Ligand extraction script

```
from Bio.PDB import PDBParser

pdb_file = r"C:\Users\athar\Downloads\SASDL82_fit1_model1.pdb"
parser = PDBParser(QUIET=True)
structure = parser.get_structure("protein", pdb_file)

ligands = set()

for model in structure:
    for chain in model:
        for residue in chain:
            # HETATM residues are non-standard (ligands, ions)
            if residue.id[0] != " ":
                ligands.add(residue.resname)

print("Ligands found in the PDB file:", ligands)
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from Bio.PDB import PDBParser, PDBIO, Select
from rdkit import Chem
from rdkit.Chem import AllChem, DataStructs
import py3Dmol

import warnings
warnings.filterwarnings("ignore")

PDB_FILE = r"C:\Users\athar\Downloads\SASDL82_fit1_model1.pdb"
OUTPUT_DIR = "ligand_files"
RESULT_CSV = "protein_ligand_docking_results.csv"

IGNORE_RESIDUES = {"HOH", "WAT", "H2O", "NA", "CL", "CA", "K", "MG", "ZN", "MN", "SO4"}

RDKIT_SEED = 42
np.random.seed(42)

class LigandSelect(Select):
    def __init__(self, ligand_name):
        self.ligand_name = ligand_name
    def accept_residue(self, residue):
        return residue.get_resname() == self.ligand_name
```

```
parser = PDBParser(QUIET=True)
try:
    structure = parser.get_structure("protein", PDB_FILE)
except Exception as e:
    print(f"ERROR reading file: {e}")
    sys.exit(1)

ligands_found = []
for model in structure:
    for chain in model:
        for residue in chain:
            resname = residue.get_resname().strip()
            if residue.id[0] != " " and resname not in IGNORE_RESIDUES:
                if resname not in ligands_found:
                    ligands_found.append(resname)

print("Detected ligands:", ligands_found)
if not ligands_found:
    print("No valid ligands found. Exiting.")
    sys.exit(0)

os.makedirs(OUTPUT_DIR, exist_ok=True)
io = PDBIO()
ligand_paths = []
for lig in ligands_found:
    outpath = os.path.join(OUTPUT_DIR, f"{lig}.pdb")
    io.set_structure(structure)
    io.save(outpath, LigandSelect(lig))
    ligand_paths.append(outpath)

rdkit_mols = []
rdkit_names = []

for path in ligand_paths:
    name = os.path.basename(path).replace(".pdb", "")
    mol = Chem.MolFromPDBFile(path, removeHs=False, sanitize=False)
    if mol is None:
        print(f"WARNING: Could not parse {path}")
        continue
    try:
        Chem.SanitizeMol(mol)
    except Exception:
        try:
            Chem.SanitizeMol(mol, Chem.SANITIZE_ALL ^ Chem.SANITIZE_PROPERTIES)
        except Exception:
            pass

    mol = Chem.AddHs(mol, addCoords=True)
    try:
        params = AllChem.ETKDGv3()
```

```
            params.randomSeed = RDKIT_SEED
            res = AllChem.EmbedMolecule(mol, params)
            if res != 0:
                AllChem.EmbedMolecule(mol, AllChem.ETKDG())
        except Exception as ex:
            print(f"Embedding failed for {name}: {ex}")

        try:
            AllChem.MMFFOptimizeMolecule(mol)
        except Exception:
            pass

        rdkit_mols.append(mol)
        rdkit_names.append(name)

    if not rdkit_mols:
        print("No ligands successfully converted. Exiting.")
        sys.exit(0)

    fps = []
    for mol in rdkit_mols:
        try:
            fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048)
        except Exception:
            fp = AllChem.GetMorganFingerprintAsBitVect(Chem.RemoveHs(mol), radius=2, nBits=2048)
        fps.append(fp)

    ref_fp = fps[0]
    ref_name = rdkit_names[0]

    def sim_to_score(sim, low=-4.0, high=-12.0):
        return round(low + (high - low) * sim, 3)

    scores, sims = [], []
    for fp in fps:
        sim = DataStructs.TanimotoSimilarity(ref_fp, fp)
        sims.append(sim)
        scores.append(sim_to_score(sim))

    results_df = pd.DataFrame({
        "Ligand": rdkit_names,
        f"Tanimoto_Similarity_to_{ref_name}": sims,
        "Pseudo_Docking_Score_kcal_per_mol": scores
    }).sort_values("Pseudo_Docking_Score_kcal_per_mol").reset_index(drop=True)

    print("\n=== Docking / Virtual Screening Results ===")
    print(results_df)
    results_df.to_csv(RESULT_CSV, index=False)
    print(f"\nResults saved to {RESULT_CSV}")

    best_row = results_df.iloc[0]
```

```
best_name = best_row["Ligand"]
best_score = best_row["Pseudo_Docking_Score_kcal_per_mol"]

print("\n=== BEST LIGAND ===")
print(f"Ligand: {best_name}")
print(f"Predicted Docking Score: {best_score} kcal/mol")
print(f"Reference Ligand: {ref_name}")

plt.figure(figsize=(max(6, len(results_df)*0.6), 4))
sns.barplot(x="Ligand", y="Pseudo_Docking_Score_kcal_per_mol", data=results_df,
palette="viridis")
plt.xticks(rotation=45, ha="right")
plt.title("Pseudo Docking Scores (lower = better)")
plt.ylabel("Docking Score (kcal/mol)")
plt.tight_layout()
plt.show()

best_file = os.path.join(OUTPUT_DIR, f"{best_name}.pdb")
with open(PDB_FILE, "r") as f:
    pdb_block = f.read()
with open(best_file, "r") as f:
    lig_block = f.read()

view = py3Dmol.view(width=700, height=500)
view.addModel(pdb_block, 'pdb')
view.setStyle({'cartoon': {'color': 'spectrum'}})
view.addModel(lig_block, 'pdb')
view.setStyle({'model': 1}, {'stick': {'radius': 0.25, 'color': 'red'}})
view.setBackgroundColor('0xeeeeee')
view.zoomTo()
view.show()

sdf_out = r"C:\Users\athar\OneDrive\Desktop\Practical\Bioinformatics\results.sdf"
writer = Chem.SDWriter(sdf_out)
for mol, name in zip(rdkit_mols, rdkit_names):
    mol.SetProp("_Name", name)
    writer.write(mol)
writer.close()
print(f"\nExtracted ligand SDF saved to {sdf_out}")
```

explain this code line by line

Alright, Srushti 😊 — this is a **very complete and well-written Python script** for **extracting ligands from a PDB file**, **processing them with RDKit**, **estimating pseudo docking scores**, and **visualizing results** in 3D using **py3Dmol**.

Let's go through it **line-by-line (with explanation)** 👇

🧩 Imports and Setup

python

```python
from Bio.PDB import PDBParser
```

- Imports the **PDBParser** class from **Biopython**.
- This is used to **read and parse PDB (Protein Data Bank)** files (protein structure files).

---

python

```python
pdb_file = r"C:\Users\athar\Downloads\SASDL82_fit1_model1.pdb"
parser = PDBParser(QUIET=True)
structure = parser.get_structure("protein", pdb_file)
```

- **`pdb_file`** is the path to your **`.pdb`** structure file.
- **`PDBParser(QUIET=True)`** prevents warning messages while parsing.
- **`get_structure()`** loads the PDB structure and returns a **Structure object** that contains models → chains → residues → atoms.

🧱 Finding Ligands (non-protein residues)

python

```python
ligands = set()
for model in structure:
    for chain in model:
        for residue in chain:
            if residue.id[0] != " ":
                ligands.add(residue.resname)
print("Ligands found in the PDB file:", ligands)
```

- Iterates through **model** → **chain** → **residue** hierarchy.
- **`residue.id[0] != " "`** means **non-standard residue** (like HETATM = ligand/ion/water).
- Adds unique ligand names to a set.
- Prints them.

⚙️ Additional Imports and Setup

python

```python
import os, sys, numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
from Bio.PDB import PDBParser, PDBIO, Select
from rdkit import Chem
from rdkit.Chem import AllChem, DataStructs
import py3Dmol
import warnings
warnings.filterwarnings("ignore")
```

- These libraries are for:
    - **os/sys**: file & system handling

- **numpy/pandas**: numerical + data manipulation
- **matplotlib/seaborn**: plotting
- **RDKit**: molecule handling, fingerprints, 3D embedding
- **py3Dmol**: 3D structure visualization in notebooks
- **warnings**: to suppress unnecessary warnings.

🗂️ File Paths and Constants

```python
PDB_FILE = r"C:\Users\athar\Downloads\SASDL82_fit1_model1.pdb"
OUTPUT_DIR = "ligand_files"
RESULT_CSV = "protein_ligand_docking_results.csv"
IGNORE_RESIDUES = {"HOH", "WAT", "H2O", "NA", "CL", "CA", "K", "MG", "ZN", "MN", "SO4"}
RDKIT_SEED = 42
np.random.seed(42)
```

- **`OUTPUT_DIR`** → folder where extracted ligands will be saved.
- **`RESULT_CSV`** → where results will be stored.
- **`IGNORE_RESIDUES`** → common **solvent or ion residues** to ignore.
- Seeds are fixed for **reproducibility**.

🎯 Ligand Selection Helper Class

```python
class LigandSelect(Select):
    def __init__(self, ligand_name):
        self.ligand_name = ligand_name
    def accept_residue(self, residue):
        return residue.get_resname() == self.ligand_name
```

- A **custom subclass** of **`Select`** used by **`PDBIO`** to **save only specific residues (ligands)**.
- It returns **`True`** only for residues matching the given ligand name.

📖 Parsing the PDB

```python
parser = PDBParser(QUIET=True)
try:
    structure = parser.get_structure("protein", PDB_FILE)
except Exception as e:
    print(f"ERROR reading file: {e}")
    sys.exit(1)
```

- Loads the PDB file and exits gracefully if it fails.

🔍 Detect Ligands (again, but with filtering)

```python
ligands_found = []
for model in structure:
    for chain in model:
        for residue in chain:
            resname = residue.get_resname().strip()
            if residue.id[0] != " " and resname not in IGNORE_RESIDUES:
                if resname not in ligands_found:
                    ligands_found.append(resname)
```

- Scans the structure and collects **unique ligand names**, skipping water/ions.

```python
print("Detected ligands:", ligands_found)
if not ligands_found:
    print("No valid ligands found. Exiting.")
    sys.exit(0)
```

- Prints detected ligands and stops if none found.

💾 Extract Each Ligand into Separate PDB Files

```python
os.makedirs(OUTPUT_DIR, exist_ok=True)
io = PDBIO()
ligand_paths = []
for lig in ligands_found:
    outpath = os.path.join(OUTPUT_DIR, f"{lig}.pdb")
    io.set_structure(structure)
    io.save(outpath, LigandSelect(lig))
    ligand_paths.append(outpath)
```

- Creates output folder.
- Uses **PDBIO** to write ligand-only PDBs.
- Each ligand gets saved as **`ligand_name.pdb`**.

🧪 Convert Ligands to RDKit Molecules

```python
rdkit_mols = []
rdkit_names = []

for path in ligand_paths:
    name = os.path.basename(path).replace(".pdb", "")
    mol = Chem.MolFromPDBFile(path, removeHs=False, sanitize=False)
```

- Reads each ligand file as an RDKit molecule (without sanitizing yet).

---

python

```python
    if mol is None:
        print(f"WARNING: Could not parse {path}")
        continue
```

- Skips if RDKit fails to parse.

🧼 Molecule Sanitization and Optimization

python

```python
    try:
        Chem.SanitizeMol(mol)
    except Exception:
        try:
            Chem.SanitizeMol(mol, Chem.SANITIZE_ALL ^ Chem.SANITIZE_PROPERTIES)
        except Exception:
            pass
```

- “Sanitization” = checking valency, aromaticity, etc.
- Tries fallback sanitization if full one fails.

---

python

```python
    mol = Chem.AddHs(mol, addCoords=True)
```

- Adds **explicit hydrogens** with 3D coordinates.

---

python

```python
    try:
        params = AllChem.ETKDGv3()
        params.randomSeed = RDKIT_SEED
        res = AllChem.EmbedMolecule(mol, params)
        if res != 0:
            AllChem.EmbedMolecule(mol, AllChem.ETKDG())
    except Exception as ex:
        print(f"Embedding failed for {name}: {ex}")
```

- Generates **3D coordinates** using RDKit’s ETKDG algorithm.
- If embedding fails, retries with a simpler method.

---

python

```python
    try:
        AllChem.MMFFOptimizeMolecule(mol)
    except Exception:
        pass
```

- Runs **MMFF94 force-field optimization** to relax structure geometry.

```python
    rdkit_mols.append(mol)
    rdkit_names.append(name)
```

- Stores the successfully created ligand molecules.

⚠️ Exit if No Molecules Found

```python
if not rdkit_mols:
    print("No ligands successfully converted. Exiting.")
    sys.exit(0)
```

🧠 Generate Molecular Fingerprints

```python
fps = []
for mol in rdkit_mols:
    try:
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048)
    except Exception:
        fp = AllChem.GetMorganFingerprintAsBitVect(Chem.RemoveHs(mol), radius=2, nBits=2048)
    fps.append(fp)
```

- Converts molecules to **Morgan fingerprints** (circular fingerprints used for similarity).
- Each fingerprint is a 2048-bit vector representing molecular features.

🧮 Compute Similarity and “Docking Scores”

```python
ref_fp = fps[0]
ref_name = rdkit_names[0]
```

- Uses the **first ligand** as a reference molecule.

```python
def sim_to_score(sim, low=-4.0, high=-12.0):
    return round(low + (high - low) * sim, 3)
```

- Converts similarity (0–1) to a **pseudo docking score** (lower is better).
  (Not real docking, just a similarity-based approximation.)

```python
scores, sims = [], []
for fp in fps:
    sim = DataStructs.TanimotoSimilarity(ref_fp, fp)
    sims.append(sim)
    scores.append(sim_to_score(sim))
```

- Computes **Tanimoto similarity** between reference ligand and each ligand.
- Converts it to a docking score.

📊 Store Results in a DataFrame

```python
results_df = pd.DataFrame({
    "Ligand": rdkit_names,
    f"Tanimoto_Similarity_to_{ref_name}": sims,
    "Pseudo_Docking_Score_kcal_per_mol": scores
}).sort_values("Pseudo_Docking_Score_kcal_per_mol").reset_index(drop=True)
```

- Builds a results table and sorts by docking score.

```python
print("\n=== Docking / Virtual Screening Results ===")
print(results_df)
results_df.to_csv(RESULT_CSV, index=False)
print(f"\nResults saved to {RESULT_CSV}")
```

- Displays and saves results to a CSV file.

🏆 Select Best Ligand

```python
best_row = results_df.iloc[0]
best_name = best_row["Ligand"]
best_score = best_row["Pseudo_Docking_Score_kcal_per_mol"]

print("\n=== BEST LIGAND ===")
print(f"Ligand: {best_name}")
print(f"Predicted Docking Score: {best_score} kcal/mol")
print(f"Reference Ligand: {ref_name}")
```

- Picks the **ligand with the lowest (best) score**.

📈 Visualize Results

```python
plt.figure(figsize=(max(6, len(results_df)*0.6), 4))
sns.barplot(x="Ligand", y="Pseudo_Docking_Score_kcal_per_mol", data=results_df,
palette="viridis")
plt.xticks(rotation=45, ha="right")
plt.title("Pseudo Docking Scores (lower = better)")
plt.ylabel("Docking Score (kcal/mol)")
plt.tight_layout()
plt.show()
```

- Plots a bar chart of docking scores using Seaborn/Matplotlib.

🧬 3D Visualization

```
python

best_file = os.path.join(OUTPUT_DIR, f"{best_name}.pdb")
with open(PDB_FILE, "r") as f:
    pdb_block = f.read()
with open(best_file, "r") as f:
    lig_block = f.read()

view = py3Dmol.view(width=700, height=500)
view.addModel(pdb_block, 'pdb')
view.setStyle({'cartoon': {'color': 'spectrum'}})
view.addModel(lig_block, 'pdb')
view.setStyle({'model': 1}, {'stick': {'radius': 0.25, 'color': 'red'}})
view.setBackgroundColor('0xeeeeee')
view.zoomTo()
view.show()
```

- Loads the full protein and the **best ligand** into a 3D viewer.
- Protein → cartoon view; ligand → red sticks.
- Renders it interactively with **`py3Dmol`**.

💾 Export Ligands as SDF

```
python

sdf_out = r"C:\Users\athar\OneDrive\Desktop\Practical\Bioinformatics\results.sdf"
writer = Chem.SDWriter(sdf_out)
for mol, name in zip(rdkit_mols, rdkit_names):
    mol.SetProp("_Name", name)
    writer.write(mol)
writer.close()
print(f"\nExtracted ligand SDF saved to {sdf_out}")
```

- Saves all ligands into a **single SDF file** (standard for small molecules).

🧠 Summary

Step	Purpose
Parse PDB	Read protein structure
Identify ligands	Detect non-standard residues
Extract ligands	Save each as individual PDB
Convert to RDKit	Handle molecular structure in RDKit

Step	Purpose
Generate fingerprints	For similarity comparison
Compute pseudo docking scores	Based on similarity
Visualize	Plot scores & show 3D protein-ligand complex
Save results	Export as CSV + SDF

If you want, I can show you how to **modify this code to perform *real docking*** using **AutoDock Vina** or **prepare it for machine learning screening** — would you like me to explain that next?

what is rdkit

Great question, Srushti 🌿 — let's break it down clearly 👇

🧪 What is RDKit?

RDKit is a **powerful open-source cheminformatics toolkit** written in C++ and Python.
It's widely used in **computational chemistry, bioinformatics, and drug discovery** for working with small molecules and chemical structures.

🧬 In simple terms

RDKit helps you:

> "Read, write, analyze, and modify molecules programmatically."

You can think of it like **NumPy for chemistry** — instead of numbers and arrays, you work with **atoms, bonds, and molecules**.

⚙️ What RDKit Can Do

Here are the main things RDKit is great at 👇

Category	Example Tasks
Molecule I/O	Read/write `.sdf`, `.mol`, `.pdb`, `.smiles`, etc.
Structure Representation	Represent molecules as `Mol` objects with atoms, bonds, coordinates
2D/3D Structure Generation	Generate 2D depictions or 3D conformers using ETKDG
Descriptors & Fingerprints	Compute molecular fingerprints, molecular weight, logP, etc.
Similarity & Searching	Compare molecules using **Tanimoto similarity**, substructure searches
Molecular Manipulation	Add/remove atoms, hydrogens, modify bonds
Visualization	Draw molecules in Jupyter or save 2D images
Force Field Optimization	Use MMFF94 or UFF to optimize 3D conformations
Machine Learning Features	Create numerical descriptors for QSAR/QSPR models

💻 Example: Basic Usage

```
python

from rdkit import Chem
from rdkit.Chem import AllChem, Draw

# Create a molecule from a SMILES string
mol = Chem.MolFromSmiles("CCO")  # Ethanol

# Add hydrogens and generate 3D coordinates
mol = Chem.AddHs(mol)
AllChem.EmbedMolecule(mol)

# Optimize structure
AllChem.MMFFOptimizeMolecule(mol)

# Draw the molecule
Draw.MolToImage(mol)
```

➡️ This snippet:

1. Creates ethanol (`CCO`).
2. Adds hydrogens.
3. Generates a 3D structure.
4. Optimizes geometry.
5. Displays the molecule visually.

🧠 Why It's Important

RDKit is **essential in bioinformatics & drug discovery pipelines**, especially for:

- Virtual screening
- Docking preparation
- Ligand-based similarity searches
- Molecular descriptor generation