Supplementary Information

Computational screening of Indian phytochemicals as candidate efflux-pump inhibitors of *Salmonella* AcrB

Pranoy Mathur

Inventure Academy; Bengaluru, India

Contents

S1 Software versions

S2 Data availability

S3 Supplementary tables

Table S1 Redocking of crystallographic actives

Table S2 Docking scores across three random seeds

Table S3 Compounds that failed to dock

S4 MM-GBSA output

S5 Code (scripts 01–20)

S1 — Software versions

Open Babel 3.1.0, AutoDock Vina 1.2.7, RDKit 2026.03, scikit-learn 1.9.0, PyMOL 3.1.6.1, OpenMM via the Making-it-Rain workflow, AmberTools (pytraj, MMPBSA.py), ProLIF.

S2 — Data availability

The complete code and data archive is deposited at Zenodo, `https://doi.org/10.5281/zenodo.23356930`. It contains the files listed below. The scripts reproduced in Section S5 of this document are the same scripts as those in the archive.

Code

`01_download_smiles`	canonical SMILES for the filtered IMPPAT library
`02_download_2d_structures`	IMPPAT 2D SDFs used for descriptor calculation
`03_prepare_ligands`	Open Babel preparation of all ligands and benchmarks
`04_dock_library`	docking of the full library into the T-state model
`05_redock_minocycline`	redocking of the 4DX5 co-crystallised ligand
`06_redock_rmsd`	in-place heavy-atom RMSD, no superposition
`07_pose_recovery`	pose recovery for the crystallographic actives
`08_redock_D13_9001`	D13-9001 redock and search-box coverage
`09_dock_enrichment_set`	docking of 7 actives and 496 DUD-E decoys
`10_enrichment_metrics`	ROC-AUC and enrichment factors
`11_seed_robustness`	three independent random seeds
`12_extract_md_snapshots`	five conformers from the MD trajectory
`13_build_ensemble_receptors`	docking-ready receptors from those snapshots
`14_dock_ensemble`	conformer-ensemble docking and rank correlations

`15_build_ml_dataset`	15 RDKit descriptors for every docked compound
`16_ml_scaffold_split`	scaffold split, external test set, held-out importance
`17_permutation_test`	1000-permutation significance test
`18_binding_site_contacts`	contact table regenerated from the deposited poses
`19_figure2_permutation_importance`	Figure 2
`20_figure3_inputs`	inputs for the Figure 3 render

Data

`ligand_smiles.csv`	SMILES for the screened library
`actives_smiles.csv`	SMILES for the known AcrB actives
`results_Tstate.csv`	docking scores for all 863 compounds plus PAβN and NMP
`failed_dockings.txt`	the two compounds that failed to dock, and why
`ml_dataset.csv`	the 15 RDKit descriptors and the T-state docking score for each compound
`contact_table_Tstate.csv`	binding-site contacts of the five top hits
`ADMET_table_Tstate.csv`	predicted ADMET properties of the 17 primary hits
`DBP_alignment_table.csv`	pocket residue mapping from E. coli onto the model
`benchmark_scores.csv`	docking scores for the enrichment benchmark set
`pose_recovery_summary.csv`	RMSD and box coverage for each crystallographic active
`robustness_scores.csv`	scores across the three random seeds
`ensemble_scores.csv`	scores across the five MD-derived conformers
`receptor/`	the T-state homology model, PDB and PDBQT
`docked_poses_top5/`	top-ranked docked pose of each of the five best hits
`md_analysis/`	Cα RMSD, ligand-pocket distances, control set, MM-GBSA output

Notes

Ligand three-dimensional coordinate generation with Open Babel is stochastic, so re-running 03_prepare_ligands does not reproduce byte-identical input files. The deposited PDBQT inputs, not the script, define the exact structures docked.

The molecular dynamics production run and its trajectory analysis were carried out in Google Colab using the Making-it-Rain OpenMM workflow, with pytraj for Cα RMSD and ligand-pocket distances, ProLIF for per-residue contact frequencies, and MMPBSA.py for the MM-GBSA calculation. The numerical outputs of that analysis are provided in md_analysis/. The trajectory files themselves are too large to include and are available on request.

Every script carries a single comment line stating what it does. Scripts are numbered in the order the Methods describe them. Paths at the top of each script assume it is run from the folder containing the data files.

S3 — Supplementary tables

Table S1 | Redocking of crystallographic actives

active	source_pdb	redock_score_kcalmol	best_rmsd_A	crystal_atoms_in_box	note
minocycline	4DX5	-8.95	1.05	33/33	template ligand; defines the box centre
doxorubicin	4DX7	-9.09	2.8	39/39	fully enclosed by the search box
D13-9001	3W9H	-8.81	3.44	41/49	pyridopyrimidine; partially enclosed
MBX2319	5ENO	-8.87	3.41	22/29	pyranopyridine; partially enclosed
MBX2931	5ENP	-7.98	3.81	30/37	pyranopyridine; partially enclosed

Generated by 07_pose_recovery and 08_redock_D13_9001. RMSD is in-place heavy-atom RMSD with no superposition. “crystal atoms in box” is the number of crystallographic ligand heavy atoms falling inside the search box.

Table S2 | Docking scores across three random seeds

ligand_id	seed	affinity
IMPHY001793	101	-11.02
IMPHY000398	101	-10.91
IMPHY005217	101	-10.75
IMPHY004477	101	-10.42
IMPHY005683	101	-10.43
IMPHY008805	101	-10.47
IMPHY013919	101	-10.33
IMPHY012881	101	-10.28
IMPHY015068	101	-10.21
IMPHY009261	101	-10.19
IMPHY013011	101	-9.838
IMPHY013920	101	-10.13
IMPHY010647	101	-10.05
IMPHY015025	101	-9.939
IMPHY005344	101	-9.99
IMPHY005303	101	-9.996
IMPHY013057	101	-9.981
IMPHY006056	101	-9.929
IMPHY011145	101	-9.937
IMPHY014974	101	-9.88
IMPHY011951	101	-9.883
IMPHY000566	101	-9.863
IMPHY000852	101	-9.834
IMPHY008826	101	-9.914
IMPHY002777	101	-10.03
IMPHY001779	101	-9.853
IMPHY012187	101	-9.779
IMPHY007819	101	-9.831
IMPHY009376	101	-9.788
IMPHY010168	101	-9.777
IMPHY013144	101	-9.746
IMPHY005764	101	-9.71
IMPHY014814	101	-9.734
IMPHY003717	101	-9.671
IMPHY001942	101	-9.714
IMPHY011147	101	-9.69
IMPHY007459	101	-9.693
IMPHY014234	101	-9.678
IMPHY013880	101	-8.869
IMPHY014375	101	-9.728
IMPHY011301	101	-9.63
IMPHY001591	101	-8.257
IMPHY005825	101	-9.529
IMPHY006055	101	-9.611

ligand_id	seed	affinity
IMPHY010187	101	-9.552
IMPHY000425	101	-9.584
IMPHY005658	101	-9.562
IMPHY004754	101	-9.574
IMPHY003598	101	-9.57
IMPHY011146	101	-9.297
IMPHY001793	202	-11.02
IMPHY000398	202	-10.9
IMPHY005217	202	-10.71
IMPHY004477	202	-10.43
IMPHY005683	202	-10.4
IMPHY008805	202	-10.41
IMPHY013919	202	-10.34
IMPHY012881	202	-10.26
IMPHY015068	202	-10.26
IMPHY009261	202	-10.18
IMPHY013011	202	-10.16
IMPHY013920	202	-10.11
IMPHY010647	202	-10.04
IMPHY015025	202	-9.967
IMPHY005344	202	-9.993
IMPHY005303	202	-9.995
IMPHY013057	202	-9.919
IMPHY006056	202	-9.903
IMPHY011145	202	-9.91
IMPHY014974	202	-9.875
IMPHY011951	202	-9.921
IMPHY000566	202	-9.853
IMPHY000852	202	-9.793
IMPHY008826	202	-9.919
IMPHY002777	202	-9.851
IMPHY001779	202	-9.846
IMPHY012187	202	-9.762
IMPHY007819	202	-9.784
IMPHY009376	202	-9.829
IMPHY010168	202	-9.762
IMPHY013144	202	-9.711
IMPHY005764	202	-9.669
IMPHY014814	202	-9.457
IMPHY003717	202	-9.669
IMPHY001942	202	-9.682
IMPHY011147	202	-9.707
IMPHY007459	202	-9.69
IMPHY014234	202	-9.684
IMPHY013880	202	-8.171

ligand_id	seed	affinity
IMPHY014375	202	-9.675
IMPHY011301	202	-9.67
IMPHY001591	202	-9.622
IMPHY005825	202	-9.572
IMPHY006055	202	-9.6
IMPHY010187	202	-9.552
IMPHY000425	202	-9.579
IMPHY005658	202	-9.555
IMPHY004754	202	-9.574
IMPHY003598	202	-9.542
IMPHY011146	202	-9.518
IMPHY001793	303	-11.05
IMPHY000398	303	-10.89
IMPHY005217	303	-10.68
IMPHY004477	303	-10.43
IMPHY005683	303	-10.42
IMPHY008805	303	-10.47
IMPHY013919	303	-10.33
IMPHY012881	303	-10.35
IMPHY015068	303	-10.23
IMPHY009261	303	-10.16
IMPHY013011	303	-10.11
IMPHY013920	303	-10.13
IMPHY010647	303	-10.05
IMPHY015025	303	-9.996
IMPHY005344	303	-9.994
IMPHY005303	303	-9.983
IMPHY013057	303	-9.854
IMPHY006056	303	-9.959
IMPHY011145	303	-9.91
IMPHY014974	303	-9.876
IMPHY011951	303	-9.874
IMPHY000566	303	-9.855
IMPHY000852	303	-9.809
IMPHY008826	303	-9.815
IMPHY002777	303	-9.833
IMPHY001779	303	-9.851
IMPHY012187	303	-9.756
IMPHY007819	303	-9.82
IMPHY009376	303	-9.79
IMPHY010168	303	-9.852
IMPHY013144	303	-9.745
IMPHY005764	303	-9.797
IMPHY014814	303	-9.725
IMPHY003717	303	-9.256

ligand_id	seed	affinity
IMPHY001942	303	-9.709
IMPHY011147	303	-9.689
IMPHY007459	303	-9.678
IMPHY014234	303	-9.691
IMPHY013880	303	-9.656
IMPHY014375	303	-9.727
IMPHY011301	303	-9.661
IMPHY001591	303	-9.648
IMPHY005825	303	-9.576
IMPHY006055	303	-9.575
IMPHY010187	303	-9.532
IMPHY000425	303	-9.567
IMPHY005658	303	-9.564
IMPHY004754	303	-9.587
IMPHY003598	303	-9.561
IMPHY011146	303	-9.278

Generated by 11_seed_robustness. The 50 top-ranked compounds of the primary run were redocked with three independent AutoDock Vina random seeds. Affinities are in kcal/mol.

Table S3 | Compounds that failed to dock

Compounds that failed to dock in the T-state run (2 of 865)

IMPHY005215 multi-component SMILES (hydrochloride salt); Open Babel wrote two disconnected fragments in one PDBQT file, which Vina cannot dock
IMPHY017899 multi-component SMILES (hydrochloride salt); same cause as above

Both were excluded from all subsequent analysis, leaving 863 successful dockings.
The exclusion was not based on binding score, so it is unlikely to have introduced affinity-based selection bias.

S4 — MM-GBSA output

Verbatim output of MMPBSA.py (file `md_analysis/FINAL_RESULTS_MMPBSA.dat` in the deposited archive).

```
| Run on Thu Sep  3 18:41:09 2026
|
|Input file:
|------------------------------------------------------------
|&general
|  endframe=500,  interval=50, strip_mask=:WAT:Na+:Cl-:Mg+:K+,
|/
|&gb
| igb=2, saltcon=0.15,
|/
|&pb
| istrng=0.15, inp=2, radiopt=0, prbrad=1.4,
|/-----------------------------------------------------------
|MMPBSA.py Version=14.0
|Solvated complex topology file:  /content/drive/MyDrive/AcrB_MD/SYS_gaff2_nw.prmtop
|Complex topology file:           com.prmtop
|Receptor topology file:          rec.prmtop
|Ligand topology file:            ligand.prmtop
|Initial mdcrd(s):                /content/drive/MyDrive/AcrB_MD/prot_lig_prod1-20_nw.dcd
|
|Receptor mask:                  ":1-1032"
|Ligand mask:                    ":1033"
|Ligand residue name is "LIG"
|
|Calculations performed using 10.98 complex frames.
|Poisson Boltzmann calculations performed using internal PBSA solver in mmpbsa_py_energy
|
|Generalized Born ESURF calculated using 'LCPO' surface areas
|
|All units are reported in kcal/mole.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

GENERALIZED BORN:

Complex:
Energy Component            Average              Std. Dev.   Std. Err. of Mean
-------------------------------------------------------------------------------
VDWAALS                  -8243.0619              50.5965             16.0000
EEL                     -71449.2515             158.9351             50.2597
EGB                     -10972.7953              97.3357             30.7803
ESURF                      343.2960               3.3630              1.0635

G gas                   -79692.3134             124.2688             39.2972
G solv                  -10629.4993              98.8765             31.2675

TOTAL                   -90321.8127              60.3059             19.0704


Receptor:
Energy Component            Average              Std. Dev.   Std. Err. of Mean
-------------------------------------------------------------------------------
VDWAALS                  -8203.1046              50.4258             15.9460
EEL                     -71460.9164             156.9536             49.6331
EGB                     -10979.6139              94.9897             30.0384
ESURF                      344.9057               3.3805              1.0690

G gas                   -79664.0210             122.3161             38.6798
G solv                  -10634.7082              96.6174             30.5531

TOTAL                   -90298.7292              61.1987             19.3527


Ligand:
Energy Component            Average              Std. Dev.   Std. Err. of Mean
-------------------------------------------------------------------------------
VDWAALS                     -3.2568               0.7121              0.2252
EEL                         25.9521               3.1403              0.9931
EGB                        -21.6498               1.1963              0.3783
ESURF                        3.3003               0.0258              0.0082

G gas                       22.6953               3.1424              0.9937
```

G solv	-18.3494	1.2046	0.3809
TOTAL	4.3459	2.0762	0.6565

Differences (Complex - Receptor - Ligand):

Energy Component	Average	Std. Dev.	Std. Err. of Mean
VDWAALS	-36.7006	2.1527	0.6808
EEL	-14.2871	7.1065	2.2473
EGB	28.4683	5.1268	1.6212
ESURF	-4.9100	0.3337	0.1055
DELTA G gas	-50.9877	7.5617	2.3912
DELTA G solv	23.5583	4.9359	1.5609
DELTA TOTAL	-27.4294	3.2319	1.0220

POISSON BOLTZMANN:

Complex:

Energy Component	Average	Std. Dev.	Std. Err. of Mean
VDWAALS	-8243.0619	50.5965	16.0000
EEL	-71449.2515	158.9351	50.2597
EPB	-10264.2494	81.4630	25.7609
ENPOLAR	7668.6662	20.2173	6.3933
EDISPER	-4020.1856	36.4961	11.5411
G gas	-79692.3134	124.2688	39.2972
G solv	-6615.7689	87.6186	27.7074
TOTAL	-86308.0823	80.8650	25.5718

Receptor:

Energy Component	Average	Std. Dev.	Std. Err. of Mean
VDWAALS	-8203.1046	50.4258	15.9460
EEL	-71460.9164	156.9536	49.6331
EPB	-10273.6418	79.5429	25.1537
ENPOLAR	7657.4377	19.5335	6.1770
EDISPER	-4032.2289	36.6134	11.5782
G gas	-79664.0210	122.3161	38.6798
G solv	-6648.4330	85.6641	27.0894
TOTAL	-86312.4540	80.7901	25.5481

Ligand:

Energy Component	Average	Std. Dev.	Std. Err. of Mean
VDWAALS	-3.2568	0.7121	0.2252
EEL	25.9521	3.1403	0.9931
EPB	-20.4175	1.1344	0.3587
ENPOLAR	39.9373	0.2097	0.0663
EDISPER	-37.7965	0.2201	0.0696
G gas	22.6953	3.1424	0.9937
G solv	-18.2767	1.0706	0.3385
TOTAL	4.4186	2.1815	0.6899

Differences (Complex - Receptor - Ligand):

Energy Component	Average	Std. Dev.	Std. Err. of Mean
VDWAALS	-36.7006	2.1527	0.6808
EEL	-14.2871	7.1065	2.2473
EPB	29.8099	4.7356	1.4975
ENPOLAR	-28.7088	1.6361	0.5174
EDISPER	49.8398	1.1569	0.3659

DELTA G gas	-50.9877	7.5617	2.3912
DELTA G solv	50.9409	3.8070	1.2039
DELTA TOTAL	-0.0468	4.9868	1.5770

S5 — Code

The twenty scripts below are reproduced verbatim from the deposited archive. Long lines are wrapped for the page; the wrapping is typographic only.

01_download_smiles.py

```
# Fetches canonical SMILES for the filtered IMPPAT library from the IMPPAT database.

import csv, os, time, urllib.request, urllib.error
from rdkit import Chem

CSV_PATH = "ligands/library_865.csv"
OUT      = "ligand_smiles.csv"
DELAY    = 0.4
TIMEOUT  = 30

URLS = [
    "https://cb.imsc.res.in/imppat/images/2D/SDF/{id}_2D.sdf",
    "https://cb.imsc.res.in/imppat/images/2D/MOL/{id}_2D.mol",
    "https://cb.imsc.res.in/imppat/images/3D/SDF/{id}_3D.sdf",
]

def read_ids(path):
    ids=[]
    for row in csv.reader(open(path, newline="")):
        if row and row[0].strip().startswith("IMPHY"): ids.append(row[0].strip())
    return ids

def fetch(url):
    try:
        with urllib.request.urlopen(url, timeout=TIMEOUT) as r:
            b = r.read().decode("utf-8", "ignore")
            return b if ("V2000" in b or "V3000" in b) else None
    except Exception:
        return None

def to_smiles(block):
    m = Chem.MolFromMolBlock(block, sanitize=True)
    if m is None:
        m = Chem.MolFromMolBlock(block, sanitize=False)
        if m is not None:
            try: Chem.SanitizeMol(m)
            except Exception: return None
    return Chem.MolToSmiles(m) if m else None

def main():
    ids = read_ids(CSV_PATH); print(f"{len(ids)} IDs")
    rows=[]; fail=[]
    for i, lid in enumerate(ids, 1):
        smi=None
        for tmpl in URLS:
            block = fetch(tmpl.format(id=lid))
            if block:
                smi = to_smiles(block)
                if smi: break
        (rows.append((lid, smi)) if smi else fail.append(lid))
        if i % 50 == 0: print(f"  {i}/{len(ids)} done, {len(fail)} failed")
        time.sleep(DELAY)
    with open(OUT, "w", newline="") as fh:
        w=csv.writer(fh); w.writerow(["ligand_id","smiles"]); w.writerows(rows)
    print(f"\nWrote {OUT}: {len(rows)} SMILES, {len(fail)} failed")
    if fail: print("failed IDs:", fail[:20], "..." if len(fail)>20 else "")

if __name__ == "__main__":
    main()
```

02_download_2d_structures.py

```
# Downloads the IMPPAT 2D SDF files used for RDKit descriptor calculation.

import csv, os, time, urllib.request

CSV_PATH = "ligands/library_865.csv"
OUT_DIR  = "ligands/sdf"
DELAY    = 0.3
TIMEOUT  = 30
HEADERS  = {"User-Agent": "Mozilla/5.0 (research; SMILES fetch)"}
CANDIDATES = [
    "https://cb.imsc.res.in/imppat/images/2D/SDF/{id}.sdf",
    "https://cb.imsc.res.in/imppat/images/3D/SDF/{id}_3D.sdf",
]

def read_ids(path):
    ids=[]
    for row in csv.reader(open(path, newline="")):
        if row and row[0].strip().startswith("IMPHY"): ids.append(row[0].strip())
    return ids

def get(url):
    try:
        req = urllib.request.Request(url, headers=HEADERS)
        with urllib.request.urlopen(req, timeout=TIMEOUT) as r:
            t = r.read().decode("utf-8", "ignore")
            return t if "$$$$" in t else None
    except Exception:
        return None

def main():
    os.makedirs(OUT_DIR, exist_ok=True)
    ids = read_ids(CSV_PATH); print(f"{len(ids)} IDs")
    ok=0; fail=[]
    for i, lid in enumerate(ids, 1):
        dst = os.path.join(OUT_DIR, lid + ".sdf")
        if os.path.exists(dst) and os.path.getsize(dst) > 0:
            ok += 1; continue
        text=None
        for tmpl in CANDIDATES:
            text = get(tmpl.format(id=lid))
            if text: break
        if text: open(dst, "w").write(text); ok += 1
        else: fail.append(lid)
        if i % 50 == 0: print(f"  {i}/{len(ids)}  ok={ok} fail={len(fail)}")
        time.sleep(DELAY)
    print(f"\nDone: {ok} saved to {OUT_DIR}/, {len(fail)} failed")
    if fail: print("failed:", fail[:20], "..." if len(fail)>20 else "")

if __name__ == "__main__":
    main()
```

03_prepare_ligands.py

```
# Prepares all ligands and the two benchmarks with Open Babel: pH 7.4 protonation, MMFF94 3D generation, PDBQT
        output.

import csv, os
from openbabel import pybel
pybel.ob.obErrorLog.SetOutputLevel(0)
OUT="ligands_prepped"; os.makedirs(OUT, exist_ok=True)
jobs=[(r["ligand_id"], r["smiles"]) for r in csv.DictReader(open("ligand_smiles.csv"))]
jobs.append(("PAbN","O=C(Nc1cc2c(cc1)cccc2)[C@@H](NC(=O)[C@@H](N)Cc1ccccc1)CCCN=C(N)N"))
jobs.append(("NMP","N1(CCNCC1)Cc1c2c(ccc1)cccc2"))
done=0; fail=[]
for i,(lid,smi) in enumerate(jobs,1):
    out=os.path.join(OUT, lid+".pdbqt")
    if os.path.exists(out) and os.path.getsize(out)>0: done+=1; continue
    try:
        m=pybel.readstring("smi", smi)
        m.OBMol.AddHydrogens(False, True, 7.4)
        m.make3D(forcefield="mmff94", steps=100)
        m.write("pdbqt", out, overwrite=True)
```

```
        done+=1
    except Exception:
        fail.append(lid)
    if i%25==0:
        open("prep_progress.txt","w").write(f"{i}/{len(jobs)} done={done} fail={len(fail)}\n")
open("prep_progress.txt","w").write(f"COMPLETE {done}/{len(jobs)} fail={len(fail)} {fail[:20]}\n")
```

04_dock_library.py

```
# Docks the full library and the PAbN/NMP benchmarks into the T-state AcrB model with AutoDock Vina.

import os, csv, subprocess, glob

VINA   = "vina.exe"
CONFIG = "config_Tstate.txt"
LIGDIR = "ligands_prepped"
OUTDIR = "docking/out_Tstate"
RESULTS= "results_Tstate.csv"

def done_ids():
    ids=set()
    if os.path.exists(RESULTS):
        for row in csv.reader(open(RESULTS)):
            if row and row[0]!="ligand_id": ids.add(row[0])
    return ids

def best_affinity(stdout):

    for line in stdout.splitlines():
        p=line.split()
        if len(p)>=2 and p[0]=="1":
            try: return float(p[1])
            except ValueError: pass
    return None

def main():
    os.makedirs(OUTDIR, exist_ok=True)
    for need in (VINA, CONFIG, LIGDIR):
        if not os.path.exists(need):
            print("MISSING:", need); return
    ligs=sorted(glob.glob(os.path.join(LIGDIR,"*.pdbqt")))
    done=done_ids()
    if not os.path.exists(RESULTS):
        open(RESULTS,"w").write("ligand_id,affinity_kcal_mol\n")
    print(f"{len(ligs)} ligands, {len(done)} already done")
    for i,lig in enumerate(ligs,1):
        lid=os.path.splitext(os.path.basename(lig))[0]
        if lid in done: continue
        out=os.path.join(OUTDIR, lid+"_out.pdbqt")
        cmd=[VINA,"--config",CONFIG,"--ligand",lig,"--out",out]
        try:
            r=subprocess.run(cmd, capture_output=True, text=True, timeout=600)
            aff=best_affinity(r.stdout)
        except Exception as e:
            aff=None; print("  ERR", lid, e)
        if aff is not None:
            with open(RESULTS,"a") as f: f.write(f"{lid},{aff}\n")
        if i%25==0: print(f"  {i}/{len(ligs)}  last={lid} {aff}")
    print("Done ->", RESULTS)

if __name__=="__main__":
    main()
```

05_redock_minocycline.py

```
# Redocks the co-crystallised minocycline from 4DX5 into the finished model.

from vina import Vina
import numpy as np, json, traceback
try:
    v=Vina(sf_name='vina', verbosity=0)
    v.set_receptor('model_01.pdbqt')
    v.set_ligand_from_file('minocycline_lig.pdbqt')
```

```
    v.compute_vina_maps(center=[35.75,-60.85,-41.81], box_size=[22,22,22])
    v.dock(exhaustiveness=8, n_poses=10)
    v.write_poses('minocycline_redock.pdbqt', n_poses=10, overwrite=True)
    sc=[float(x) for x in v.energies(n_poses=10)[:,0]]
    def heavy(lines):
        out=[]
        for l in lines:
            if l.startswith(("ATOM","HETATM")):
                nm=l[12:16].strip()
                if nm.startswith("H"): continue
                out.append((nm[0].upper(), np.array([float(l[30:38]),float(l[38:46]),float(l[46:54])])))
        return out
    crys=heavy(open("minocycline_4DX5.pdb"))
    poses=[];cur=[]
    for l in open("minocycline_redock.pdbqt"):
        if l.startswith("MODEL"): cur=[]
        elif l.startswith("ENDMDL"): poses.append(cur)
        else: cur.append(l)
    def rmsd(a,b):
        b=list(b);used=[False]*len(b);sq=0;n=0
        for el,pa in a:
            bj=-1;bd=1e9
            for j,(e2,pb) in enumerate(b):
                if used[j] or e2!=el: continue
                d=float(np.sum((pa-pb)**2))
                if d<bd:bd=d;bj=j
            if bj>=0:used[bj]=True;sq+=bd;n+=1
        return (sq/n)**0.5 if n else None
    rl=[rmsd(crys,heavy(p)) for p in poses]
    b=min(rl);bi=rl.index(b)
    json.dump({"scores":sc,"rmsd_per_pose":rl,"best_rmsd":b,"best_pose":bi+1,
               "best_pose_score":sc[bi],"crystal_heavy_atoms":len(crys)},
              open("redock_result.json","w"), indent=2)
    print("DONE")
except Exception:
    open("redock_result.json","w").write('{"error":"'+traceback.format_exc().replace(chr(10),' ')+'"}')
    raise
```

06_redock_rmsd.py

```
# Computes heavy-atom RMSD in place, without superposition, between an experimental pose and its redocked pose.

import argparse, sys
from rdkit import Chem
from rdkit.Chem import AllChem
def load_mol(path):
    p = path.lower()
    if p.endswith(".sdf"):
        return [m for m in Chem.SDMolSupplier(path, removeHs=True) if m]
    if p.endswith(".mol2"):
        m = Chem.MolFromMol2File(path, removeHs=True); return [m] if m else []
    if p.endswith(".pdb"):
        m = Chem.MolFromPDBFile(path, removeHs=True); return [m] if m else []

    if p.endswith(".pdbqt"):
        lines=[]
        for ln in open(path):
            if ln.startswith(("ATOM","HETATM")): lines.append(ln[:66]+"\n")
            elif ln.startswith(("MODEL","ENDMDL","END")): lines.append(ln)
        m = Chem.MolFromPDBBlock("".join(lines), removeHs=True); return [m] if m else []
    raise ValueError("Unsupported format: "+path)

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--crystal", required=True)
    ap.add_argument("--docked", required=True)
    a = ap.parse_args()
    ref_list = load_mol(a.crystal)
    if not ref_list: sys.exit("Could not read crystal ligand: "+a.crystal)
    ref = ref_list[0]
    poses = load_mol(a.docked)
    if not poses: sys.exit("Could not read docked poses: "+a.docked)

    best=None
    for i,pose in enumerate(poses):
```

```
        try:

            rms = AllChem.GetBestRMS(Chem.Mol(pose), Chem.Mol(ref))
        except Exception as e:
            print(f"  pose {i}: RMS failed ({e}) -- check atom-name/order match", file=sys.stderr); continue
        print(f"  pose {i+1}: heavy-atom RMSD = {rms:.2f} A")
        if best is None or rms < best: best = rms
    if best is None: sys.exit("No RMSD could be computed. Ensure both files are the SAME molecule.")
    print(f"\nBEST heavy-atom RMSD = {best:.2f} A  ->  {'PASS (<2.0)' if best<2.0 else 'FAIL (>=2.0)'}")

if __name__ == "__main__":
    main()
```

07_pose_recovery.py

```
# Runs pose recovery for the four crystallographic AcrB actives and reports RMSD per compound.

import numpy as np, sys, os, csv
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit import RDLogger; RDLogger.DisableLog("rdApp.*")
from meeko import MoleculePreparation, PDBQTWriterLegacy, PDBQTMolecule, RDKitMolCreate
from vina import Vina

BOX=dict(center=[35.75,-60.85,-41.81], box_size=[22,22,22])
ACT=[
 ("MBX2319","benchmark_actives/5QG_ideal.sdf","5ENO.pdb","C","benchmark_actives/5ENO_5QG_crystal.pdb",8),
 ("MBX2931","benchmark_actives/5QF_ideal.sdf","5ENP.pdb","C","benchmark_actives/5ENP_5QF_crystal.pdb",8),
 ("D13-9001","benchmark_actives/P9D_ideal.sdf","3W9H.pdb","B","benchmark_actives/3W9H_P9D_crystal.pdb",4),
 ("doxorubicin","benchmark_actives/DM2_ideal.sdf","4DX7.pdb","B","benchmark_actives/4DX7_DM2_crystal.pdb",8),
]
def ca(fn, chain=None):
    d={}
    for l in open(fn):
        if l.startswith("ATOM") and l[12:16].strip()=="CA" and (chain is None or l[21]==chain):
            d[int(l[22:26])]=np.array([float(l[30:38]),float(l[38:46]),float(l[46:54])])
    return d
def kabsch(P,Q):
    Pc=P-P.mean(0); Qc=Q-Q.mean(0)
    H=Pc.T@Qc; U,S,Vt=np.linalg.svd(H); d=np.sign(np.linalg.det(Vt.T@U.T))
    R=Vt.T@np.diag([1,1,d])@U.T
    return R, Q.mean(0)-R@P.mean(0)
def heavy(fn_or_lines):
    lines=open(fn_or_lines) if isinstance(fn_or_lines,str) else fn_or_lines
    out=[]
    for l in lines:
        if l.startswith(("ATOM","HETATM")):
            nm=l[12:16].strip()
            if nm[:1]=="H": continue
            out.append((nm[0].upper(), np.array([float(l[30:38]),float(l[38:46]),float(l[46:54])])))
    return out
def rmsd(a,b):
    b=list(b); used=[False]*len(b); s=0;n=0
    for el,pa in a:
        bj=-1;bd=1e18
        for j,(e2,pb) in enumerate(b):
            if used[j] or e2!=el: continue
            dd=float(((pa-pb)**2).sum())
            if dd<bd: bd=dd;bj=j
        if bj>=0: used[bj]=True;s+=bd;n+=1
    return (s/n)**0.5 if n else float('nan')

mca=ca("model_01.pdb")
RES="benchmark_actives/pose_recovery.csv"
done=set()
if os.path.exists(RES):
    done={r[0] for r in csv.reader(open(RES)) if r and r[0]!="active"}
else:
    open(RES,"w").write("active,redock_score,best_rmsd,n_ca_aligned\n")

for name,sdf,cryst,chain,ligpdb,exh in ACT:
    if name in done: continue
    cca=ca(cryst, chain)
    common=sorted(set(mca)&set(cca))
    P=np.array([cca[i] for i in common]); Q=np.array([mca[i] for i in common])
```

```
    R,t=kabsch(P,Q)

    ref=[(el, R@xyz+t) for el,xyz in heavy(ligpdb)]

    mol=Chem.MolFromMolFile(sdf, removeHs=False)
    setups=MoleculePreparation().prepare(mol)
    pdbqt,ok,_=PDBQTWriterLegacy.write_string(setups[0])
    lp=f"benchmark_actives/{name}_lig.pdbqt"; open(lp,"w").write(pdbqt)
    v=Vina(sf_name='vina',verbosity=0)
    v.set_receptor('model_01.pdbqt'); v.set_ligand_from_file(lp)
    v.compute_vina_maps(**BOX); v.dock(exhaustiveness=exh,n_poses=10)
    outp=f"benchmark_actives/{name}_redock.pdbqt"
    v.write_poses(outp,n_poses=10,overwrite=True)
    sc=float(v.energies(n_poses=1)[0][0])

    poses=[];cur=[]
    for l in open(outp):
        if l.startswith("MODEL"): cur=[]
        elif l.startswith("ENDMDL"): poses.append(cur)
        else: cur.append(l)
    best=min(rmsd(ref, heavy(p)) for p in poses)
    with open(RES,"a") as f: f.write(f"{name},{sc:.2f},{best:.2f},{len(common)}\n")
    print(f"{name}: redock {sc:.2f} kcal/mol, best RMSD {best:.2f} A ({len(common)} CA aligned)")
print("DONE")
```

08_redock_D13_9001.py

```
# Completes the D13-9001 pose-recovery run and reports search-box coverage for every active.

import os, csv, subprocess, sys
import numpy as np

VINA   = "vina.exe"
RECEPTOR = "model_01.pdbqt"
BOX_CENTRE = np.array([35.75, -60.85, -41.81])
BOX_SIZE   = 22.0
EXHAUST    = 8
OUT_CSV    = "benchmark_actives/pose_recovery.csv"

ACTIVES = [
    ("D13-9001",    "3W9H.pdb", "B", "benchmark_actives/3W9H_P9D_crystal.pdb",
                                     "benchmark_actives/D13-9001_lig.pdbqt"),
    ("doxorubicin", "4DX7.pdb", "B", "benchmark_actives/4DX7_DM2_crystal.pdb", None),
    ("MBX2319",     "5ENO.pdb", "C", "benchmark_actives/5ENO_5QG_crystal.pdb", None),
    ("MBX2931",     "5ENP.pdb", "C", "benchmark_actives/5ENP_5QF_crystal.pdb", None),
]

def ca(fn, chain=None):
    d = {}
    for l in open(fn):
        if l.startswith("ATOM") and l[12:16].strip() == "CA" and (chain is None or l[21] == chain):
            d[int(l[22:26])] = np.array([float(l[30:38]), float(l[38:46]), float(l[46:54])])
    return d

def heavy(lines):
    out = []
    for l in lines:
        if l.startswith(("ATOM", "HETATM")):
            nm = l[12:16].strip()
            if nm[:1] == "H":
                continue
            out.append((nm[0].upper(),
                        np.array([float(l[30:38]), float(l[38:46]), float(l[46:54])])))
    return out

def kabsch(P, Q):
    Pc, Qc = P - P.mean(0), Q - Q.mean(0)
    U, S, Vt = np.linalg.svd(Pc.T @ Qc)
    d = np.sign(np.linalg.det(Vt.T @ U.T))
    R = Vt.T @ np.diag([1, 1, d]) @ U.T
    return R, Q.mean(0) - R @ P.mean(0)

def rmsd_inplace(ref, pose):
    b = list(pose); used = [False] * len(b); s = 0.0; n = 0
    for el, pa in ref:
```

```
        bj, bd = -1, 1e18
        for j, (e2, pb) in enumerate(b):
            if used[j] or e2 != el:
                continue
            dd = float(((pa - pb) ** 2).sum())
            if dd < bd:
                bd, bj = dd, j
        if bj >= 0:
            used[bj] = True; s += bd; n += 1
    return (s / n) ** 0.5 if n else float("nan")

def transformed_reference(cryst, chain, ligpdb):
    m, c = ca("model_01.pdb"), ca(cryst, chain)
    common = sorted(set(m) & set(c))
    R, t = kabsch(np.array([c[i] for i in common]), np.array([m[i] for i in common]))
    ref = [(el, R @ xyz + t) for el, xyz in heavy(open(ligpdb))]
    return ref, len(common)

def coverage(ref):
    pts = np.array([x for _, x in ref])
    inside = int(np.all(np.abs(pts - BOX_CENTRE) <= BOX_SIZE / 2, axis=1).sum())
    return inside, len(pts)

def main():
    for need in (VINA, RECEPTOR, "model_01.pdb"):
        if not os.path.exists(need):
            sys.exit(f"MISSING: {need} (run from the Acrb_Project folder)")

    print("=" * 68)
    print("SEARCH-BOX COVERAGE OF EACH CRYSTAL REFERENCE POSE")
    print("=" * 68)
    refs = {}
    for name, cryst, chain, ligpdb, _ in ACTIVES:
        if not os.path.exists(cryst):
            print(f"  {name:<13} skip ({cryst} not found)"); continue
        ref, nca = transformed_reference(cryst, chain, ligpdb)
        ins, tot = coverage(ref)
        refs[name] = ref
        print(f"  {name:<13} {ins:>3}/{tot:<3} atoms inside the {BOX_SIZE:.0f} A box"
              f"   ({nca} CA aligned)")

    name, cryst, chain, ligpdb, lig = ACTIVES[0]
    if lig is None or not os.path.exists(lig):
        sys.exit(f"\nMISSING prepped ligand: {lig}")

    cfg = "benchmark_actives/config_D13.txt"
    with open(cfg, "w") as f:
        f.write(f"receptor = {RECEPTOR}\n")
        f.write(f"center_x = {BOX_CENTRE[0]}\ncenter_y = {BOX_CENTRE[1]}\ncenter_z = {BOX_CENTRE[2]}\n")
        f.write(f"size_x = {BOX_SIZE}\nsize_y = {BOX_SIZE}\nsize_z = {BOX_SIZE}\n")
        f.write(f"exhaustiveness = {EXHAUST}\nnum_modes = 10\n")

    out = "benchmark_actives/D13-9001_redock.pdbqt"
    print(f"\ndocking {name} (exhaustiveness {EXHAUST}, 10 poses) -- this is the slow one...")
    try:
        r = subprocess.run([VINA, "--config", cfg, "--ligand", lig, "--out", out],
                           capture_output=True, text=True, timeout=3600)
    except subprocess.TimeoutExpired:
        sys.exit("Vina exceeded 60 min. Report D13-9001 as attempted but not completed.")
    if not os.path.exists(out):
        print(r.stdout[-1500:]); print(r.stderr[-800:])
        sys.exit("Vina produced no output. Report as attempted but not completed.")

    score = None
    for l in r.stdout.splitlines():
        p = l.split()
        if len(p) >= 2 and p[0] == "1":
            try: score = float(p[1]); break
            except ValueError: pass

    poses, cur = [], []
    for l in open(out):
        if l.startswith("MODEL"): cur = []
        elif l.startswith("ENDMDL"): poses.append(cur)
        else: cur.append(l)

    ref = refs[name]
```

```
    rl = [rmsd_inplace(ref, heavy(p)) for p in poses]
    best = min(rl); bi = rl.index(best) + 1
    ins, tot = coverage(ref)

    print(f"\n  top score      : {score} kcal/mol")
    print(f"  poses returned : {len(poses)}")
    print(f"  per-pose RMSD  : {', '.join(f'{x:.2f}' for x in rl)}")
    print(f"  BEST RMSD      : {best:.2f} A  (pose {bi})")
    print(f"  box coverage   : {ins}/{tot} crystal atoms inside the box")

    rows = []
    if os.path.exists(OUT_CSV):
        rows = [r for r in csv.reader(open(OUT_CSV))]
    if not rows:
        rows = [["active", "redock_score", "best_rmsd", "n_ca_aligned"]]
    rows = [r for r in rows if not (r and r[0] == name)]
    _, nca = transformed_reference(cryst, chain, ligpdb)
    rows.append([name, f"{score:.2f}" if score else "", f"{best:.2f}", str(nca)])
    with open(OUT_CSV, "w", newline="") as f:
        csv.writer(f).writerows(rows)
    print(f"\nappended to {OUT_CSV}")
    print("\nNOTE: report this RMSD together with the box-coverage figure. A reference")
    print("pose only partly enclosed by the search box cannot be reproduced in full,")
    print("so coverage is part of the honest interpretation of the number.")

if __name__ == "__main__":
    main()
```

09_dock_enrichment_set.py

```
# Docks the seven known AcrB actives and 496 property-matched DUD-E decoys under the identical protocol.

import os, csv, subprocess, glob

VINA="vina.exe"; CONFIG="config_Tstate.txt"
LIGDIR="benchmark_prepped"; OUTDIR="docking/out_bench"; RESULTS="benchmark_scores.csv"
def best_affinity(stdout):
    for line in stdout.splitlines():
        p=line.split()
        if len(p)>=2 and p[0]=="1":
            try: return float(p[1])
            except ValueError: pass
    return None

def done_ids():
    d=set()
    if os.path.exists(RESULTS):
        for r in csv.reader(open(RESULTS)):
            if r and r[0]!="id": d.add(r[0])
    return d

def main():
    os.makedirs(OUTDIR, exist_ok=True)
    for need in (VINA, CONFIG, LIGDIR):
        if not os.path.exists(need): print("MISSING:", need); return
    ligs=sorted(glob.glob(os.path.join(LIGDIR,"*.pdbqt")))
    done=done_ids()
    if not os.path.exists(RESULTS): open(RESULTS,"w").write("id,affinity_kcal_mol\n")
    print(f"{len(ligs)} ligands, {len(done)} already done")
    for i,lig in enumerate(ligs,1):
        lid=os.path.splitext(os.path.basename(lig))[0]
        if lid in done: continue
        out=os.path.join(OUTDIR, lid+"_out.pdbqt")
        try:
            r=subprocess.run([VINA,"--config",CONFIG,"--ligand",lig,"--out",out],
                             capture_output=True,text=True,timeout=600)
            aff=best_affinity(r.stdout)
        except Exception as e:
            aff=None; print("ERR",lid,e)
        if aff is not None:
            with open(RESULTS,"a") as f: f.write(f"{lid},{aff}\n")
        if i%25==0: print(f"  {i}/{len(ligs)}  last={lid} {aff}")
    print("Done ->", RESULTS)

if __name__=="__main__":
```

```
    main()
```

10_enrichment_metrics.py

```
# Computes ROC-AUC and enrichment factors at 1, 5 and 10 percent from the benchmark docking scores.

import argparse, numpy as np, pandas as pd, matplotlib
matplotlib.use("Agg"); import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, roc_curve
def enrichment_factor(labels_sorted, frac):
    n = len(labels_sorted); ntop = max(1, int(round(frac*n)))
    actives_total = labels_sorted.sum()
    if actives_total == 0: return float("nan")
    return (labels_sorted[:ntop].sum()/ntop) / (actives_total/n)

def bedroc(labels_sorted, alpha=20.0):

    n = len(labels_sorted); ra = labels_sorted.sum()/n
    if ra in (0, 1): return float("nan")
    ranks = np.where(labels_sorted == 1)[0] + 1
    rie_num = np.sum(np.exp(-alpha * ranks / n))
    rie_den = (ra / n) * (1 - np.exp(-alpha)) / (np.exp(alpha / n) - 1)
    rie = rie_num / rie_den
    factor = (ra * np.sinh(alpha / 2)) / (np.cosh(alpha / 2) - np.cosh(alpha / 2 - alpha * ra))
    return rie * factor

def main():
    ap = argparse.ArgumentParser(); ap.add_argument("--csv", required=True); a=ap.parse_args()
    df = pd.read_csv(a.csv).dropna(subset=["vina_score","label"])
    df = df.sort_values("vina_score", ascending=True).reset_index(drop=True)
    lab = df["label"].astype(int).values

    auc = roc_auc_score(lab, -df["vina_score"].values)
    ef1, ef5 = enrichment_factor(lab, 0.01), enrichment_factor(lab, 0.05)
    bd = bedroc(lab, 20.0)
    print(f"N = {len(df)}  ({lab.sum()} actives / {(lab==0).sum()} decoys)")
    print(f"ROC-AUC = {auc:.3f}")
    print(f"EF 1%   = {ef1:.2f}")
    print(f"EF 5%   = {ef5:.2f}")
    print(f"BEDROC(a=20) = {bd:.3f}")
    fpr,tpr,_ = roc_curve(lab, -df["vina_score"].values)
    plt.figure(figsize=(5,5)); plt.plot(fpr,tpr,color="#1a7f7f",lw=2,label=f"AUC={auc:.3f}")
    plt.plot([0,1],[0,1],"--",color="grey"); plt.xlabel("False positive rate")
    plt.ylabel("True positive rate"); plt.title("AcrB DBP docking enrichment")
    plt.legend(); plt.tight_layout(); plt.savefig("enrichment_roc.png", dpi=200)
    print("Wrote enrichment_roc.png")

if __name__ == "__main__":
    main()
```

11_seed_robustness.py

```
# Re-docks the top compounds at three independent random seeds and measures rank agreement and hit-set overlap.

import os, csv, subprocess, math

VINA   = "vina.exe"
CONFIG = "config_Tstate.txt"
LIGDIR = "ligands_prepped"
RESULTS= "results_Tstate.csv"
OUT    = "robustness_scores.csv"
TOPN   = 50
SEEDS  = [101, 202, 303]
HITSET = 17

def best_affinity(stdout):
    for line in stdout.splitlines():
        p=line.split()
        if len(p)>=2 and p[0]=="1":
            try: return float(p[1])
            except ValueError: pass
    return None
def load_top():
```

```
    rows=[(r[0],float(r[1])) for r in csv.reader(open(RESULTS)) if r and r[0].startswith("IMPHY")]
    rows.sort(key=lambda x:x[1])
    return [lid for lid,_ in rows[:TOPN]]

def done_set():
    d=set()
    if os.path.exists(OUT):
        for r in csv.reader(open(OUT)):
            if r and r[0]!="ligand_id":
                try: d.add((r[0],int(r[1])))
                except (ValueError, IndexError): continue
    return d

def spearman(a,b):
    ids=sorted(set(a)&set(b))
    ra={x:i for i,x in enumerate(sorted(ids,key=lambda z:a[z]))}
    rb={x:i for i,x in enumerate(sorted(ids,key=lambda z:b[z]))}
    n=len(ids); d2=sum((ra[x]-rb[x])**2 for x in ids)
    return 1-6*d2/(n*(n*n-1)) if n>1 else float('nan')

def main():
    if not all(os.path.exists(x) for x in (VINA,CONFIG,LIGDIR,RESULTS)):
        print("Missing one of:", VINA, CONFIG, LIGDIR, RESULTS); return
    top=load_top(); done=done_set()
    if not os.path.exists(OUT): open(OUT,"w").write("ligand_id,seed,affinity\n")
    total=len(top)*len(SEEDS); i=0
    for seed in SEEDS:
        for lid in top:
            i+=1
            if (lid,seed) in done: continue
            lig=os.path.join(LIGDIR,lid+".pdbqt")
            cmd=[VINA,"--config",CONFIG,"--ligand",lig,"--seed",str(seed),
                 "--out",os.path.join("docking","out_Tstate",f"{lid}_s{seed}.pdbqt")]
            try:
                r=subprocess.run(cmd,capture_output=True,text=True,timeout=600)
                aff=best_affinity(r.stdout)
            except Exception as e:
                aff=None; print("ERR",lid,seed,e)
            if aff is not None:
                with open(OUT,"a") as f: f.write(f"{lid},{seed},{aff}\n")
            if i%20==0: print(f"  {i}/{total}  {lid} seed {seed} -> {aff}")

    data={s:{} for s in SEEDS}
    for r in csv.reader(open(OUT)):
        if r and r[0]!="ligand_id":
            try: data[int(r[1])][r[0]]=float(r[2])
            except (ValueError, IndexError, KeyError): continue
    print("\n=== ROBUSTNESS ACROSS SEEDS ===")
    seeds=[s for s in SEEDS if data[s]]
    for i in range(len(seeds)):
        for j in range(i+1,len(seeds)):
            print(f"Spearman rho (seed {seeds[i]} vs {seeds[j]}) =
        {spearman(data[seeds[i]],data[seeds[j]]):.3f}")
    tops=[set(sorted(data[s],key=lambda z:data[s][z])[:HITSET]) for s in seeds]
    for i in range(len(seeds)):
        for j in range(i+1,len(seeds)):
            inter=len(tops[i]&tops[j]); uni=len(tops[i]|tops[j])
            print(f"Top-{HITSET} Jaccard (seed {seeds[i]} vs {seeds[j]}) = {inter/uni:.3f}  ({inter}/{HITSET}
        shared)")

    allids=set().union(*[set(data[s]) for s in seeds])
    spreads=[]
    for lid in allids:
        vals=[data[s][lid] for s in seeds if lid in data[s]]
        if len(vals)>1:
            m=sum(vals)/len(vals); sd=(sum((v-m)**2 for v in vals)/len(vals))**0.5; spreads.append(sd)
    if spreads: print(f"\nmean per-compound score spread across seeds = {sum(spreads)/len(spreads):.3f}
        kcal/mol")
    print("Done -> robustness_scores.csv")

if __name__=="__main__":
    main()
```

12_extract_md_snapshots.py

```
# Extracts five protein conformers from the 100 ns MD trajectory for ensemble docking.

import pytraj as pt, os

D      = "/content/drive/MyDrive/AcrB_MD"
TOP    = f"{D}/SYS_gaff2_nw.prmtop"
TRAJ   = f"{D}/prot_lig_prod1-20_nw.dcd"
OUTDIR = f"{D}/snapshots"
PROT   = ":1-1032"

os.makedirs(OUTDIR, exist_ok=True)

traj = pt.load(TRAJ, TOP)
print(f"loaded: {traj.n_frames} frames, {traj.n_atoms} atoms, "
      f"{traj.top.n_residues} residues")

prot = traj[PROT]
print(f"protein selection {PROT}: {prot.n_atoms} atoms, "
      f"{prot.top.n_residues} residues   (expect ~1032)")

n = prot.n_frames
frames = [int(round(f * (n - 1))) for f in (0.20, 0.40, 0.60, 0.80, 1.00)]
print("extracting frames:", frames, "  (~20/40/60/80/100 ns)")

for k, fr in enumerate(frames, 1):
    out = os.path.join(OUTDIR, f"snap{k}.pdb")
    pt.write_traj(out, prot, frame_indices=[fr], overwrite=True, options="pdb")
    print(f"  snap{k}.pdb  <- frame {fr}")

print("\nDone. Download the 5 snap*.pdb files from Drive into:")
print(r"   C:\Users\prano\OneDrive\Desktop\Acrb_Project\snapshots\ ")
print("Then run:  make_ensemble_receptors.py   ->   dock_ensemble.py")
```

13_build_ensemble_receptors.py

```
# Builds docking-ready receptors from the MD snapshots so that atom typing is identical across conformers.

import os, glob, sys
import numpy as np

TEMPLATE_PDBQT = "model_01.pdbqt"
SNAPDIR        = "snapshots"
BOX_CENTER     = np.array([35.75, -60.85, -41.81])
DBP_RESIDUES   = [136, 178, 610, 615, 617, 628]
MIN_MATCH_RATE = 0.95

CANON = {"HIE":"HIS","HID":"HIS","HIP":"HIS","CYX":"CYS","CYM":"CYS",
         "ASH":"ASP","GLH":"GLU","LYN":"LYS"}
def canon(rn):
    rn = rn.strip().upper()
    if len(rn) == 4 and rn[0] in "NC":
        rn = rn[1:]
    return CANON.get(rn, rn)

def parse_pdbqt(path):
    lines, rec = [], []
    for ln in open(path):
        lines.append(ln.rstrip("\n"))
        if ln.startswith(("ATOM", "HETATM")):
            rec.append({
                "i":    len(lines) - 1,
                "name": ln[12:16].strip(),
                "res":  ln[17:20].strip(),
                "seq":  int(ln[22:26]),
                "xyz":  np.array([float(ln[30:38]), float(ln[38:46]), float(ln[46:54])]),
                "type": ln[77:79].strip() if len(ln) >= 79 else "",
            })
    return lines, rec

def parse_pdb(path):
    rec = []
    for ln in open(path):
```

```
        if ln.startswith(("ATOM", "HETATM")):
            rec.append({
                "name": ln[12:16].strip(),
                "res":  ln[17:20].strip(),
                "seq":  int(ln[22:26]),
                "xyz":  np.array([float(ln[30:38]), float(ln[38:46]), float(ln[46:54])]),
            })
    return rec

def kabsch(P, Q):
    Pc, Qc = P - P.mean(0), Q - Q.mean(0)
    U, S, Vt = np.linalg.svd(Pc.T @ Qc)
    d = np.sign(np.linalg.det(Vt.T @ U.T))
    R = Vt.T @ np.diag([1, 1, d]) @ U.T
    return R, Q.mean(0) - R @ P.mean(0)

def best_offset(snap, model):
    s_ca = {r["seq"] for r in snap  if r["name"] == "CA"}
    m_ca = {r["seq"] for r in model if r["name"] == "CA"}
    if not s_ca or not m_ca:
        sys.exit("ERROR: no CA atoms found -- check the input files.")
    best, bo = -1, 0
    lo = min(m_ca) - max(s_ca)
    hi = max(m_ca) - min(s_ca)
    for o in range(lo, hi + 1):
        n = len({s + o for s in s_ca} & m_ca)
        if n > best:
            best, bo = n, o
    return bo, best, len(m_ca)

def main():
    if not os.path.exists(TEMPLATE_PDBQT):
        sys.exit(f"ERROR: {TEMPLATE_PDBQT} not found. Run from the Acrb_Project folder.")
    snaps = sorted(glob.glob(os.path.join(SNAPDIR, "snap*.pdb")))
    if not snaps:
        sys.exit(f"ERROR: no snap*.pdb in {SNAPDIR}/ -- run the Colab cell and "
                 f"download the files first.")

    tpl_lines, tpl = parse_pdbqt(TEMPLATE_PDBQT)
    tpl_heavy = [r for r in tpl if not r["type"].startswith("H")]
    print(f"template: {len(tpl)} atoms ({len(tpl_heavy)} heavy) from {TEMPLATE_PDBQT}\n")

    ref_pk = np.mean([r["xyz"] for r in tpl
                      if r["seq"] in DBP_RESIDUES and r["name"] == "CA"], axis=0)
    print(f"reference DBP CA centroid sits {np.linalg.norm(ref_pk - BOX_CENTER):.2f} A "
          f"from the box centre (expected offset -- CA atoms are behind the sidechains)\n")

    for sp in snaps:
        tag = os.path.splitext(os.path.basename(sp))[0]
        snap = parse_pdb(sp)
        off, n_ov, n_model_ca = best_offset(snap, tpl)

        s_ca = {r["seq"] + off: r["xyz"] for r in snap if r["name"] == "CA"}
        m_ca = {r["seq"]: r["xyz"] for r in tpl if r["name"] == "CA"}
        common = sorted(set(s_ca) & set(m_ca))
        R, t = kabsch(np.array([s_ca[i] for i in common]),
                      np.array([m_ca[i] for i in common]))

        lut, hs = {}, {}
        for r in snap:
            key = (r["seq"] + off, r["name"])
            xyz = R @ r["xyz"] + t
            lut[key] = xyz
            if r["name"][:1] == "H" or (len(r["name"]) > 1 and r["name"][0].isdigit()
                                        and r["name"][1] == "H"):
                hs.setdefault(r["seq"] + off, []).append([xyz, False])

        s_res = {r["seq"] + off: canon(r["res"]) for r in snap if r["name"] == "CA"}
        m_res = {r["seq"]: canon(r["res"]) for r in tpl if r["name"] == "CA"}
        bad = [i for i in common if s_res.get(i) != m_res.get(i)]

        out = list(tpl_lines)
        direct = viaH = missed = 0
        for r in tpl:
            key = (r["seq"], r["name"])
            xyz = lut.get(key)
            if xyz is not None:
```

```
                direct += 1
            elif r["type"].startswith("H"):

                par = min((h for h in tpl_heavy if h["seq"] == r["seq"]),
                          key=lambda h: np.sum((h["xyz"] - r["xyz"])**2), default=None)
                pxyz = lut.get((par["seq"], par["name"])) if par is not None else None
                if pxyz is not None:
                    cands = [c for c in hs.get(r["seq"], [])
                             if not c[1] and np.linalg.norm(c[0] - pxyz) < 1.35]
                    if cands:
                        c = min(cands, key=lambda c: np.linalg.norm(c[0] - pxyz))
                        c[1] = True; xyz = c[0]; viaH += 1
            if xyz is None:
                missed += 1
                continue
            ln = out[r["i"]]
            out[r["i"]] = (ln[:30] + f"{xyz[0]:8.3f}{xyz[1]:8.3f}{xyz[2]:8.3f}" + ln[54:])

        rate = (direct + viaH) / len(tpl)

        pk = [lut[(i, "CA")] for i in DBP_RESIDUES if (i, "CA") in lut]
        shift = np.linalg.norm(np.mean(pk, 0) - ref_pk) if pk else float("nan")

        outp = os.path.join(SNAPDIR, tag + ".pdbqt")
        if rate < MIN_MATCH_RATE:
            print(f"{tag}: MATCH RATE {rate:.1%} TOO LOW -- not written. "
                  f"direct={direct} viaH={viaH} missed={missed}")
            print("       -> send me this line and I'll fix the atom-name mapping.\n")
            continue
        open(outp, "w").write("\n".join(out) + "\n")

        print(f"{tag}:")
        print(f"   residue offset {off:+d}, {n_ov}/{n_model_ca} CA matched, "
              f"{len(bad)} residue-identity mismatches")
        print(f"   coordinates transferred: {rate:.1%} "
              f"(direct {direct}, via-parent H {viaH}, missed {missed})")
        print(f"   DBP pocket moved {shift:.2f} A relative to model_01 "
              f"({'OK -- well inside the 22 A box' if shift < 5 else 'WARNING -- re-centre the box'})")
        print(f"   -> {outp}\n")

    print("Next: run  dock_ensemble.py")

if __name__ == "__main__":
    main()
```

14_dock_ensemble.py

```
# Docks the conformer-ensemble compound set across the five receptors and reports rank correlations and hit
        retention.

import os, csv, glob, subprocess, sys, itertools, statistics as st

VINA    = "vina.exe"
CONFIG  = "config_Tstate.txt"
LIGDIR  = "ligands_prepped"
SNAPDIR = "snapshots"
REF     = "results_Tstate.csv"
OUT     = "ensemble_scores.csv"
OUTDIR  = os.path.join("docking", "out_ensemble")
TOPN    = 50
NSTRAT  = 100
HITSET  = 17

def best_affinity(stdout):
    for line in stdout.splitlines():
        p = line.split()
        if len(p) >= 2 and p[0] == "1":
            try:
                return float(p[1])
            except ValueError:
                pass
    return None

def load_ref():
    rows = [(r[0], float(r[1])) for r in csv.reader(open(REF))
```

```
            if r and r[0].startswith("IMPHY")]
    rows.sort(key=lambda x: x[1])
    return rows

def select_ligands(ref_rows):
    top = [lid for lid, _ in ref_rows[:TOPN]]
    rest = ref_rows[TOPN:]
    if NSTRAT > 0 and rest:
        step = len(rest) / float(NSTRAT)
        strat = [rest[min(len(rest) - 1, int(i * step))][0] for i in range(NSTRAT)]
    else:
        strat = []
    seen, out = set(), []
    for lid in top + strat:
        if lid not in seen:
            seen.add(lid); out.append(lid)
    return out, set(top)

def done_set():
    d = set()
    if os.path.exists(OUT):
        for r in csv.reader(open(OUT)):
            if r and r[0] != "ligand_id":
                try:
                    d.add((r[0], r[1]))
                except IndexError:
                    continue
    return d

def spearman(a, b, ids=None):
    ids = sorted((set(a) & set(b)) if ids is None else
                 [i for i in ids if i in a and i in b])
    n = len(ids)
    if n < 3:
        return float("nan")
    ra = {x: i for i, x in enumerate(sorted(ids, key=lambda z: a[z]))}
    rb = {x: i for i, x in enumerate(sorted(ids, key=lambda z: b[z]))}
    return 1 - 6 * sum((ra[x] - rb[x]) ** 2 for x in ids) / (n * (n * n - 1))

def make_config(snap_pdbqt, path):
    with open(path, "w") as f:
        for ln in open(CONFIG):
            if ln.strip().lower().startswith("receptor"):
                f.write(f"receptor = {snap_pdbqt}\n")
            else:
                f.write(ln)

def main():
    for need in (VINA, CONFIG, LIGDIR, REF, SNAPDIR):
        if not os.path.exists(need):
            sys.exit(f"MISSING: {need}  (run from the Acrb_Project folder)")
    snaps = sorted(glob.glob(os.path.join(SNAPDIR, "snap*.pdbqt")))
    if not snaps:
        sys.exit("No snap*.pdbqt -- run make_ensemble_receptors.py first.")

    os.makedirs(OUTDIR, exist_ok=True)
    ref_rows = load_ref()
    ref = {lid: sc for lid, sc in ref_rows}
    ligs, topset = select_ligands(ref_rows)
    done = done_set()
    if not os.path.exists(OUT):
        open(OUT, "w").write("ligand_id,conformer,affinity\n")

    sel = [ref[l] for l in ligs]
    total = len(ligs) * len(snaps)
    print(f"{len(ligs)} compounds ({TOPN} top + {len(ligs)-TOPN} range-spanning) "
          f"x {len(snaps)} conformers = {total} dockings")
    print(f"   selected rigid-score range: {min(sel):.2f} to {max(sel):.2f} kcal/mol")
    print(f"   {len(done)} already done, {total - len(done)} to run\n")

    i = 0
    for snap in snaps:
        tag = os.path.splitext(os.path.basename(snap))[0]
        cfg = os.path.join(SNAPDIR, f"config_{tag}.txt")
        make_config(snap, cfg)
        for lid in ligs:
            i += 1
```

```
        if (lid, tag) in done:
            continue
        lig = os.path.join(LIGDIR, lid + ".pdbqt")
        if not os.path.exists(lig):
            print("  missing ligand:", lig); continue
        try:
            r = subprocess.run(
                [VINA, "--config", cfg, "--ligand", lig,
                 "--out", os.path.join(OUTDIR, f"{lid}_{tag}.pdbqt")],
                capture_output=True, text=True, timeout=900)
            aff = best_affinity(r.stdout)
        except Exception as e:
            aff = None; print("  ERR", lid, tag, e)
        if aff is not None:
            with open(OUT, "a") as f:
                f.write(f"{lid},{tag},{aff}\n")
        if i % 50 == 0:
            print(f"  {i}/{total}  {lid} @ {tag} -> {aff}")

data = {}
for r in csv.reader(open(OUT)):
    if r and r[0] != "ligand_id":
        try:
            data.setdefault(r[1], {})[r[0]] = float(r[2])
        except (ValueError, IndexError):
            continue
confs = sorted(data)
docked = [l for l in ligs if any(l in data[c] for c in confs)]
top50 = [l for l in docked if l in topset]

print("\n" + "=" * 70)
print("ENSEMBLE DOCKING ACROSS MD-DERIVED T-STATE CONFORMERS")
print("=" * 70)

sc = [ref[l] for l in docked]
sds = [st.pstdev([data[c][l] for c in confs if l in data[c]])
       for l in docked if sum(l in data[c] for c in confs) > 1]
noise = sum(sds) / len(sds)
print(f"\ncompounds analysed: {len(docked)}  (of which {len(top50)} are primary-set)")
print(f"rigid-score range analysed: {min(sc):.2f} to {max(sc):.2f} "
      f"= {max(sc)-min(sc):.2f} kcal/mol")
print(f"mean conformer-to-conformer SD: {noise:.2f} kcal/mol")
print(f"signal-to-noise over the analysed range: {(max(sc)-min(sc))/noise:.1f}")

print("\n--- (1) BROAD DISCRIMINATION: full analysed range ---")
full = []
for c in confs:
    rho = spearman(ref, data[c], docked)
    full.append(rho)
    print(f"  rigid vs {c}:  rho = {rho:+.3f}  (n={sum(l in data[c] for l in docked)})")
if full:
    print(f"  mean rho across conformers = {sum(full)/len(full):+.3f}")

print("\n--- (2) FINE RANKING: within the near-tied top set only ---")
sub = []
for c in confs:
    rho = spearman(ref, data[c], top50)
    sub.append(rho)
    print(f"  rigid vs {c}:  rho = {rho:+.3f}")
if sub:
    print(f"  mean rho within top set = {sum(sub)/len(sub):+.3f}")

print("\n--- (3) DO THE CONFORMERS AGREE WITH EACH OTHER? ---")
print("    (if this is much higher than rigid-vs-conformer, the rigid")
print("     crystal-template geometry is the outlier, not random noise)")
inter_full, inter_top = [], []
for a, b in itertools.combinations(confs, 2):
    rf = spearman(data[a], data[b], docked)
    rt = spearman(data[a], data[b], top50)
    inter_full.append(rf); inter_top.append(rt)
    print(f"  {a} vs {b}:  full rho = {rf:+.3f}   top-set rho = {rt:+.3f}")
if inter_full:
    print(f"  mean inter-conformer rho: full = {sum(inter_full)/len(inter_full):+.3f}, "
          f"top-set = {sum(inter_top)/len(inter_top):+.3f}")

print("\n--- (4) CONSENSUS ENSEMBLE SCORING ---")
best = {l: min(data[c][l] for c in confs if l in data[c]) for l in docked}
```

```
    mean = {l: sum(data[c][l] for c in confs if l in data[c])
               / sum(l in data[c] for c in confs) for l in docked}
    rigid_top = set(sorted(ref, key=lambda z: ref[z])[:HITSET])
    for nm, d in (("ensemble-BEST (min across conformers)", best),
                  ("ensemble-MEAN", mean)):
        ov = len(set(sorted(d, key=lambda z: d[z])[:HITSET]) & rigid_top)
        print(f"\n  {nm}:  rho vs rigid = {spearman(ref, d, docked):+.3f}, "
              f"top-{HITSET} overlap with rigid = {ov}/{HITSET}")
        print("    top 10: " + ", ".join(sorted(d, key=lambda z: d[z])[:10]))

    print("\n--- (5) PER-HIT PERSISTENCE ---")
    tops = {c: set(sorted(data[c], key=lambda z: data[c][z])[:HITSET]) for c in confs}
    surv = []
    for lid in sorted(rigid_top, key=lambda z: ref[z]):
        if lid not in data[confs[0]] and not any(lid in data[c] for c in confs):
            continue
        k = sum(1 for c in confs if lid in tops[c])
        surv.append(k)
        vals = [data[c][lid] for c in confs if lid in data[c]]
        m = sum(vals) / len(vals)
        sd = st.pstdev(vals) if len(vals) > 1 else 0.0
        print(f"  {lid}  rigid {ref[lid]:>7.2f} | ensemble {m:>7.2f} +/- {sd:.2f} "
              f"| best {min(vals):>7.2f} | retained {k}/{len(confs)}")
    if surv:
        rob = sum(1 for k in surv if k >= (len(confs) + 1) // 2)
        print(f"\n  {rob}/{len(surv)} primary hits retained in a MAJORITY of conformers")
        print(f"  mean retention = {sum(surv)/len(surv):.1f}/{len(confs)}")

    print("\n" + "=" * 70)
    print("HOW TO READ THIS")
    print("=" * 70)
    print("Compare (1) and (2). The likely and entirely reportable outcome is that")
    print("BROAD discrimination survives (compounds that docked well still dock")
    print("better than compounds that docked badly) while FINE ranking inside the")
    print("near-tied top set does not. Report both numbers explicitly: it is a")
    print("precise, quantitative justification for presenting a prioritised")
    print("candidate SET rather than a ranked prediction -- which is the framing")
    print("the manuscript already uses.")
    print("\nUse (3) to distinguish real conformational sensitivity (conformers")
    print("agree with each other, disagree with the rigid model) from pure noise")
    print("(nothing agrees with anything).")
    print("\nUse (4) to name the conformationally robust candidates -- those are")
    print("the ones to put forward for experimental testing.")
    print(f"\nDone -> {OUT}")

if __name__ == "__main__":
    main()
```

15_build_ml_dataset.py

```
# Computes the 15 RDKit descriptors for every docked compound and writes the machine-learning dataset.

import csv
import os
import time
import urllib.request
import urllib.error

from rdkit import Chem
from rdkit.Chem import Descriptors, Lipinski, rdMolDescriptors
from rdkit import RDLogger
RDLogger.DisableLog("rdApp.*")

RESULTS   = "results.csv"
SDF_DIR   = "ligands/sdf"
OUT_CSV   = "ml_dataset.csv"
URL_TMPL  = "https://cb.imsc.res.in/imppat/images/2D/SDF/{id}.sdf"
DELAY     = 0.3
TIMEOUT   = 30

def compute_descriptors(mol):
    return {
        "MW":               Descriptors.MolWt(mol),
        "LogP":             Descriptors.MolLogP(mol),
        "TPSA":             Descriptors.TPSA(mol),
```

```
        "HBD":              Lipinski.NumHDonors(mol),
        "HBA":              Lipinski.NumHAcceptors(mol),
        "RotBonds":         Descriptors.NumRotatableBonds(mol),
        "AromaticRings":    rdMolDescriptors.CalcNumAromaticRings(mol),
        "RingCount":        rdMolDescriptors.CalcNumRings(mol),
        "HeavyAtoms":       mol.GetNumHeavyAtoms(),
        "FractionCSP3":     Descriptors.FractionCSP3(mol),
        "MolarRefractivity":Descriptors.MolMR(mol),
        "HeteroAtoms":      rdMolDescriptors.CalcNumHeteroatoms(mol),
        "SaturatedRings":   rdMolDescriptors.CalcNumSaturatedRings(mol),
        "AliphaticRings":   rdMolDescriptors.CalcNumAliphaticRings(mol),
        "QED":              Descriptors.qed(mol),
    }

def read_results(path):
    rows = []
    with open(path, newline="") as f:
        for row in csv.reader(f):
            if row and row[0] != "ligand_id":
                rows.append((row[0], float(row[1])))
    return rows

def download_sdf(cid, path, headers):
    if os.path.exists(path) and os.path.getsize(path) > 0:
        return True
    url = URL_TMPL.format(id=cid)
    try:
        req = urllib.request.Request(url, headers=headers)
        with urllib.request.urlopen(req, timeout=TIMEOUT) as resp:
            data = resp.read().decode("utf-8", errors="replace")
        if "$$$$" in data:
            with open(path, "w") as f:
                f.write(data)
            return True
    except Exception:
        pass
    return False

def main():
    if not os.path.exists(RESULTS):
        print(f"ERROR: {RESULTS} not found. Run from your Acrb_Project folder.")
        return
    os.makedirs(SDF_DIR, exist_ok=True)

    results = read_results(RESULTS)
    print(f"Read {len(results)} compounds from {RESULTS}")

    headers = {"User-Agent": "Mozilla/5.0 (academic research)"}
    rows_out = []
    dl_fail, parse_fail = [], []

    for i, (cid, affinity) in enumerate(results, 1):
        sdf_path = os.path.join(SDF_DIR, f"{cid}.sdf")

        if not download_sdf(cid, sdf_path, headers):
            dl_fail.append(cid)
            continue
        if not (os.path.exists(sdf_path) and os.path.getsize(sdf_path) > 0):
            dl_fail.append(cid)
            continue

        mol = Chem.MolFromMolFile(sdf_path)
        if mol is None:
            parse_fail.append(cid)
            continue

        desc = compute_descriptors(mol)
        desc["ligand_id"] = cid
        desc["affinity_kcal_mol"] = affinity
        rows_out.append(desc)

        if i % 50 == 0 or i == len(results):
            print(f"  {i}/{len(results)}  ok={len(rows_out)} dl_fail={len(dl_fail)}
        parse_fail={len(parse_fail)}")
        time.sleep(DELAY)

    if rows_out:
```

```
        desc_cols = [k for k in rows_out[0].keys() if k not in ("ligand_id", "affinity_kcal_mol")]
        fieldnames = ["ligand_id"] + desc_cols + ["affinity_kcal_mol"]
        with open(OUT_CSV, "w", newline="") as f:
            w = csv.DictWriter(f, fieldnames=fieldnames)
            w.writeheader()
            for r in rows_out:
                w.writerow(r)

    print("\n=== DONE ===")
    print(f"  compounds with full descriptors : {len(rows_out)}")
    print(f"  download failures               : {len(dl_fail)}")
    print(f"  RDKit parse failures            : {len(parse_fail)}")
    print(f"  saved -> {OUT_CSV}")
    if dl_fail or parse_fail:
        with open("ml_dataset_failures.txt", "w") as f:
            for c in dl_fail:    f.write(f"{c}\tdownload_failed\n")
            for c in parse_fail: f.write(f"{c}\tparse_failed\n")
        print("  (failures listed in ml_dataset_failures.txt)")

if __name__ == "__main__":
    main()
```

16_ml_scaffold_split.py

```
# Runs the scaffold-split classifier, the external test set and the held-out permutation importance.

import csv, sys, io
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import GroupKFold, StratifiedKFold, cross_val_predict
from sklearn.metrics import roc_auc_score, r2_score, mean_squared_error
from sklearn.inspection import permutation_importance
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

N_PERM      = 1000
N_REPEATS   = 20
EXT_FRAC    = 0.20
SEED        = 0

class Tee:
    def __init__(self, path):
        self.f = open(path, "w", encoding="utf-8"); self.t = sys.stdout
    def write(self, s): self.t.write(s); self.f.write(s)
    def flush(self): self.t.flush(); self.f.flush()
sys.stdout = Tee("ml_permimp_heldout_results.txt")

new = {r[0]: float(r[1]) for r in csv.reader(open("results_Tstate.csv"))
       if r and r[0] != "ligand_id"}
D = {r["ligand_id"]: r for r in csv.DictReader(open("ml_dataset.csv"))}
S = {r["ligand_id"]: r["murcko_scaffold"] for r in csv.DictReader(open("ligand_smiles.csv"))}

FEAT = ["MW","LogP","TPSA","HBD","HBA","RotBonds","AromaticRings","RingCount",
        "HeavyAtoms","FractionCSP3","MolarRefractivity","HeteroAtoms",
        "SaturatedRings","AliphaticRings","QED"]

ids = [i for i in D if i in S and S[i] and i in new]
X   = np.array([[float(D[i][f]) for f in FEAT] for i in ids])
y   = np.array([new[i] for i in ids])
g   = pd.factorize(np.array([S[i] for i in ids]))[0]

print("="*70)
print("ML RE-ANALYSIS (scaffold-split, held-out permutation importance)")
print("="*70)
print(f"{len(ids)} compounds, {len(set(g))} unique Murcko scaffolds")

rng = np.random.default_rng(SEED)
uniq_scaf = np.unique(g)
rng.shuffle(uniq_scaf)
n_ext = max(1, int(round(EXT_FRAC * len(uniq_scaf))))
ext_scaf = set(uniq_scaf[:n_ext].tolist())
is_ext = np.array([s in ext_scaf for s in g])
print(f"\nEXTERNAL TEST SET: {n_ext}/{len(uniq_scaf)} scaffolds held out "
```

```
      f"({is_ext.sum()}/{len(ids)} compounds). No CV loop ever sees these.")

lo, hi = np.percentile(y, 33), np.percentile(y, 67)
print(f"\nClass cutoffs:")
print(f"  33rd percentile = {lo:.3f} kcal/mol  (more-favorable docking-score class)")
print(f"  67th percentile = {hi:.3f} kcal/mol  (less-favorable docking-score class)")

m  = (y <= lo) | (y >= hi)
Xc, yc, gc, extc = X[m], (y[m] <= lo).astype(int), g[m], is_ext[m]
print(f"  {m.sum()} compounds in the two extreme classes "
      f"({(yc==1).sum()} favorable / {(yc==0).sum()} unfavorable)")

dev = ~extc
Xd, yd, gd = Xc[dev], yc[dev], gc[dev]
Xe, ye     = Xc[extc], yc[extc]

gkf = GroupKFold(5)

def cvauc(labels, Xin, gin, n_est=400):
    c = RandomForestClassifier(n_estimators=n_est, random_state=SEED, n_jobs=-1)
    p = cross_val_predict(c, Xin, labels, cv=gkf, groups=gin,
                          method="predict_proba", n_jobs=-1)[:, 1]
    return roc_auc_score(labels, p)

auc_scaf = cvauc(yd, Xd, gd)
clf_r = RandomForestClassifier(n_estimators=400, random_state=SEED, n_jobs=-1)
p_rand = cross_val_predict(clf_r, Xd, yd,
                           cv=StratifiedKFold(5, shuffle=True, random_state=SEED),
                           method="predict_proba", n_jobs=-1)[:, 1]
auc_rand = roc_auc_score(yd, p_rand)

print("\n" + "-"*70)
print("CLASSIFICATION")
print("-"*70)
print(f"random-split AUC   = {auc_rand:.3f}   (analogues can span folds -> optimistic)")
print(f"scaffold-split AUC = {auc_scaf:.3f}   (no scaffold shared across folds)")
print(f"leakage from random splitting = {auc_rand - auc_scaf:+.3f} AUC")

clf_ext = RandomForestClassifier(n_estimators=400, random_state=SEED, n_jobs=-1)
clf_ext.fit(Xd, yd)
auc_ext = roc_auc_score(ye, clf_ext.predict_proba(Xe)[:, 1])
print(f"EXTERNAL test-set AUC (unseen scaffolds, evaluated once) = {auc_ext:.3f}")

reg = RandomForestRegressor(n_estimators=400, random_state=SEED, n_jobs=-1)
yh  = cross_val_predict(reg, X[~is_ext], y[~is_ext], cv=GroupKFold(5),
                        groups=g[~is_ext], n_jobs=-1)
print("\n" + "-"*70)
print("REGRESSION")
print("-"*70)
print(f"scaffold-split R2   = {r2_score(y[~is_ext], yh):.3f}")
print(f"scaffold-split RMSE = {mean_squared_error(y[~is_ext], yh)**0.5:.3f} kcal/mol")

reg_ext = RandomForestRegressor(n_estimators=400, random_state=SEED, n_jobs=-1)
reg_ext.fit(X[~is_ext], y[~is_ext])
ye_pred = reg_ext.predict(X[is_ext])
print(f"EXTERNAL test-set R2   = {r2_score(y[is_ext], ye_pred):.3f}")
print(f"EXTERNAL test-set RMSE = {mean_squared_error(y[is_ext], ye_pred)**0.5:.3f} kcal/mol")

print("\n" + "-"*70)
print("PERMUTATION IMPORTANCE -- computed ONLY on held-out folds")
print("-"*70)
print("For each of 5 scaffold-grouped folds: fit on the 4 training folds, then")
print("permute each feature in the held-out fold and measure the AUC drop on")
print("that held-out fold. Values below are the mean +/- SD across the 5 folds.")

fold_imps = []
for k, (tr, te) in enumerate(gkf.split(Xd, yd, groups=gd), 1):
    mdl = RandomForestClassifier(n_estimators=400, random_state=SEED, n_jobs=-1)
    mdl.fit(Xd[tr], yd[tr])
    if len(np.unique(yd[te])) < 2:
        print(f"  fold {k}: skipped (single class in held-out fold)")
        continue
    r = permutation_importance(mdl, Xd[te], yd[te], scoring="roc_auc",
                               n_repeats=N_REPEATS, random_state=SEED, n_jobs=-1)
    fold_imps.append(r.importances_mean)
    print(f"  fold {k}: held-out n={len(te)}, base AUC="
          f"{roc_auc_score(yd[te], mdl.predict_proba(Xd[te])[:,1]):.3f}")
```

```
fold_imps = np.array(fold_imps)
imp_mean, imp_sd = fold_imps.mean(0), fold_imps.std(0)

order = np.argsort(imp_mean)
plt.figure(figsize=(8, 5))
plt.barh([FEAT[i] for i in order], imp_mean[order], xerr=imp_sd[order],
         color="#1a7f7f", ecolor="#444444", capsize=3)
plt.xlabel("Permutation importance on held-out folds (mean AUC drop)")
plt.axvline(0, color="k", lw=0.8)
plt.tight_layout()
plt.savefig("figure2_permutation_Tstate.png", dpi=200)

print("\nRanked held-out permutation importances (mean AUC drop +/- SD across folds):")
for i in order[::-1]:
    print(f"  {FEAT[i]:<18} {imp_mean[i]:+.4f} +/- {imp_sd[i]:.4f}")
print("\n-> figure2_permutation_Tstate.png rewritten (held-out version).")

print("\n" + "-"*70)
print(f"PERMUTATION TEST (N = {N_PERM})")
print("-"*70)
rng2 = np.random.default_rng(42)
null = np.array([cvauc(rng2.permutation(yd), Xd, gd, n_est=120)
                 for _ in range(N_PERM)])
exc = int((null >= auc_scaf).sum())
p   = (exc + 1) / (N_PERM + 1)
print(f"observed scaffold-split AUC = {auc_scaf:.3f}")
print(f"null distribution: mean {null.mean():.3f}, SD {null.std():.3f}, max {null.max():.3f}")
print(f"exceedance count = {exc}/{N_PERM}")
print(f"p = ({exc} + 1) / ({N_PERM} + 1) = {p:.4g}")
if exc == 0:
    print(f"NOTE: zero exceedances, so p is at the 1/{N_PERM+1} floor "
          f"= {1/(N_PERM+1):.4g}; report it as p < {1/(N_PERM+1):.3g}, not as an exact value.")

print("\n" + "="*70)
print("DONE. Everything above is saved in ml_permimp_heldout_results.txt")
print("="*70)
```

17_permutation_test.py

```
# Runs the 1000-permutation significance test on the scaffold-split classifier.

import os

for _v in ("OMP_NUM_THREADS", "OPENBLAS_NUM_THREADS", "MKL_NUM_THREADS",
           "NUMEXPR_NUM_THREADS", "VECLIB_MAXIMUM_THREADS"):
    os.environ[_v] = "1"

import csv, time, sys
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GroupKFold
from sklearn.metrics import roc_auc_score

N_PERM   = 1000
N_TREES  = 120
EXT_FRAC = 0.20
SEED     = 0
CKPT     = "perm_null.csv"

new = {r[0]: float(r[1]) for r in csv.reader(open("results_Tstate.csv"))
       if r and r[0] != "ligand_id"}
D = {r["ligand_id"]: r for r in csv.DictReader(open("ml_dataset.csv"))}
S = {r["ligand_id"]: r["murcko_scaffold"] for r in csv.DictReader(open("ligand_smiles.csv"))}
FEAT = ["MW","LogP","TPSA","HBD","HBA","RotBonds","AromaticRings","RingCount",
        "HeavyAtoms","FractionCSP3","MolarRefractivity","HeteroAtoms",
        "SaturatedRings","AliphaticRings","QED"]

ids = [i for i in D if i in S and S[i] and i in new]
X = np.array([[float(D[i][f]) for f in FEAT] for i in ids])
y = np.array([new[i] for i in ids])
g = pd.factorize(np.array([S[i] for i in ids]))[0]

rng = np.random.default_rng(SEED)
uniq = np.unique(g); rng.shuffle(uniq)
```

```
ext = set(uniq[:max(1, int(round(EXT_FRAC * len(uniq))))].tolist())
is_ext = np.array([s in ext for s in g])

lo, hi = np.percentile(y, 33), np.percentile(y, 67)
m = (y <= lo) | (y >= hi)
Xc, yc, gc, extc = X[m], (y[m] <= lo).astype(int), g[m], is_ext[m]
dev = ~extc
Xd, yd, gd = Xc[dev], yc[dev], gc[dev]

print(f"{len(ids)} compounds | development set for the test: {len(yd)} "
      f"({(yd==1).sum()} favorable / {(yd==0).sum()} unfavorable), "
      f"{len(set(gd))} scaffolds")

folds = list(GroupKFold(5).split(Xd, yd, groups=gd))

def cvauc(labels, n_trees=N_TREES):
    oof = np.zeros(len(labels), dtype=float)
    for tr, te in folds:
        mdl = RandomForestClassifier(n_estimators=n_trees, random_state=SEED,
                                     n_jobs=1)
        mdl.fit(Xd[tr], labels[tr])
        oof[te] = mdl.predict_proba(Xd[te])[:, 1]
    return roc_auc_score(labels, oof)

t0 = time.time()
auc_obs = cvauc(yd, n_trees=400)
print(f"observed scaffold-split AUC = {auc_obs:.3f}   "
      f"[{time.time()-t0:.1f}s for the 400-tree reference fit]")

null = []
if os.path.exists(CKPT):
    null = [float(r[0]) for r in csv.reader(open(CKPT)) if r]
    print(f"resuming: {len(null)} permutations already on disk")
else:
    open(CKPT, "w").close()

rng2 = np.random.default_rng(42)
for _ in range(len(null)):
    rng2.permutation(yd)

start_n = len(null)
t0 = time.time()
try:
    while len(null) < N_PERM:
        v = cvauc(rng2.permutation(yd))
        null.append(v)
        with open(CKPT, "a") as f:
            f.write(f"{v}\n")
        n = len(null)
        this_session = n - start_n
        if this_session <= 5 or this_session % 20 == 0:
            rate = (time.time() - t0) / this_session
            eta = (N_PERM - n) * rate
            print(f"  {n}/{N_PERM}  {rate:.2f}s/perm  ETA {eta/60:.1f} min  "
                  f"(exceedances so far: {sum(1 for x in null if x >= auc_obs)})")
except KeyboardInterrupt:
    print(f"\nstopped early at N = {len(null)} -- that is still a valid test.")

null = np.array(null)
N = len(null)
exc = int((null >= auc_obs).sum())
p = (exc + 1) / (N + 1)

print("\n" + "=" * 60)
print(f"PERMUTATION TEST (N = {N})")
print("=" * 60)
print(f"observed scaffold-split AUC = {auc_obs:.3f}")
print(f"null: mean {null.mean():.3f}, SD {null.std():.3f}, "
      f"min {null.min():.3f}, max {null.max():.3f}")
print(f"95th percentile of null = {np.percentile(null, 95):.3f}")
print(f"exceedance count = {exc}/{N}")
print(f"p = ({exc} + 1) / ({N} + 1) = {p:.4g}")
if exc == 0:
    print(f"\nNOTE: zero exceedances -> p is at the 1/{N+1} floor. Report it as")
    print(f"      'p < {1/(N+1):.3g} (0/{N} permutations reached the observed AUC)'")
    print(f"      rather than as an exact value. Stating the exceedance count is")
with open("permutation_test_results.txt", "w") as f:
```

```
    f.write(f"observed AUC {auc_obs:.4f}\nN {N}\nexceedance {exc}\np {p:.6g}\n"
            f"null_mean {null.mean():.4f}\nnull_sd {null.std():.4f}\n"
            f"null_max {null.max():.4f}\n")
print("\nsaved -> permutation_test_results.txt")
```

18_binding_site_contacts.py

```
# Regenerates the binding-site contact table from the deposited docked poses at a 4 A heavy-atom cutoff.

import csv, os, sys

CUTOFF   = 4.0
RECEPTOR = "model_01.pdb"
POSEDIR  = os.path.join("docking", "out_Tstate")
OUT      = "contact_table_Tstate_regenerated.csv"

HITS = ["IMPHY001793", "IMPHY000398", "IMPHY005217", "IMPHY004477", "IMPHY005683"]

DBP = {46, 89, 128, 136, 139, 176, 178, 273, 277, 279, 563, 566, 570,
       573, 610, 612, 615, 617, 620, 626, 628}

def ligand_heavy_atoms(path):
    pts, started = [], False
    for line in open(path):
        if line.startswith("MODEL"):
            started = True
            continue
        if line.startswith("ENDMDL") and started:
            break
        if line.startswith(("ATOM", "HETATM")):
            if line[77:79].strip().startswith("H"):
                continue
            pts.append((float(line[30:38]), float(line[38:46]), float(line[46:54])))
    return pts

def receptor_residues(path):
    res = {}
    for line in open(path):
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip().startswith("H"):
            continue
        key = (int(line[22:26]), line[17:20].strip())
        res.setdefault(key, []).append(
            (float(line[30:38]), float(line[38:46]), float(line[46:54])))
    return res

def contacts(res, lig, cut):
    c2 = cut * cut
    out = set()
    for (num, name), atoms in res.items():
        for a in atoms:
            if any((a[0]-b[0])**2 + (a[1]-b[1])**2 + (a[2]-b[2])**2 <= c2 for b in lig):
                out.add((num, name))
                break
    return out

def main():
    if not os.path.exists(RECEPTOR):
        sys.exit(f"MISSING {RECEPTOR} -- run from the Acrb_Project folder")
    res = receptor_residues(RECEPTOR)

    found = {}
    for h in HITS:
        p = os.path.join(POSEDIR, f"{h}_out.pdbqt")
        if not os.path.exists(p):
            sys.exit(f"MISSING pose file {p}")
        found[h] = contacts(res, ligand_heavy_atoms(p), CUTOFF)

    allres = sorted({r for s in found.values() for r in s}, key=lambda z: z[0])

    with open(OUT, "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["Residue", "In_DBP"] + HITS + ["N_hits"])
        for num, name in allres:
```

```
            row = [f"{name}{num}", "yes" if num in DBP else ""]
            n = 0
            for h in HITS:
                hit = (num, name) in found[h]
                row.append("X" if hit else "")
                n += hit
            w.writerow(row + [n])

    print(f"{len(allres)} contacting residues at a {CUTOFF} A heavy-atom cutoff")
    print(f"  in the DBP     : {sum(1 for n,_ in allres if n in DBP)}")
    print(f"  outside the DBP: {sum(1 for n,_ in allres if n not in DBP)}")
    universal = [f"{nm}{n}" for n, nm in allres
                 if all((n, nm) in found[h] for h in HITS)]
    print(f"  contacted by all five hits: {', '.join(universal)}")
    print(f"\nwritten -> {OUT}")
    print("Compare against contact_table_Tstate.csv; they should be identical.")

if __name__ == "__main__":
    main()
```

19_figure2_permutation_importance.py

```
# Draws Figure 2, the held-out permutation importance chart.

import os
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
OUT = os.path.join("manuscript_assets", "Figure2_permutation_importance.png")

DATA = [
    ("RingCount",          0.0250, 0.0091),
    ("LogP",               0.0236, 0.0098),
    ("RotBonds",           0.0156, 0.0060),
    ("MolarRefractivity",  0.0137, 0.0122),
    ("FractionCSP3",       0.0101, 0.0049),
    ("HeteroAtoms",        0.0086, 0.0041),
    ("AromaticRings",      0.0080, 0.0023),
    ("HBA",                0.0049, 0.0040),
    ("AliphaticRings",     0.0029, 0.0044),
    ("TPSA",               0.0015, 0.0023),
    ("HeavyAtoms",         0.0006, 0.0074),
    ("QED",               -0.0010, 0.0017),
    ("MW",                -0.0011, 0.0060),
    ("SaturatedRings",    -0.0015, 0.0033),
    ("HBD",               -0.0017, 0.0018),
]

TEAL = "#1b7f79"
GREY = "#c3c8cc"
EDGE = "#4d5358"

plt.rcParams.update({
    "font.family": "DejaVu Sans",
    "font.size": 13,
    "axes.linewidth": 1.2,
    "xtick.major.width": 1.2,
    "ytick.major.width": 1.2,
})

names = [d[0] for d in DATA][::-1]
means = [d[1] for d in DATA][::-1]
sds   = [d[2] for d in DATA][::-1]

solid = [(m - s) > 0 for m, s in zip(means, sds)]
colors = [TEAL if s else GREY for s in solid]

fig, ax = plt.subplots(figsize=(9.0, 6.4))
y = range(len(names))
ax.barh(y, means, xerr=sds, color=colors, edgecolor="none",
        error_kw=dict(ecolor=EDGE, elinewidth=1.3, capsize=3.5, capthick=1.3),
        height=0.68, zorder=3)
ax.axvline(0, color="black", linewidth=1.1, zorder=4)

ax.set_yticks(list(y))
```

```
ax.set_yticklabels(names)
ax.set_ylim(-0.75, len(names) - 0.25)
ax.set_xlabel("Permutation importance on held-out folds\n(mean AUC drop ± SD across 5 scaffold-grouped folds)",
              labelpad=10)
ax.tick_params(axis="y", length=0)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.grid(axis="x", color="#e6e9eb", linewidth=0.9, zorder=0)
ax.set_axisbelow(True)

handles = [plt.Rectangle((0, 0), 1, 1, color=TEAL),
           plt.Rectangle((0, 0), 1, 1, color=GREY)]
ax.legend(handles,
          ["interval excludes zero", "interval includes zero"],
          loc="lower right", frameon=False, fontsize=12,
          handlelength=1.3, borderaxespad=0.6)

fig.tight_layout()
os.makedirs("manuscript_assets", exist_ok=True)
fig.savefig(OUT, dpi=300, bbox_inches="tight", facecolor="white")
print(f"written -> {OUT}")
```

20_figure3_inputs.py

```
# Prepares the pose and contact list used to render Figure 3.

import os

LIG_ID  = "IMPHY001793"
POSE    = os.path.join("docking", "out_Tstate", f"{LIG_ID}_out.pdbqt")
RECEPT  = "model_01.pdb"
OUTDIR  = os.path.join("manuscript_assets", "figure3")
CUTOFF  = 4.0

ADMAP = {"A": "C", "NA": "N", "NS": "N", "OA": "O", "OS": "O",
         "SA": "S", "HD": "H", "HS": "H"}

def top_pose_lines(path):
    out, inside = [], False
    for line in open(path):
        if line.startswith("MODEL"):
            inside = True
            continue
        if line.startswith("ENDMDL"):
            break
        if inside and line.startswith(("ATOM", "HETATM")):
            out.append(line.rstrip("\n"))
    return out

def to_pdb(lines):
    pdb = []
    for i, l in enumerate(lines, 1):
        adtype = l[77:79].strip()
        el = ADMAP.get(adtype, adtype)[:2]
        x, y, z = l[30:38], l[38:46], l[46:54]
        name = (el + str(i))[:4]
        pdb.append(f"HETATM{i:5d} {name:<4s}LIG A   1    "
                   f"{x}{y}{z}  1.00  0.00          {el:>2s}")
    pdb.append("END")
    return pdb

def receptor_residues(path):
    res = {}
    for l in open(path):
        if not l.startswith("ATOM"):
            continue
        if l[12:16].strip().startswith("H"):
            continue
        res.setdefault((int(l[22:26]), l[17:20].strip()), []).append(
            (float(l[30:38]), float(l[38:46]), float(l[46:54])))
    return res

THREE_TO_ONE = {"PHE": "Phe", "VAL": "Val", "GLN": "Gln", "LEU": "Leu",
                "ILE": "Ile", "ALA": "Ala", "GLY": "Gly", "TYR": "Tyr",
```

```
                "MET": "Met", "SER": "Ser", "THR": "Thr", "ASN": "Asn",
                "GLU": "Glu", "ASP": "Asp", "LYS": "Lys", "ARG": "Arg",
                "TRP": "Trp", "PRO": "Pro", "HIS": "His", "CYS": "Cys"}

def main():
    for f in (POSE, RECEPT):
        if not os.path.exists(f):
            raise SystemExit(f"MISSING {f} -- run from the Acrb_Project folder")
    os.makedirs(OUTDIR, exist_ok=True)

    lines = top_pose_lines(POSE)
    heavy = [(l[77:79].strip(), (float(l[30:38]), float(l[38:46]), float(l[46:54])))
             for l in lines if not l[77:79].strip().startswith("H")]
    pdb_path = os.path.join(OUTDIR, f"{LIG_ID}_pose.pdb")
    open(pdb_path, "w").write("\n".join(to_pdb(lines)) + "\n")
    print(f"pose -> {pdb_path}  ({len(lines)} atoms, {len(heavy)} heavy)")

    res = receptor_residues(RECEPT)
    c2 = CUTOFF * CUTOFF
    hits = []
    for (num, nm), atoms in res.items():
        if any((a[0]-b[0])**2 + (a[1]-b[1])**2 + (a[2]-b[2])**2 <= c2
               for a in atoms for _, b in heavy):
            hits.append((num, nm))
    hits.sort()

    labels = [f"{THREE_TO_ONE.get(nm, nm.title())}{num}" for num, nm in hits]
    sel = "resi " + "+".join(str(n) for n, _ in hits)

    txt = os.path.join(OUTDIR, "contacts.txt")
    with open(txt, "w") as f:
        f.write(f"{LIG_ID} (Glabroisoflavanone B), Vina top pose, "
                f"{CUTOFF} A heavy-atom cutoff\n")
        f.write(f"{len(hits)} contacting residues: {', '.join(labels)}\n\n")
        f.write("PyMOL selection:\n" + sel + "\n")
    print(f"{len(hits)} contacts: {', '.join(labels)}")
    print(f"contacts -> {txt}")

    pml = f"""# figure3.pml -- Figure 3, binding pose of Glabroisoflavanone B
# Run:  pymol -cq revision_scripts/../manuscript_assets/figure3/figure3.pml
# or open PyMOL, File > Run Script, and pick this file.
# Paths below are relative to the Acrb_Project folder, so START PYMOL THERE
# (File > Working Directory, or `cd` before launching).

reinitialize
bg_color white
set ray_opaque_background, 1
set orthoscopic, 1
set antialias, 2
set ray_shadows, 0
set cartoon_transparency, 0.55
set label_size, 20
set label_color, black
set label_font_id, 7
set dash_color, grey40
set dash_gap, 0.35
set dash_width, 2.5

load {RECEPT}, receptor
load manuscript_assets/figure3/{LIG_ID}_pose.pdb, ligand

hide everything
show cartoon, receptor
color grey80, receptor

# --- pocket residues: the {len(hits)} within {CUTOFF} A of the pose ---
select pocket, receptor and ({sel}) and not (name C+N+O)
show sticks, pocket
color skyblue, pocket and elem C
util.cnc("pocket")

# --- ligand ---
show sticks, ligand
color yellow, ligand and elem C
util.cnc("ligand")
set stick_radius, 0.16, ligand
set stick_radius, 0.13, pocket
```

```
# --- polar contacts, drawn not asserted ---
distance polar, ligand, pocket, 3.5, mode=2
hide labels, polar

# --- labels, one per residue, on CA so they do not sit on top of the sticks ---
label receptor and ({sel}) and name CA, "%s%s" % (one_letter[resn], resi)

orient ligand
zoom ligand, 5
turn x, -10

# STOP HERE the first time. Rotate/zoom by hand until the pocket reads well,
# then run the two lines below from the PyMOL command bar.
ray 2400, 1800
png manuscript_assets/Figure3_binding_pose.png, dpi=300
"""
    pml_path = os.path.join(OUTDIR, "figure3.pml")
    open(pml_path, "w").write(pml)
    print(f"script -> {pml_path}")

if __name__ == "__main__":
    main()
```