Step 0 — Pre-model filtering and normalization

Before any physics-based modeling, the input PETase sequence set must be aggressively filtered and normalized, because no docking or MD protocol can rescue nonfunctional folds. All sequences are screened for the presence of the canonical PETase catalytic triad (Ser160–Asp206–His237 in IsPETase numbering), the PETase-specific disulfide bond (Cys203–Cys239 or structural equivalent), and a complete α/β-hydrolase fold; truncated sequences, partial domains, and proteins lacking these features are discarded. Signal peptides are predicted with SignalP and removed so that all sequences correspond to the mature enzyme. Redundant or near-identical sequences are clustered and reduced to a representative set. This step mirrors the filtering logic used by Erickson, Norton-Baker, and Alam, and typically removes a large fraction of false positives before any structural modeling.

```
# remove signal peptides

signalp6 --fastafile input.fasta --organism bacteria --output_dir
signalp_out

# extract mature sequences (example)

python extract_mature.py input.fasta signalp_out/summary.signalp6

# HMM search for PETase DLH domain

hmmsearch --cut_tc PF01738.hmm mature_sequences.fasta > hmm.out

# filter for catalytic residues (Ser160, Asp206, His237) and
disulfide Cys

python filter_petase.py mature_sequences.fasta hmm.out >
filtered.fasta

# filter_petase.py (sketch)

from Bio import SeqIO

for rec in SeqIO.parse("mature.fasta","fasta"):

    seq = str(rec.seq)

    if all(pos < len(seq) for pos in [159,205,236]) and seq[159]=="S"
and seq[205]=="D" and seq[236]=="H":

        print(f">{rec.id}\n{seq}")
```

Step 1 — Fix the PET substrate model once and permanently

A single substrate model must be selected and frozen for the entire pipeline to preserve comparability across variants. The appropriate choice for ranking PET-to-TPA activity is the hydroxyethyl-capped PET trimer, 2-HE(MHET)$_3$, which best approximates polymer-like geometry while remaining computationally tractable and is the substrate used (explicitly or implicitly) by Tournier, Garcia-Meseguer, Cui, and Burgin. The ligand is built once, geometry-optimized, parameterized consistently, and reused without modification for all enzymes and all conditions. BHET or MHET are not used for primary ranking because they represent downstream intermediates rather than PET itself.

```
# rdkit_build_pet_trimer.py

from rdkit import Chem

from rdkit.Chem import AllChem

smiles = "OCCOC(=O)c1ccc(cc1)C(=O)OCCOC(=O)c1ccc(cc1)C(=O)OCCO"

mol = Chem.MolFromSmiles(smiles)

mol = Chem.AddHs(mol)

AllChem.EmbedMolecule(mol)

AllChem.MMFFOptimizeMolecule(mol)

Chem.MolToMolFile(mol, "PET_trimer.mol")

# AmberTools parameterization (GAFF2 + AM1-BCC)

antechamber -i PET_trimer.mol -fi mol -o PET_trimer.mol2 -fo mol2 -c
bcc -at gaff2

parmchk2 -i PET_trimer.mol2 -f mol2 -o PET_trimer.frcmod
```

Step 2 — Anchor all modeling to a reference PETase–PET pose family

Docking does not begin from random placement. Instead, all modeling is anchored to a canonical PETase–terephthalate binding geometry derived from IsPETase–HEMT/PET-analog complexes reported in the literature. The terephthalate ring of the PET trimer is positioned in the aromatic clamp region defined by Y87, W159, W185, and F243 (IsPETase numbering), with the scissile ester carbonyl oriented toward Ser160 Oγ. This reference pose defines a “pose family” rather than a single rigid structure, allowing limited flexibility while preserving chemically meaningful geometry. All subsequent docking and refinement steps operate as constrained

variations around this anchor, which is why independent studies converge to nearly identical binding modes.

```
# align candidate PETase to IsPETase reference

tm_align candidate.pdb ispetase_ref.pdb > tmalign.out

# superimpose PET trimer from reference complex

import pymol

pymol.cmd.load("candidate.pdb")

pymol.cmd.load("ispetase_pet_ref.pdb")

pymol.cmd.align("candidate","ispetase_pet_ref")

pymol.cmd.save("candidate_with_pet.pdb")
```

Step 3 — Perform constrained docking focused on catalytic plausibility

For each PETase structure, constrained docking of the PET trimer is performed with the protein held rigid and the ligand flexible, using a docking box centered on the catalytic serine and large enough to accommodate the trimer. A moderate number of poses (on the order of 100–200) are generated per enzyme. Immediately after docking, hard geometric filters are applied: any pose in which the distance between the scissile ester carbonyl carbon and Ser160 Oγ exceeds approximately 4 Å is discarded, as are poses lacking reasonable aromatic stacking against the clamp residues or exhibiting severe steric clashes. Docking scores are recorded but treated as weak signals; the dominant outputs of this step are the number and quality of catalytically competent poses rather than the raw affinity value.

```
# prepare receptor and ligand

prepare_receptor4.py -r candidate.pdb -o candidate.pdbqt

prepare_ligand4.py -l PET_trimer.mol2 -o PET_trimer.pdbqt

# vina.conf

receptor = candidate.pdbqt

ligand = PET_trimer.pdbqt

center_x = <Ser160_Og_x>
```

```
center_y = <Ser160_Og_y>

center_z = <Ser160_Og_z>

size_x = 24

size_y = 24

size_z = 24

exhaustiveness = 20

num_modes = 200

vina --config vina.conf --out docked.pdbqt --log dock.log

# filter poses by catalytic distance

from Bio.PDB import *

parser = PDBParser()

for pose in poses:

    if ser_og.distance(ester_c) < 4.0:

        keep(pose)
```

Step 4 — Encode pH dependence via protonation-state modeling

To model activity under acidic and alkaline assay conditions without doubling the entire simulation cost, pH effects are introduced at the level of protein protonation. Each PETase structure is prepared twice using pKa prediction (e.g., PROPKA): once corresponding to pH 5.5 and once corresponding to pH 9.0. Particular attention is paid to the protonation state of the catalytic histidine and nearby titratable residues that influence electrostatics and hydrogen bonding in the active site. The constrained docking protocol is then repeated independently for each protonation state, producing two condition-specific sets of docking-derived geometric features per enzyme. This approach captures most of the pH sensitivity relevant for ranking without requiring separate long MD simulations for every condition.

```
# pH 5.5

propka3 candidate.pdb --pH 5.5

pdb2pqr --ff=amber candidate.pdb candidate_pH5.5.pdb
```

```
# pH 9.0

propka3 candidate.pdb --pH 9.0

pdb2pqr --ff=amber candidate.pH9.pdb

# repeat docking separately for each protonation state

vina --config vina_pH5.5.conf

vina --config vina_pH9.conf
```

Step 5 — Apply short MD refinement only to top candidates

Molecular dynamics refinement is reserved for a small, high-confidence subset of variants selected based on docking geometry and pose consistency, typically the top 1–3% per condition. For these candidates, the best docked pose is solvated and subjected to short explicit-solvent MD simulations on the order of 5–20 ns, sufficient to relax docking artifacts and test pose stability. During MD, the primary observables are not reaction energetics but structural stability metrics: persistence of the Ser–ester distance within catalytic range, maintenance of oxyanion-hole hydrogen bonds, ligand RMSD, and mobility of key loops such as those containing W159 and W185. This level of MD mirrors the refinement strategy used by Cui, Wu, and Burgin while remaining feasible at scale.

```
# tleap

source leaprc.protein.ff14SB

source leaprc.gaff2

loadamberparams PET_trimer.frcmod

PET = loadmol2 PET_trimer.mol2

REC = loadpdb candidate_with_pet.pdb

complex = combine {REC PET}

solvatebox complex TIP3PBOX 10.0

addions complex Na+ 0

saveamberparm complex complex.prmtop complex.inpcrd
```

```
# MD protocol (min → heat → short MD)

pmemd.cuda -O -i min.in -p complex.prmtop -c complex.inpcrd -o
min.out

pmemd.cuda -O -i heat.in -p complex.prmtop -c min.rst -o heat.out

pmemd.cuda -O -i md.in -p complex.prmtop -c heat.rst -o md.out

# analyze stability

cpptraj complex.prmtop <<EOF

trajin md.nc

rms ligand :PET

distance ser_ester :160@OG :PET@C

EOF
```

Step 6 — Optionally rescore refined poses with MM/GBSA

For MD-refined complexes, MM/GBSA calculations may be performed on a limited number of trajectory frames as an optional energetic rescore. These calculations provide an approximate binding free energy decomposition that can help distinguish otherwise similar candidates, but they are treated strictly as secondary features. Consistent with the PETase literature, MM/GBSA values are never interpreted as absolute measures of activity and are not used without supporting geometric and stability evidence.

```
MMPBSA.py -O \

  -i mmpbsa.in \

  -cp complex.prmtop \

  -rp receptor.prmtop \

  -lp ligand.prmtop \

  -y md.nc

# mmpbsa.in

&general
```

```
  startframe=1, endframe=500, interval=5

/

&gb

  igb=5, saltcon=0.1

/
```

Step 7 — Aggregate features into condition-specific activity rankings

For each PETase under each condition (pH 5.5 and pH 9.0), docking-derived geometry metrics, pose consistency measures, and—where available—MD stability and MM/GBSA features are combined into a single catalytic-competence score using a weighted scheme that emphasizes geometry and stability over raw binding energy. This score is a proxy for relative enzymatic activity rather than a direct prediction of μmol TPA produced. Enzymes are ranked separately for each condition, and the resulting rankings can be calibrated against experimentally characterized PETases to map scores onto expected relative TPA production. This final aggregation step reflects the consensus philosophy across the PETase field: physics-informed structural features, calibrated by experiment, are the most reliable way to rank large variant sets.

```
import pandas as pd

df = pd.read_csv("features.csv")
df["score"] = (
    -0.2 * df["vina_score"]
    -2.0 * df["ser_ester_dist"]
    +1.5 * df["pi_stack_score"]
    +1.0 * df.get("md_stability",0)
)

rank_pH55 = df[df.pH==5.5].sort_values("score", ascending=False)
rank_pH9  = df[df.pH==9.0].sort_values("score", ascending=False)

rank_pH55.to_csv("ranking_pH5.5.csv")
rank_pH9.to_csv("ranking_pH9.0.csv")
```

SNAKEMAKE THE PIPELINE

```
petase_pipeline/
├── Snakefile
├── config.yaml
├── envs/
│   ├── vina.yaml
│   ├── amber.yaml
│   ├── rdkit.yaml
│   └── propka.yaml
├── scripts/
│   ├── build_pet_trimer.py
│   ├── filter_docking.py
│   ├── extract_features.py
│   └── aggregate_scores.py
├── data/
│   ├── proteins/
│   │   ├── P001.pdb
│   │   ├── P002.pdb
│   │   └── ...
│   └── ligands/
│       └── PET_trimer.mol2
└── results/
```

```
proteins: "data/proteins/*.pdb"

ligand:
  name: PET_trimer
  mol2: data/ligands/PET_trimer.mol2

docking:
  box_size: 24
  exhaustiveness: 20
  poses: 200
  ser_resid: 160

conditions:
  - name: pH55
    pH: 5.5
  - name: pH9
    pH: 9.0

md:
  run: false        # set true for top hits only
```

```
  ns: 10
import glob
from pathlib import Path

configfile: "config.yaml"

PROTEINS = [Path(p).stem for p in glob.glob(config["proteins"])]
CONDS = [c["name"] for c in config["conditions"]]

rule all:
    input:
        expand("results/{cond}/scores.csv", cond=CONDS)

# ------------------------------
# Protonation (pH-specific)
# ------------------------------
rule protonate:
    input:
        pdb="data/proteins/{protein}.pdb"
    output:
        pdb="results/{cond}/prot/{protein}.pdb"
    params:
        pH=lambda wc: next(c["pH"] for c in config["conditions"] if
c["name"] == wc.cond)
    conda: "envs/propka.yaml"
    shell:
        """
        propka3 {input.pdb} --pH {params.pH}
        pdb2pqr --ff=amber {input.pdb} {output.pdb}
        """

# ------------------------------
# Docking
# ------------------------------
rule dock:
    input:
        pdb="results/{cond}/prot/{protein}.pdb",
        ligand=config["ligand"]["mol2"]
    output:
        pdbqt="results/{cond}/dock/{protein}.pdbqt",
        log="results/{cond}/dock/{protein}.log"
    conda: "envs/vina.yaml"
    shell:
        """
        prepare_receptor4.py -r {input.pdb} -o rec.pdbqt
```

```
        prepare_ligand4.py -l {input.ligand} -o lig.pdbqt

        vina \
          --receptor rec.pdbqt \
          --ligand lig.pdbqt \
          --center_x 0 --center_y 0 --center_z 0 \
          --size_x {config[docking][box_size]} \
          --size_y {config[docking][box_size]} \
          --size_z {config[docking][box_size]} \
          --exhaustiveness {config[docking][exhaustiveness]} \
          --num_modes {config[docking][poses]} \
          --out {output.pdbqt} \
          --log {output.log}
        """

# ------------------------------
# Catalytic filtering
# ------------------------------
rule filter_docking:
    input:
        dock="results/{cond}/dock/{protein}.pdbqt"
    output:
        filt="results/{cond}/filtered/{protein}.pdb"
    shell:
        """
        python scripts/filter_docking.py {input.dock} > {output.filt}
        """

# ------------------------------
# Feature extraction
# ------------------------------
rule features:
    input:
        pdb="results/{cond}/filtered/{protein}.pdb"
    output:
        csv="results/{cond}/features/{protein}.csv"
    shell:
        """
        python scripts/extract_features.py {input.pdb} > {output.csv}
        """

# ------------------------------
# Aggregate per condition
# ------------------------------
rule aggregate:
```

```
input:
    expand("results/{cond}/features/{protein}.csv",
           cond=lambda wc: wc.cond,
           protein=PROTEINS)
output:
    "results/{cond}/scores.csv"
shell:
    """
    python scripts/aggregate_scores.py {input} > {output}
    """
```