**Python Notebook Series for Protein Structure Analysis (Practical Workflows)**

A screenshot of a computer

AI-generated content may be incorrect.

**Overview**

To complement the **protein structure analysis training**, we have prepared a structured set of Python Jupyter notebooks. These six notebooks (listed below) align with the topics identified in the training plan, focusing on practical skills like parsing structure files, exploring molecular interactions, visualizing 3D structures, and using **AlphaFold** for predictions. Each notebook is designed to be **stand-alone** – containing all necessary imports, setup steps, code cells, markdown explanations, and sample outputs (figures, printed tables, etc.). This way, trainees can run them sequentially or individually, and even refer back to them as templates for future projects.

**Environment and Tools:** All notebooks use Python 3 and open-source libraries. We recommend setting up a Conda environment or using Google Colab for easy installation of dependencies. Key packages include **Biopython** (for PDB/mmCIF parsing)[[1]](https://www.ebi.ac.uk/training/online/courses/alphafold/advanced-modeling-and-applications-of-predicted-protein-structures/customising-alphafold-structure-predictions/outputs-from-colabfold/), **py3Dmol** (for 3D visualization inside notebooks), and the **PyMOL** API (for programmatic molecular graphics). Each notebook begins with a section to install or load these requirements. For example, a notebook might start with:

!pip install biopython py3Dmol pymol-open-source

along with instructions to the user on any additional steps (PyMOL on some systems may need a conda install or an existing PyMOL executable[[1]](https://www.ebi.ac.uk/training/online/courses/alphafold/advanced-modeling-and-applications-of-predicted-protein-structures/customising-alphafold-structure-predictions/outputs-from-colabfold/)). We also use Python’s standard libraries (e.g., requests for web access) when needed. The notebooks are geared to run on a typical laptop with internet access (for fetching PDB files or using Colab for AlphaFold).

**Structure of Notebooks:** The notebooks are organized by topic and complexity, gradually building the user’s skills:

1. **Notebook 1: Retrieving and Parsing PDB/mmCIF Files**
2. **Notebook 2: Basic Structure Exploration and Analysis**
3. **Notebook 3: Visualization of Structures in Jupyter**
4. **Notebook 4: Protein–Ligand Interaction Analysis**
5. **Notebook 5: Protein–Protein Interface Analysis**
6. **Notebook 6: AlphaFold Prediction and Model Inspection**

Each notebook contains a mix of **markdown cells** (explaining the steps, theory background, and instructions), **code cells** (performing the tasks), and **output examples** (textual results or images). Plenty of comments are included in code to explain what each block is doing, so that even those with only basic Python knowledge can understand and learn. We also incorporate internal best practices from prior workshops – for instance, clear commentary and step-by-step breakdown of tasks, similar to how an earlier internal bioinformatics workshop notebook walked through a pandas data analysis example[[2]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/_layouts/15/Doc.aspx?action=edit\&mobileredirect=true\&wdorigin=Sharepoint\&DefaultItemOpen=1\&sourcedoc=%7bbe54a777-837c-4bdd-915f-99e96cb3f5aa%7d\&wd=target%28/Meetings.one/%29\&wdpartid=%7ba966a277-911a-481e-819b-99315f05dcec%7d%7b1%7d\&wdsectionfileid=%7b834961bc-3d55-4bc6-ac25-0093ace038f1%7d).

Below is a breakdown of each notebook with its focus, contents, and how it ties into the training topics:

**Notebook 1: *“Accessing Protein Structures – Retrieval & Parsing”***

**Purpose & Scope:** This first notebook introduces how to obtain protein structure files from the Protein Data Bank and parse them using Biopython. It sets the foundation by showing participants how to programmatically fetch data and read the structural content into Python. It covers both the old PDB format and the modern mmCIF format, with emphasis on the latter (since PDBx/mmCIF is now the standard[[3]](https://biopython.org/docs/latest/Tutorial/chapter_pdb.html)). This corresponds to the training’s **Data Formats and Repositories** module (PDB/mmCIF understanding and usage).

**Contents:**

* *Setup:* Import Biopython’s Bio.PDB module (particularly PDBList, MMCIFParser, and PDBParser).
* *Retrieving a Structure:* Demonstrate two methods to download a structure by its PDB ID:
  1. Using **Biopython’s PDBList** utility. For example, the code: <pre loop-creation-data="{"dataType":"Code","data":{"codeLanguage":"Python","code":"from Bio.PDB import PDBList\npdbl = PDBList()\nfilename = pdbl.retrieve*pdb*file(\"1MBN\", file*format=\"mmCif\")\n`` &quot;}}" unfurl="true" lang="en-us">from Bio.PDB import PDBList pdbl = PDBList() filename = pdbl.retrieve\_pdb\_file("1MBN", file\_format="mmCif") ¨G0G </pre><span></span></span> This loads the structure of myoglobin into the variablestructure[3](https://biopython.org/docs/latest/Tutorial/chapter\_pdb.html). We include a note thatget*structure` requires an ID and file name; the ID is just a user label (we use the PDB ID for clarity).
* *Inspecting Basic Info:* We then guide the user through the hierarchy of the structure object. Code loops through the models, chains, and residues to print a summary:
  1. How many models (for NMR ensembles, etc. Myoglobin 1MBN has 1 model).
  2. What chains exist (e.g., chain A only in myoglobin). We print something like *“Chains: A”*.
  3. Count of residues in each chain (e.g., chain A has 153 residues for myoglobin).
  4. List of any heterogroups (ligands or ions). In 1MBN, we expect to find a heme group (HEM) and an oxygen or hydroxide (OH) ligand. The code can detect non-polymers by checking residue type. We demonstrate printing: *“Heterogens: HEM, OH”*. This confirms parsing grabbed the ligand data too. Biopython makes it easy to iterate: <pre loop-creation-data="{"dataType":"Code","data":{"codeLanguage":"Python","code":"for model in structure:\n for chain in model:\n print(chain.id, \"has\", len(chain), \"residues\")\n for res in chain:\n if res.id[0] != \" \":\n print(\" HETATM:\", res.get\_resname(), res.id)\n`` &quot;}}" unfurl="true" lang="en-us">for model in structure: for chain in model: print(chain.id, "has", len(chain), "residues") for res in chain: if res.id[0] != " ": print(" HETATM:", res.get\_resname(), res.id) ¨G1G </pre><span></span></span> This would create an embedded viewer of 1MBN, showing chain A as a cartoon colored by spectrum (rainbow from N- to C-terminus) and the heme ligand as sticks, then auto-zoom. We explain each step:addModelloads the molecule (here we show using a PDB string read from file),setStyleandaddStyleapply visualization styles to subsets of the model, andshow` renders it. Because this is 3D interactive, the user can rotate the protein right in the notebook.
  5. We note that instead of reading from a local file, one can also fetch directly via URL, or even use the structure object from Biopython (though integration is easiest by writing a PDB and then loading it, or using py3Dmol’s ability to parse PDB text).
* *Views and Representations:* The notebook then guides the user through a few common visualization tasks:
  1. Show the structure in different representations: cartoon (ribbon) for overall fold, sticks for detailed sidechains, surface for shape. We demonstrate toggling these. For example, “display chain A in cartoon and color by secondary structure” or “display only ligand in ball-and-stick”.
  2. Highlight a specific region: e.g., if the protein has a ligand, we focus on the binding site. We can center the view on the heme by using zoomTo with a selection (like xyzview.zoomTo({'resn':'HEM'}) to zoom in on the heme). Additionally, we can add labels to certain atoms (py3Dmol can add labels with addLabel).
  3. Show interactions: While py3Dmol doesn’t automatically find hydrogen bonds, we can visually indicate potential interactions by showing a ligand and nearby residues in detailed style. (In Notebook 4 we will systematically find contacts, but here it’s more visual: e.g., highlight His93 and the heme iron.)
  4. Multiple models: If a structure has multiple models (like an NMR ensemble), show how to load all and perhaps animate through them or just display the first.
* *User Interaction:* Encourage the user to play with the viewer. For instance, after setting up a view, the notebook might instruct: “Try rotating the structure to see the pocket where the heme is buried.” Because this is interactive, it’s an engaging step.
* *Saving Images:* Although the focus is on interactive use, we mention that for publication-quality images one might use PyMOL or other software. However, py3Dmol views can be saved via screenshot or notebook export if needed. If possible, we include a tip: e.g., xyzview.png() can output a static PNG of the current view (depending on environment).

**Example Outputs:**

* The primary output is the **embedded 3D view**. There won’t be a lot of printed text besides headings and instructions. But for verification, we might also print a line like *“Rendering 1MBN: chain A (cartoon, rainbow colored), HEM ligand (sticks)”* so if someone runs it in a non-interactive environment they know what was intended.
* Perhaps we include a static image snapshot in the notebook as well (for the record). For example, an image of the myoglobin structure with the heme displayed, and maybe arrows or labels in a markdown cell to illustrate key parts (since a static screenshot can be annotated in the notebook markdown if needed with Markdown image syntax, but that requires pre-saving an image).

**Learning Outcomes:** By the end of Notebook 3, users can visualize any PDB structure quickly in a notebook, which is very useful for communication and inspection. They will have learned how to adjust representations and focus on regions of interest – effectively a mini crash-course in molecular visualization concepts but using code. This demystifies PyMOL a bit too, because py3Dmol’s JSON-like selection syntax is analogous to PyMOL’s selection language. It prepares them for Notebook 5, where we use PyMOL in scripting (but here they’ve already seen colored cartoons and such, so the concept carries over). Most importantly, it adds an interactive, **visual understanding** to the data they’ve been parsing, which reinforces the connection between sequences/residues and the actual 3D structure.

**Notebook 4: *“Protein–Ligand Interaction Analysis”***

**Purpose & Scope:** This is a key notebook focusing on analyzing interactions between a protein and a bound ligand (small molecule). It directly supports the training topic of **Protein–Ligand interactions**, teaching users how to identify which protein residues are near a ligand and might be interacting (via hydrogen bonds, hydrophobic contacts, etc.). We use **Biopython’s NeighborSearch** functionality to systematically find nearby atoms[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html), and then process those to get a list of neighboring residues. We also incorporate visualization of the findings – e.g., highlighting the ligand and its neighbors either with py3Dmol (as an extension of Notebook 3) or by generating a PyMOL image. This notebook is somewhat the practical culmination of earlier ones: it uses parsing (from Notebook 1), selection logic (from Notebook 2), and visualization (from Notebook 3) together for a specific analytical task.

**Contents:**

* *Selecting a Test Complex:* We choose a protein structure that has a clear ligand. We might use a different example than myoglobin here to broaden exposure – for instance **HIV protease with a bound inhibitor** (PDB ID 1HVR or similar) or a kinase with a small molecule (e.g., PDB 1E9X – CDK2 with a staurosporine inhibitor). For concreteness, say we use **1HVR**, which is HIV-1 protease dimer with a peptidomimetic inhibitor (JEI ligand). We ensure the PDB file is loaded (via Biopython as before).
* *Identifying the Ligand Molecule:* We programmatically find the ligand’s residue in the structure. Typically, the ligand can be identified because it’s a hetero-residue with a specific resname (not an amino acid). We can search for residues where res.id[0] == 'H\_' or simply exclude known polymer types. In 1HVR, the inhibitor has a known resname (e.g., “JEI”).
  + The code will locate that and store it as ligand\_residue. We print the ligand’s name, number of atoms, etc., just to confirm: *“Ligand found: JEI (36 atoms)”*.
* *Neighbor Search for Protein–Ligand Contacts:* Using Biopython’s NeighborSearch[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html):
  + First, gather all protein atoms (we can exclude the ligand’s atoms and any waters). For example:
  + from Bio.PDB import NeighborSearch
  + protein\_atoms = [atom for atom in structure.get\_atoms() if atom.get\_parent().get\_id()[0] == ' ']
  + ns = NeighborSearch(protein\_atoms)
  + ligand\_atoms = ligand\_residue.get\_atoms()
  + neighbors = set()
  + for atom in ligand\_atoms:
  + close\_atoms = ns.search(atom.get\_coord(), 5.0) # within 5 Å
  + neighbors.update(close\_atoms)

```

This finds all protein atoms within 5.0 Å of any ligand atom (5 Å is a typical cutoff to consider potential interactions). Using a set ensures we don’t double-count.

* + Then we map each atom in neighbors back to its residue (atom.get\_parent() gives the residue)[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html). We create a set of those residues. We exclude the ligand itself if somehow included (shouldn’t be, since we only searched protein atoms).
  + Now we have a set of protein residues near the ligand. We iterate through and print them in a human-readable way: e.g., *“Neighbor residues within 5Å of ligand JEI: Asp25 (Chain A), Asp25 (Chain B), Gly27 (B), Ile50 (A), Ile50 (B),…”*. In HIV protease, it’s a symmetric dimer so likely certain residues from both chains show up around the inhibitor. We include the chain and residue number. We might also sort them by distance or by name.
  + Optionally, we refine by distance or type of interaction: for example, mark which of these are within 3.5 Å (possible hydrogen bonds or salt bridges) vs those at 4–5 Å (likely hydrophobic contacts). We can do that by checking atom pairs: if any polar atom in the residue is very close to a polar atom of ligand, we tag it as a hydrogen bond candidate. But detailed hydrogen bond criteria (angle, etc.) might be too much – a simple distance cutoff is a rough proxy.
  + If the ligand or protein has metal coordination or other interactions, we’d handle similarly. (Not in this example, but e.g., a heme and histidine as in myoglobin could be analyzed similarly – indeed, our neighbor search earlier found His93 near the heme’s iron[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html).)
* *Analyzing Interactions:* For each neighboring residue, discuss what interaction it might form:
  + If it’s an acidic residue near a basic group of ligand (Asp/Glu near an amine), likely a salt bridge or H-bond.
  + If it’s a polar residue (Ser/Thr/Tyr) near ligand, likely H-bond.
  + If it’s nonpolar (Leu/Ile/Phe) around a nonpolar part of ligand, that’s hydrophobic contact.

We can automate a bit: categorize residues by polarity (using a predefined list of polar vs hydrophobic AAs) and print a summary: e.g., *“Likely hydrogen-bonding residues: Asp25 (COO⁻ group), backbone N of Gly27; Hydrophobic contacts: Ile50 side chain, Phe53 side chain…”*. It might be hard to be 100% accurate without geometric analysis, so we clearly state these are *predictions based on distance*.

This analysis part connects to theory the user learned – we actually show in data form the interactions we conceptually expect.

* *Visualization of the Binding Site:* We incorporate a visualization to complement the text output:
  + Using **py3Dmol**: we can create a focused view of just the ligand and its neighbor residues. For instance, show the ligand in sticks and the neighbor residues in sticks as well, while perhaps showing the rest of protein in a transparent surface or not at all. We can even use .setStyle to hide everything then .addStyle on the selection of neighbor residues plus ligand.
  + Using **PyMOL (scripted)**: Alternatively, we might demonstrate generating a PyMOL image for publication. For example, use PyMOL’s API (cmd.distance) to draw dashed lines between certain atoms to represent H-bonds. However, this might overlap with Notebook 5’s scope. We could keep Notebook 4’s visualization in py3Dmol for interactivity, and leave advanced PyMOL visualization to Notebook 5.
  + We ensure important contacts are visible – e.g., color the ligand distinctly and maybe color polar neighbor residues blue, nonpolar neighbors green, etc., to differentiate types. py3Dmol can color by element or chain, but we can manually set color for selections.
  + The user can rotate this binding site view to see the pocket shape.
* *Example (HIV protease)*: We explain in markdown that HIV protease’s inhibitor sits in the enzyme’s active site between two flaps, and typically the two Asp25 residues (one from each chain) are crucial (they act as acid-base catalysts and often form hydrogen bonds to the inhibitor’s scissile bond analog). If our analysis lists Asp25 from chain A and B, we highlight that as expected key interactions[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html). This cross-references the theory mention that proteases use Aspartates in the active site.

**Example Outputs:**

* Textual: A list of neighbor residues: Residues within 5Å of ligand JEI: Chain A: ASP25, THR26, GLY27, ILE50 Chain B: ASP25, GLY27, ILE50, PRO81 (Just as an example pattern – actual ones from 1HVR may differ slightly.) Then additional notes like “Asp25 (both chains) – likely H-bond or ionic; Ile50 (both chains) – hydrophobic contact from flap region; Gly27 – close contact in pocket; Pro81 – near ligand tail, hydrophobic.”
* Visual: An embedded 3D view focusing on the binding site, or a static image. If static, perhaps we include an illustrative figure with dashed lines for H-bonds. But given this is a notebook, interactive is better. The notebook can show an initial oriented view that clearly shows the ligand and labeled neighbor residues.
* If we incorporate PyMOL for a nicer image: the notebook might output “binding\_site.png” that we created via PyMOL’s cmd.png() after setting up the scene, and then display it. (However, to keep things simple, py3Dmol should suffice. We can always mention that one can refine visualization in PyMOL.)

**Learning Outcomes:** This is a pivotal exercise – by doing this, the trainee learns a practical skill of any structural biologist or chemist: how to analyze a complex to figure out what interactions stabilize it. They practice searching spatially (using a KD-tree neighbor search from Biopython which is efficient[[5]](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html)), which can be applied to other problems too (like finding clashes or metal coordination). They also tie back to chemistry fundamentals (seeing which residues are polar/nonpolar and how they interact). Having both a list and a visual representation helps them correlate numbers with the 3D structure – for instance, they might see that two aspartates on either side of the ligand are symmetrically placed (which is a known feature of HIV protease’s active site).

After Notebook 4, the user should be capable of writing a short script to identify binding site residues for any protein-ligand PDB they encounter – a very useful task in drug discovery or protein engineering contexts. They also get a template for making pretty binding site figures for presentations, which can be directly useful in their job.

**Notebook 5: *“Protein–Protein Interface Analysis”***

**Purpose & Scope:** This notebook addresses interactions in protein–protein complexes. Many principles overlap with Notebook 4, but we focus on interfaces between two protein chains or subunits rather than a small ligand. The goal is to identify interface residues, perhaps calculate the interface area (if feasible with available tools or simple approximation), and discuss the nature of protein–protein contacts. We use similar techniques (neighbor search across chains). It reinforces the training’s **Protein–Protein interaction** section with a hands-on example.

**Contents:**

* *Selecting a Complex:* We pick a PDB of a protein complex – for example, an **antibody-antigen complex** (PDB 1IGC is a classic antibody-antigen), or an **enzyme-inhibitor protein complex** (PDB 1AVX is a benchmark complex of two proteins). Another option: **Hemoglobin** (PDB 1A3N or 2HHB), which has multiple subunits and well-known interfaces (between α and β subunits). Let’s say we use the antibody-antigen for variety.
* *Isolating Chains:* We identify the two (or more) chains that form the interface of interest. In an antibody-antigen PDB, there may be multiple chains (like Heavy chain, Light chain for the antibody, and the antigen protein). We decide which interface to analyze (e.g., heavy+light vs antigen, combined as the antibody side vs antigen side). For simplicity, we can treat all antibody chains as one set and the antigen as another.
  + We create two lists: atoms\_in\_set1 and atoms\_in\_set2. For example, set1 = all atoms in chains A and B (the antibody), set2 = all atoms in chain C (the antigen).
* *Neighbor Search Across Interface:* Use NeighborSearch on one set’s atoms and search for neighbors from the other set within a cutoff (say 5 Å again):
  + Similar to Notebook 4, but now we iterate over atoms of chain A and find neighbors in chain B. Or use two searches (one for each direction) and combine results.
  + We gather residues on chain A that have any atom near chain B, and vice versa. Essentially, these are interface residues on each side.
  + We list them: e.g., *“Antibody chain (H+L) interface residues: Tyr33 (H chain), Trp52 (H), Ser95 (L), …; Antigen chain interface residues: Glu274, Lys276, …”*.
  + If we detect the same residue pairs that are neighbors, that’s expected since proximity is mutual. We ensure to capture both sides.
* *Interface Area (optional):* Calculating solvent accessible surface area (SASA) and interface area requires more complex libraries (BioPython doesn’t directly do SASA, but we might use an external tool or skip detailed area calc). Instead, we can do a simple proxy: count number of inter-chain atom pairs within 5Å as a rough measure of interface “size” or count of contacts. Or use the residue count as a measure: e.g., “~15 residues on each protein form the interface”. If we had PyMOL open, PyMOL can compute interface area via get\_area before and after splitting the complex, but doing that programmatically might be advanced. We might mention that there are tools like PISA or PDBe that can compute interface areas more rigorously (citing an internal doc that mentioned PISA).
* *Interaction Character:* We analyze what types of residues are at the interface:
  + Are they mostly hydrophobic? (Often protein interfaces have a hydrophobic core).
  + Are there salt bridges? (Often a few charged residues at the rim of the interface). We can tally the interface residues by type (similar to composition analysis from Notebook 2) to see enrichment. For an antibody-antigen, one expects some hotspots (like maybe a Tyrosine or Tryptophan often provides a lot of contact area). We highlight any notable ones: e.g., if there’s a pair of oppositely charged residues facing each other across the interface, we note the potential salt bridge.
* *Visualization of Interface:*
  + We show the two proteins and highlight the interface residues (perhaps color them red). Using py3Dmol again: display both chains in surface representation, color interface residues differently (if we can select them by residue number).
  + Alternatively, display one chain in surface and the other in cartoon to illustrate how one fits into the other.
  + If using PyMOL: We could load the complex into PyMOL via script and use PyMOL’s show surface, chain X and color red, (chain X and resi 50 51 52 ... ).
  + Perhaps an easier approach: simply shrink the view to just the interface residues – but that might be less informative because context is useful. Better to show the whole complex but emphasize interface region.
  + If applicable, show hydrogen bonds across the interface using dashed lines (but that may require identifying them first, which we can do by checking distances between donor-acceptor pairs similarly to ligand H-bond check).
* *Example (Antibody-Antigen):* The notebook might point out something like: “The antibody-antigen interface is dominated by a set of tyrosine and tryptophan residues on the antibody that insert into a pocket on the antigen. This is typical as aromatic residues often contribute to binding energy (a known concept of antibody paratopes). Also, note the salt bridge between LysX of antigen and AspY of antibody at the periphery.” We tie this to theory from the training about hotspots and the nature of protein interfaces.

**Example Outputs:**

* A list of interface residues for each chain, possibly formatted for clarity: Interface residues on Chain A (antibody): Y33, W52, N54, Y102, … Interface residues on Chain C (antigen): E272, F274, K276, Y278, … Approx. interface contact count: 110 atom-atom contacts within 5Å.
* If we attempt an area calc via an external script (not guaranteed), we might output: *“Estimated interface area ~1200 Å²”*, but only if we can justify it. Perhaps we skip numeric area to avoid unverified info.
* Visual: a 3D view of the complex with interface highlighted, or separate views of each partner’s interface surface. Could even do a “split” view: show each protein’s interface surface after separating them (like two halves facing each other) – though that’s complex to implement in the notebook, it is conceptually interesting (we might just discuss it instead of actually doing it).
* Additional info: The notebook might also output the number of hydrogen bonds or salt bridges if we check those specifically (like “3 putative hydrogen bonds across the interface (within 3.5 Å between polar atoms)”).

**Learning Outcomes:** After this, the user can analyze protein-protein interfaces similarly to how they did protein-ligand. They see the commonalities: spatial neighbor search works for any interacting partners. They also appreciate differences: protein interfaces are larger and often hydrophobic, so the output might be a larger set of residues and a pattern (lots of nonpolars clustered). This gives them the ability to study things like oligomerization interfaces or protein complex binding sites in their own work. Additionally, by doing both ligand and protein interfaces, they better grasp the concept of interaction “footprints” – e.g., a small ligand touches only a pocket of maybe 10 residues, whereas a protein can touch a broad surface on another protein. They can use these notebooks to, say, analyze a mutation’s effect: if a mutated residue appears in the interface list, that indicates it might disrupt binding.

**Notebook 6: *“AlphaFold Structure Prediction & Confidence Analysis”***

**Purpose & Scope:** The final notebook is about using **AlphaFold** to predict a protein structure (if an experimental one is not available) and analyzing the confidence metrics of that prediction. This addresses the training topic of **working with AlphaFold predictions**. Since running the full AlphaFold pipeline requires heavy computation and specialized setup, we guide the user to use **AlphaFold Colab** (an interactive Google Colab provided by DeepMind or the ColabFold by the community) for obtaining a model, and then use our notebook to examine the result. The notebook, thus, is partly instructive (telling the user how to get a model from Colab) and partly analytical (what to do with the model after obtaining it). We also incorporate the concept of pLDDT (per-residue confidence) and how to interpret it, which was covered in theory.

**Contents:**

* *Colab Notebook Instructions:* We provide step-by-step instructions (with screenshots or descriptions) for using the official **AlphaFold Colab** notebook :
  1. Open the Colab URL (we’d provide the link, e.g., to AlphaFold2 or ColabFold notebook).
  2. Input the amino acid sequence of the protein of interest (the user should have this prepared, possibly from UniProt or from a FASTA file).
  3. Run the Colab cells to generate the prediction. We note that it may take 5-20 minutes depending on the protein length and Colab’s resources.
  4. Download the resulting PDB file (AlphaFold usually outputs one or several ranked models and some plots). We make it clear this part requires internet and Colab (outside of our Jupyter environment) – effectively telling the user “go do this, then come back with the model file.” We advise using **MMseqs2 option** in ColabFold for better predictions, etc., but keep it simple.
* *Using AlphaFold Database:* As an alternative to running a new prediction, we mention the **AlphaFold Protein Structure Database** with 200+ million precomputed models. If the user’s protein is in that database (which it likely is if it’s a known sequence from a known organism), they can simply download it:
  1. We show how to find a model by UniProt ID or gene name on the AlphaFold DB website, or even programmatically fetch it. For example, download link construction: https://alphafold.ebi.ac.uk/files/AF-<UniProtID>-F1-model\_v4.pdb (AlphaFold DB provides PDBs named with the UniProt accession).
  2. We provide a code snippet using requests to fetch a model. E.g., for human insulin (UniProt P01308):
  3. af\_id = "P01308"
  4. url = f"https://alphafold.ebi.ac.uk/files/AF-{af\_id}-F1-model\_v4.pdb"
  5. resp = requests.get(url)
  6. with open("P01308\_AF.pdb", "w") as f:
  7. f.write(resp.text)

This way, even if user cannot run Colab, they can get a model for many proteins in seconds.

* 1. We then load the downloaded model with Biopython’s parser as before.
* *Analyzing the Predicted Model:* Now that we have a predicted structure (from either method), we focus on **confidence metrics**:
  1. We explain that AlphaFold’s per-residue confidence is given by pLDDT scores, which are stored in the **B-factor field** of the PDB file. Using Biopython, we can access each atom’s bfactor (atom.bfactor) and thus each residue’s average pLDDT.
  2. We iterate over residues and extract pLDDT. Perhaps create a list of (residue number, pLDDT). We then plot this: a simple line plot of pLDDT vs residue index. This reproduces the kind of confidence plot AlphaFold itself provides[[1][1]](https://www.ebi.ac.uk/training/online/courses/alphafold/advanced-modeling-and-applications-of-predicted-protein-structures/customising-alphafold-structure-predictions/outputs-from-colabfold/).
     + The plot will highlight regions below 50 (very low confidence, presumably unstructured loops) and above 90 (very high confidence). We add horizontal lines at pLDDT=50 and 90 for reference.
     + If certain regions have low confidence, we mark them. For example, “Residues 120-135 have pLDDT ~40, indicating a likely disordered loop or uncertain structure.”
  3. We also parse the AlphaFold output for any *PAE (Predicted Alignment Error)* if it’s available. The Colab usually gives a PAE heatmap image; we might just mention that and encourage the user to look at it (the ColabFold output includes a \_PAE.png).
  4. If using ColabFold, multiple models (rank1, rank2, etc.) might be produced. We can compare their pLDDT distributions or just focus on the top-ranked model.
* *Structural Features:* We encourage examining the predicted model structurally:
  1. Use py3Dmol to visualize it, perhaps coloring by pLDDT. py3Dmol can color by b-factor: since pLDDT is in b-factor, we can do:
  2. view = py3Dmol.view()
  3. view.addModel(open('model.pdb','r').read(),'pdb')
  4. view.setStyle({'cartoon':{}})
  5. # Color by b-factor (pLDDT): say blue for high, red for low
  6. view.setColorByProperty({'model':0}, 'b')
  7. view.show()

If setColorByProperty with 'b' works, it will color structure from blue (low B = high confidence) to red (high B = low confidence) as per typical conventions. We verify this if possible. Otherwise, we can manually set color: iterate residues, and use addStyle with specific color for ranges of bfactor (like if bfactor <50, color red, 50-90 yellow, >90 blue). The visual will clearly show flexible loops in red, confident core in blue.

* 1. If an experimental structure exists for comparison (in some cases, we could compare AlphaFold prediction to a PDB structure if available): We might demonstrate aligning the predicted model to a known PDB structure of the same protein to see accuracy. However, that requires having a pair, and if the protein had a known structure we might not be predicting it. Perhaps we pick an example where we do have both, to show AlphaFold’s reliability. For instance, alphaFold model vs actual for a small protein (many AF papers show e.g. comparing AF to real structure of some enzyme).
     + Use Superimposer to align predicted model to real structure and compute RMSD.
     + Show that secondary structure aligns well if pLDDT was high.
     + Where they differ, check if that region had low pLDDT (likely yes).
     + This reinforces trusting high-confidence parts and being cautious with low-confidence ones.
  2. If no known structure, we at least emphasize how to use the model: focus on the high confidence regions for any conclusions, and treat low confidence parts as “model could be wrong here or the region is disordered in reality”.
* *AlphaFold Multimer (FYI):* If relevant (like predicting a complex or multimeric structure), mention that AlphaFold has a multimer mode (AlphaFold-Multimer) that can predict protein complexes (AlphaFold DB also has some complexes, or one can use ColabFold for that). If the user’s interest extends to predicting an interface (like two protein binding), they could use ColabFold’s complex prediction by inputting both sequences. We won’t dive deep, just note it exists, since our training mostly covered monomeric prediction.

**Example Outputs:**

* A pLDDT plot graphic with a caption like *“Residue confidence plot: Most of the protein is predicted with high confidence (pLDDT > 90), except two loops around residues 50-60 and 120-130 which are lower confidence.”*
* Some printed statistics: e.g., *“Average pLDDT = 88.7. Residues with pLDDT < 50: 54-60, 121-130”*.
* If structure alignment done: *“AlphaFold model vs PDB X-ray structure RMSD = 1.2 Å over 200 Cα atoms, with main differences in the N-terminal tail (model is disordered for residues 1-5, which had low pLDDT).”* This illustrates excellent accuracy except in low-confidence regions – a teaching point.
* The 3D view colored by confidence (which in AlphaFold’s scheme is usually blue = very confident, green = confident, yellow = low, orange/red = very low). We mimic that color scale in py3Dmol. The output is interactive: one can zoom into a red region to see which part of the protein it is (maybe a tail or a flexible loop).

**Learning Outcomes:** The user learns how to get structural models for proteins lacking them, using state-of-the-art AI. They also learn **critical evaluation** of predicted models: not all parts of a model are equal, and the notebook shows how to pinpoint which parts are reliable. This nurtures a habit of not blindly trusting predictions, aligning well with the theoretical caution taught (AlphaFold gives confidence scores – use them!). Additionally, they get exposure to analyzing a structure similarly regardless of source – once the model is loaded, they can apply previous notebooks’ techniques to it (e.g., if they predicted a enzyme structure, they could then run a ligand docking or use Notebook 4-style analysis on it if they have a guess for a ligand position, etc.). We also indirectly introduce them to using external computational notebooks (Colab), broadening their toolset.

**Notebook Organization and Delivery**

We will organize these notebooks in a logical sequence, naming them clearly:

* 01\_Retrieve\_and\_Parse\_Structures.ipynb
* 02\_Structure\_Exploration.ipynb
* 03\_Visualization\_in\_Notebook.ipynb
* 04\_Protein\_Ligand\_Interactions.ipynb
* 05\_Protein\_Protein\_Interface.ipynb
* 06\_AlphaFold\_Prediction\_Analysis.ipynb

Each notebook starts with a markdown header explaining its goals and prerequisites (if any, e.g., “Notebook 6 requires internet access to use AlphaFold Colab”). They will be stored in an accessible location (e.g., a shared folder on our internal training SharePoint or GitHub if appropriate) and cross-referenced. For instance, Notebook 4 might say “Make sure you have run Notebook 1 to understand how to load structures”.

We have also looked into any **internal templates or existing notebooks** that we can adapt:

* The internal *Bioinf Workshop* notes show an emphasis on providing notebooks for hands-on practice[[2]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/_layouts/15/Doc.aspx?action=edit\&mobileredirect=true\&wdorigin=Sharepoint\&DefaultItemOpen=1\&sourcedoc=%7bbe54a777-837c-4bdd-915f-99e96cb3f5aa%7d\&wd=target%28/Meetings.one/%29\&wdpartid=%7ba966a277-911a-481e-819b-99315f05dcec%7d%7b1%7d\&wdsectionfileid=%7b834961bc-3d55-4bc6-ac25-0093ace038f1%7d). We carry that philosophy here. If any internal code was available for, say, neighbor searching or visualization, we would integrate it, but most of our code is custom written for this purpose. We did integrate internal knowledge on Pandas usage style (lots of comments, stepwise approach) from those notes[[2]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/_layouts/15/Doc.aspx?action=edit\&mobileredirect=true\&wdorigin=Sharepoint\&DefaultItemOpen=1\&sourcedoc=%7bbe54a777-837c-4bdd-915f-99e96cb3f5aa%7d\&wd=target%28/Meetings.one/%29\&wdpartid=%7ba966a277-911a-481e-819b-99315f05dcec%7d%7b1%7d\&wdsectionfileid=%7b834961bc-3d55-4bc6-ac25-0093ace038f1%7d).
* If there are existing internal notebooks on structural analysis (perhaps from other teams), we would review them. For example, an internal project maybe used Biopython to analyze PDBs for something – if found, we’d borrow useful snippets. (Our searches didn’t surface a specific one, so presumably this is new development.)
* We ensured that all notebooks follow a consistent style for ease of use: clear section headings, code separated by concept, and summary text after each major result (for pedagogical reinforcement).

In conclusion, by completing these six notebooks, trainees will gain **practical experience in all key areas of protein structure analysis**:

* Reading and interpreting structural data,
* Visualizing structures and binding sites,
* Identifying molecular interactions in complexes,
* Utilizing modern AI tools for structure prediction and critically evaluating their output.

This comprehensive set can serve as a reusable toolkit for the trainees’ future work (they can plug in their protein of interest into these notebooks to perform similar analyses). The structured approach, from basic to advanced, ensures that even a beginner can start at Notebook 1 and progressively build skills to tackle the more advanced Notebook 6. All notebooks are ready for immediate use in the training sessions, and we’ll encourage participants to experiment with them (e.g., as exercises, they can modify the code to analyze a structure of their choice, with our guidance).

Moreover, the combination of textual summary and code results within the notebooks will make them a self-documenting resource – after the training, someone could read through the notebooks and understand the workflow without needing separate lecture notes. This dual nature of the notebooks – as both learning exercises and reference material – maximizes their value for the organization.

**References**

[1] [Outputs from ColabFold | AlphaFold - EMBL-EBI](https://www.ebi.ac.uk/training/online/courses/alphafold/advanced-modeling-and-applications-of-predicted-protein-structures/customising-alphafold-structure-predictions/outputs-from-colabfold/)

[2] [Bioinf Workshop](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/_layouts/15/Doc.aspx?action=edit&mobileredirect=true&wdorigin=Sharepoint&DefaultItemOpen=1&sourcedoc=%7bbe54a777-837c-4bdd-915f-99e96cb3f5aa%7d&wd=target(/Meetings.one/)&wdpartid=%7ba966a277-911a-481e-819b-99315f05dcec%7d%7b1%7d&wdsectionfileid=%7b834961bc-3d55-4bc6-ac25-0093ace038f1%7d)

[3] [Going 3D: The PDB module — Biopython 1.85 documentation](https://biopython.org/docs/latest/Tutorial/chapter_pdb.html)

[4] [Bio.PDB.PDBList module — Biopython 1.85 documentation](https://biopython.org/docs/latest/api/Bio.PDB.PDBList.html)

[5] [Analyzing MMCIF Files using Biopython - MolSSI Education](https://education.molssi.org/python-scripting-biochemistry/chapters/biopython_mmcif.html)