTIVFSSKKGLDKVITVQKTVT`
`TIKTIVILDSKVDYRGYQCLDTFIKRNTPPGFQASSFKTVEVDRKEQVALIMNSSGSTGLPKGVQLTHENIVTRFSHARDPI`
`YGNQVSPGTAVLTVVPFHHGFGMFTTLGYLICGFRVVMLTKFDEETFLKTLQDYKCTSVILVPTLFAILNKSELLNKYDLSN`
`LVEIASGGAPLSKEVGEAVARRFNLPGVRQGYGLTETTSAIIITPEGDDKPGASGKVVPLFKAKVIDLDTKKSLGPNRRGEV`
`CVKGPMLMKGYVNNPEATKELIDEEGWLHTGDIGYYDEEKHFFIVDRLKSLIKYKGYQVPPAELESVLLQHPSIFDAGVAGV`
`PDPVAGELPGAVVVLESGKNMTEKEVMDYVASQVSNAKRLRGGVRFVDEVPKGLTGKIDGRAIREILKKPV`

Like what we did for item 2, we combined those two sequences together. The Chai-1 results are:

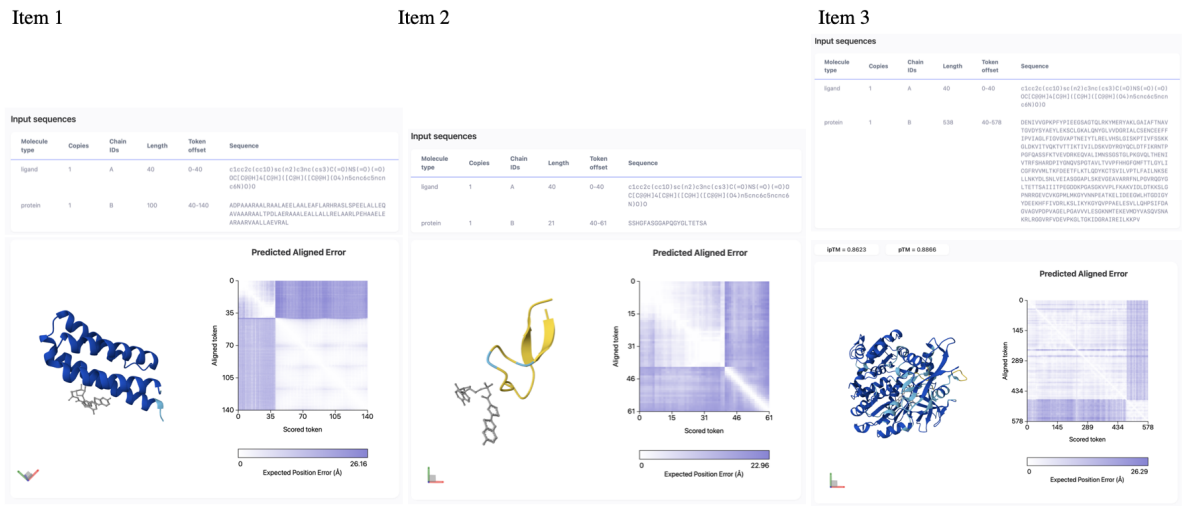

Figure 9: Results from Chai-1 on item 1, item 2 and item 3

To perform energy calculations in PyRosetta, we encountered issues caused by missing heavy atoms and unrecognized residues in the PDB files. To resolve this:

- **Residue Adjustment:** Residues labeled as `UNK` (unknown) were manually replaced with `LIG` (a placeholder), enabling PyRosetta to parse the structures without errors.
- **CONECT Removal:** All `CONECT` records were removed, as they caused parsing issues but did not affect the overall structure.
- **Structural Verification:** After adjustments, the PDB files were visualized in PyMOL, and the overall structures remained unchanged.

Following these steps, we successfully calculated the total energies for the adjusted complexes. The results are as follows:

```
Total energy of the complex1: -166.35117528912963
Total energy of the complex1: 42.545012912266344
Total energy of the complex1: -308.0231482716453
```

The total energy values for the adjusted complexes provide insights into their relative stability. Negative energies indicate favorable and stable configurations, while positive energies suggest instability. Among the complexes:

- **Complex3** has the lowest energy (-308.02), suggesting the highest stability.
- **Complex1** is also stable but less so (-166.35).
- **Complex2** has a positive energy (42.54), indicating an unfavorable or strained configuration.

The code we used is:

```
from pyrosetta import *
from pyrosetta.teaching import *

# Initialize PyRosetta (no extra options if ligand params are not needed)
init()

# Load the complex PDB file
pose1 = pose_from_pdb("ligand_protein_complex1.pdb")
pose2 = pose_from_pdb("ligand_protein_complex2.pdb")
pose3 = pose_from_pdb("ligand_protein_complex3.pdb")

# Get the standard Rosetta scoring function
scorefxn = get_fa_scorefxn()

# Calculate the total energy of the complex
total_energy1 = scorefxn(pose1)
total_energy2 = scorefxn(pose2)
total_energy3 = scorefxn(pose3)

# Print the results
print("Total energy of the complex1:", total_energy1)
print("Total energy of the complex1:", total_energy2)
print("Total energy of the complex1:", total_energy3)
```