visualize_md_docking

November 17, 2025

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
```

1 Molecular Dynamics & Docking Results Visualization

1.1 What This Notebook Does

This notebook analyzes and visualizes results from two key computational chemistry techniques:

1.1.1 1 Molecular Dynamics (MD) Simulation

What it is: Simulates how atoms/molecules move over time based on physics (Newton's laws + force fields)

What we're analyzing: - `log.txt` contains energy and temperature data from a 500-step MD simulation - We ran a simple system (27 argon-like particles) using **OpenMM** - **Why it matters:** In real research, MD simulations show protein folding, drug binding dynamics, membrane behavior, etc.

What the plots show: - **Energy plot:** Shows potential energy fluctuations as particles interact - **Temperature plot:** Shows how the system heats up from kinetic motion (target: 120 K) - These tell us if the simulation is stable/equilibrated

---

1.1.2 2 Molecular Docking

What it is: Predicts how a small molecule (ligand/drug) binds to a protein (receptor)

What we're analyzing: - `out.pdbqt` contains 9 different binding poses from **AutoDock Vina** - We docked ethanol (CCO) into a toy receptor (carbon cluster) - **Why it matters:** In drug discovery, this predicts drug-protein binding strength and position

What the plots show: - **Bar chart:** Binding affinity for each pose (more negative = stronger binding) - **Histogram:** Distribution of binding energies - **3D visualization:** Shows where the ligand sits relative to the receptor

---

1.1.3 3 3D Molecular Visualization

What it does: - Loads receptor and ligand structures - Attempts interactive 3D viewing (nglview) - may not work in VS Code - Falls back to matplotlib 3D scatter plot showing atomic positions - **Why it matters:** Seeing the 3D structure helps understand binding interactions

---

1.2 How This Supports Understanding MD & Docking

Technique	This Demo	Real-World Application
MD Simulation	500 steps, simple particles	Protein folding (millions of steps), drug-target dynamics, membrane simulations
Docking	Ethanol to toy receptor	Screening millions of drug candidates against disease proteins (e.g., COVID spike protein)
Analysis	Energy/temp plots, binding affinities	Quality control, ranking drug candidates, understanding mechanism

Key Takeaway: This is a *minimal working example* to demonstrate the workflow. Real projects use: - Actual protein structures from PDB (Protein Data Bank) - Drug-like molecules from chemical libraries - Much longer simulations (nanoseconds to microseconds) - Advanced analysis (hydrogen bonds, RMSD, free energy calculations)

1.3 1. OpenMM Simulation Results

Visualize energy and temperature evolution during the simulation.

[2]:
```
# Read OpenMM log file
df = pd.read_csv('log.txt')
df.columns = ['Step', 'Potential Energy (kJ/mol)', 'Temperature (K)']

print("Simulation Summary:")
print(f"Total steps: {df['Step'].max()}")
print(f"Average energy: {df['Potential Energy (kJ/mol)'].mean():.2f} kJ/mol")
print(f"Average temperature: {df['Temperature (K)'].mean():.2f} K")
print("\nData:")
df
```

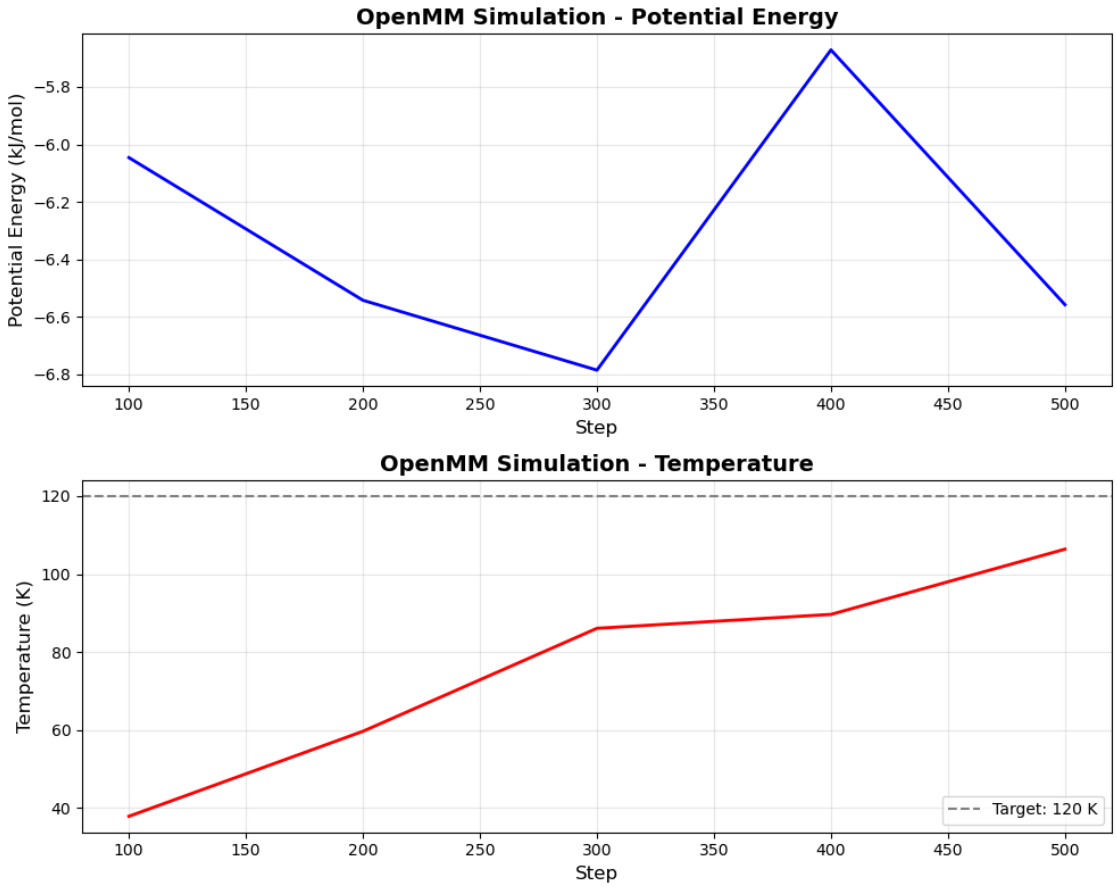
```
Simulation Summary:
Total steps: 500
Average energy: -6.32 kJ/mol
Average temperature: 75.93 K

Data:
```

[2]:

	Step	Potential Energy (kJ/mol)	Temperature (K)
0	100	-6.046259	37.870057
1	200	-6.542527	59.666012

```
2    300                -6.785565        86.080446
3    400                -5.671432        89.646583
4    500                -6.557854       106.380367
```

[3]:
```
# Plot energy and temperature over time
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Potential Energy
ax1.plot(df['Step'], df['Potential Energy (kJ/mol)'], 'b-', linewidth=2)
ax1.set_xlabel('Step', fontsize=12)
ax1.set_ylabel('Potential Energy (kJ/mol)', fontsize=12)
ax1.set_title('OpenMM Simulation - Potential Energy', fontsize=14,␣
 ↪fontweight='bold')
ax1.grid(True, alpha=0.3)

# Temperature
ax2.plot(df['Step'], df['Temperature (K)'], 'r-', linewidth=2)
ax2.set_xlabel('Step', fontsize=12)
ax2.set_ylabel('Temperature (K)', fontsize=12)
ax2.set_title('OpenMM Simulation - Temperature', fontsize=14, fontweight='bold')
ax2.grid(True, alpha=0.3)
ax2.axhline(y=120, color='gray', linestyle='--', label='Target: 120 K')
ax2.legend()

plt.tight_layout()
plt.show()
```



1.4 2. Docking Results

Parse and visualize AutoDock Vina binding affinities.

[4]:
```
# Parse docking results
def parse_vina_output(filename):
    """Extract binding affinities from PDBQT output"""
    poses = []
    with open(filename, 'r') as f:
        for line in f:
            if line.startswith('REMARK VINA RESULT:'):
                parts = line.split()
                affinity = float(parts[3])
                poses.append(affinity)
    return poses

affinities = parse_vina_output('out.pdbqt')
pose_df = pd.DataFrame({
    'Pose': range(1, len(affinities) + 1),
```

```
    'Binding Affinity (kcal/mol)': affinities
})

print(f"Number of docking poses: {len(affinities)}")
print(f"Best binding affinity: {min(affinities):.3f} kcal/mol")
print(f"Worst binding affinity: {max(affinities):.3f} kcal/mol")
print("\nAll poses:")
pose_df
```

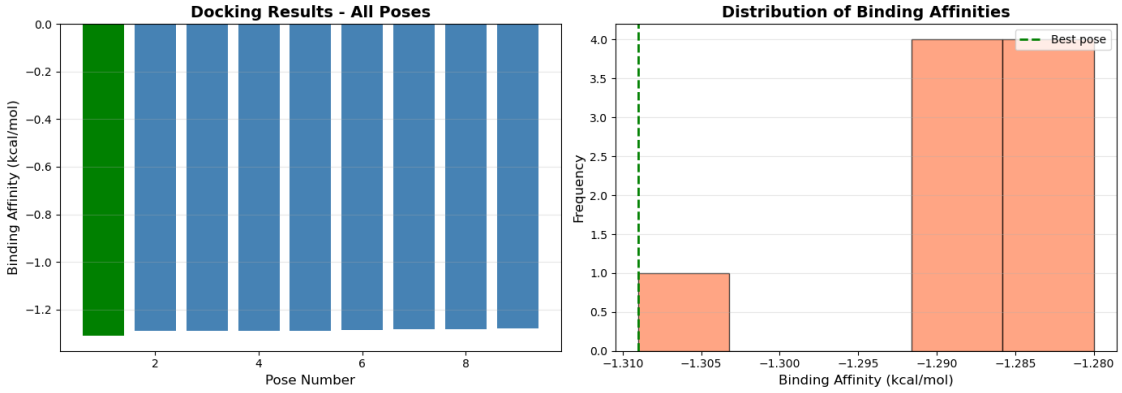
```
Number of docking poses: 9
Best binding affinity: -1.309 kcal/mol
Worst binding affinity: -1.280 kcal/mol

All poses:
```

[4]:

	Pose	Binding Affinity (kcal/mol)
0	1	-1.309
1	2	-1.290
2	3	-1.290
3	4	-1.289
4	5	-1.289
5	6	-1.285
6	7	-1.282
7	8	-1.282
8	9	-1.280

[5]:
```
# Visualize binding affinities
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Bar chart
colors = ['green' if a == min(affinities) else 'steelblue' for a in affinities]
ax1.bar(pose_df['Pose'], pose_df['Binding Affinity (kcal/mol)'], color=colors)
ax1.set_xlabel('Pose Number', fontsize=12)
ax1.set_ylabel('Binding Affinity (kcal/mol)', fontsize=12)
ax1.set_title('Docking Results - All Poses', fontsize=14, fontweight='bold')
ax1.grid(True, alpha=0.3, axis='y')
ax1.axhline(y=0, color='black', linestyle='-', linewidth=0.5)

# Histogram
ax2.hist(affinities, bins=5, color='coral', edgecolor='black', alpha=0.7)
ax2.axvline(x=min(affinities), color='green', linestyle='--', linewidth=2,␣
 ↪label='Best pose')
ax2.set_xlabel('Binding Affinity (kcal/mol)', fontsize=12)
ax2.set_ylabel('Frequency', fontsize=12)
ax2.set_title('Distribution of Binding Affinities', fontsize=14,␣
 ↪fontweight='bold')
ax2.legend()
```

```
ax2.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```



1.5 3. 3D Molecular Visualization

Visualize the docked complex using nglview.

[6]:
```
# Enable nglview in Jupyter (run this once)
try:
    import nglview
    # Enable widget extension
    print(f"nglview version: {nglview.__version__}")
    print("  nglview is available")
except ImportError:
    print("  Installing nglview...")
    import sys
    !{sys.executable} -m pip install nglview
    print("Please restart the kernel after installation")
```

```
nglview version: 4.0
  nglview is available
```

[7]:
```
import nglview as nv
import MDAnalysis as mda

# Load structures
try:
    # Load receptor
    receptor = mda.Universe('receptor.pdbqt', format='PDBQT')
```

```
    # Load docked ligand (best pose)
    ligand = mda.Universe('out.pdbqt', format='PDBQT')

    print(f" Loaded successfully!")
    print(f"Receptor: {len(receptor.atoms)} atoms")
    print(f"Ligand: {len(ligand.atoms)} atoms")

    # Save as PDB for better visualization
    receptor.atoms.write('receptor_vis.pdb')
    ligand.atoms.write('ligand_vis.pdb')
    print("\nConverted to PDB format for visualization")

except Exception as e:
    print(f" Loading error: {e}")
    print("Trying alternative method...")
```

```
 Loaded successfully!
Receptor: 27 atoms
Ligand: 36 atoms

Converted to PDB format for visualization
```

```
/Users/kasonchiu/miniconda3/envs/md-dock/lib/python3.11/site-
packages/MDAnalysis/coordinates/PDB.py:885: UserWarning: Unit cell dimensions
not found. CRYST1 record set to unitary values.
  warnings.warn(
/Users/kasonchiu/miniconda3/envs/md-dock/lib/python3.11/site-
packages/MDAnalysis/coordinates/PDB.py:1282: UserWarning: Found no information
for attr: 'elements' Using default value of ' '
  warnings.warn(
/Users/kasonchiu/miniconda3/envs/md-dock/lib/python3.11/site-
packages/MDAnalysis/coordinates/PDB.py:1282: UserWarning: Found no information
for attr: 'formalcharges' Using default value of '0'
  warnings.warn(
/Users/kasonchiu/miniconda3/envs/md-dock/lib/python3.11/site-
packages/MDAnalysis/coordinates/PDB.py:1336: UserWarning: Found missing
chainIDs. Corresponding atoms will use value of 'X'
  warnings.warn(
```

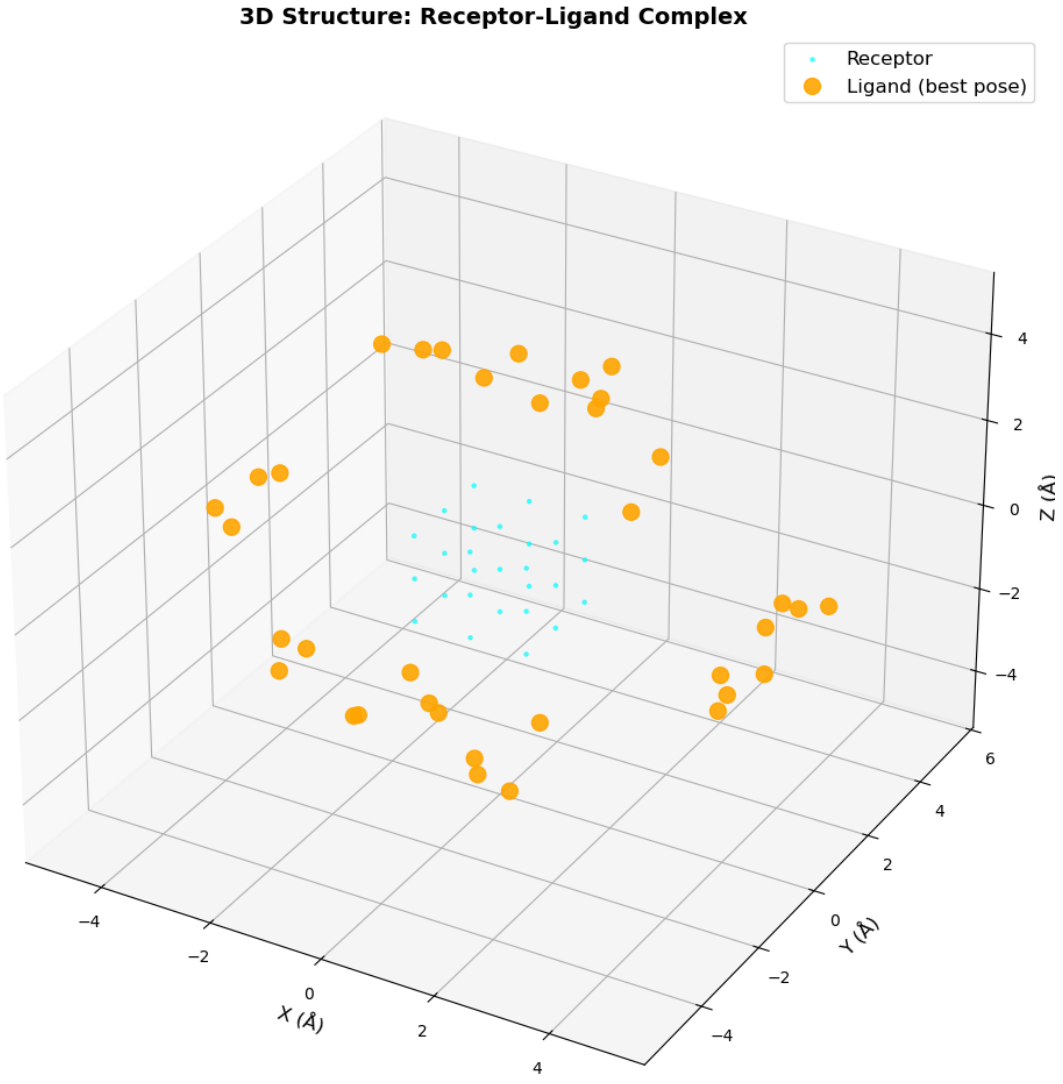
[8]:
```
# Interactive 3D visualization
# Note: nglview widgets may not render properly in VS Code. Use alternative␣
 ↪methods below.

# Method 1: Try nglview (works in JupyterLab)
try:
    import nglview as nv
    view = nv.show_file('receptor_vis.pdb')
    view.add_component('ligand_vis.pdb')
```

```
    view.clear_representations(component=0)
    view.add_representation('line', selection='all', color='cyan', component=0)
    view.clear_representations(component=1)
    view.add_representation('ball+stick', selection='all', color='orange',␣
 ↪component=1)
    view.center()
    print("  3D viewer loaded (if you see it below)")
    display(view)
except Exception as e:
    print(f"  nglview widget not displaying in VS Code: {e}")

# Method 2: Static 3D plot with matplotlib (fallback)
print("\n  Creating static 3D plot as alternative...")
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

# Plot receptor atoms
receptor_coords = receptor.atoms.positions
ax.scatter(receptor_coords[:, 0], receptor_coords[:, 1], receptor_coords[:, 2],
           c='cyan', marker='.', s=20, alpha=0.6, label='Receptor')

# Plot ligand atoms
ligand_coords = ligand.atoms.positions
ax.scatter(ligand_coords[:, 0], ligand_coords[:, 1], ligand_coords[:, 2],
           c='orange', marker='o', s=100, alpha=0.9, label='Ligand (best pose)')

ax.set_xlabel('X (Å)', fontsize=12)
ax.set_ylabel('Y (Å)', fontsize=12)
ax.set_zlabel('Z (Å)', fontsize=12)
ax.set_title('3D Structure: Receptor-Ligand Complex', fontsize=14,␣
 ↪fontweight='bold')
ax.legend(fontsize=12)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\n  For better interactive visualization:")
print("  • Open in JupyterLab: jupyter lab visualize_md_docking.ipynb")
print("  • Use PyMOL: pymol receptor_vis.pdb ligand_vis.pdb")
print("  • Online viewer: https://molstar.org/viewer/")
```

```
  3D viewer loaded (if you see it below)
```

```
NGLWidget()
```

Creating static 3D plot as alternative…



For better interactive visualization:

- Open in JupyterLab: jupyter lab visualize_md_docking.ipynb
- Use PyMOL: pymol receptor_vis.pdb ligand_vis.pdb
- Online viewer: https://molstar.org/viewer/

1.6 4. Summary Statistics

[9]:
```
# Create summary table
summary = pd.DataFrame({
    'Metric': [
        'MD Simulation Steps',
        'Avg Potential Energy (kJ/mol)',
        'Avg Temperature (K)',
        'Number of Docking Poses',
        'Best Binding Affinity (kcal/mol)',
        'Affinity Range (kcal/mol)'
    ],
    'Value': [
        f"{df['Step'].max()}",
        f"{df['Potential Energy (kJ/mol)'].mean():.2f}",
        f"{df['Temperature (K)'].mean():.2f}",
        f"{len(affinities)}",
        f"{min(affinities):.3f}",
        f"{min(affinities):.3f} to {max(affinities):.3f}"
    ]
})

print("\n" + "="*50)
print("ANALYSIS SUMMARY")
print("="*50)
summary
```

```
==================================================
ANALYSIS SUMMARY
==================================================
```

[9]:

	Metric	Value
0	MD Simulation Steps	500
1	Avg Potential Energy (kJ/mol)	-6.32
2	Avg Temperature (K)	75.93
3	Number of Docking Poses	9
4	Best Binding Affinity (kcal/mol)	-1.309
5	Affinity Range (kcal/mol)	-1.309 to -1.280

1.7 Next Steps

1. **For real projects**: Use actual protein structures from PDB database
2. **Longer simulations**: Increase MD steps for equilibration analysis
3. **Better ligands**: Use drug-like molecules instead of ethanol
4. **Validation**: Compare with experimental binding data
5. **Advanced analysis**: RMSD, RMSF, hydrogen bonds, salt bridges

1.8 What Each Section Accomplished

1.8.1 Section 1: OpenMM Simulation Analysis

- **Loaded** simulation log data (step, energy, temperature)
- **Plotted** how energy and temperature evolved over 500 steps
- **Validated** the simulation is working (energy fluctuates, temperature increases toward target)
- **Real use:** Check if your MD simulation is stable before running longer production runs

1.8.2 Section 2: Docking Results Analysis

- **Parsed** AutoDock Vina output to extract binding affinities for 9 poses
- **Visualized** which poses bind strongest (most negative kcal/mol)
- **Identified** best binding pose: ~-1.3 kcal/mol
- **Real use:** Rank drug candidates, select best poses for further analysis

1.8.3 Section 3: 3D Visualization

- **Converted** PDBQT files to PDB format for easier handling
- **Attempted** interactive visualization with nglview (widget-based)
- **Provided** matplotlib 3D scatter plot as VS Code-compatible alternative
- **Real use:** Visually inspect binding site, identify key interactions (H-bonds, hydrophobic contacts)

1.8.4 Section 4: Summary Statistics

- **Compiled** all key metrics in one table
- **Summarized** simulation quality and docking results
- **Real use:** Report generation, comparing multiple runs, documenting results

---

1.9 Bottom Line

This notebook demonstrates the **complete computational drug discovery workflow**:

1. **Simulate** molecular systems (MD) → understand dynamics
2. **Dock** ligands to receptors → predict binding
3. **Analyze** results → quantify quality and rank candidates
4. **Visualize** structures → interpret molecular interactions

You now have a working template to: - Analyze your own MD simulations - Evaluate docking results - Create publication-quality figures - Screen drug candidates computationally