PRISM-4D Complete Parallel Worktree Development Blueprint

10 Worktrees — Pharma-Grade SOTA Pipeline

Source Documents Consolidated:

- FEP + GenPhore Implementation Blueprints
- PRISM-4D SOTA Platform Assessment (Pharma-Grade Readiness Evaluation)

TABLE OF CONTENTS

---

1. ARCHITECTURE OVERVIEW

```
~/Desktop/Prism4D-bio/                          ← MAIN (sota-dev branch,
READ-ONLY during sprint)
│
│   FOUNDATION (merge first)
├── ../wt-interfaces/                           ← WT-0: Interface contracts
+ shared types
```

```
├── ../wt-interfaces-v2/                    ← WT-9: Interface extensions
for new WTs
│
│   GENERATION + FILTERING (parallel)
├── ../wt-genphore/                         ← WT-1: Spike→Pharmacophore
+ PhoreGen/PGMG
├── ../wt-filters/                          ← WT-3: Multi-stage
filtering + ranking
│
│   PRE/POST PROCESSING (parallel with above)
├── ../wt-preprocessing/                    ← WT-5: Tautomers + membrane
embedding
├── ../wt-explicit-solvent/                 ← WT-6: Explicit solvent
refinement + water maps
│
│   VALIDATION (depends on above)
├── ../wt-fep/                              ← WT-2: OpenFE ABFE/RBFE
pipeline
├── ../wt-ensemble-scoring/                 ← WT-7: Ensemble rescoring +
P_open + IE
│
│   INTEGRATION (merge last)
├── ../wt-orchestrator/                     ← WT-4: End-to-end pipeline
+ reporting
└── ../wt-interactive-viewer/               ← WT-8: HTML5/WebGL
deliverable
```

---

2. SOTA GAP ANALYSIS

Cross-Reference: SOTA Evaluation vs. Worktree Coverage

#	SOTA Requirement	Eval Tier	Worktree	Status
1	ABFE/FEP Integration	Bleeding Edge	WT-2	✅ COVERED
2	Generative Molecule Design	Bleeding Edge	WT-1	✅ COVERED
3	Multi-Stage Filtering	Standard	WT-3	✅ COVERED
4	Publication Reporting	Standard	WT-4	✅ COVERED
5	Tautomer/Protomer Enumeration (pH 7.4)	MVP	WT-5	✅ COVERED
6	Explicit Solvent Refinement (TIP3P/OPC)	MVP	WT-6	✅ COVERED
7	Ensemble-Based Rescoring (MM-GBSA averaging)	MVP	WT-7	✅ COVERED
8	Water Map / Hydration Site Analysis	MVP	WT-6	✅ COVERED
9	Membrane Embedding (GPCRs/Ion Channels)	Market Expansion	WT-5	✅ COVERED
10	Cryptic Pocket P_open (transient analysis)	Bleeding Edge	WT-7	✅ COVERED
11	Interactive HTML5/WebGL Deliverable	Bleeding Edge	WT-8	✅ COVERED
12	Markov State Models for ensemble weighting	Bleeding Edge	WT-7	✅ COVERED
13	Interaction Entropy (replace NMA)	Standard	WT-7	✅ COVERED

All 13 SOTA requirements addressed. 0 gaps remaining.

Gap Severity Context

TIER 1 — MVP Blockers (Gaps 5-8): Without tautomers, explicit solvent, ensemble scoring, and water maps, the platform is stuck in "Academic/Screening" tier. Pharma clients will reject results. These are addressed by WT-5, WT-6, and WT-7.

TIER 2 — Market Expansion (Gap 9): ~60% of drug targets are membrane proteins. WT-5 opens the GPCR/ion channel market.

TIER 3 — Big Pharma Differentiation (Gaps 10-13): P_open, interactive viewer, MSMs, and IE separate PRISM from academic tools. Addressed by WT-7 and WT-8.

---

3. SETUP COMMANDS (ALL 10 WORKTREES)

```
cd ~/Desktop/Prism4D-bio

# Create ALL feature branches from sota-dev
git branch feat/interfaces         sota-dev
git branch feat/interfaces-v2      sota-dev
git branch feat/genphore           sota-dev
git branch feat/fep-pipeline       sota-dev
git branch feat/filters            sota-dev
git branch feat/orchestrator       sota-dev
git branch feat/preprocessing      sota-dev
git branch feat/explicit-solvent   sota-dev
git branch feat/ensemble-scoring   sota-dev
git branch feat/interactive-viewer sota-dev

# Create ALL worktrees
git worktree add ../wt-interfaces       feat/interfaces
git worktree add ../wt-interfaces-v2    feat/interfaces-v2
git worktree add ../wt-genphore         feat/genphore
git worktree add ../wt-filters          feat/filters
git worktree add ../wt-fep              feat/fep-pipeline
git worktree add ../wt-orchestrator     feat/orchestrator
git worktree add ../wt-preprocessing    feat/preprocessing
git worktree add ../wt-explicit-solvent feat/explicit-solvent
git worktree add ../wt-ensemble-scoring feat/ensemble-scoring
git worktree add ../wt-interactive-viewer feat/interactive-viewer
```

---

4. WT-0: INTERFACE CONTRACTS + SHARED TYPES

Branch: `feat/interfaces`

Duration: 2-3 days

Depends On: Nothing

Merge Target: `sota-dev` (merge BEFORE all others start coding)

Purpose

Define the data contracts between ALL worktrees so they can develop in parallel against stable interfaces without needing each other's code. Without stable interfaces, WT-1 doesn't know what format to output, WT-3 doesn't know what format to accept, WT-2 doesn't know what docking result looks like, and WT-4 can't wire anything together. This WT eliminates ALL integration risk.

File Ownership (EXCLUSIVE — no other WT touches these)

```
scripts/
├── interfaces/
│   ├── __init__.py
│   ├── spike_pharmacophore.py       # SpikePharmacophore dataclass
│   ├── generated_molecule.py        # GeneratedMolecule dataclass
│   ├── filtered_candidate.py        # FilteredCandidate dataclass
│   ├── docking_result.py            # DockingResult dataclass
(extends existing)
│   ├── fep_result.py                # FEPResult dataclass
│   ├── pipeline_config.py           # PipelineConfig (paths,
thresholds, flags)
│   └── residue_mapping.py           # ResidueMapping
(PDB↔Topology↔UniProt)
```

Key Interface Definitions

```
# spike_pharmacophore.py
@dataclass
class PharmacophoreFeature:
    feature_type: str          # AR, PI, NI, HBD, HBA, HY
    x: float; y: float; z: float
    intensity: float           # PRISM spike intensity (0-1)
    source_spike_type: str     # BNZ, TYR, CATION, ANION, etc.
```

```
    source_residue_id: int     # Topology residue ID
    source_residue_name: str   # e.g., "TYR142"
    wavelength_nm: float       # UV excitation wavelength
    water_density: float       # Local solvent accessibility

@dataclass
class ExclusionSphere:
    x: float; y: float; z: float
    radius: float              # Angstrom
    source_atom: str           # e.g., "CA:ALA145"

@dataclass
class SpikePharmacophore:
    target_name: str           # e.g., "KRAS_G12C"
    pdb_id: str
    pocket_id: int
    features: List[PharmacophoreFeature]
    exclusion_spheres: List[ExclusionSphere]
    pocket_centroid: Tuple[float, float, float]
    pocket_lining_residues: List[int]   # Topology IDs
    prism_run_hash: str        # SHA256 of PRISM binary + input
    creation_timestamp: str

    def to_phoregen_json(self) -> dict: ...
    def to_pgmg_posp(self) -> str: ...
    def to_docking_box(self, padding: float = 4.0) -> dict: ...


# generated_molecule.py
@dataclass
class GeneratedMolecule:
    smiles: str
    mol_block: str             # 3D SDF block
    source: str                # "phoregen" | "pgmg"
    pharmacophore_match_score: float
    matched_features: List[str]  # Which pharmacophore features
matched
    generation_batch_id: str


# filtered_candidate.py
@dataclass
class FilteredCandidate:
    molecule: GeneratedMolecule
    qed_score: float
    sa_score: float
    lipinski_violations: int
    pains_alerts: List[str]
    tanimoto_to_nearest_known: float
    nearest_known_cid: str     # PubChem CID
    cluster_id: int
    passed_all_filters: bool
```

```
    rejection_reason: Optional[str]


# fep_result.py
@dataclass
class FEPResult:
    compound_id: str
    delta_g_bind: float        # kcal/mol
    delta_g_error: float       # +/- kcal/mol
    method: str                # "ABFE" | "RBFE"
    n_repeats: int
    convergence_passed: bool
    hysteresis_kcal: float
    overlap_minimum: float     # Minimum lambda-window overlap
    max_protein_rmsd: float    # Angstrom during FEP
    restraint_correction: float
    charge_correction: float
    vina_score_deprecated: Optional[float]
    spike_pharmacophore_match: str  # "4/5 features within 2.0A"
    classification: str        # NOVEL_HIT | RECAPITULATED |
WEAK_BINDER | FAILED_QC
    raw_data_path: str         # Path to OpenFE output directory
```

Deliverable

- All dataclasses with serialization (JSON, pickle)
- Unit tests with mock data for each interface
- `scripts/interfaces/README.md` documenting every field
- **Merged to `sota-dev` before other WTs begin coding**

---

5. WT-9: INTERFACE EXTENSIONS (V2)

Branch: `feat/interfaces-v2`

Duration: 2-3 days

Depends On: WT-0 merged

Merge Target: `sota-dev` (merge BEFORE WT-5/6/7/8 start coding)

Purpose

Extend the WT-0 interface contracts with new dataclasses required by the continuation worktrees (WT-5 through WT-8). Does NOT modify any existing WT-0 interfaces — purely additive.

File Ownership (EXCLUSIVE)

```
scripts/
├── interfaces/
│   ├── tautomer_state.py            # TautomerState, TautomerEnsemble
│   ├── explicit_solvent_result.py   # ExplicitSolventResult
│   ├── water_map.py                 # HydrationSite, WaterMap
│   ├── ensemble_score.py            # EnsembleMMGBSA,
InteractionEntropy
│   ├── pocket_dynamics.py           # PocketDynamics (P_open,
lifetimes)
│   ├── membrane_system.py           # MembraneSystem,
LipidComposition
│   └── viewer_payload.py            # ViewerPayload (for interactive
HTML5)
```

Key Interface Definitions

```
# tautomer_state.py
@dataclass
class TautomerState:
    smiles: str                    # Canonical SMILES of this tautomer
    parent_smiles: str             # Original input SMILES
    protonation_ph: float          # pH at which this state was
generated
    charge: int                    # Net formal charge
    pka_shifts: List[Tuple[int, float]]  # (atom_idx, predicted_pKa)
    population_fraction: float     # Boltzmann weight at target pH
    source_tool: str               # "dimorphite_dl" | "openeye" |
"pkasolver"

@dataclass
class TautomerEnsemble:
    parent_smiles: str
    states: List[TautomerState]
    dominant_state: TautomerState  # Highest population at target pH
    target_ph: float               # Default 7.4
    enumeration_method: str


# explicit_solvent_result.py
@dataclass
class ExplicitSolventResult:
    pocket_id: int
    simulation_time_ns: float
    water_model: str               # "TIP3P" | "OPC" | "TIP4P-Ew"
    force_field: str               # "ff19SB" | "ff14SB" | "CHARMM36m"
```

```
    pocket_stable: bool            # Did pocket remain open?
    pocket_rmsd_mean: float        # Angstrom
    pocket_rmsd_std: float
    pocket_volume_mean: float      # Angstrom^3
    pocket_volume_std: float
    n_structural_waters: int       # Conserved waters (>80% occupancy)
    trajectory_path: str
    water_map: Optional['WaterMap']
    snapshot_frames: List[int]     # Frame indices used for ensemble
scoring


# water_map.py
@dataclass
class HydrationSite:
    x: float; y: float; z: float
    occupancy: float               # Fraction of simulation time
occupied
    delta_g_transfer: float        # kcal/mol (negative = happy,
positive = unhappy)
    entropy_contribution: float    # -TdS (kcal/mol)
    enthalpy_contribution: float   # dH (kcal/mol)
    n_hbonds_mean: float
    classification: str            # "CONSERVED_HAPPY" |
"CONSERVED_UNHAPPY" | "BULK"
    displaceable: bool             # True if dG_transfer > +1.0
kcal/mol

@dataclass
class WaterMap:
    pocket_id: int
    hydration_sites: List[HydrationSite]
    n_displaceable: int
    max_displacement_energy: float
    total_displacement_energy: float
    grid_resolution: float         # Angstrom (typically 0.5)
    analysis_frames: int


# ensemble_score.py
@dataclass
class EnsembleMMGBSA:
    compound_id: str
    delta_g_mean: float            # kcal/mol (ensemble-averaged)
    delta_g_std: float
    delta_g_sem: float
    n_snapshots: int
    snapshot_interval_ps: float
    decomposition: Dict[str, float]  # {"vdw": ..., "elec": ...,
"gb": ..., "sa": ...}
    per_residue_contributions: Dict[int, float]
    method: str                    # "MMGBSA_ensemble" |
```

```
"MMPBSA_ensemble"

@dataclass
class InteractionEntropy:
    compound_id: str
    minus_t_delta_s: float         # -TdS (kcal/mol)
    delta_h: float                 # dH (kcal/mol)
    delta_g_ie: float              # dH + (-TdS)
    n_frames: int
    convergence_block_std: float


# pocket_dynamics.py
@dataclass
class PocketDynamics:
    pocket_id: int
    p_open: float                  # 0-1
    p_open_error: float            # Bootstrap error
    mean_open_lifetime_ns: float
    mean_closed_lifetime_ns: float
    n_opening_events: int
    druggability_classification: str  # "STABLE_OPEN" | "TRANSIENT" |
"RARE_EVENT"
    volume_autocorrelation_ns: float
    msm_state_weights: Optional[Dict[int, float]]


# membrane_system.py
@dataclass
class MembraneSystem:
    lipid_composition: Dict[str, float]  # {"POPC": 0.7, "CHOL": 0.3}
    bilayer_method: str
    n_lipids: int
    membrane_thickness: float      # Angstrom
    protein_orientation: str       # "OPM" | "manual" | "PPM"
    opm_tilt_angle: float
    system_size: Tuple[float, float, float]
    total_atoms: int
    equilibration_protocol: str


# viewer_payload.py
@dataclass
class ViewerPayload:
    target_name: str
    pdb_structure: str             # PDB text
    pocket_surfaces: List[dict]
    spike_positions: List[dict]
    water_map_sites: List[dict]
    ligand_poses: List[dict]
    lining_residues: List[int]
    p_open: Optional[float]
```

```
    metadata: dict
```

Deliverable

- All new dataclasses with JSON serialization + unit tests
- Backward-compatible with WT-0 interfaces
- Merged to `sota-dev` before WT-5/6/7/8 begin coding

---

6. WT-1: SPIKE → PHARMACOPHORE + GENERATIVE MODELS

Branch: `feat/genphore`

Duration: Week 1-2

Depends On: WT-0 merged to sota-dev

Merge Target: `sota-dev`

Purpose

Convert PRISM spike output to pharmacophore format, run PhoreGen + PGMG generation, output `GeneratedMolecule` objects.

File Ownership (EXCLUSIVE)

```
scripts/
├── genphore/
│   ├── __init__.py
│   ├── spike_to_pharmacophore.py    # PRISM spike JSON → SpikePharmacophore
│   ├── run_phoregen.py              # PhoreGen wrapper
│   ├── run_pgmg.py                  # PGMG wrapper
│   ├── generate.py                  # Unified entry point (calls both)
│   └── README.md
│
tools/                                # External tool installations
├── PhoreGen/                         # git clone (gitignored)
└── PGMG/                            # git clone (gitignored)
│
envs/
├── phoregen.yml                      # conda env spec
```

```
└── pgmg.yml                          # conda env spec
│
tests/
└── test_genphore/
    ├── test_spike_to_pharmacophore.py
    ├── test_phoregen_wrapper.py
    ├── test_pgmg_wrapper.py
    └── fixtures/
        ├── mock_spike_output.json
        └── expected_pharmacophore.json
```

Key Implementation

`spike_to_pharmacophore.py`:

```
Input:  PRISM spike JSON (from nhs_rt_full output)
Output: SpikePharmacophore (from interfaces)

Steps:
1. Parse spike JSON -> extract per-type (BNZ/TYR/CATION/ANION/etc.)
2. For each type within detected pocket:
   - Compute intensity-weighted centroid
   - Map spike_type -> pharmacophore feature_type
   - Record source residue, wavelength, water_density
3. Generate exclusion spheres from pocket lining residue heavy atoms
4. Validate: >=2 features required, coordinates in protein reference
frame
5. Return SpikePharmacophore with .to_phoregen_json()
and .to_pgmg_posp()
```

`run_phoregen.py`:

```
Input:  SpikePharmacophore
Output: List[GeneratedMolecule]

Steps:
1. Write SpikePharmacophore.to_phoregen_json() to temp file
2. Call PhoreGen sampling (subprocess or Python import)
3. Parse output SDF -> validate with RDKit -> wrap as
GeneratedMolecule
4. Tag each with source="phoregen", pharmacophore_match_score
```

`run_pgmg.py`:

```
Input:  SpikePharmacophore (max 8 features -- truncate by intensity)
Output: List[GeneratedMolecule]
```

```
Steps:
1. Write SpikePharmacophore.to_pgmg_posp() to temp file
2. Call PGMG generate.py (subprocess)
3. Parse output SMILES -> 3D conformer via RDKit ETKDG -> wrap as
GeneratedMolecule
4. Tag each with source="pgmg"
```

Does NOT Touch

- `scripts/fep/` (WT-2), `scripts/filters/` (WT-3), `scripts/pipeline/` (WT-4)
- `scripts/gpu_dock.py` (existing), any PRISM Rust code

Deliverable

- Working `spike_to_pharmacophore.py` tested on KRAS/TEAD2 spike outputs
- PhoreGen generating 1000 molecules from PRISM pharmacophore on RTX 5080
- PGMG generating 10000 molecules as fast fallback
- All output as `List[GeneratedMolecule]` per interface spec
- Conda env YAMLs for reproducible setup

Integration Test

```
python scripts/genphore/generate.py \
    --spike-json snapshots/kras_site1/spikes.json \
    --output-dir /tmp/genphore_test/ \
    --n-phoregen 100 --n-pgmg 1000
# Produces: /tmp/genphore_test/molecules.sdf + molecules_meta.json
```

---

7. WT-3: MULTI-STAGE FILTERING + RANKING

Branch: `feat/filters`

Duration: Week 1-2 (parallel with WT-1)

Depends On: WT-0 merged to sota-dev

Merge Target: `sota-dev`

Purpose

Take raw `GeneratedMolecule` lists, apply 6-stage filtering cascade, output ranked `FilteredCandidate` list ready for docking and FEP.

File Ownership (EXCLUSIVE)

```
scripts/
├── filters/
│   ├── __init__.py
│   ├── filter_pipeline.py           # Main entry point
│   ├── stage1_validity.py           # RDKit sanitization
│   ├── stage2_druglikeness.py       # Lipinski, QED, SA score
│   ├── stage3_pains.py              # PAINS substructure alerts
│   ├── stage4_pharmacophore.py      # Re-validate 3D pharmacophore
overlap
│   ├── stage5_novelty.py            # Tanimoto vs PubChem/known
compounds
│   ├── stage6_diversity.py          # Butina clustering +
representative selection
│   ├── ranking.py                   # Multi-objective Pareto ranking
│   └── README.md
│
data/
├── pains_catalog.csv
└── reference_fingerprints/
│
tests/
└── test_filters/
    ├── test_filter_pipeline.py
    ├── test_each_stage.py
    └── fixtures/
        ├── mock_generated_molecules.json
        └── expected_filtered_output.json
```

Key Implementation

`filter_pipeline.py`:

```
Input:  List[GeneratedMolecule] + SpikePharmacophore + PipelineConfig
Output: List[FilteredCandidate] (ranked, top-N)

Pipeline: Stage 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> Rank

Each stage: takes list, returns filtered list, logs
N_in/N_out/rejection_reasons.
Never modifies molecules -- only annotates + filters.

Config: top_n=5, qed_threshold=0.3, sa_threshold=6.0,
        tanimoto_novelty_cutoff=0.85, diversity_cutoff=0.4,
        min_pharmacophore_matches=3,
pharmacophore_distance_tolerance=1.5A
```

`stage4_pharmacophore.py`**:** Re-validates 3D overlap with PRISM spike pharmacophore. For each molecule: identify features, compute 3D distance to spike centroids, require >=3/N features matched within 1.5A.

`stage5_novelty.py`**:** Morgan fingerprint (radius=2, 2048 bits) vs PubChem + patent databases. Reject if Tanimoto > 0.85 to any known compound.

`stage6_diversity.py`**:** Butina clustering (cutoff 0.4), select best representative per cluster, ensure >= ceil(N/2) distinct clusters in top-N.

Does NOT Touch

- `scripts/genphore/` (WT-1), `scripts/fep/` (WT-2), `scripts/pipeline/` (WT-4)

Deliverable

- 6-stage filter pipeline producing ranked `FilteredCandidate` list
- Each stage independently testable
- Full audit trail per molecule

Integration Test

```
python scripts/filters/filter_pipeline.py \
    --molecules /tmp/genphore_test/molecules_meta.json \
    --pharmacophore /tmp/genphore_test/pharmacophore.json \
    --top-n 5 --output /tmp/filter_test/candidates.json
```

---

8. WT-5: PRE-PROCESSING ENHANCEMENTS

Branch: `feat/preprocessing`

Duration: Week 1-2

Depends On: WT-9 merged to sota-dev

Merge Target: `sota-dev`

SOTA Gaps Closed: #5 (Tautomers), #9 (Membrane)

Purpose

Add biological context awareness: tautomer/protomer enumeration at physiological pH, automated membrane embedding for transmembrane targets, and PDB fixing.

File Ownership (EXCLUSIVE)

```
scripts/
├── preprocessing/
│   ├── __init__.py
│   ├── tautomer_enumeration.py      # Ligand protonation state
generation
│   ├── membrane_builder.py          # Automated lipid bilayer
embedding
│   ├── protein_fixer.py             # Loop modeling, missing residues
│   ├── target_classifier.py         # Soluble vs membrane auto-detect
│   └── README.md
│
envs/
└── preprocessing.yml
│
tests/
└── test_preprocessing/
    ├── test_tautomer_enumeration.py
    ├── test_membrane_builder.py
    ├── test_target_classifier.py
    └── fixtures/
        ├── ligand_with_ionizable_groups.sdf
        ├── beta2_adrenergic.pdb
        └── kras_soluble.pdb
```

Key Implementation

tautomer_enumeration.py

Tool: Dimorphite-DL (Apache 2.0, rule-based + pKa) + RDKit MolStandardize (canonical tautomers) + PROPKA (protein titratable residues).

```
Input:  SMILES or SDF (single or batch from PhoreGen/PGMG)
Output: TautomerEnsemble per ligand

Algorithm:
1. Canonicalize input (RDKit)
2. Enumerate tautomers (RDKit TautomerEnumerator)
3. For each tautomer:
   a. Predict pKa of ionizable groups (Dimorphite-DL)
   b. Generate protonation states at pH 7.4 +/- 1.0
   c. Compute Boltzmann population at target pH
4. Filter: keep states with population > 1%
5. Return TautomerEnsemble with dominant_state flagged

CRITICAL: ALL ligands entering gpu_dock.py, PhoreGen, or FEP must pass
through this.
If dominant state charge != 0, triggers charge correction in WT-2 FEP
```

```
QC.
```

Pipeline insertion: `[WT-1 output] --> tautomer_enumeration.py --> [WT-3 filter_pipeline]`

membrane_builder.py

Tool: packmol-memgen (GPL, AMBER-compatible, scriptable CLI) + OPM database for orientation.

```
Input:  PDB + target_classifier result ("membrane" | "soluble")
Output: MembraneSystem

If "soluble" -> skip. If "membrane":
1. Query OPM database for orientation (tilt angle, depth)
2. If not in OPM -> PPM server prediction
3. Run packmol-memgen: POPC(70%)/CHOL(30%), 80Ax80A
4. Solvate + neutralize with 150mM NaCl
5. 6-step CHARMM-GUI equilibration protocol
Wall time: ~1 hour per membrane target on RTX 5080
```

target_classifier.py

Auto-detect membrane vs soluble: OPM database (definitive) -> UniProt annotation -> hydrophobicity belt detection -> manual override via `--membrane` flag.

protein_fixer.py

PDBFixer (OpenMM, MIT): missing residues/atoms, non-standard residues (MSE->MET), alternate conformations, crystal contact warnings.

Deliverable

- Correct pH 7.4 protonation states for drug-like molecules
- Membrane builder tested on beta2-adrenergic receptor
- Target classifier with OPM + UniProt integration

---

9. WT-6: EXPLICIT SOLVENT REFINEMENT + WATER MAPS

Branch: `feat/explicit-solvent`

Duration: Week 2-3

Depends On: WT-9 merged to sota-dev

Merge Target: `sota-dev`

SOTA Gaps Closed: #6 (Explicit Solvent), #8 (Water Map Analysis)

Purpose

Re-simulate PRISM-detected pockets with explicit water to confirm stability. Compute hydration site thermodynamics identifying displaceable waters.

File Ownership (EXCLUSIVE)

```
scripts/
├── explicit_solvent/
│   ├── __init__.py
│   ├── pocket_refinement.py         # Explicit solvent MD on detected
pocket
│   ├── water_map_analysis.py        # Hydration site thermodynamics
│   ├── structural_water_finder.py   # Conserved water identification
│   ├── solvent_setup.py             # TIP3P/OPC solvation + ions
│   └── README.md
│
envs/
└── explicit_solvent.yml
│
tests/
└── test_explicit_solvent/
    ├── test_pocket_refinement.py
    ├── test_water_map.py
    └── fixtures/
        ├── prism_detected_pocket.json
        └── expected_water_sites.json
```

Key Implementation

`pocket_refinement.py`

This is the single most important QC gate in the entire pipeline.

```
Input:  PDB + PRISM spike JSON (detected pocket with lining residues)
Output: ExplicitSolventResult

Protocol:
1. Take PRISM output structure (post-detection snapshot)
2. Solvate: TIP3P (default) or OPC (high-accuracy)
```

```
   Truncated octahedron, 12A buffer, 150 mM NaCl
3. Minimize: 5000 steps steepest descent
4. NVT equilibration: 500 ps, 300K, Langevin thermostat
5. NPT equilibration: 500 ps, 1 bar, Monte Carlo barostat
6. Production NPT MD: 10 ns (MVP) or 50 ns (publication)
   Save coordinates every 10 ps
7. Analysis:
   a. Pocket RMSD (lining residue Ca over trajectory)
   b. Pocket volume (POVME or fpocket on frames)
   c. Classification:
      - RMSD < 2.0A, volume sigma < 20% -> STABLE
      - RMSD 2.0-3.5A or sigma 20-40% -> METASTABLE
      - RMSD > 3.5A or closes -> COLLAPSED
   d. Structural water identification (>80% occupancy)

Wall time (RTX 5080, OpenMM GPU):
- 10 ns: ~15-30 min
- 50 ns: ~1.5-3 hours

CRITICAL: If COLLAPSED -> flag "unvalidated", do NOT proceed to FEP.
```

Pipeline insertion:

```
PRISM Detection (implicit, 2 min) -> pocket_refinement.py (explicit,
30min-3hr) -> [REST OF PIPELINE]
                                        Only if STABLE or METASTABLE
```

water_map_analysis.py

Tool: SSTMap (GPL, Python, grid-based Inhomogeneous Solvation Theory) + GIST/cpptraj (validation).

```
Input:  Explicit solvent trajectory + pocket centroid + lining
residues
Output: WaterMap

Algorithm:
1. Define region: 8A sphere around pocket centroid
2. Grid at 0.5A resolution
3. Per voxel over trajectory:
   a. Water occupancy
   b. dH: water-protein + water-water energy vs bulk
   c. -TdS: orientational + translational entropy vs bulk
   d. dG_transfer = dH + (-TdS)
4. Cluster into discrete HydrationSites
5. Classify:
   - dG < -1.0 -> CONSERVED_HAPPY (don't displace)
   - dG > +1.0 -> CONSERVED_UNHAPPY (displace for gain)
   - Occupancy < 0.3 -> BULK
6. total_displacement_energy = sum(dG of unhappy waters)
```

```
7. Report: "Displacing 3 unhappy waters yields +4.2 kcal/mol"

PRISM leverage:
- water_density field correlates with hydration sites
- High water_density + high spike -> H-bond pharmacophore
- Low water_density near hydrophobic spikes -> cavity
- Water map validates spike assignments from first principles
```

structural_water_finder.py

Identify conserved waters (>80% occupancy, >=2 H-bonds to protein) that must be PRESERVED in docking. Output coordinates as fixed waters for gpu_dock.py.

Deliverable

- Explicit solvent pocket stability confirmation/rejection
- Water map with dG per hydration site
- Tested on KRAS G12C (known structural waters in Switch II)

---

10. WT-2: OPENFE ABFE/RBFE PIPELINE

Branch: `feat/fep-pipeline`

Duration: Week 2-3

Depends On: WT-0 merged to sota-dev

Merge Target: `sota-dev`

Purpose

Set up OpenFE + GROMACS GPU-FEP infrastructure, build PRISM->OpenFE bridge, execute ABFE calculations, analyze results.

File Ownership (EXCLUSIVE)

```
scripts/
├── fep/
│   ├── __init__.py
│   ├── prism_to_openfe.py          # DockingResult -> OpenFE
ChemicalSystem
│   ├── prepare_abfe.py             # ABFE protocol configuration
│   ├── prepare_rbfe.py             # RBFE for lead optimization
(Phase 2)
```

```
│   ├── run_fep.py                    # Execution engine (local GPU or
SLURM)
│   ├── analyze_fep.py                # Result gathering + QC checks
│   ├── fep_qc.py                     # QC gate implementations
│   ├── restraint_selector.py         # Boresch restraint from PRISM
lining residues
│   └── README.md
│
envs/
└── fep.yml
│
tests/
└── test_fep/
    ├── test_prism_to_openfe.py
    ├── test_prepare_abfe.py
    ├── test_analyze_fep.py
    ├── test_fep_qc.py
    └── fixtures/
        ├── mock_docking_result.json
        ├── mock_protein.pdb
        └── mock_ligand.sdf
```

Key Implementation

`prism_to_openfe.py`:

```
Input:  FilteredCandidate + DockingResult + SpikePharmacophore
Output: OpenFE AlchemicalNetwork (ABFE or RBFE)

Steps:
1. Load protein PDB + docked ligand SDF
2. Create OpenFE SmallMoleculeComponent + ProteinComponent +
SolventComponent
3. Build ChemicalSystem
4. Select Boresch restraint atoms from PRISM lining residues
5. Configure ABFE: 42 lambda-windows (complex) + 31 (solvent),
   5 ns/window, 3 repeats, REST2 for flexible pockets
6. Output AlchemicalNetwork as JSON
```

`restraint_selector.py`:

```
Input:  SpikePharmacophore.pocket_lining_residues + docked ligand pose
Output: Boresch restraint (3 protein atoms + 3 ligand atoms)

1. Select Ca atoms from lining residues with lowest B-factor
2. Select ligand heavy atoms nearest high-intensity spike positions
3. Compute restraint score (Aldeghi method)
4. Select lowest-score restraint, validate stability
```

`fep_qc.py`:

```
QC Gates (automatic pass/fail):
1. Lambda-window overlap >= 0.03
2. Forward/reverse hysteresis <= 1.5 kcal/mol
3. Protein Ca RMSD <= 4.0A
4. Ligand stays in pocket (COM < 5A from initial)
5. >= 2/3 repeats converge within 1.0 kcal/mol
```

`analyze_fep.py`:

```
1. openfe gather -> raw dG + error
2. Apply Boresch restraint analytical correction
3. Apply PB charge correction if needed
4. Run all QC gates
5. Classify: dG < -6 + QC pass -> NOVEL_HIT | RECAPITULATED
6. Output FEPResult objects
```

Deliverable

- Working ABFE on T4 lysozyme benzene benchmark
- GROMACS GPU-FEP confirmed on RTX 5080
- All output as `FEPResult` per interface spec
- Wall-time benchmarks (actual ns/day)

Integration Test

```
python scripts/fep/prepare_abfe.py \
    --protein tests/test_fep/fixtures/mock_protein.pdb \
    --ligand tests/test_fep/fixtures/mock_ligand.sdf \
    --output-dir /tmp/fep_test/

python scripts/fep/run_fep.py --network /tmp/fep_test/network.json --
gpu 0 --dry-run
```

---

11. WT-7: ENSEMBLE SCORING + P_OPEN + INTERACTION ENTROPY

Branch: `feat/ensemble-scoring`

Duration: Week 2-4

Depends On: WT-9 merged; reads WT-6 trajectory output

Merge Target: `sota-dev`

SOTA Gaps Closed: #7 (Ensemble Rescoring), #10 (P_open), #12 (MSM), #13 (IE)

Purpose

Replace single-snapshot MM-GBSA with ensemble-averaged scoring. Quantify pocket open probability. Compute interaction entropy. Optionally build Markov State Models.

File Ownership (EXCLUSIVE)

```
scripts/
├── ensemble/
│   ├── __init__.py
│   ├── ensemble_mmgbsa.py           # MM-GBSA averaged over MD
snapshots
│   ├── interaction_entropy.py       # IE method (replaces NMA)
│   ├── pocket_popen.py              # P_open from PRISM trajectories
│   ├── msm_builder.py               # Markov State Models (optional,
P3)
│   ├── block_analysis.py            # Block averaging for convergence
│   └── README.md
│
envs/
└── ensemble.yml
│
tests/
└── test_ensemble/
    ├── test_ensemble_mmgbsa.py
    ├── test_interaction_entropy.py
    ├── test_popen.py
    └── fixtures/
```

Key Implementation

ensemble_mmgbsa.py

Tool: MMPBSA.py (AmberTools, GPL) or gmx_MMPBSA (GROMACS wrapper).

```
Input:  Protein-ligand complex trajectory (from WT-6)
Output: EnsembleMMGBSA

1. Extract snapshots every 100 ps (100 frames for 10 ns)
2. Per frame: strip solvent, compute MM-GBSA
```

```
3. Average: dG_mean, dG_std, dG_sem (block analysis)
4. Per-residue decomposition for hot-spot identification

BEFORE: Single pose -> -28 kcal/mol (unrealistic)
AFTER:  100 frames -> -8.2 +/- 1.4 kcal/mol (realistic)

Wall time: ~5-15 min for 100 frames
```

interaction_entropy.py

Method: Duan et al., JACS 2016. Replaces obsolete NMA.

```
-TdS_IE = kT * ln < exp[ beta(E_int - <E_int>) ] >

1. Per frame: compute E_int(protein, ligand) -- VdW + Coulomb
2. Fluctuation: delta_E = E_int - <E_int>
3. -TdS = kT * ln <exp(beta * delta_E)>
4. dG_IE = dH (from MM-GBSA) + (-TdS_IE)

Advantages over NMA: no Hessian (10-100x faster), anharmonic, reuses
MD trajectory.
Wall time: negligible (same E_int data as MM-GBSA)
```

pocket_popen.py

```
Input:  PRISM 20 multi-stream trajectories + pocket lining residues
Output: PocketDynamics

1. Per stream: compute pocket volume at each frame (POVME/fpocket)
2. Binary trajectory: open[t] = V(t) > threshold
3. Aggregate: P_open = mean(fraction_open), error = bootstrap across
streams
4. Lifetimes: mean open/closed durations
5. Classify:
   P_open > 0.5 -> STABLE_OPEN (good druggability)
   0.1 < P_open < 0.5 -> TRANSIENT (moderate)
   P_open < 0.1 -> RARE_EVENT (poor)

PRISM leverage: 20 streams = 20 independent estimates + cryo-thermal
sampling
```

msm_builder.py (P3 priority, optional)

deeptime/PyEMMA: TICA -> k-means -> MSM -> stationary distribution -> weight ensemble poses. Defer unless targeting IDP clients.

Deliverable

- Ensemble MM-GBSA producing realistic dG with error bars
- Interaction Entropy replacing NMA
- P_open quantification tested on KRAS G12C

---

12. WT-4: ORCHESTRATOR + REPORTING

Branch: `feat/orchestrator`

Duration: Week 3-4

Depends On: WT-0 merged; integrates ALL other WT deliverables

Merge Target: `sota-dev` (second-to-last merge)

Purpose

Wire everything into the end-to-end pipeline. Single command: PDB -> novel validated hits. Generate publication-quality reports and anti-leakage audit trail.

File Ownership (EXCLUSIVE)

```
scripts/
├── pipeline/
│   ├── __init__.py
│   ├── prism_fep_pipeline.py        # Master orchestrator
│   ├── pipeline_config.yaml         # Default configuration
│   ├── audit_trail.py               # Anti-leakage audit logging
│   ├── report_generator.py          # Publication-quality output
│   ├── report_templates/
│   │   ├── hit_report.md.j2         # Jinja2 template per compound
│   │   └── campaign_report.md.j2    # Full target campaign summary
│   └── README.md
│
tests/
└── test_pipeline/
    ├── test_end_to_end.py
    ├── test_audit_trail.py
    └── fixtures/
        └── kras_g12c_golden/
```

Key Implementation

`prism_fep_pipeline.py`: Master script calling all WT deliverables:

```
Stages:
1.  [WT-5] target_classifier -> soluble/membrane
2.  [WT-5] protein_fixer / membrane_builder
3.  [EXISTING] prism-prep -> topology
4.  [EXISTING] nhs_rt_full -> spike JSON + pocket detection
5.  [WT-6] pocket_refinement -> explicit solvent validation (QC GATE)
6.  [WT-6] water_map_analysis -> hydration sites
7.  [WT-7] pocket_popen -> P_open quantification
8.  [WT-1] spike_to_pharmacophore -> SpikePharmacophore
9.  [WT-1] generate -> List[GeneratedMolecule]
10. [WT-5] tautomer_enumeration -> correct protonation
11. [WT-3] filter_pipeline -> List[FilteredCandidate] (top N)
12. [EXISTING] gpu_dock.py -> docked poses
13. [WT-7] ensemble_mmgbsa + interaction_entropy
14. [WT-2] prepare_abfe + run_fep + analyze_fep -> List[FEPResult]
15. [WT-4] report_generator -> publication report
16. [WT-4] audit_trail -> anti-leakage verification

CLI:
  python scripts/pipeline/prism_fep_pipeline.py \
      --pdb input.pdb --config pipeline_config.yaml \
      --output-dir results/kras_campaign/ \
      --skip-fep  # Optional: stop before FEP
```

`audit_trail.py`: Logs timestamps, input hashes, database queries, classification justification. Proves PRISM ran blind, external data used ONLY post-detection.

`report_generator.py`: Per-compound hit reports + campaign summary. Structure, dG (ABFE + ensemble GBSA + IE), pharmacophore match, drug-likeness, novelty, P_open, water map, classification, residue mapping, anti-leakage audit.

Deliverable

- Single-command end-to-end pipeline
- Publication-quality markdown reports
- Full anti-leakage audit trail
- Integration test on KRAS G12C golden case

Integration Test

```
python scripts/pipeline/prism_fep_pipeline.py \
    --pdb tests/fixtures/kras.pdb \
    --config scripts/pipeline/pipeline_config.yaml \
    --output-dir /tmp/pipeline_test --skip-fep
```

NOTE: WT-4 needs a second pass (WT-4b, ~2-3 days) after WT-5/6/7 merge to integrate the new stages into the master orchestrator.

---

13. WT-8: INTERACTIVE HTML5/WebGL DELIVERABLE

Branch: `feat/interactive-viewer`

Duration: Week 3-4

Depends On: WT-9 merged; reads all WT outputs via ViewerPayload

Merge Target: `sota-dev` (LAST to merge)

SOTA Gaps Closed: #11 (Interactive Viewer)

Purpose

Replace static PyMOL scripts and PDFs with a self-contained interactive HTML5 viewer. Clients open it in any browser, including iPad.

File Ownership (EXCLUSIVE)

```
scripts/
├── viewer/
│   ├── __init__.py
│   ├── build_viewer.py              # Data -> HTML5 artifact
│   ├── viewer_template.html         # Base Mol* template
│   ├── components/
│   │   ├── pocket_surface.js
│   │   ├── spike_overlay.js
│   │   ├── water_map_layer.js
│   │   ├── ligand_viewer.js
│   │   ├── report_panel.js
│   │   └── controls.js
│   └── README.md
│
tests/
└── test_viewer/
    ├── test_build_viewer.py
    └── fixtures/
        └── mock_viewer_payload.json
```

Key Implementation

Tool: Mol* (MIT, powers RCSB PDB, handles large systems, custom representations).

```
Input:  ViewerPayload
Output: Single self-contained .html file (< 10 MB)

Architecture:
- Mol* from CDN (or embedded for offline)
- Protein as base64 PDB in <script> tag
- Pre-computed pocket surfaces as mesh
- Spike positions as custom geometry
- Water map sites as colored spheres (red=unhappy, blue=happy)
- Ligand poses as separate entities
- Collapsible report side panel

Interactive Features:
- Toggle layers: surface, pocket, spikes, water map, ligands,
residues, H-bonds
- Pocket panel: target, P_open, druggability, displaceable waters,
quality score
- Per-ligand panel: SMILES, dG (ABFE + ensemble), QED/SA, Tanimoto,
classification
- Export: PNG, PDB, SDF
```

Deliverable

- Self-contained HTML5 viewer
- Tested on iPad Safari
- < 10 MB per target

---

14. MERGE ORDER + INTEGRATION STRATEGY

```
FOUNDATION:
Day 1-3:    WT-0  ============ -> MERGE to sota-dev
Day 3-5:    WT-9  ============ -> MERGE to sota-dev (interface
extensions)

PARALLEL DEVELOPMENT (all start after their foundation merges):
Day 3-10:   WT-1  ======================== -> MERGE (genphore)
Day 3-10:   WT-3  ======================== -> MERGE (filters)
Day 5-14:   WT-5  ======================== -> MERGE (preprocessing)
Day 5-18:   WT-6  ==================================== -> MERGE
(explicit solvent)

VALIDATION (depends on above):
Day 7-18:   WT-2  ==================================== -> MERGE (FEP)
Day 7-21:   WT-7  ============================================ -> MERGE
(ensemble)
```

```
INTEGRATION (merge last):
Day 14-23:  WT-4  ================================== -> MERGE
(orchestrator)
Day 14-25:  WT-8  ================================ -> MERGE (viewer,
LAST)
```

Merge Protocol

Step 1: WT-0 -> sota-dev (Day 3)

```
cd ~/Desktop/Prism4D-bio
git merge feat/interfaces --no-ff -m "feat: shared interface contracts
for FEP+GenPhore pipeline"
```

Gate: All dataclasses have unit tests. Mock data validates serialization round-trip.

Step 2: WT-9 -> sota-dev (Day 5)

```
git merge feat/interfaces-v2 --no-ff -m "feat: interface extensions
for preprocessing, solvent, ensemble, viewer"
```

Gate: All new dataclasses have unit tests. No modifications to WT-0 interfaces.

Step 3: WT-1 -> sota-dev (Day 10)

```
cd ../wt-genphore && git rebase sota-dev
python scripts/genphore/generate.py --spike-json
tests/fixtures/mock_spike_output.json --output-dir /tmp/test
cd ~/Desktop/Prism4D-bio
git merge feat/genphore --no-ff -m "feat: spike->pharmacophore +
PhoreGen/PGMG generation"
```

Gate: Generates valid molecules. Output conforms to GeneratedMolecule interface.

Step 4: WT-3 -> sota-dev (Day 10, after WT-1)

```
cd ../wt-filters && git rebase sota-dev
python scripts/filters/filter_pipeline.py --molecules
/tmp/test/molecules_meta.json --pharmacophore
/tmp/test/pharmacophore.json --top-n 5 --output /tmp/filter_test
cd ~/Desktop/Prism4D-bio
git merge feat/filters --no-ff -m "feat: 6-stage filtering with
novelty + diversity"
```

Gate: Top-5 are diverse, drug-like, novel. Audit trail complete.

Step 5: WT-5 -> sota-dev (Day 14)

```
cd ../wt-preprocessing && git rebase sota-dev
python scripts/preprocessing/tautomer_enumeration.py --smiles
"CC(=O)Oc1ccccc1C(=O)O" --ph 7.4
cd ~/Desktop/Prism4D-bio
git merge feat/preprocessing --no-ff -m "feat: tautomer enumeration +
membrane builder + protein fixer"
```

Gate: Correct protonation for known drugs. Membrane builder works on beta2-AR.

Step 6: WT-6 -> sota-dev (Day 18)

```
cd ../wt-explicit-solvent && git rebase sota-dev
python scripts/explicit_solvent/pocket_refinement.py --pdb
tests/fixtures/kras.pdb --spike-json tests/fixtures/kras_spikes.json -
-time-ns 10 --dry-run
cd ~/Desktop/Prism4D-bio
git merge feat/explicit-solvent --no-ff -m "feat: explicit solvent
refinement + water map analysis"
```

Gate: Pocket classification correct for KRAS. Water map identifies known structural waters.

Step 7: WT-2 -> sota-dev (Day 18)

```
cd ../wt-fep && git rebase sota-dev
python scripts/fep/prepare_abfe.py --protein
tests/fixtures/mock_protein.pdb --ligand
tests/fixtures/mock_ligand.sdf --output-dir /tmp/fep_test --dry-run
cd ~/Desktop/Prism4D-bio
git merge feat/fep-pipeline --no-ff -m "feat: OpenFE ABFE/RBFE
pipeline with GPU-FEP + QC gates"
```

Gate: OpenFE setup runs. Dry-run validates. GPU-FEP confirmed on RTX 5080.

Step 8: WT-7 -> sota-dev (Day 21)

```
cd ../wt-ensemble-scoring && git rebase sota-dev
python scripts/ensemble/pocket_popen.py --trajectories
tests/fixtures/multi_stream/ --pocket-residues "12,60-69"
cd ~/Desktop/Prism4D-bio
git merge feat/ensemble-scoring --no-ff -m "feat: ensemble MM-GBSA +
interaction entropy + P_open"
```

Gate: Ensemble MM-GBSA realistic. P_open matches literature for KRAS Switch II.

Step 9: WT-4 -> sota-dev (Day 23)

```
cd ../wt-orchestrator && git rebase sota-dev
python scripts/pipeline/prism_fep_pipeline.py --pdb
tests/fixtures/kras.pdb --config scripts/pipeline/pipeline_config.yaml
--output-dir /tmp/pipeline_test --skip-fep
cd ~/Desktop/Prism4D-bio
git merge feat/orchestrator --no-ff -m "feat: end-to-end orchestrator
+ reporting + audit trail"
```

Gate: All stages run (skip-fep mode). Report generated. Audit trail valid.

Step 10: WT-8 -> sota-dev (Day 25, LAST)

```
cd ../wt-interactive-viewer && git rebase sota-dev
python scripts/viewer/build_viewer.py --payload
tests/fixtures/mock_viewer_payload.json --output /tmp/test_report.html
cd ~/Desktop/Prism4D-bio
git merge feat/interactive-viewer --no-ff -m "feat: interactive
HTML5/Mol* viewer"
```

Gate: HTML5 opens in Chrome + Safari. Interactive features work. < 10 MB.

Step 11: Cleanup

```
git worktree remove ../wt-interfaces
git worktree remove ../wt-interfaces-v2
git worktree remove ../wt-genphore
git worktree remove ../wt-fep
git worktree remove ../wt-filters
git worktree remove ../wt-orchestrator
git worktree remove ../wt-preprocessing
git worktree remove ../wt-explicit-solvent
git worktree remove ../wt-ensemble-scoring
git worktree remove ../wt-interactive-viewer

git branch -d feat/interfaces feat/interfaces-v2 feat/genphore
feat/fep-pipeline feat/filters feat/orchestrator feat/preprocessing
feat/explicit-solvent feat/ensemble-scoring feat/interactive-viewer
```

---

15. CONFLICT PREVENTION MATRIX

File/Directory	WT-0	WT-9	WT-1	WT-3	WT-5	WT-6	WT-2	WT-7	WT-4	WT-8
`interfaces/` (original)	**W**	—	r	r	r	r	r	r	r	r
`interfaces/` (v2)	—	**W**	—	—	r	r	—	r	r	r
`scripts/genphore/`	—	—	**W**	—	—	—	—	—	r	—
`scripts/filters/`	—	—	—	**W**	—	—	—	—	r	—
`scripts/preprocessing/`	—	—	—	—	**W**	—	—	—	r	—
`scripts/explicit_solvent/`	—	—	—	—	—	**W**	—	—	r	—
`scripts/fep/`	—	—	—	—	—	—	**W**	—	r	—
`scripts/ensemble/`	—	—	—	—	—	—	—	**W**	r	—
`scripts/pipeline/`	—	—	—	—	—	—	—	—	**W**	—
`scripts/viewer/`	—	—	—	—	—	—	—	—	—	**W**
`envs/phoregen.yml`	—	—	**W**	—	—	—	—	—	—	—
`envs/pgmg.yml`	—	—	**W**	—	—	—	—	—	—	—
`envs/fep.yml`	—	—	—	—	—	—	**W**	—	—	—
`envs/preprocessing.yml`	—	—	—	—	**W**	—	—	—	—	—
`envs/explicit_solvent.yml`	—	—	—	—	—	**W**	—	—	—	—
`envs/ensemble.yml`	—	—	—	—	—	—	—	**W**	—	—
`data/pains_catalog.csv`	—	—	—	**W**	—	—	—	—	—	—
Existing PRISM Rust code	—	—	—	—	—	—	—	—	—	—
`CLAUDE.md` / `.claude/`	—	—	—	—	—	—	—	—	—	—

Zero overlap in WRITE columns across all 10 worktrees = zero merge conflicts.

---

16. FULL INTEGRATED PIPELINE DIAGRAM

```
INPUT: PDB file
  target_classifier.py [WT-5] -> soluble/membrane
      |
      +-- SOLUBLE --> protein_fixer.py [WT-5]
      +-- MEMBRANE --> membrane_builder.py [WT-5] (OPM + packmol +
equilibration)
      |
      v
PRISM-4D Detection (BLIND) -- 2 min [EXISTING]
  nhs_rt_full --multi-stream 20 --multi-scale
  Output: Spike JSON + lining residues + pocket coords
      |
      v
Explicit Solvent Refinement [WT-6] -- 30 min to 3 hr
  pocket_refinement.py: TIP3P/OPC -> 10-50 ns explicit MD
  *** QC GATE: STABLE / METASTABLE / COLLAPSED ***
  If COLLAPSED -> STOP, pocket not validated
      |  [STABLE or METASTABLE only]
      +---> Water Map Analysis [WT-6]: unhappy waters, dG per site
      +---> Pocket P_open [WT-7]: open probability from 20 PRISM
```

```
streams
      |
      v
Spike -> Pharmacophore Translation [WT-1]
  Enriched with: water map data + P_open + explicit-solvent coords
      |
      +---> PhoreGen [WT-1]: 1000 mols, ~10 min
      +---> PGMG [WT-1]: 10000 mols, ~30 sec
      |
      v
Tautomer/Protomer Enumeration [WT-5]
  pH 7.4 +/- 1.0, all generated molecules
      |
      v
6-Stage Filtering Pipeline [WT-3]
  Validity -> Drug-likeness -> PAINS -> Pharmacophore -> Novelty ->
Diversity
      |
      v
GPU Docking [EXISTING]
  With: explicit structural waters from WT-6
  With: correct protonation states from WT-5
      |
      +---> Ensemble MM-GBSA + IE [WT-7]: 10 ns MD/pose, 100 frames
      +---> ABFE Validation [WT-2]: OpenFE + GROMACS GPU-FEP, 1-3 days
      |
      v
Classification & Report [WT-4]
  Per compound: dG (ABFE + ensemble GBSA + IE), novelty, drug-likeness
  Per pocket: NOVEL/RECAPITULATED, P_open, water map
  Anti-leakage audit trail
      |
      v
Interactive HTML5 Viewer [WT-8]
  Mol* + pocket surface + spike overlay + water map + ligand poses
  Opens on iPad, < 10 MB
```

---

17. DEPENDENCY GRAPH

```
WT-0 (interfaces) ------------------------------------------------+
  |                                                               |
  +---> WT-1 (genphore)                                           |
  +---> WT-2 (fep)                                                |
  +---> WT-3 (filters)                                            |
  |                                                               |
  |    WT-9 (interfaces v2) -------------------------------------+
  |      |                                                        |
```

```
  |      +---> WT-5 (preprocessing)                                    |
  |      +---> WT-6 (explicit solvent)                                 |
  |      +---> WT-7 (ensemble) <-- reads WT-6 trajectory output        |
  |      +---> WT-8 (viewer) <-- reads all WT outputs                  |
  |                                                                     |
  +---> WT-4 (orchestrator) <-- integrates ALL <----------------------+
  +---> WT-8 (viewer, LAST)

Merge order:
  WT-0 -> WT-9 -> {WT-1 || WT-3 || WT-5 || WT-6} -> {WT-2 || WT-7} ->
WT-4 -> WT-8
```

---

18. WALL-TIME ESTIMATES (FULL SOTA PIPELINE)

Stage	Time (RTX 5080)	Worktree
Target classification	~1 sec	WT-5
Protein fixing	~30 sec	WT-5
Membrane embedding*	~1 hr	WT-5 (*membrane only)
PRISM detection	~2 min	Existing
Explicit solvent refinement	30 min - 3 hr	WT-6
Water map analysis	~10 min	WT-6
P_open calculation	~5 min	WT-7
Spike -> pharmacophore	~1 min	WT-1
PhoreGen generation	~10 min	WT-1
Tautomer enumeration	~2 min	WT-5
6-stage filtering	~5 min	WT-3
GPU docking	~10 min	Existing
Ensemble MM-GBSA + IE	~1-2 hr	WT-7
ABFE validation	~30-60 hr	WT-2
Report generation	~5 min	WT-4
Interactive viewer build	~2 min	WT-8
Total (soluble, with FEP)	**~1.5-3.5 days**	
Total (membrane, with FEP)	**~2-4 days**	
Total (soluble, no FEP)	**~2-5 hours**	MVP mode

---

19. PARALLEL DEVELOPMENT RULES

1.

2. **NEVER modify files outside your WRITE column.** If you need a change in another WT's territory, open an issue and wait for them to expose it via the interface.
3.
4. **All inter-WT communication goes through `scripts/interfaces/`.** No direct imports between genphore<->fep<->filters. Everything serializes through the shared dataclasses.
5.
6. **Each WT has its own conda environment.** No shared environments. Pipeline (WT-4) activates the right env per stage via subprocess.
7.
8. **Tests use mock data from interfaces, not other WT outputs.** WT-2 tests with mock DockingResult, not actual gpu_dock.py output. WT-3 tests with mock GeneratedMolecule, not actual PhoreGen output.
9.
10. **WT-0 and WT-9 merge FIRST and are FROZEN after merge.** If interfaces need changes during development, ALL WTs must agree, and it goes through a hotfix branch merged to sota-dev (then all WTs rebase).
11.
12. **WT-8 merges LAST.** It's the final presentation layer.
13.
14. **Each WT's README.md documents its standalone test command.** Anyone should be able to clone, install deps, and run the WT's integration test independently.

---

20. CLAUDE CODE COMMANDS

```
# Foundation
cd ~/Desktop/wt-interfaces       && claude    # WT-0
cd ~/Desktop/wt-interfaces-v2    && claude    # WT-9

# Generation + Filtering
cd ~/Desktop/wt-genphore         && claude    # WT-1
cd ~/Desktop/wt-filters          && claude    # WT-3

# Pre/Post Processing
cd ~/Desktop/wt-preprocessing    && claude    # WT-5
cd ~/Desktop/wt-explicit-solvent && claude    # WT-6
```

```
# Validation
cd ~/Desktop/wt-fep              && claude    # WT-2
cd ~/Desktop/wt-ensemble-scoring && claude    # WT-7

# Integration
cd ~/Desktop/wt-orchestrator     && claude    # WT-4
cd ~/Desktop/wt-interactive-viewer && claude  # WT-8

# Each worktree inherits:
# - Managed CLAUDE.md (anti-leakage firewalls, QC protocols)
# - All MCP servers (AlphaFold, UniProt, PubChem, STRING, gget, etc.)
# - Anti-confabulation hooks
# - rust-analyzer plugin
```

Each Claude Code instance sees the SAME managed rules but works on DIFFERENT files. No conflicts possible.

SUMMARY

Worktree	Branch	Purpose	SOTA Tier	Start	Merge
WT-0	feat/interfaces	Interface contracts	Foundation	Day 1	Day 3
WT-9	feat/interfaces-v2	Interface extensions	Foundation	Day 3	Day 5
WT-1	feat/genphore	De novo generation	Bleeding Edge	Day 3	Day 10
WT-3	feat/filters	Filtering cascade	Standard	Day 3	Day 10
WT-5	feat/preprocessing	Tautomers + Membrane	MVP + Market	Day 5	Day 14
WT-6	feat/explicit-solvent	Solvent + Water Maps	MVP	Day 5	Day 18
WT-2	feat/fep-pipeline	ABFE/RBFE	Bleeding Edge	Day 7	Day 18
WT-7	feat/ensemble-scoring	Ensemble + P_open + IE	MVP + SOTA	Day 7	Day 21
WT-4	feat/orchestrator	Pipeline + Reporting	Standard	Day 14	Day 23
WT-8	feat/interactive-viewer	HTML5 Viewer	Bleeding Edge	Day 14	Day 25

10 worktrees. 0 file conflicts. 13/13 SOTA requirements covered. 25-day sprint.