RF3 for Antibody/Antigen Structure Prediction

Worked examples for **RF3** and **AtomWorks**, walking through a small vaccine and antibody discovery pipeline end to end.

A short story (your week with RF3)

You're a structural biologist who just joined a small vaccine and antibody discovery group. Over this tutorial you will walk through the modeling end of a full pipeline:

- **Monday morning.** Warm up on the structure I/O tools (AtomWorks).
- **Monday afternoon.** Design an immunogen candidate: the SARS-CoV-2 RBD (6M0J).
- **Tuesday.** Model glycans on the immunogen and see how the surface gets masked.
- **Wednesday.** First antibody back from immunization: a single-chain nanobody (3OGO). Model it from sequence and grade RF3 against the crystal.
- **Thursday.** A collaborator sends over the Trastuzumab Fab (1N8Z) so you can practice paired-chain folding.
- **Friday morning.** The whole point: dock the SARS-CoV cross-reactive antibody CR3022 onto the RBD (6W41). Antibody/antigen docking from sequence is notoriously hard. Are the confidence metrics warning you?
- **Friday afternoon.** Use the experimental RBD template to fix the antigen and let RF3 focus on the docking.

By the end you will have run six RF3 jobs, inspected their confidences, and built a feel for when to trust the model and when to bring in extra information.

RF3 Overview

RF3 exhibits strong performance at antibody-antigen structure prediction, as validated by the external FoldBench benchmark:

Model	Protein-Protein	Antibody-Antigen	Protein-Ligand
RosettaFold3*	72.44%	37.50%	57.28%
AlphaFold 3	70.87%	47.95%	67.59%
Boltz-2*	70.54%	25.00%	53.90%
OpenFold 3 (preview)	68.22%	34.29%	40.85%
Chai-1	66.95%	18.31%	49.28%
HelixFold 3	66.67%	28.17%	50.68%
Protenix	64.80%	38.36%	53.25%
Boltz-1	64.10%	31.43%	51.33%

Tutorial Overview

Section	Topic
0	Setup and environment
1	AtomWorks: loading and inspecting structures
2	SARS-CoV-2 RBD monomer prediction (immunogen)
3	Adding glycans to the RBD
4	Nanobody (VHH) prediction
5	Paired antibody (Fab) prediction
6	Antibody/antigen complex prediction
7	Template-guided prediction
8	Tips and practical guidance

Requirements: Python 3.12, CUDA GPU, `rc-foundry[all]`, `atomworks[ml]`

Section 0: Setup and Environment

Installation

If you are working on Vanderbilt molgraph machines you can activate the existing environment:

```
conda activate foundry  # unlikely to work on non-Vanderbilt machines

# Get to the tutorial directory
cd /usr/people/workshop/rosetta_workshop/tutorials/rf3-tutorial
```

If you are doing the tutorial from a different machine you should create a virtual environment. This may take a few minutes.

```
conda create -n foundry python=3.12 -y
conda activate foundry
pip install "rc-foundry[all]" "atomworks[ml]" py3Dmol matplotlib pandas biotite
foundry install rf3  # download RF3 weights (~2 GB)
```

This will also work with `uv` *and* `pip` *if your university SLURM system does not allow Anaconda.*

If you have a Mac with an M1/M2/M3/M4 chip you can still use RF3 on your GPU, but you have to follow the instructions in the Foundry README.

If your machine does not have a GPU that will work with RF3, but your university has a SLURM-based compute cluster, you can reserve a GPU with `salloc`. For example during the workshop we will use:

```
ssh login.accre.vu
salloc --partition=batch_gpu --account=p_meiler_acc --gres=gpu:nvidia_l40s:1 --ntasks=6 --time=2
```

And then to send the workshop files there, run this from the terminal of your local computer:

```
rsync rf3-tutorial/ USERNAME@login.accre.vu:~/
```

Project layout

The tutorial folder we provide looks like this. Everything except this tutorial file lives in `input_files/`:

```
rf3-tutorial/
├── input_files/
│   ├── jsons/              # RF3 input configs (.json)
│   ├── msas/               # Pre-computed MSAs (.a3m)
│   ├── structs/            # Pre-downloaded reference structures (.cif)
│   ├── seqs/               # Fasta files for the MSA step
│   └── scripts/            # All Python scripts (helpers.py, generate_msa.py,
│                           #   download_cif.py, inspect_structure.py, etc.)
├── output_files/           # Cached example predictions (fallback if anything fails)
└── rf3_antibody_tutorial.md
```

From the project root, create your working directory and `cd` into it:

```
mkdir -p working_dir && cd working_dir
```

All bash and python commands in the rest of the tutorial assume you are inside **`working_dir/`**. Every script in `../input_files/scripts/` is written to import `helpers.py` from its own directory, so you can always invoke them from `working_dir/` like:

```
python ../input_files/scripts/<script_name>.py [args]
```

Quick GPU check

Before running anything that costs compute, confirm RF3 can see a GPU:

```
python -c "import torch; print('GPU available:', torch.cuda.is_available()); \
print('GPU:', torch.cuda.get_device_name(0)) if torch.cuda.is_available() else None"
```

You should see `GPU available: True` and a device name (an L40S, H100, or H200 in our case).

Two reusable commands you'll see throughout

Two helper scripts in `../input_files/scripts/` show up repeatedly:

```
# Download any PDB structure as a CIF into your working directory
python ../input_files/scripts/download_cif.py PDB_ID [MORE_IDs...]

# Generate an MSA from a fasta file (writes .a3m to working_dir/)
python ../input_files/scripts/generate_msa.py path/to/sequence.fasta
```

Both have pre-computed fallbacks under `../input_files/structs/` and `../input_files/msas/` if a server is slow.

Section 1: Meet your tools (AtomWorks)

Before predicting anything, get comfortable loading a structure. **AtomWorks** (built on Biotite) is the data layer underneath RF3, and you will use it both to prepare templates and to evaluate predictions.

You will work with PDB 3OGO, an anti-GFP nanobody (a single-chain antibody from a camelid, ~120 residues). It's a tidy starting example since the whole binding apparatus lives in one short chain. You will come back to this exact protein in Section 4 when you predict it from sequence.

1.1 Download the crystal structure

```
python ../input_files/scripts/download_cif.py 3OGO
```

This writes `3OGO.cif` into your working directory. A pre-downloaded copy is also at `../input_files/structs/3OGO.cif`

1.2 Inspect the file

```
python ../input_files/scripts/inspect_structure.py 3OGO.cif
```

This prints the chains in the file, the number of residues per chain, and the one-letter sequence of each chain. 3OGO contains the nanobody bound to GFP, so you'll see more than one protein chain. The nanobody is chain E.

1.3 Render it in 3D

```
python ../input_files/scripts/visualize_structure.py 3OGO.cif \
    --chain E --out 3ogo_nanobody.html
```

Open `3ogo_nanobody.html` in your browser and rotate the nanobody around. The immunoglobulin fold is clear: two beta sheets sandwiched together. The three CDR loops at one end are what we care about for binding.

If you prefer PyMOL or ChimeraX, the same CIF works directly: `pymol 3OGO.cif`.

Section 2: Predict the SARS-CoV-2 RBD as an immunogen candidate

Early in the pandemic the full SARS-CoV-2 spike was solved (6VSB), but the receptor binding domain was only partially resolved. Many groups theorized that an immunogen focused on the RBD alone could be very effective at eliciting antibodies that block ACE2 binding.

This is exactly the kind of structure prediction you would do to design or sanity-check a vaccine immunogen.

2.1 Generate an MSA

The RBD sequence lives in `../input_files/seqs/sars2_rbd_wt.fasta`. If you want to see what it looks like:

```
cat ../input_files/seqs/sars2_rbd_wt.fasta
```

You should see a single ~190-residue sequence beginning with `TNLCPFGE`.

Generate the MSA via the ColabFold MMseqs2 server:

```
python ../input_files/scripts/generate_msa.py ../input_files/seqs/sars2_rbd_wt.fasta
```

This writes `sars2_rbd_wt.a3m` into your working directory. A pre-computed copy is at `../input_files/msas/sars2_rb` the JSON we'll use in 2.2 points at that path so the workshop run is reproducible.

2.2 Run RF3

Take a quick look at the input config:

```
cat ../input_files/jsons/sars2_rbd_wt.json
```

The main components:

```
"name": "sars2_rbd_aa",
"components": [
  {
    "seq": "TNLCPFGE...",
    "msa_path": "../input_files/msas/sars2_rbd_wt.a3m",
    "chain_id": "R"
  }
]
```

Fold:

```
rf3 fold inputs='../input_files/jsons/sars2_rbd_wt.json' \
    n_recycles=3 diffusion_batch_size=2 num_steps=50 out_dir="."
```

2.3 Check the confidence

Open `sars2_rbd_aa/seed-0_sample-0/sars2_rbd_aa_seed-0_sample-0_summary_confidences.json` and look at `overall_plddt` and `ptm`. For a monomer, both above 0.7 means the model is likely accurate.

```
cat sars2_rbd_aa/seed-0_sample-0/sars2_rbd_aa_seed-0_sample-0_summary_confidences.json
```

2.4 Visualize colored by pLDDT

```
python ../input_files/scripts/visualize_structure.py \
    sars2_rbd_aa/sars2_rbd_aa_model.cif --plddt --out rbd_plddt.html
```

Open the HTML and inspect. From low to high confidence the colors go red, orange, yellow, green, blue in a gradient. The core of the RBD is blue which means very high confidence. The flexible RBM (receptor binding motif, the tip that contacts ACE2) is often the lowest-confidence patch at more of a green/cyan color which is still high enough confidence to trust.

2.5 Compare to crystal

The RBD has been solved many times. We'll compare against 6M0J chain B, which is the same RBD in complex with ACE2. Download it (we'll reuse this CIF as a template in Section 7):

```
python ../input_files/scripts/download_cif.py 6M0J
```

Compare prediction to experimental:

```
python ../input_files/scripts/compare_structures.py \
    sars2_rbd_aa/sars2_rbd_aa_model.cif 6M0J.cif \
    --pred-chain R --ref-chain B --out rbd_overlay.html
```

You should see a Cα RMSD around 1 to 2 Å, which is quite good. Open `rbd_overlay.html` to see the prediction (blue) on the crystal (orange).

Caveat: 6M0J is in RF3's training set, so this comparison is on the optimistic end of what to expect on a fully novel target.

- No glycans were predicted. For real immunogen design you care about which surface is shielded, which is the point of Section 3.

Section 3: Add glycans to the RBD

The native RBD is glycosylated, and glycans matter for immunogen design because they can shield surfaces from antibody recognition. Here we'll model the same RBD with two different glycan types and look at which residues end up masked.

3.1 Predict where glycans go

Use the NetNGlyc server, or find N-X-(S/T) motifs by hand on the reference structure (a surface-exposed asparagine, followed by a non-proline residue, followed by serine or threonine).

3.2 Choose which glycan to place

Ideally you would have experimental glycoproteomics data from this virus or a closely related one. Without that, modeling a high-mannose glycan is a reasonable default. To check which surfaces our immunogen exposes, we will model **both** a high-mannose Man5 glycan and a biantennary complex glycan. RF3 can batch both jobs in a single call.

Glycan 1: biantennary complex — NAG-NAG-BMA-(MAN-NAG-GAL-SIA)2

```
Asn ── NAG ── NAG ── BMA
                     ├── α1,3 ── MAN ── NAG ── GAL ── SIA
                     └── α1,6 ── MAN ── NAG ── GAL ── SIA
```

Glycan 2: high-mannose Man5 — NAG-NAG-BMA-(MAN)3

```
Asn ── NAG ── NAG ── BMA
                     ├── α1,3 ── MAN ── MAN
                     └── α1,6 ── MAN
```

Both JSON configs already point at the same `sars2_rbd_wt.a3m` MSA from Section 2.

3.3 Run RF3 (batched)

```
rf3 fold \
    inputs='[../input_files/jsons/sars2_rbd_complex_glyc.json, ../input_files/jsons/sars2_rbd_man
    n_recycles=3 diffusion_batch_size=2 num_steps=50 out_dir="."
```

3.4 Visualize the glycoproteins

```
python ../input_files/scripts/visualize_structure.py \
    sars2_rbd_complex_glyc/sars2_rbd_complex_glyc_model.cif \
    --plddt --out rbd_complex_glyc.html

python ../input_files/scripts/visualize_structure.py \
    sars2_rbd_man5_glyc/sars2_rbd_man5_glyc_model.cif \
    --plddt --out rbd_man5_glyc.html
```

The visualization script renders glycan residues (NAG, BMA, MAN, FUC, SIA, GAL, etc.) as purple sticks layered on top of the protein cartoon, so they pop out visually.

Open both HTML files and note the residues around the glycan attachment site that would be at least intermittently shielded from antibodies. Glycans are flexible in vivo, so a single snapshot is just one pose from a wide conformational ensemble. For a serious surface-accessibility analysis you would run several seeds and average exposure across the predicted poses.

Section 4: Predict the anti-GFP nanobody from sequence

You've designed an immunogen. After immunization in a llama, the first antibody back is a single-chain VHH nanobody. Goal: predict it from sequence alone and grade RF3 against the crystal (3OGO from Section 1).

4.1 Generate an MSA

This time, let's assume you were not given an existing fasta file and you need to create one yourself. But what is a fasta file?

A fasta file consists of 2 things:

1. A header which starts with ">" and is followed by a name (e.g. >nanobody_vhh). For our specific MSA generation script the header has the same name as the output MSA file.
2. The amino acid sequence spread across as many lines as you want. Try to avoid spaces and extraneous characters.

To make the header line run:

```
echo ">nanobody_vhh" > nanobody_vhh.fasta
```

To make the sequence line run:

```
python ../input_files/scripts/inspect_structure.py 3OGO.cif
```

and copy the amino acid sequence for chain E. Then add it to the file with:

```
echo "QVQLVESGGALVQPGGSLRLSCAASGFPVNRYSMRWYRQAPGKEREWVAGMSSAGDRSSY" >> nanobody_vhh.fasta
echo "EDSVKGRFTISRDDARNTVYLQMNSLKPEDTAVYYCNVNVGFEYWGQGTQVTVSS" >> nanobody_vhh.fasta
```

Finally, create the MSA with:

```
python ../input_files/scripts/generate_msa.py nanobody_vhh.fasta
```

Pre-computed fallbacks at `../input_files/seqs/nanobody_vhh.fasta` and `../input_files/msas/nanobody_vhh.a`

4.2 Inspect the RF3 input JSON

```
cat ../input_files/jsons/nanobody_vhh.json
```

Key fields:

- `name`: identifier for this prediction; becomes the output folder name
- `components`: each chain has its own entry
  - `seq`: amino acid sequence
  - `chain_id`: chain label in the output structure
  - `msa_path`: path to the .a3m

4.3 Run RF3

```
rf3 fold inputs='../input_files/jsons/nanobody_vhh.json' \
    n_recycles=3 diffusion_batch_size=2 num_steps=50 out_dir="."
```

This takes about 1 minute on an L40S. RF3 writes the predicted structure and confidence files to `./nanobody_vhh/seed-0_sample-0/`.

4.4 Visualize colored by pLDDT

```
python ../input_files/scripts/visualize_structure.py \
    nanobody_vhh/nanobody_vhh_model.cif --plddt --out nanobody_pred_plddt.html
```

Lower confidence concentrates in the CDR loops, especially CDR3.

4.5 Compare to the crystal

The 3OGO CIF is already in your working directory from Section 1 (otherwise re-run `python ../input_files/scripts/download_cif.py 3OGO`).

```
python ../input_files/scripts/analyze_nanobody.py
```

This script:

1. Loads your prediction from `./nanobody_vhh/`
2. Loads the 3OGO crystal (chain E) from `../input_files/structs/3OGO.cif`
3. Computes Cα RMSD after superposition
4. Plots per-residue Cα distance with CDR regions highlighted (`nanobody_per_residue.png`)
5. Saves an overlay HTML (`nanobody_overlay.html`)

You should see Cα RMSD comfortably under 1 Å, with most of the deviation concentrated in CDR3 and the first and last few residues, which are very flexible. That is the typical RF3 fingerprint: framework regions are nearly perfect, CDR loops carry most of the uncertainty.

Caveat: 3OGO is in RF3's training set, so this particular prediction is biased upward. Expect lower confidence and slightly higher deviation on novel targets.

Section 5: Predict a paired Fab (Trastuzumab)

Goal: Practice multi-chain antibody folding on a well-characterized therapeutic, the **Trastuzumab Fab** (Herceptin, anti-HER2). Antibodies have 2 chains (VH and VL) and so require two MSAs. We will then compare to an experimental structure of it, 1N8Z.

5.1 Generate the two MSAs

```
python ../input_files/scripts/generate_msa.py ../input_files/seqs/trastuzumab_vh.fasta
python ../input_files/scripts/generate_msa.py ../input_files/seqs/trastuzumab_vl.fasta
```

Pre-computed fallbacks at `../input_files/msas/trastuzumab_vh.a3m` and `trastuzumab_vl.a3m`.

5.2 Download the reference crystal

```
python ../input_files/scripts/download_cif.py 1N8Z
```

5.3 Inspect the multi-chain JSON

```
cat ../input_files/jsons/trastuzumab_fab.json
```

You'll see two entries in `components`, one per chain, each with its own `msa_path`. There is no special "this is the heavy chain" field. RF3 just gets two chains plus their MSAs and figures out the pairing geometry from the immunoglobulin fold.

5.4 Run RF3

```
rf3 fold inputs='../input_files/jsons/trastuzumab_fab.json' \
    n_recycles=3 diffusion_batch_size=2 num_steps=50 out_dir="."
```

5.5 Analyze the prediction

```
python ../input_files/scripts/analyze_fab.py
```

The script prints per-chain Cα RMSD versus the 1N8Z crystal, overall pLDDT, pTM, iPTM, and the VH-VL min iPAE. It also saves:

- `fab_pred_plddt.html`: prediction colored by pLDDT
- `fab_pae.png`: PAE heatmap with chain boundaries marked

5.6 Interpreting the PAE heatmap

The PAE matrix shows RF3's confidence in the **relative positions** of every pair of residues. For multi-chain predictions, the **off-diagonal blocks** are where you look: low PAE between chains means RF3 is confident about how those chains sit relative to each other.

For Trastuzumab you should see:

- Crisp low-error blocks along the diagonal (each chain folds confidently on its own)
- A noticeably dark off-diagonal **VH/VL block**: RF3 is confident about the pairing

That second point is the key one. VH/VL pairing is an evolutionarily conserved interface, so RF3 reliably gets it right. That confidence does **not** transfer to antibody/antigen interfaces, as you are about to see.

For a Fab you see an interesting phenomenon different from what you see when modeling just the variable (Fv) region of an antibody. Because there is a flexible hinge separating the heavy or light chain into 2 domains you see a "checkerboard pattern" where within each domain there is high confidence but between the Fv and the constant region there is lower confidence.

Section 6: Predict an antibody/antigen complex (the hard part)

Goal: Predict **CR3022** (a SARS-CoV cross-reactive antibody) bound to the **SARS-CoV-2 RBD**. We will compare to the experimental structure PDB 6W41.

Antibody/antigen docking from MSAs alone is notoriously difficult. There is no strong evolutionary covariance signal between an antibody and its target antigen (they did not co-evolve over millions of years). RF3 may fold each chain correctly but place them wrong relative to each other.

To decrease computational complexity we crop the antibody to its Fv region (the heavy and light variable domains only). This is the only part of the antibody that engages the antigen, and it folds well on its own.

6.1 Generate the antibody MSAs

```
python ../input_files/scripts/generate_msa.py ../input_files/seqs/cr3022_vh.fasta
python ../input_files/scripts/generate_msa.py ../input_files/seqs/cr3022_vl.fasta
```

Pre-computed fallbacks at `../input_files/msas/cr3022_vh.a3m` and `cr3022_vl.a3m`. The antigen MSA is the same `sars2_rbd_wt.a3m` you generated back in Section 2.

6.2 Download the reference crystal

```
python ../input_files/scripts/download_cif.py 6W41
```

6.3 Run the folding job

```
rf3 fold inputs='../input_files/jsons/cr3022_rbd_complex.json' \
    n_recycles=10 diffusion_batch_size=5 num_steps=200 out_dir="."
```

You can set n_recycles=3, diffusion_batch_size=2, and num_steps=50 if this takes too long or crashes, but in a real world scenario you should use the parameters above

6.4 Look at the confidences

```
python ../input_files/scripts/analyze_complex.py cr3022_rbd_complex
```

This prints overall pLDDT and iPTM, the pairwise min iPAE between all chain pairs, and Cα RMSDs of each chain versus the 6W41 crystal. It also saves `cr3022_rbd_complex_pae.png` (PAE heatmap with chain boundaries marked).

6.5 Understanding min iPAE

`chain_pair_pae_min` (min iPAE) is the key interface confidence metric. It reports the **minimum predicted aligned error** between each pair of chains, reflecting how confident the model is about the best-resolved part of the interface.

min iPAE	Interpretation
< 5	High confidence: interface likely correct
5 to 12	Moderate: some contacts may be right
> 12	Low confidence: docking is uncertain

VH-VL min iPAE will be very low (conserved pairing, exactly as in Section 5). The interesting numbers are VH-RBD and VL-RBD. In a typical workshop run these will land in the teens, which means the docking is barely above random guessing. The chain pTMs will look fine and pLDDT will look great because each chain folded well on its own.

This is the central lesson of antibody/antigen prediction: **per-chain confidence is not interface confidence.** If you only looked at pLDDT here you'd be falsely reassured.

Open `cr3022_rbd_complex_pae.png` and look at the off-diagonal blocks between antibody chains and the RBD. Compare to Section 5's PAE heatmap. The diagonal blocks look similar (each chain folds well), but the antibody-to-RBD off-diagonals are washed out. That is exactly what high min iPAE looks like visually.

The question, then: how do we make this work?

Section 7: Template-guided prediction

RF3's templating system lets you **fix known structural regions** and predict only the unknown parts. This is the key to reliable antibody/antigen complex prediction.

How it works:

- Provide a structure via `"path"` in a JSON component
- Use `"template_selection"` to specify which chains and residues to fix
- Syntax: `"CHAIN/RESNAME/RESID_RANGE"` (e.g. `"R/*/1-194"` fixes residues 1-194 of chain R)
- Unfixed residues are predicted from scratch by the diffusion model

Your strategy: use the experimental RBD structure from 6M0J as a template for the antigen, and let RF3 only figure out where the antibody docks.

7.1 Prepare the template

6M0J is ACE2 bound to the RBD. You only want the RBD chain (chain B), and you need to rename it to avoid colliding with the antibody chain IDs (H, L). The 6M0J CIF is already in your working directory from Section 2 (if not, run `python ../input_files/scripts/download_cif.py 6M0J`).

```
python ../input_files/scripts/prepare_rbd_template.py
```

This crops 6M0J chain B (the RBD) and rewrites it as chain R into `rbd_template.cif`. Verify:

```
python ../input_files/scripts/inspect_structure.py rbd_template.cif
```

You should see a single chain R with 194 residues.

CIF-based templating is a very powerful feature of RF3, but it can be picky about file formatting. When preparing templates I recommend using AtomWorks (as we did here) rather than hand-editing CIF files. I also recommend reading the full RF3 README for a deeper understanding of CIF templating.

7.2 Look at the templated JSON

```
cat ../input_files/jsons/cr3022_rbd_templated.json
```

The differences from Section 6's JSON:

- The RBD component now has a `"path"` pointing at the template CIF instead of just a sequence
- There is a top-level `"template_selection": ["R"]` telling RF3 to fix chain R

7.3 Run the templated fold

```
rf3 fold inputs='../input_files/jsons/cr3022_rbd_templated.json' \
    n_recycles=10 diffusion_batch_size=5 num_steps=200 out_dir="."
```

You can set n_recycles=3, diffusion_batch_size=2, and num_steps=50 if this takes too long or crashes, but expect worse results

7.4 Compare against the un-templated run

Re-run the same analysis script, pointing at the new prediction:

```
python ../input_files/scripts/analyze_complex.py cr3022_rbd_templated
```

Two things should jump out:

1. The min iPAE between antibody and antigen should drop noticeably (often into the single digits). RF3 is now confident about the docking.
2. The Cα RMSD of the predicted RBD versus the 6W41 crystal will be near zero (we just gave RF3 the answer for that piece), and the overall complex will line up much more closely with the experimental structure.

Open both PAE heatmaps (`cr3022_rbd_complex_pae.png` and `cr3022_rbd_templated_pae.png`) side by side. The off-diagonal antibody-to-RBD blocks are dramatically darker in the templated case. That is what a confident interface looks like.

7.5 Compare against the experimental complex

The per-chain RMSDs above superimpose each chain independently — they tell you how well each chain *folds*, not where it *docks*. A cleaner docking question is: after aligning the RBD chains from the two complexes, how far is the predicted antibody from where it should be? This is the real equivalent to the chain_pair_pae_min values which are estimates.

```
python ../input_files/scripts/compare_docking.py
```

This superimposes the predicted RBD (chain R) onto the crystal RBD (6W41 chain C), applies that rigid-body transform to the full prediction, then reports the Cα RMSD over the combined antibody (H+L vs 6W41 chains A+B). It also saves `docking_overlay.html` — open it to see both complexes in the same reference frame, with the RBDs coinciding and the antibodies either overlapping (good docking) or diverging.

The conceptual takeaway

Templating is essentially telling RF3: *"The antigen already folded in the absence of the antibody. Now figure out where the antibody binds."* Without a template, RF3 is free to deform the antigen to create false contacts. With a fixed antigen, the only flexibility is in the docking, which is what we wanted to predict in the first place. There are limitations which is that some antibody-antigen binding events cause large deformations to the antigen and in these cases you are likely to miss the correct binding pose using this technique.

You should also check the iPAE confidence metrics. How well do they line up with the visual quality of the docking? In general RF3's confidence aligns well with actual performance, but occasionally it can be very confident even when the model is wrong, so visual inspection still matters.

Key points:

- **Always inspect chains** before templating. PDB files often contain binding partners, symmetry mates, or other unwanted molecules.
- **Rename chains** that would clash with antibody chain IDs (H, L).
- Crystal and cryo-EM templates are more reliable than predicted ones.
- The more you template, the more confident the model will appear. Confidence metrics should not be directly compared across approaches with different amounts of templating.

Section 8: Tips and Practical Guidance

Speed vs. quality

Setting	`num_steps`	`n_recycles`	`diffusion_batch_size`	Use case
Quick	50	3	1-2	Exploration, workshops
Production	200	10	5-10	Final predictions, publications

For antibody/antigen complexes I also recommend repeating over 5 seeds at production settings and picking the one with the lowest chain_pair_pae_min between the antibody and antigen.

When to use templates vs. MSAs

Scenario	Approach
Sequence only	MSA-only prediction
Crystal or cryo-EM structure of antigen	Crop to binding domain, template it
Predicted structure of antigen	Template (verify prediction quality first)
Known antibody framework	Template framework, predict CDRs
Both structures known	Template both, predict CDRs in antigen context

Templating strategies

- **Crop to the binding domain.** Include 10 to 20 residues of buffer around the epitope for structural context.
- **Rename chains** to avoid conflicts with antibody chains (H, L).
- You can also template a **predicted** antigen structure (e.g. from an RF3 monomer prediction or AlphaFold DB). Just replace the `"path"` in the template input.

Confidence metrics reference

Metric	What it measures	Confident
pLDDT	Per-residue fold confidence (0-1)	> 0.9 very high, > 0.7 confident
min iPAE	Best interface contact confidence	< 5 high, 5-12 moderate, > 12 low
iPTM	Overall interface quality	> 0.8
pTM	Overall fold quality	> 0.8
PAE matrix	Pairwise position confidence	Visual: check off-diagonal blocks

For complex predictions, **min iPAE** (`chain_pair_pae_min`) is the most reliable interface metric. iPTM averages over all inter-chain pairs and can be diluted by large domains.

Resources

- **RF3 (Foundry):** github.com/RosettaCommons/foundry
- **AtomWorks:** github.com/RosettaCommons/atomworks
- **RFantibody** (antibody design): github.com/RosettaCommons/RFantibody
- **Biotite** (structural bioinformatics): biotite-python.org
- **py3Dmol** (3D visualization): 3dmol.csb.pitt.edu
- **ColabFold** (MSA generation): github.com/sokrypton/ColabFold

PDB structures used

PDB	Description	Used in
3OGO	Anti-GFP nanobody (VHH)	Sections 1, 4
6M0J	ACE2 / SARS-CoV-2 RBD complex	Sections 2, 7
1N8Z	Trastuzumab Fab / HER2 complex	Section 5
6W41	CR3022 Fab / SARS-CoV-2 RBD complex	Section 6