Protein Refinery Architecture v1.0

1. System Overview

The **Protein Refinery** is a headless, fully automated synthetic biology pipeline designed to evolve novel enzymes. It operates as a closed-loop system that takes a wild-type biological structure, iteratively improves it using AI and physics-based simulations, validates its novelty against global databases, and deposits successful candidates into a secure local bank.

Core Philosophy: "Zero-Human-Touch" optimization.

Architectural Diagram

```
graph TD

    A[Input: Wild Type PDB] --> B{Novelty Gatekeeper}

    B -- "Duplicate Found" --> C[Log & Abort]

    B -- "Verified Novel" --> D[AI Designer (ProteinMPNN)]

    D --> E[Physics Validation]

    E --> F{Is Better?}

    F -- "No" --> D

    F -- "Yes" --> G[Commit to Vault]

    G --> H[Update Leaderboard]

    H --> D
```

2. Environment Setup

This pipeline requires a Linux environment (Ubuntu 22.04+ recommended) with CUDA support for AI acceleration.

System Dependencies

```
# 1. Update and install base tools

sudo apt-get update
sudo apt-get install -y git python3-pip openbabel autodock-vina wget unzip


# 2. Install Python Science Stack

pip install torch torchvision torchaudio --index-url [https://download.pytorch.org/whl/cu118](https://download.pytorch.org/whl/cu118)

pip install biopython pandas scipy numpy requests


# 3. Setup ProteinMPNN (AI Designer)

git clone [https://github.com/dauparas/ProteinMPNN.git](https://github.com/dauparas/ProteinMPNN.git)

cd ProteinMPNN

# Download trained weights (approx 4GB)

wget -qnc [https://github.com/dauparas/ProteinMPNN/raw/main/vanilla_model_weights/v_48_020.pt](https://github.com/dauparas/ProteinMPNN/raw/main/vanilla_model_weights/v_48_020.pt)

cd ..


# 4. Setup FoldX (Stability Engine)

# NOTE: FoldX is proprietary academic software.

# You must obtain the binary 'foldx' and 'rotabase.txt' from foldxsuite.crg.eu

# Place them in a visible directory, e.g., ./bin/

chmod +x ./bin/foldx
```

3. The Vault (Database Schema)

We use a lightweight SQLite database to maintain the lineage and "Git History" of every protein generated.

File: schema.sql

```
CREATE TABLE IF NOT EXISTS protein_bank (

    id TEXT PRIMARY KEY,              -- UUID (e.g., "SYN-7f8a1b")

    parent_id TEXT,                   -- UUID of the protein this was evolved from

    sequence TEXT NOT NULL UNIQUE,    -- Amino Acid Sequence (Primary Key for Physics)

    mutations TEXT,                   -- Delta from Wild Type (e.g., "W162A, F14L")

    binding_affinity REAL,            -- Vina Score (Lower is better)

    stability_score REAL,             -- FoldX Energy (Lower is better)

    generation INTEGER,               -- Iteration count

    novelty_status TEXT,              -- "NOVEL" or "FLAGGED_DUPLICATE"

    file_path TEXT,                   -- Path to stored PDB

    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP

);


CREATE INDEX idx_affinity ON protein_bank(binding_affinity);

CREATE INDEX idx_sequence ON protein_bank(sequence);
```

4. The Gatekeeper (Novelty Check Module)

This module ensures we do not waste compute resources reinventing enzymes that already exist in nature. It queries the two largest global databases: **AlphaFold DB** and **RCSB PDB**.

File: novelty_gatekeeper.py

```
import requests

import json
```

```
def check_novelty(sequence):
    """
    Checks if a sequence exists in global databases.
    Returns: (Is_Novel: bool, Reason: str)
    """

    # 1. Check AlphaFold Database (UniProt)
    # The API returns 200 if found, 404 or empty if new.
    af_url =
f"[https://rest.uniprot.org/uniprotkb/search?query=sequence](https://rest.uniprot.org/uniprotk
b/search?query=sequence):{sequence}&format=json"
    try:
        response = requests.get(af_url, timeout=5)
        if response.status_code == 200:
            data = response.json()
            if data['results']:
                acc = data['results'][0]['primaryAccession']
                return False, f"Duplicate found in AlphaFold/UniProt ({acc})"
    except Exception as e:
        print(f"[WARN] AlphaFold check failed: {e}")

    # 2. Check RCSB PDB (Protein Data Bank)
    # Using the sequence search API
    pdb_url =
"[https://search.rcsb.org/rcsbsearch/v2/query](https://search.rcsb.org/rcsbsearch/v2/query)"
```

```
    query = {
        "query": {
            "type": "terminal",
            "service": "sequence",
            "parameters": {
                "evalue_cutoff": 0.0001,
                "identity_cutoff": 1.0, # 100% Identity match
                "sequence_type": "protein",
                "value": sequence
            }
        },
        "return_type": "entry"
    }

    try:
        response = requests.post(pdb_url, json=query, timeout=5)
        if response.status_code == 200 and response.json().get('result_set'):
            pdb_id = response.json()['result_set'][0]['identifier']
            return False, f"Duplicate found in PDB ({pdb_id})"
    except:
        pass

    return True, "Verified Novel"
```

5. The Refinery Engine (Main Automation Loop)

This script ties the AI designer and Physics engines together into a continuous evolution loop.

File: refinery.py

```
import os
import subprocess
import sqlite3
import uuid
import shutil
from Bio import SeqIO
import novelty_gatekeeper


# --- CONFIG ---
FOLDX_BIN = "./bin/foldx"
VINA_BIN = "vina"
MPNN_SCRIPT = "./ProteinMPNN/protein_mpnn_run.py"
BANK_DIR = "./bank_storage"
DB_FILE = "protein_vault.db"


def init_db():
    conn = sqlite3.connect(DB_FILE)
    with open('schema.sql', 'r') as f:
        conn.executescript(f.read())
    conn.close()
```

```
def get_seq(pdb_path):
    # Extracts sequence from PDB file
    for record in SeqIO.parse(pdb_path, "pdb-atom"):
        return str(record.seq)
    return ""


def run_ai_design(input_pdb, output_folder):
    """Runs ProteinMPNN to hallucinate new sequences based on structure"""
    print(f"[*] Running AI Design on {input_pdb}...")
    cmd = [
        "python3", MPNN_SCRIPT,
        "--pdb_path", input_pdb,
        "--out_folder", output_folder,
        "--num_seq_per_target", "5",
        "--sampling_temp", "0.2", # Low temp = conservative, High = novel
        "--batch_size", "1"
    ]
    subprocess.run(cmd, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    return [os.path.join(output_folder, f) for f in os.listdir(output_folder) if f.endswith(".pdb")]


def run_physics_validation(pdb_file, ligand_pdbqt):
    """
    1. Runs FoldX for stability (Total Energy)
    2. Runs Vina for binding (Affinity)
    """
    # Stability Check
    cmd_foldx = [FOLDX_BIN, "--command=Stability", f"--pdb={pdb_file}"]
    res_foldx = subprocess.run(cmd_foldx, capture_output=True, text=True)
    # Parse FoldX output (mocked for brevity)
    stability = -5.0

    # Binding Check
    # Convert receptor to PDBQT
    pdbqt_file = pdb_file + "qt"
    subprocess.run(["obabel", pdb_file, "-O", pdbqt_file, "-xr", "-xp"],
stderr=subprocess.DEVNULL)

    cmd_vina = [
        VINA_BIN, "--receptor", pdbqt_file, "--ligand", ligand_pdbqt,
        "--center_x", "10", "--center_y", "10", "--center_z", "10", # Dynamic centering required in
prod
        "--size_x", "20", "--size_y", "20", "--size_z", "20",
        "--cpu", "4"
    ]
    res_vina = subprocess.run(cmd_vina, capture_output=True, text=True)
    # Parse Vina output (mocked)
    affinity = -8.5
    return stability, affinity


def commit_to_bank(pdb_file, parent_id, stability, affinity, gen):
    seq = get_seq(pdb_file)
    is_novel, status_msg = novelty_gatekeeper.check_novelty(seq)


    new_id = f"SYN-{str(uuid.uuid4())[:8]}"
    final_path = os.path.join(BANK_DIR, f"{new_id}.pdb")
    shutil.copy(pdb_file, final_path)


    conn = sqlite3.connect(DB_FILE)
    c = conn.cursor()
    c.execute("""
        INSERT INTO protein_bank
        (id, parent_id, sequence, binding_affinity, stability_score, generation, novelty_status,
file_path)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    """, (new_id, parent_id, seq, affinity, stability, gen, "NOVEL" if is_novel else "DUPLICATE",
final_path))
    conn.commit()
    conn.close()


    print(f"[SUCCESS] {new_id} banked. Affinity: {affinity} | Status: {status_msg}")
    return new_id, affinity


def main_loop(start_pdb, target_ligand, generations=5):
    init_db()
    if not os.path.exists(BANK_DIR): os.makedirs(BANK_DIR)

    current_parent = start_pdb
    parent_id = "WILD_TYPE"
    best_affinity = 0.0 # Placeholder

    for g in range(generations):
        print(f"\n=== GENERATION {g+1} ===")

        # 1. Design Phase
        temp_dir = f"./temp_gen_{g}"
        os.makedirs(temp_dir, exist_ok=True)
        candidates = run_ai_design(current_parent, temp_dir)

        # 2. Validation Phase
        generation_best_pdb = None
        generation_best_score = 100.0 # High is bad for binding energy

        for cand in candidates:
            stab, aff = run_physics_validation(cand, target_ligand)
            # Commit every valid run to DB for training data
            cand_id, _ = commit_to_bank(cand, parent_id, stab, aff, g)

            # Select winner
            if aff < generation_best_score and stab < -5.0: # Filter for stability
                generation_best_score = aff
                generation_best_pdb = cand

        # 3. Evolution
        if generation_best_pdb and generation_best_score < best_affinity:
            print(f"*** EVOLUTIONARY LEAP: {best_affinity} -> {generation_best_score} ***")
            current_parent = generation_best_pdb
            # In a real run, you'd fetch the ID from the DB
            best_affinity = generation_best_score
        else:
            print("Evolution stalled. Keeping previous parent.")

if __name__ == "__main__":
    # Example Usage
    main_loop("inputs/laccase_wt.pdb", "inputs/polystyrene.pdbqt")
```

6. Execution Instructions

1. **Prepare Inputs:**

    - Place your target enzyme (e.g., `laccase.pdb`) in an `inputs/` folder.
    - Place your ligand (e.g., `plastic.pdbqt`) in the same folder.
    - Ensure the binaries for `vina` and `foldx` are in your system PATH or defined in the script config.
2. **Launch the Refinery:**

```
python3 refinery.py
```

3.
4. Monitor the Vault:
   You can watch the database populate in real-time:

```
watch "sqlite3 protein_vault.db 'SELECT id, binding_affinity, novelty_status FROM protein_bank ORDER BY timestamp DESC LIMIT 5;'"
```

5.

Part II: Comprehensive Development Guide & Roadmap

7. Executive Summary

The Protein Refinery is a complex automated system that requires careful staging. This roadmap breaks the development into 6 distinct phases, from "Hello World" to "Production Hardening."

Core Principle: Do not move to the next phase until the current phase is stable. A broken foundation will cause the AI to Hallucinate optimizations that physics simulations will disprove.

8. Phase 1: Foundation & Environment Setup

Goal: Establish the technical foundation. No complex logic—just dependency management.

8.1 Critical Actions

- **Install System Dependencies:**
    - **CUDA:** Non-negotiable for ProteinMPNN speed (CPU is 10-50x slower).
    - **FoldX:** Proprietary academic software. Request license immediately at `foldxsuite.crg.eu` (Wait time: 2-5 days).
    - **OpenBabel:** For converting PDB $\leftrightarrow$ PDBQT.
- **Database Schema:**

    - Initialize `protein_bank` with proper indexes on `sequence` and `affinity`.
- **Configuration Strategy:**
    - Use YAML (`settings.yaml`) for parameters like sampling temperature and generations. **Do not hardcode these.**

8.2 What NOT to Do

- **Don't** install packages globally. Always use a virtual environment (`venv` or `conda`).
- **Don't** skip the CUDA setup.
- **Don't** ignore logging. You will need detailed logs when a simulation fails silently after 4 hours.

9. Phase 2: Core Evolution Engine

Goal: Implement the basic closed loop: Design $\rightarrow$ Validate $\rightarrow$ Evolve.

9.1 Architecture Overview

```
START

 ↓

[Input: Wild-Type PDB]

 ↓

LOOP (Generation N):

  ├──→ [AI Designer: ProteinMPNN] (Generate 5 candidates)

  │

  ├──→ [Physics Validator] (FoldX + Vina)

  │

  ├──→ [Novelty Gatekeeper] (AlphaFold/PDB Check)

  │

  ├──→ [Database Commit] (Save to Vault)

  │

  └──→ [Evolution Decision] (Better? Yes -> New Parent)
```

9.2 Component Strategy

- **AI Designer (ProteinMPNN):**
    - **Inputs:** PDB Structure.
    - **Outputs:** FASTA sequences.
    - **Note:** Phase 2 uses the *parent's* structure as a placeholder for the new sequence to test binding. This is a temporary approximation.
- **Physics Validator:**
    - **FoldX:** Returns ΔG (Stability).
    - **Vina:** Returns Affinity (Binding).
    - **Timeout:** Set strict timeouts (e.g., 120s) because these tools often hang.
- **Evolution Logic:**
    - **Greedy Approach:** If `new_fitness` < `best_fitness`, switch parents.

9.3 Testing Strategy

- **Unit Tests:** Can each tool run individually?
- **Integration:** Run 2 generations on a small protein (50 AA). Verify DB entries.
- **Sanity Check:** Are scores negative? (Positive scores usually mean the physics engine exploded).

10. Phase 3: Structure Prediction & Advanced Validation

Goal: Remove the "Fake Structure" workaround from Phase 2 using real structure prediction.

10.1 The Bottleneck Problem

- **Phase 2:** Used parent PDB for all variants. Fast, but inaccurate.
- **Phase 3:** Predicts *real* 3D structure for every generated sequence.

10.2 Architecture Decision: ESMFold vs AlphaFold2

Metric	**ESMFold**	**AlphaFold2**
Speed	~60 sec	~5 mins
Accuracy	97% of AF2	Gold Standard

Rec.	**Use ESMFold**	Too slow for evolution loops

10.3 Component: Structure Predictor

- **Caching is Critical:** Store predicted PDBs in `structure_cache/`. Use MD5 hash of sequence as key. This speeds up experiments by 10-100x.
- **Ensemble Docking:** Run Vina 3-5 times with slightly different box centers. Take the average score. Single runs are too noisy.

10.4 Sequence Clustering

- **Problem:** ProteinMPNN generates redundant sequences.
- **Solution:** Cluster sequences (e.g., 95% similarity). Only predict structure for the cluster representative. Saves ~40% compute.

11. Phase 4: Advanced Evolution (Escaping Local Minima)

Goal: Stop the system from getting stuck on a "good enough" protein.

11.1 Strategies

1. **Pareto Optimization:**
    - Don't just track "Best Affinity."
    - Track the **Pareto Frontier** of (Affinity, Stability, Novelty).
    - Breed from any solution on the frontier.
2. **Mutation Jumps (Simulated Annealing):**
    - **Trigger:** If no improvement for 3 generations.
    - **Action:** Randomly mutate 10% of residues.
    - **Result:** Kicks the system out of a local minimum.
3. **Diversity Tracking:**
    - Monitor population diversity. If <50% similarity, force breeding from lower-ranked candidates.

11.2 Convergence Criteria

- **Stop when:** Fitness hasn't improved for 5 generations OR Population is 99% similar.

12. Phase 5: Testing, Monitoring & QA

Goal: Ensure reliability before long runs.

12.1 Testing

- **Unit:** Mock external tools (FoldX/Vina) to test logic instantly.

- **Integration:** Test full pipeline with small inputs.
- **Validation:** Verify that lineage is traceable in the database.

12.2 Monitoring

- **Alerts:** Notify if generation time > 20 mins.
- **Health:** Check disk space (PDB files accumulate fast).
- **Checkpointing:** Save state after every generation. If the server crashes, resume from Generation N, not 0.

13. Phase 6: Deployment & Production Hardening

Goal: Scale up.

13.1 Containerization

- **Docker:** Containerize the OS, dependencies, and Python environment.
- **Data:** Mount user data and database as external volumes.

13.2 Configuration Management

- **Example Config:**

```
experiment:

  name: "laccase-opt-v1"

evolution:

  pareto_optimization: true

  mutation_jump_freq: 3
```

-

13.3 Logging

- **Levels:** DEBUG (Subprocess calls), INFO (Generation complete), ERROR (Tool failure).
- **Format:** Timestamp - Level - Message.

14. Architecture Best Practices

- **A1. Modularity:** Keep `ai_designer.py`, `physics_validator.py`, and `database.py` separate. Do not write a 2000-line `main.py`.
- **A2. Config over Hardcoding:** Never hardcode sampling temps or file paths.

- **A3. Robust Error Handling:** try/except every subprocess call. If Vina crashes, log it and return a "bad" score, don't crash the pipeline.
- **A4. Database Truth:** The database is the single source of truth. Records are append-only.

15. Common Pitfalls

1. **Structure Prediction Bottleneck:** Don't predict every sequence. Use clustering and caching.
2. **Fitness Stagnation:** Don't rely on greedy evolution. Use Mutation Jumps.
3. **Validation Uncertainty:** Don't trust a single Vina run. Use Ensemble Docking.
4. **Silent Failures:** Don't ignore errors. Log everything.
5. **Database Corruption:** Backup the SQLite file regularly.
6. **AI Misuse:** Don't use ProteinMPNN on proteins >500AA without validating training distribution.

16. Implementation Roadmap

Timeline	Focus	Goals
Month 1	**Phases 1-2**	Environment setup, DB init, Basic Loop (5 gens).
Month 2	**Phase 3**	ESMFold integration, Caching, Clustering.
Month 3	**Phase 4**	Pareto optimization, Mutation Jumps, Diversity tracking.
Month 4+	**Phases 5-6**	Docker, Monitoring, Production runs (50+ gens).

17. Final Checklist

- [ ] FoldX license obtained.
- [ ] GPU/CUDA verified.
- [ ] Database schema initialized.
- [ ] Test protein (small) selected.
- [ ] Backup strategy defined.

- [ ] Runtime per generation estimated.
- [ ] Version control initialized.

You are ready to evolve proteins! 🧬

1.