Step-by-Step of Molecular Docking

1. Introduction to Molecular Docking

Molecular docking is a computational technique used to predict how two molecules, such as a protein and DNA, interact with each other to form a stable complex. The goal is to estimate the strength of binding and identify the binding modes.
In my project, I performed molecular docking of the RFX1 gene's transcription factor using a systematic approach. Below, I describe each and every step I performed, along with the exact commands I used.

---

2. Methodology

Step 1: Creating Necessary Folders

Command:

bash

```
mkdir Binding_Affinity Identify_Loss_of_Favorable_Interactions MUT_Docking Wild_Docking
Electrostatic_Repulsion Input Steric_Clashes
```

Explanation:

- I used the `mkdir` command to create multiple folders at once.
- These folders are used to organize different parts of the molecular docking analysis:
  - `Binding_Affinity`: for analyzing binding strength.
  - `Identify_Loss_of_Favorable_Interactions`: to store unfavorable interactions.
  - `MUT_Docking`: for mutant protein docking results.
  - `Wild_Docking`: for wild-type protein docking results.
  - `Electrostatic_Repulsion`: for electrostatic clash studies.
  - `Input`: for storing input PDB files.
  - `Steric_Clashes`: for steric clash studies.

Step 2: Copying Required Files

Command:

bash

```
cp /home/zero/RFX1/Phylogeny/TF_structure.pdb /home/zero/RFX1/Molecular_Docking/Input
&& cp /home/zero/RFX1/Phylogeny/TF_structure_mutations.pdb
/home/zero/RFX1/Molecular_Docking/Input
```

Explanation:

- I copied two PDB structure files from the Phylogeny folder to the Molecular Docking Input folder.
- `cp` is used to copy files.
- First, I copied `TF_structure.pdb` (wild-type).
- Then, I copied `TF_structure_mutations.pdb` (mutated).
- Both files were placed in the `/Input` folder for further processing.

Step 3: Separating DNA and Protein Using PyMOL

Command (PyMOL Script):

```
pml

load complex.pdb
remove resn HOH
remove solvent
select protein_only, polymer.protein
save MUT_Protein.pdb, protein_only
select dna_only, polymer.nucleic
save MUT_DNA.pdb, dna_only
quit
```

Run in Terminal:

```
bash

pymol -c split_complex.pml
```

Explanation:

- I loaded the PDB complex file in PyMOL.
- `remove resn HOH` and `remove solvent` removed all water molecules.
- `select protein_only, polymer.protein` selected only the protein part.
- `save MUT_Protein.pdb, protein_only` saved the protein.
- `select dna_only, polymer.nucleic` selected the DNA part.
- `save MUT_DNA.pdb, dna_only` saved the DNA.
- `quit` closed PyMOL after saving.
- I executed the script in command-line mode using `pymol -c split_complex.pml`.

Step 4: Converting PDB Files to PDBQT Format

Command:

Using AutoDockTools:

```bash
prepare_receptor4.py -r protein.pdb -o protein.pdbqt
prepare_ligand4.py -l dna.pdb -o dna.pdbqt
```

Or using Open Babel:

```bash
obabel MUT_Protein.pdb -O MUT_Protein.pdbqt
obabel MUT_DNA.pdb -O MUT_DNA.pdbqt
```

Explanation:

- I converted the PDB files into PDBQT format, required for docking with AutoDock Vina.
- Using AutoDockTools scripts:
  - `prepare_receptor4.py`: prepared the protein.
  - `prepare_ligand4.py`: prepared the DNA.
- Alternatively, I used `obabel` (Open Babel tool) to convert the files.

Step 5: Creating Config.txt File for Docking

Command:

(for Wild_Docking)

```txt
receptor = Wild_Protein.pdbqt
ligand = Wild_DNA.pdbqt
center_x = 38.6266
center_y = 38.2328
center_z = 20.8931
size_x = 30
size_y = 30
size_z = 30
exhaustiveness = 8
num_modes = 10
energy_range = 4
```

(for MUT_Docking)

```txt
receptor = MUT_Protein.pdbqt
ligand = MUT_DNA.pdbqt
center_x = 38.6266
center_y = 38.2328
center_z = 20.8931
size_x = 30
size_y = 30
```

```
size_z = 30
exhaustiveness = 8
num_modes = 10
energy_range = 4
```

Explanation:

- I manually created `config.txt` files.
- It specifies:
  - Which receptor and ligand to use.
  - The docking grid center (x, y, z).
  - The size of the docking box.
  - Search exhaustiveness.
  - Number of docking poses generated.
  - Energy range for poses.

Step 6: Running Docking (for Wild and Mutant)

Wild_Docking.sh Script

Command:

```
bash

nano Wild_Docking.sh
```

Then, I wrote the following script:

```
bash

#!/bin/bash

# === Filenames ===
RECEPTOR_PDBQT="Wild_Protein.pdbqt"
LIGAND_PDBQT="Wild_DNA.pdbqt"
RECEPTOR_PDB="Wild_Protein.pdb"
CONFIG="config.txt"
OUTPUT="docked_output.pdbqt"
LOG="docking_log.txt"

# === Step 1: Auto-calculate docking box center from PDB ===
echo "[*] Calculating docking box center from $RECEPTOR_PDB..."
CENTER=$(grep "^ATOM" "$RECEPTOR_PDB" | awk '{x+=$6; y+=$7; z+=$8; n++} END {if(n>0)
print x/n, y/n, z/n; else print "0 0 0"}')
CX=$(echo $CENTER | cut -d' ' -f1)
CY=$(echo $CENTER | cut -d' ' -f2)
CZ=$(echo $CENTER | cut -d' ' -f3)

# === Step 2: Create config file ===
echo "[*] Creating config file..."
cat > "$CONFIG" <<EOF
receptor = $RECEPTOR_PDBQT
ligand = $LIGAND_PDBQT
```

```
center_x = $CX
center_y = $CY
center_z = $CZ
size_x = 30
size_y = 30
size_z = 30
exhaustiveness = 8
num_modes = 10
energy_range = 4
EOF

# === Step 3: Run docking ===
echo "[*] Running AutoDock Vina..."
vina --config "$CONFIG" --out "$OUTPUT" --log "$LOG"

# === Step 4: Convert output to PDB (if Open Babel is available) ===
if [ -f "$OUTPUT" ]; then
  echo "[*] Converting docked PDBQT to PDB..."
  obabel "$OUTPUT" -O docked_result.pdb > /dev/null
  echo "[✓] Docking complete!"
  echo "[📁] Docked structure: docked_result.pdb"
  echo "[📜] Docking log: $LOG"
else
  echo "[✗] Docking failed. Check $LOG for details."
fi
```

Running the script:

```bash
chmod +x Wild_Docking.sh
./Wild_Docking.sh
```

Wild_Docking.sh Script Explanation (Line-by-Line)

```bash
#!/bin/bash
```

- This line tells the system that the script should be run using Bash shell.

```bash
# === Filenames ===
RECEPTOR_PDBQT="Wild_Protein.pdbqt"
LIGAND_PDBQT="Wild_DNA.pdbqt"
RECEPTOR_PDB="Wild_Protein.pdb"
CONFIG="config.txt"
OUTPUT="docked_output.pdbqt"
LOG="docking_log.txt"
```

- Here, I define variables for filenames that will be used throughout the script:
  - `RECEPTOR_PDBQT` = protein in PDBQT format,
  - `LIGAND_PDBQT` = DNA in PDBQT format,
  - `RECEPTOR_PDB` = protein in normal PDB format,

    - `CONFIG` = configuration file,
    - `OUTPUT` = output file of docking,
    - `LOG` = file where docking logs are saved.

```bash
# === Step 1: Auto-calculate docking box center from PDB ===
echo "[*] Calculating docking box center from $RECEPTOR_PDB..."
```

- Prints a message that docking box center calculation is starting.

```bash
CENTER=$(grep "^ATOM" "$RECEPTOR_PDB" | awk '{x+=$6; y+=$7; z+=$8; n++} END {if(n>0)
print x/n, y/n, z/n; else print "0 0 0"}')
```

- This line:
    - Searches all atoms (`grep "^ATOM"`) in the PDB file,
    - Extracts their X, Y, and Z coordinates,
    - Calculates the **average X, Y, Z** values using `awk`,
    - These averages give the **center** of the docking box.

```bash
CX=$(echo $CENTER | cut -d' ' -f1)
CY=$(echo $CENTER | cut -d' ' -f2)
CZ=$(echo $CENTER | cut -d' ' -f3)
```

- This splits the center coordinates into three separate values:
    - `CX` = center X-coordinate,
    - `CY` = center Y-coordinate,
    - `CZ` = center Z-coordinate.

```bash
# === Step 2: Create config file ===
echo "[*] Creating config file..."
```

- Prints a message saying I am creating a configuration file.

```bash
cat > "$CONFIG" <<EOF
(receptor, ligand, center, size, exhaustiveness, etc.)
EOF
```

- Creates the **config.txt** file using the previously defined variables.

- `cat > "$CONFIG"` starts writing into the file.
- `<<EOF` and `EOF` define where the writing starts and ends.

```
bash

# === Step 3: Run docking ===
echo "[*] Running AutoDock Vina..."
vina --config "$CONFIG" --out "$OUTPUT" --log "$LOG"
```

- Prints a message that docking is starting.
- Runs AutoDock Vina with:
    - `--config` to read config.txt,
    - `--out` to save docking output,
    - `--log` to save the docking process log.

```
bash

# === Step 4: Convert output to PDB (if Open Babel is available) ===
if [ -f "$OUTPUT" ]; then
  echo "[*] Converting docked PDBQT to PDB..."
  obabel "$OUTPUT" -O docked_result.pdb > /dev/null
```

- Checks if docking output file exists.
- If yes:
    - Converts PDBQT output file to a normal PDB file using Open Babel (`obabel`).

```
bash

  echo "[✓] Docking complete!"
  echo "[📁] Docked structure: docked_result.pdb"
  echo "[📜] Docking log: $LOG"
else
  echo "[✗] Docking failed. Check $LOG for details."
fi
```

- If docking succeeded:
    - Prints success message,
    - Shows location of docked PDB structure and log file.
- If docking failed:
    - Prints failure message.

MUT_Docking.sh Script

Command:

```
bash

nano MUT_Docking.sh
```

Then, I wrote the following script:

```
bash

#!/bin/bash

# === Filenames ===
RECEPTOR="MUT_Protein.pdbqt"
LIGAND="MUT_DNA.pdbqt"
CONFIG="config.txt"
OUTPUT="docked_output.pdbqt"
LOG="docking_log.txt"

# === Step 1: Auto-calculate docking box center from receptor ===
echo "[*] Calculating docking box center from $RECEPTOR..."
CENTER=$(grep "^ATOM" MUT_Protein.pdb | awk '{x+=$6; y+=$7; z+=$8; n++} END {print x/n,
y/n, z/n}')
CX=$(echo $CENTER | cut -d' ' -f1)
CY=$(echo $CENTER | cut -d' ' -f2)
CZ=$(echo $CENTER | cut -d' ' -f3)

# === Step 2: Create config file ===
echo "[*] Creating config file..."
cat > $CONFIG <<EOF
receptor = $RECEPTOR
ligand = $LIGAND
center_x = $CX
center_y = $CY
center_z = $CZ
size_x = 30
size_y = 30
size_z = 30
exhaustiveness = 8
num_modes = 10
energy_range = 4
EOF

# === Step 3: Run docking ===
echo "[*] Running AutoDock Vina..."
vina --config "$CONFIG" --out "$OUTPUT" --log "$LOG"

# === Step 4: Convert output to PDB (if Open Babel is available) ===
echo "[*] Converting docked PDBQT to PDB..."
obabel "$OUTPUT" -O docked_result.pdb > /dev/null

echo "[✓] Docking complete!"
echo "[📁] Docked structure: docked_result.pdb"
echo "[📜] Docking log: $LOG"
```

Running the script:

```
bash

chmod +x MUT_Docking.sh
./MUT_Docking.sh
```

MUT_Docking.sh Script Explanation (Line-by-Line)

```bash
#!/bin/bash
```

- Again, runs the script using Bash shell.

```bash
# === Filenames ===
RECEPTOR="MUT_Protein.pdbqt"
LIGAND="MUT_DNA.pdbqt"
CONFIG="config.txt"
OUTPUT="docked_output.pdbqt"
LOG="docking_log.txt"
```

- Defines filenames:
    - Here, MUT (Mutant) protein and DNA files are used.

```bash
# === Step 1: Auto-calculate docking box center from receptor ===
echo "[*] Calculating docking box center from $RECEPTOR..."
```

- Prints that docking box center calculation for mutant is starting.

```bash
CENTER=$(grep "^ATOM" MUT_Protein.pdb | awk '{x+=$6; y+=$7; z+=$8; n++} END {print x/n,
y/n, z/n}')
```

- Same as Wild script, calculates average X, Y, Z coordinates to find docking box center.

```bash
CX=$(echo $CENTER | cut -d' ' -f1)
CY=$(echo $CENTER | cut -d' ' -f2)
CZ=$(echo $CENTER | cut -d' ' -f3)
```

- Extracts center X, Y, Z coordinates separately.

```bash
# === Step 2: Create config file ===
echo "[*] Creating config file..."
```

- Prints a message that config file is being created.

```bash
cat > $CONFIG <<EOF
(receptor, ligand, center_x, center_y, center_z, size, etc.)
EOF
```

- Creates `config.txt` file for mutant docking.

```bash
# === Step 3: Run docking ===
echo "[*] Running AutoDock Vina..."
vina --config "$CONFIG" --out "$OUTPUT" --log "$LOG"
```

- Runs AutoDock Vina with the created configuration.

```bash
# === Step 4: Convert output to PDB (if Open Babel is available) ===
echo "[*] Converting docked PDBQT to PDB..."
obabel "$OUTPUT" -O docked_result.pdb > /dev/null
```

- Converts the docking result from `.pdbqt` to `.pdb` using Open Babel.

```bash
echo "[✓] Docking complete!"
echo "[📁] Docked structure: docked_result.pdb"
echo "[📜] Docking log: $LOG"
```

- Prints that docking is successfully complete.
- Shows where the docked PDB file and log are saved.

Step 7: Analyzing Binding Affinity

What I Did:

I made a Bash script called `Analyze_TF_DNA_Binding.sh` to calculate and compare the binding affinities between wild-type and mutant complexes.
I ran the script inside this path:

```
swift
CopyEdit
/home/zero/RFX1/Molecular_Docking
```

Command to Create Script:

```
bash

nano Analyze_TF_DNA_Binding.sh
```

Full Script:

```
bash

#!/bin/bash

# ============ CONFIG ============
WT_DIR="./Wild_Docking"
MUT_DIR="./MUT_Docking"
OUT_DIR="./Binding_Affinity"
mkdir -p "$OUT_DIR"

DEPENDENCIES=(pymol pdb2pqr apbs grep awk bc)

# ============ CHECK DEPENDENCIES ============
echo "[*] Checking required dependencies..."
missing_tools=()

for tool in "${DEPENDENCIES[@]}"; do
    if ! command -v "$tool" &> /dev/null; then
        echo "[!] Missing: $tool"
        missing_tools+=("$tool")
    else
        echo "[✓] $tool found"
    fi
done

# Attempt to auto-install common packages
if [ ${#missing_tools[@]} -ne 0 ]; then
    echo "[*] Attempting to install missing dependencies..."
    if command -v apt &> /dev/null; then
        PKG_MANAGER="sudo apt"
    elif command -v yum &> /dev/null; then
        PKG_MANAGER="sudo yum"
    else
        echo "[!] Could not detect package manager (apt or yum). Please install
manually."
        printf '%s\n' "${missing_tools[@]}"
        exit 1
    fi

    for tool in "${missing_tools[@]}"; do
        case "$tool" in
            pymol) echo "[*] Installing PyMOL..."; $PKG_MANAGER install -y pymol ||
echo "[!] PyMOL install failed." ;;
            pdb2pqr) echo "[*] Installing pdb2pqr..."; $PKG_MANAGER install -y pdb2pqr
|| echo "[!] pdb2pqr install failed." ;;
            apbs) echo "[*] Installing APBS..."; $PKG_MANAGER install -y apbs || echo
"[!] APBS install failed." ;;
            *) echo "[*] Installing $tool..."; $PKG_MANAGER install -y "$tool" || echo
"[!] $tool install failed." ;;
        esac
    done
fi

# ============ START ANALYSIS ============

echo "[*] Comparing binding affinities..."
grep "^REMARK VINA RESULT:" "$WT_DIR/docked_result.pdb" | awk '{print $4}' >
"$OUT_DIR/wt_affinity.txt"
grep "^REMARK VINA RESULT:" "$MUT_DIR/docked_result.pdb" | awk '{print $4}' >
"$OUT_DIR/mut_affinity.txt"

WT_BA=$(head -n 1 "$OUT_DIR/wt_affinity.txt")
MUT_BA=$(head -n 1 "$OUT_DIR/mut_affinity.txt")
echo "Wild-Type Binding Affinity: $WT_BA kcal/mol" >
"$OUT_DIR/Binding_Affinity_Comparison.txt"
echo "Mutant Binding Affinity: $MUT_BA kcal/mol" >>
"$OUT_DIR/Binding_Affinity_Comparison.txt"
echo "Difference: $(echo "$WT_BA - $MUT_BA" | bc) kcal/mol" >>
"$OUT_DIR/Binding_Affinity_Comparison.txt"

echo "[*] Generating PyMOL analysis script..."
cat << EOF > analyze_interactions.pml
load $WT_DIR/docked_result.pdb, WT_Complex
load $MUT_DIR/docked_result.pdb, MUT_Complex

select WT_HBonds, (WT_Complex and name N*) within 3.5 of (WT_Complex and name O*)
select MUT_HBonds, (MUT_Complex and name N*) within 3.5 of (MUT_Complex and name O*)
distance WT_HBonds_Dist, (WT_Complex and name N*), (WT_Complex and name O*), 3.5
distance MUT_HBonds_Dist, (MUT_Complex and name N*), (MUT_Complex and name O*), 3.5

select WT_Clashes, (WT_Complex and name CA) within 2.0 of (WT_Complex and name CA)
select MUT_Clashes, (MUT_Complex and name CA) within 2.0 of (MUT_Complex and name CA)
distance WT_Clash_Dist, (WT_Complex and name CA), (WT_Complex and name CA), 2.0
distance MUT_Clash_Dist, (MUT_Complex and name CA), (MUT_Complex and name CA), 2.0

save $OUT_DIR/WildType_vs_Mutant.pse
quit
EOF

echo "[*] Running PyMOL..."
pymol -cq analyze_interactions.pml

echo "[*] Running APBS electrostatics..."
pdb2pqr --ff=parse "$WT_DIR/docked_result.pdb" "$OUT_DIR/wt.pqr"
cat << EOF > "$OUT_DIR/wt.in"
(read and solve electrostatic potential for wild-type)
EOF
apbs "$OUT_DIR/wt.in" && cp "$OUT_DIR/wt.dx" "$OUT_DIR/WT_Electrostatics.dx"

pdb2pqr --ff=parse "$MUT_DIR/docked_result.pdb" "$OUT_DIR/mut.pqr"
cat << EOF > "$OUT_DIR/mut.in"
(read and solve electrostatic potential for mutant)
EOF
```

```
apbs "$OUT_DIR/mut.in" && cp "$OUT_DIR/mut.dx" "$OUT_DIR/MUT_Electrostatics.dx"

echo "✅ All results saved in $OUT_DIR/"
```

Running the Script:

```
bash

chmod +x Analyze_TF_DNA_Binding.sh
./Analyze_TF_DNA_Binding.sh
```

Analyzing Binding Affinity — Script Explanation (Line-by-Line)

```
bash

#!/bin/bash
```

- I declared that this file is a Bash script.

```
bash

# ============ CONFIG ============
WT_DIR="./Wild_Docking"
MUT_DIR="./MUT_Docking"
OUT_DIR="./Binding_Affinity"
mkdir -p "$OUT_DIR"
```

- I set three directory paths:
  - `WT_DIR` for Wild-Type docking folder,
  - `MUT_DIR` for Mutant docking folder,
  - `OUT_DIR` where output will be saved (Binding_Affinity folder).
- `mkdir -p` creates the Binding_Affinity folder if it doesn't already exist.

```
bash

DEPENDENCIES=(pymol pdb2pqr apbs grep awk bc)
```

- I made a list of all software tools needed for this analysis:
  - PyMOL, pdb2pqr, APBS, grep, awk, bc.

```
bash

# ============ CHECK DEPENDENCIES ============
echo "[*] Checking required dependencies..."
```

```
missing_tools=()
```

- I printed that dependency checking is starting.
- I initialized an empty list `missing_tools` to store any missing programs.

```
bash

for tool in "${DEPENDENCIES[@]}"; do
    if ! command -v "$tool" &> /dev/null; then
        echo "[!] Missing: $tool"
        missing_tools+=("$tool")
    else
        echo "[✓] $tool found"
    fi
done
```

- I looped through each tool and checked if it is installed:
  - If missing, I printed "[!] Missing" and added it to `missing_tools`.
  - If found, I printed "[√] found".

```
bash

if [ ${#missing_tools[@]} -ne 0 ]; then
```

- If the number of missing tools is **not zero**, then:

```
bash
CopyEdit
    echo "[*] Attempting to install missing dependencies..."
```

- I printed a message that I am trying to install missing tools.

```
bash

    if command -v apt &> /dev/null; then
        PKG_MANAGER="sudo apt"
    elif command -v yum &> /dev/null; then
        PKG_MANAGER="sudo yum"
    else
        echo "[!] Could not detect package manager (apt or yum). Please install
manually."
        printf '%s\n' "${missing_tools[@]}"
        exit 1
    fi
```

- I detected whether my system uses `apt` or `yum` for package management.
- If neither found, I printed an error and exited.

```
bash

    for tool in "${missing_tools[@]}"; do
```

- I started installing each missing tool.

---

bash

```
        case "$tool" in
            pymol) echo "[*] Installing PyMOL..."; $PKG_MANAGER install -y pymol ||
echo "[!] PyMOL install failed." ;;
            pdb2pqr) echo "[*] Installing pdb2pqr..."; $PKG_MANAGER install -y pdb2pqr
|| echo "[!] pdb2pqr install failed." ;;
            apbs) echo "[*] Installing APBS..."; $PKG_MANAGER install -y apbs || echo
"[!] APBS install failed." ;;
            *) echo "[*] Installing $tool..."; $PKG_MANAGER install -y "$tool" || echo
"[!] $tool install failed." ;;
        esac
    done
fi
```

- For each missing tool, I tried to auto-install it using my system's package manager.
- Special installation handling for PyMOL, pdb2pqr, and APBS.

---

bash

```
# ============ START ANALYSIS ============
echo "[*] Comparing binding affinities..."
```

- I printed that binding affinity comparison is starting.

---

bash

```
grep "^REMARK VINA RESULT:" "$WT_DIR/docked_result.pdb" | awk '{print $4}' >
"$OUT_DIR/wt_affinity.txt"
grep "^REMARK VINA RESULT:" "$MUT_DIR/docked_result.pdb" | awk '{print $4}' >
"$OUT_DIR/mut_affinity.txt"
```

- I searched (`grep`) for binding energy in Wild and Mutant docked files.
- Extracted (`awk`) the 4th column (binding affinity value).
- Saved Wild value to `wt_affinity.txt`, Mutant value to `mut_affinity.txt`.

---

bash

```
WT_BA=$(head -n 1 "$OUT_DIR/wt_affinity.txt")
MUT_BA=$(head -n 1 "$OUT_DIR/mut_affinity.txt")
```

- I loaded the first binding affinity value from both Wild and Mutant files.

---

bash

```
echo "Wild-Type Binding Affinity: $WT_BA kcal/mol" >
"$OUT_DIR/Binding_Affinity_Comparison.txt"
echo "Mutant Binding Affinity: $MUT_BA kcal/mol" >>
"$OUT_DIR/Binding_Affinity_Comparison.txt"
echo "Difference: $(echo "$WT_BA - $MUT_BA" | bc) kcal/mol" >>
"$OUT_DIR/Binding_Affinity_Comparison.txt"
```

- I wrote the binding affinities and their difference into `Binding_Affinity_Comparison.txt`.

```
bash

echo "[*] Generating PyMOL analysis script..."
```

- I printed that PyMOL analysis script generation is starting.

```
bash
CopyEdit
cat << EOF > analyze_interactions.pml
(load both structures, find hydrogen bonds, clashes, and save session)
EOF
```

- I created a PyMOL `.pml` script:
  - Loaded docked structures,
  - Selected hydrogen bonds and clashes,
  - Measured distances,
  - Saved session.

```
bash

echo "[*] Running PyMOL..."
pymol -cq analyze_interactions.pml
```

- I ran PyMOL in quiet mode to execute my `.pml` script.

```
bash

echo "[*] Running APBS electrostatics..."
```

- I printed that I am starting APBS electrostatic calculations.

```
bash

pdb2pqr --ff=parse "$WT_DIR/docked_result.pdb" "$OUT_DIR/wt.pqr"
```

- I converted Wild docked PDB to PQR format needed for APBS.

```
bash

cat << EOF > "$OUT_DIR/wt.in"
(APBS input file for Wild)
EOF
```

- I wrote APBS input file for Wild.

```bash
apbs "$OUT_DIR/wt.in" && cp "$OUT_DIR/wt.dx" "$OUT_DIR/WT_Electrostatics.dx"
```

- I ran APBS for Wild, and saved the electrostatic potential.

```bash
pdb2pqr --ff=parse "$MUT_DIR/docked_result.pdb" "$OUT_DIR/mut.pqr"
```

- I converted Mutant docked PDB to PQR format.

```bash
cat << EOF > "$OUT_DIR/mut.in"
(APBS input file for Mutant)
EOF
```

- I wrote APBS input file for Mutant.

```bash
apbs "$OUT_DIR/mut.in" && cp "$OUT_DIR/mut.dx" "$OUT_DIR/MUT_Electrostatics.dx"
```

- I ran APBS for Mutant, and saved the electrostatic potential.

```bash
echo "✅ All results saved in $OUT_DIR/"
```

- I printed a success message that everything was saved.

Step 8: Finding Steric Clashes

What I Did:

I created a Python script inside:

```path
/home/zero/RFX1/Molecular_Docking/Steric_Clashes
```

to identify **steric clashes** between transcription factor and DNA.

Command to Create Script:

```bash
nano steric_clash_analysis.py
```

Full Script:

python

```
from pymol import cmd
import os
import traceback

# Set paths
input_dir = "/home/zero/RFX1/Molecular_Docking/Input"
pdb_wt = os.path.join(input_dir, "TF_structure.pdb")
pdb_mut = os.path.join(input_dir, "TF_structure_mutations.pdb")
log_file = "clash_report.txt"

# Command logger
def log_cmd(step, func, *args, **kwargs):
    with open(log_file, "a") as f:
        f.write(f"\n--- Step {step}: {func.__name__}({', '.join(map(str, args))}) ---\
n")
        try:
            result = func(*args, **kwargs)
            if result is not None:
                f.write(f"Result: {result}\n")
        except Exception as e:
            f.write(f"Error: {e}\n")
            f.write(traceback.format_exc())
    return

# Start fresh
open(log_file, "w").write("Steric Clash Analysis Log\n")

# Run commands with logging
log_cmd(1, cmd.load, pdb_wt, "wt")
log_cmd(2, cmd.load, pdb_mut, "mut")
log_cmd(3, cmd.align, "mut", "wt")
log_cmd(4, cmd.select, "wt_TF", "wt and chain P")
log_cmd(5, cmd.select, "wt_DNA", "wt and chain D")
log_cmd(6, cmd.select, "mut_TF", "mut and chain P")
log_cmd(7, cmd.select, "mut_DNA", "mut and chain D")
log_cmd(8, cmd.select, "clashes", "(mut_TF within 3.0 of mut_DNA)")
log_cmd(9, cmd.distance, "close_contacts", "mut_TF", "mut_DNA", 3.0)
log_cmd(10, cmd.label, "clashes", "resn + resi + name")
log_cmd(11, cmd.color, "red", "clashes")
log_cmd(12, cmd.color, "yellow", "close_contacts")

# Distance reporter
def report_distances(sel1='mut_TF', sel2='mut_DNA', cutoff=3.0):
    atoms1 = cmd.index(sel1)
    atoms2 = cmd.index(sel2)

    with open(log_file, "a") as f:
        f.write("\n--- Steric Clash Distance Report ---\n")
        for (model1, index1) in atoms1:
            for (model2, index2) in atoms2:
                sel1_str = f"(model {model1} and index {index1})"
                sel2_str = f"(model {model2} and index {index2})"
                try:
                    dist = cmd.get_distance(sel1_str, sel2_str)
                    if dist <= cutoff:
                        a1 = cmd.get_model(sel1_str).atom[0]
                        a2 = cmd.get_model(sel2_str).atom[0]
                        line = (f"{a1.resn}{a1.resi}/{a1.name} ({model1}) --> {a2.resn}
{a2.resi}/{a2.name} ({model2}) = {dist:.2f} Å\n")
                        f.write(line)
                except Exception as e:
                    f.write(f"Error comparing {sel1_str} and {sel2_str}: {e}\n")

# Log distance report
report_distances()

# Exit PyMOL
cmd.quit()
```

Running the Script:

```
bash

pymol -cq steric_clash_analysis.py
```

Finding Steric Clashes — Script Explanation (Line-by-Line)

```
python

from pymol import cmd
import os
import traceback
```

- I imported PyMOL commands, OS operations, and error tracking.

```
python

input_dir = "/home/zero/RFX1/Molecular_Docking/Input"
pdb_wt = os.path.join(input_dir, "TF_structure.pdb")
pdb_mut = os.path.join(input_dir, "TF_structure_mutations.pdb")
log_file = "clash_report.txt"
```

- I defined input folder, paths for Wild and Mutant PDB files, and a log file.

```
python

def log_cmd(step, func, *args, **kwargs):
    with open(log_file, "a") as f:
        f.write(f"\n--- Step {step}: {func.__name__}({', '.join(map(str, args))}) ---\
n")
        try:
            result = func(*args, **kwargs)
            if result is not None:
                f.write(f"Result: {result}\n")
        except Exception as e:
            f.write(f"Error: {e}\n")
            f.write(traceback.format_exc())
    return
```

- I created a function that:

    - Runs a PyMOL command,
    - Logs the command and any errors into `clash_report.txt`.

```
python

open(log_file, "w").write("Steric Clash Analysis Log\n")
```

- I started fresh by clearing and writing a new log file.

```
python

log_cmd(1, cmd.load, pdb_wt, "wt")
log_cmd(2, cmd.load, pdb_mut, "mut")
log_cmd(3, cmd.align, "mut", "wt")
```

- I loaded Wild and Mutant structures into PyMOL,
- I aligned Mutant structure to Wild structure.

```
python

log_cmd(4, cmd.select, "wt_TF", "wt and chain P")
log_cmd(5, cmd.select, "wt_DNA", "wt and chain D")
log_cmd(6, cmd.select, "mut_TF", "mut and chain P")
log_cmd(7, cmd.select, "mut_DNA", "mut and chain D")
```

- I selected transcription factor (chain P) and DNA (chain D) separately for both Wild and Mutant.

```
python

log_cmd(8, cmd.select, "clashes", "(mut_TF within 3.0 of mut_DNA)")
log_cmd(9, cmd.distance, "close_contacts", "mut_TF", "mut_DNA", 3.0)
```

- I selected atoms where TF and DNA are **closer than 3 Å**,
- I measured distances between these close contacts.

```
python

log_cmd(10, cmd.label, "clashes", "resn + resi + name")
log_cmd(11, cmd.color, "red", "clashes")
log_cmd(12, cmd.color, "yellow", "close_contacts")
```

- I labeled the clashing atoms with residue name and number,
- I colored clashes red,
- I colored close contacts yellow.

```
python

def report_distances(sel1='mut_TF', sel2='mut_DNA', cutoff=3.0):
```

- I created a function to report distances between Mutant TF and DNA atoms.

Inside the function:

- I found atoms from mutant TF and DNA,
- For each atom pair, I calculated their distance,
- If distance was less than cutoff (3.0 Å), I logged the atom names and distance into `clash_report.txt`.

```python
report_distances()
```

- I ran the distance reporting function.

```python
cmd.quit()
```

- I quit PyMOL after finishing the analysis.

Step 9: Finding Electrostatic Repulsion

What I Did:

To find **electrostatic repulsion**, I analyzed the electrostatic surface potentials of the **mutant** and **wild-type** structures separately.

First, I went inside this path:

```path
/home/zero/RFX1/Molecular_Docking/Electrostatic_Repulsion
```

Inside, I created two folders: `MUT` and `WT`.

In each folder, I made a Bash script called `Electrostatic_Repulsion.sh`, and I ran the scripts separately to perform the analysis.

➡️ MUT Folder

Commands:

```bash
mkdir MUT
cd MUT
```

```
nano Electrostatic_Repulsion.sh
```

Full Script inside MUT:

```bash
#!/bin/bash

# Set filenames
PQR_FILE="TF_structure_mutations.pqr"
POTENTIAL_FILE="${PQR_FILE}.dx"
CLEAN_PDB="TF_structure_cleaned.pdb"
TXT_OUTPUT="potential_values.txt"
SUMMARY="electrostatics_summary.txt"
APBS_INPUT="apbs_input.in"

# Step 1: Run pdb2pqr
echo "[Step 1] Running pdb2pqr..."
pdb2pqr --ff AMBER --pdb-output "$CLEAN_PDB"
/home/zero/RFX1/Molecular_Docking/Input/TF_structure_mutations.pdb "$PQR_FILE"

# Step 2: Create APBS input file
echo "[Step 2] Generating APBS input file..."
cat <<EOF > "$APBS_INPUT"
(read section)
EOF

# Step 3: Run APBS
echo "[Step 3] Running APBS..."
apbs "$APBS_INPUT"

# Step 4: Extract potential values
echo "[Step 4] Extracting potential values..."
grep -v '^#\|^object\|^gridpositions\|^grid' "$POTENTIAL_FILE" > "$TXT_OUTPUT"

# Step 5: Analyze potential values
echo "[Step 5] Analyzing potential values..."
{ analysis block } > "$SUMMARY"

# Step 6: Generate histogram if gnuplot is installed
(if gnuplot available, generate histogram)
```

Then, I ran the script:

```bash
chmod +x Electrostatic_Repulsion.sh
./Electrostatic_Repulsion.sh
```

Full Script inside WT:

WT Folder

First, I came out of the MUT folder:

bash

```
cd ..
```

Then, I made a new directory for WT:

bash

```
mkdir WT
cd WT
```

Then, inside WT folder, I created a new script:

bash

```
nano Electrostatic_Repulsion.sh
```

Full Script inside WT:

bash

```
#!/bin/bash

# Set filenames
PQR_FILE="TF_structure.pqr"
POTENTIAL_FILE="${PQR_FILE}.dx"
CLEAN_PDB="TF_structure_cleaned.pdb"
TXT_OUTPUT="potential_values.txt"
SUMMARY="electrostatics_summary.txt"
APBS_INPUT="apbs_input.in"

# Step 1: Run pdb2pqr
echo "[Step 1] Running pdb2pqr..."
pdb2pqr --ff AMBER --pdb-output "$CLEAN_PDB"
/home/zero/RFX1/Molecular_Docking/Input/TF_structure.pdb "$PQR_FILE"

# Step 2: Create APBS input file
echo "[Step 2] Generating APBS input file..."
cat <<EOF > "$APBS_INPUT"
read
  mol pqr $PQR_FILE
end

elec
  mg-auto
  dime 129 129 129
  cglen 80.0 80.0 80.0
  fglen 40.0 40.0 40.0
  cgcent mol 1
  fgcent mol 1
  mol 1
  lpbe
  bcfl sdh
  ion charge 1.0 conc 0.150 radius 2.0
  ion charge -1.0 conc 0.150 radius 1.8
  pdie 2.0
  sdie 78.0
  srfm smol
  chgm spl2
  sdens 10.0
  srad 1.4
  swin 0.3
  temp 298.15
  calcenergy total
  calcforce no
  write pot dx $PQR_FILE
end

quit
EOF

# Step 3: Run APBS
echo "[Step 3] Running APBS..."
apbs "$APBS_INPUT"

# Step 4: Extract potential values
echo "[Step 4] Extracting potential values..."
grep -v '^#\|^object\|^gridpositions\|^grid' "$POTENTIAL_FILE" > "$TXT_OUTPUT"

# Step 5: Analyze potential values
echo "[Step 5] Analyzing potential values..."
{
echo "Analyzing potential values..."
echo "-----------------------------------"
echo "Min / Max / Mean electrostatic potentials:"
awk '{ sum+=$1; count+=1; if(min==""){min=max=$1}; if($1>max){max=$1}; if($1<min)
{min=$1} } END { print "Min:", min, "\nMax:", max, "\nMean:", sum/count }'
"$TXT_OUTPUT"

echo -e "\nHighly positive regions (> +10.0):"
awk '$1 > 10.0' "$TXT_OUTPUT" | wc -l

echo -e "\nHighly negative regions (< -10.0):"
awk '$1 < -10.0' "$TXT_OUTPUT" | wc -l
} > "$SUMMARY"

# Step 6: Generate histogram if gnuplot is installed
if command -v gnuplot &> /dev/null; then
  echo "[Step 6] Generating histogram..."
  HISTO_DATA="potential_histogram.dat"
  awk '{bin=int($1); count[bin]++} END {for (b in count) print b, count[b]}'
"$TXT_OUTPUT" | sort -n > "$HISTO_DATA"
  gnuplot -persist <<-EOF
    set terminal png size 800,600
    set output 'potential_histogram.png'
    set title "Electrostatic Potential Distribution"
    set xlabel "Potential (kT/e)"
    set ylabel "Frequency"
    plot "$HISTO_DATA" using 1:2 with boxes notitle
EOF
  echo -e "\nHistogram saved to potential_histogram.png" >> "$SUMMARY"
else
  echo "[Step 6] Gnuplot not found. Skipping histogram generation." >> "$SUMMARY"
fi

# Final log
echo -e "\n[Done] Electrostatic analysis complete. Summary written to $SUMMARY."
```

Then, I made the script executable and ran it:

```
bash

chmod +x Electrostatic_Repulsion.sh
./Electrostatic_Repulsion.sh
```

Then again, I ran the script:

```bash
chmod +x Electrostatic_Repulsion.sh
./Electrostatic_Repulsion.sh
```

Now, Line-by-Line Explanation of Electrostatic_Repulsion.sh (for both MUT and WT)

---

1. Start of the Script:

```bash
#!/bin/bash
```

- I told Linux to run this file using Bash shell.

---

2. Define Important Filenames:

```bash
PQR_FILE="TF_structure_mutations.pqr"   # (In WT script, it's TF_structure.pqr)
POTENTIAL_FILE="${PQR_FILE}.dx"
CLEAN_PDB="TF_structure_cleaned.pdb"
TXT_OUTPUT="potential_values.txt"
SUMMARY="electrostatics_summary.txt"
APBS_INPUT="apbs_input.in"
```

- I defined names for different output files:
  - `.pqr` file: to store the molecule with charges and radii,
  - `.dx` file: electrostatic potential grid,
  - cleaned PDB file,
  - text output file,
  - APBS input script,
  - analysis summary file.

---

3. Step 1: Run `pdb2pqr`

```bash
echo "[Step 1] Running pdb2pqr..."
pdb2pqr --ff AMBER --pdb-output "$CLEAN_PDB" /path/to/original.pdb "$PQR_FILE"
```

- I ran `pdb2pqr`:
  - Converted the PDB structure into PQR format,

    - Applied **AMBER** force field parameters,
    - Saved cleaned and charged versions of my structures.

---

4. Step 2: Create APBS Input File

```
bash

echo "[Step 2] Generating APBS input file..."
cat <<EOF > "$APBS_INPUT"
(read section with molecule and electrostatic settings)
EOF
```

- I made an **input file** for APBS software.
- This file tells APBS:
    - How large the grid should be,
    - How to calculate electrostatics,
    - What parameters to use (dielectric constants, ions, etc.).

---

5. Step 3: Run APBS

```
bash

echo "[Step 3] Running APBS..."
apbs "$APBS_INPUT"
```

- I ran **APBS** to compute the electrostatic potential across the molecular surface.
- The output was saved as a `.dx` file.

---

6. Step 4: Extract Useful Potential Values

```
bash

echo "[Step 4] Extracting potential values..."
grep -v '^#\|^object\|^gridpositions\|^grid' "$POTENTIAL_FILE" > "$TXT_OUTPUT"
```

- I removed all comment lines and grid information from the `.dx` file,
- I extracted only the **electrostatic potential values** into a simple text file.

---

7. Step 5: Analyze Potential Values

```
bash

echo "[Step 5] Analyzing potential values..."
{ awk block inside }
```

Inside this block:

- I calculated:

    - **Minimum potential**,
    - **Maximum potential**,
    - **Mean (average) potential**,
- I also counted:
    - How many points have highly positive values (> +10.0),
    - How many points have highly negative values (< -10.0).

All results were saved into `electrostatics_summary.txt`.

---

8. Step 6: Optional Histogram Generation

bash

```
if command -v gnuplot &> /dev/null; then
  (generate histogram plot)
else
  (skip histogram if gnuplot missing)
fi
```

- I checked if `gnuplot` is installed.
- If yes, I made a **histogram plot** showing the distribution of electrostatic potentials.
- If no, I skipped plotting.

---

9. Final Message:

bash

```
echo -e "\n[Done] Electrostatic analysis complete. Summary written to $SUMMARY."
```

- I printed that the analysis was complete and where to find the summary.

Comparing WT vs MUT Electrostatic Repulsion

What I Did:

After I analyzed the Electrostatic Repulsion separately for **MUT** and **WT**,
I wanted to **compare** the results between Wild-Type (WT) and Mutant (MUT).

So, I stayed inside this path:

path

```
/home/zero/RFX1/Molecular_Docking/Electrostatic_Repulsion
```

There, I made two comparison scripts:
➔ First: `compare_WT_vs_MUT.sh` (to calculate differences and summary)
➔ Second: `Plot_WT_vs_MUT.sh` (to plot histogram)

First Script: compare_WT_vs_MUT.sh

Command to Create:

```
bash

nano compare_WT_vs_MUT.sh
```

Full Script:

```
bash

#!/bin/bash

# Define directories
WT_DIR="WT"
MUT_DIR="MUT"

# Define files
WT_POT="$WT_DIR/potential_values.txt"
MUT_POT="$MUT_DIR/potential_values.txt"
DIFF_FILE="WT_vs_MUT_difference.txt"
COMPARISON_SUMMARY="WT_vs_MUT_summary.txt"
COMPARISON_PLOT="WT_vs_MUT_potential_comparison.png"

# Step 1: Print summary files
echo "[Step 1] WT electrostatic summary:"
cat "$WT_DIR/electrostatics_summary.txt"
echo -e "\n[Step 1] MUT electrostatic summary:"
cat "$MUT_DIR/electrostatics_summary.txt"

# Step 2: Create a histogram comparison plot
echo -e "\n[Step 2] Generating comparative histogram plot..."
gnuplot <<EOF
set terminal png size 1000,600
set output "$COMPARISON_PLOT"
set title "WT vs MUT Electrostatic Potential Distribution"
set xlabel "Potential (kT/e)"
set ylabel "Frequency"
set style fill solid 0.4 border -1
set boxwidth 0.9
plot \
  "$WT_DIR/potential_histogram.dat" using 1:2 with boxes title "WT", \
  "$MUT_DIR/potential_histogram.dat" using 1:2 with boxes title "MUT" linecolor rgb
"red"
EOF
echo "Histogram saved to $COMPARISON_PLOT"

# Step 3: Point-by-point potential difference
echo -e "\n[Step 3] Calculating potential differences..."
paste "$WT_POT" "$MUT_POT" | awk '{print $2 - $1}' > "$DIFF_FILE"

# Step 4: Statistical summary of differences
echo -e "\n[Step 4] Analyzing difference statistics..."
awk '{
  sum+=$1; count+=1;
  if(min==""){min=max=$1};
  if($1>max){max=$1};
  if($1<min){min=$1}
} END {
  print "Difference in Electrostatic Potentials (MUT - WT):" > "'$COMPARISON_SUMMARY'"
  print "--------------------------------------------------" >>
"'$COMPARISON_SUMMARY'"
  print "Min Difference:", min >> "'$COMPARISON_SUMMARY'"
  print "Max Difference:", max >> "'$COMPARISON_SUMMARY'"
  print "Mean Difference:", sum/count >> "'$COMPARISON_SUMMARY'"
}' "$DIFF_FILE"

echo "Summary saved to $COMPARISON_SUMMARY"

# Final note
echo -e "\n[Done] Comparison complete. See:"
echo " - $COMPARISON_SUMMARY for summary"
echo " - $COMPARISON_PLOT for histogram comparison"
```

Then, I ran the script:

```bash
chmod +x compare_WT_vs_MUT.sh
./compare_WT_vs_MUT.sh
```

Explaining `compare_WT_vs_MUT.sh` Line-by-Line

```bash
#!/bin/bash
```

- I declared this file is a Bash script.

```bash
# Define directories
WT_DIR="WT"
MUT_DIR="MUT"
```

- I set directory names:
    - `WT_DIR` points to the Wild-Type results folder,
    - `MUT_DIR` points to the Mutant results folder.

```bash
# Define files
WT_POT="$WT_DIR/potential_values.txt"
MUT_POT="$MUT_DIR/potential_values.txt"
DIFF_FILE="WT_vs_MUT_difference.txt"
```

```
COMPARISON_SUMMARY="WT_vs_MUT_summary.txt"
COMPARISON_PLOT="WT_vs_MUT_potential_comparison.png"
```

- I defined file names:
  - `WT_POT` = Wild-Type potential values file,
  - `MUT_POT` = Mutant potential values file,
  - `DIFF_FILE` = output file where differences will be saved,
  - `COMPARISON_SUMMARY` = summary of comparison,
  - `COMPARISON_PLOT` = name of histogram image comparing WT and MUT.

```
bash

# Step 1: Print summary files
echo "[Step 1] WT electrostatic summary:"
cat "$WT_DIR/electrostatics_summary.txt"
```

- I printed the Wild-Type summary on the terminal by displaying the contents of `electrostatics_summary.txt`.

```
bash

echo -e "\n[Step 1] MUT electrostatic summary:"
cat "$MUT_DIR/electrostatics_summary.txt"
```

- I printed the Mutant summary similarly.
- `-e "\n"` adds a blank line to separate outputs nicely.

```
bash

# Step 2: Create a histogram comparison plot
echo -e "\n[Step 2] Generating comparative histogram plot..."
```

- I printed a message that I am starting histogram generation.

```
bash

gnuplot <<EOF
(set up the plotting commands here)
EOF
```

- I used Gnuplot inside my bash script directly.
- The section between `<<EOF` and `EOF` contains plotting commands for Gnuplot.

Inside Gnuplot block:

```
bash

set terminal png size 1000,600
set output "$COMPARISON_PLOT"
set title "WT vs MUT Electrostatic Potential Distribution"
set xlabel "Potential (kT/e)"
set ylabel "Frequency"
set style fill solid 0.4 border -1
set boxwidth 0.9
```

- I set:
    - Output file as PNG image,
    - Plot size,
    - Title, labels,
    - Filled boxes for better visual clarity.

```
bash

plot \
  "$WT_DIR/potential_histogram.dat" using 1:2 with boxes title "WT", \
  "$MUT_DIR/potential_histogram.dat" using 1:2 with boxes title "MUT" linecolor rgb
"red"
```

- I plotted two histograms together:
    - Wild-Type histogram in default color,
    - Mutant histogram in red color.

```
bash

echo "Histogram saved to $COMPARISON_PLOT"
```

- I printed a message that the histogram was saved successfully.

```
bash

# Step 3: Point-by-point potential difference
echo -e "\n[Step 3] Calculating potential differences..."
paste "$WT_POT" "$MUT_POT" | awk '{print $2 - $1}' > "$DIFF_FILE"
```

- I calculated differences at each point:
    - `paste` combines the two files side by side,
    - `awk` calculates (Mutant - Wild-Type) difference,
    - Saved all differences into `WT_vs_MUT_difference.txt`.

```
bash

# Step 4: Statistical summary of differences
echo -e "\n[Step 4] Analyzing difference statistics..."
```

- I printed that statistical analysis of differences is starting.

```bash
awk '{
  sum+=$1; count+=1;
  if(min==""){min=max=$1};
  if($1>max){max=$1};
  if($1<min){min=$1}
} END {
  print "Difference in Electrostatic Potentials (MUT - WT):" > "'$COMPARISON_SUMMARY'"
  print "--------------------------------------------------" >> 
"'$COMPARISON_SUMMARY'"
  print "Min Difference:", min >> "'$COMPARISON_SUMMARY'"
  print "Max Difference:", max >> "'$COMPARISON_SUMMARY'"
  print "Mean Difference:", sum/count >> "'$COMPARISON_SUMMARY'"
}' "$DIFF_FILE"
```

- I used `awk` to:
  - Find minimum, maximum, and mean values of the electrostatic differences.
  - Saved all results into the summary file `WT_vs_MUT_summary.txt`.

```bash
echo "Summary saved to $COMPARISON_SUMMARY"
```

- I printed a confirmation that the summary was successfully saved.

```bash
# Final note
echo -e "\n[Done] Comparison complete. See:"
echo " - $COMPARISON_SUMMARY for summary"
echo " - $COMPARISON_PLOT for histogram comparison"
```

- I printed the final message telling where to find the comparison results.

Second Script: Plot_WT_vs_MUT.sh

(For extra plotting if needed separately.)

Command to Create:

```bash
nano Plot_WT_vs_MUT.sh
```

Full Script:

```bash
gnuplot -persist <<EOF
set terminal png size 1000,600
set output "WT_vs_MUT_potential_comparison.png"
set title "WT vs MUT Electrostatic Potential Distribution"
set xlabel "Potential (kT/e)"
set ylabel "Frequency"
set style fill solid 0.5 border -1
plot \
  "WT/potential_histogram.dat" using 1:2 with boxes title "WT", \
  "MUT/potential_histogram.dat" using 1:2 with boxes title "MUT" linecolor rgb "red"
EOF
```

Then, I ran this script:

```bash
chmod +x Plot_WT_vs_MUT.sh
./Plot_WT_vs_MUT.sh
```

Explaining `Plot_WT_vs_MUT.sh` Line-by-Line

```bash
gnuplot -persist <<EOF
```

- I started Gnuplot plotting directly inside the Bash script.
- `-persist` keeps the window open even after plotting.

Inside Gnuplot block:

```bash
set terminal png size 1000,600
```

- I set output format to PNG image of size 1000x600 pixels.

```bash
set output "WT_vs_MUT_potential_comparison.png"
```

- I named the output file `WT_vs_MUT_potential_comparison.png`.

```bash
set title "WT vs MUT Electrostatic Potential Distribution"
set xlabel "Potential (kT/e)"
set ylabel "Frequency"
```

- I set the title of the plot and labels for X and Y axes.

```bash
set style fill solid 0.5 border -1
```

- I set bars to be solid filled with some transparency.

```bash
plot \
  "WT/potential_histogram.dat" using 1:2 with boxes title "WT", \
  "MUT/potential_histogram.dat" using 1:2 with boxes title "MUT" linecolor rgb "red"
```

- I plotted:
  - Wild-Type histogram in one color,
  - Mutant histogram in red color,
  - Each using boxes to show frequencies.

```bash
EOF
```

- End of the Gnuplot block.

Step 10: Identify Loss of Favorable Interactions

What I Did:

In this step, I identified important favorable interactions (hydrogen bonds, salt bridges, hydrophobic contacts) between the transcription factor and DNA in the **mutated structure**.
I did this by writing a PyMOL `.pml` script and analyzing the contacts automatically.

Commands I Ran:

First, I was inside this path:

```path
/home/zero/RFX1/Molecular_Docking/Identify_Loss_of_Favorable_Interactions
```

Then, I made a new script:

```bash
nano analyze_interactions.pml
```

Full Script (analyze_interactions.pml):

```
bash

load /home/zero/RFX1/Molecular_Docking/Input/TF_structure_mutations.pdb

# Define output log
python
logfile = "/home/zero/RFX1/Molecular_Docking/Identify_Loss_of_Favorable_Interactions/
interaction_analysis_log.txt"
log = open(logfile, "w")
def logprint(s):
    print(s)
    log.write(s + "\n")
python end

# Selections
select TF, chain P
select DNA, chain D

# Hydrogen Bonds
python
from pymol import cmd
logprint("\n[Hydrogen Bonds]")
count = 0
for atm1 in cmd.index("TF and name N*"):
    for atm2 in cmd.index("DNA and name O*"):
        dist = cmd.get_distance(f"index {atm1[1]}", f"index {atm2[1]}")
        if dist < 3.5:
            a1 = cmd.get_model(f"index {atm1[1]}").atom[0]
            a2 = cmd.get_model(f"index {atm2[1]}").atom[0]
            logprint(f"H-Bond {count+1}: {a1.resn}{a1.resi}-{a1.name} (chain
{a1.chain}) <--> {a2.resn}{a2.resi}-{a2.name} (chain {a2.chain}) | Distance: {dist:.2f}
Å")
            count += 1
logprint(f"Total: {count}")
python end

# Salt Bridges
python
logprint("\n[Salt Bridges]")
count = 0
for atm1 in cmd.index("TF and (resn ARG+LYS) and name NH*+NZ"):
    for atm2 in cmd.index("DNA and name OP1+OP2+O1P+O2P"):
        dist = cmd.get_distance(f"index {atm1[1]}", f"index {atm2[1]}")
        if dist < 4.0:
            a1 = cmd.get_model(f"index {atm1[1]}").atom[0]
            a2 = cmd.get_model(f"index {atm2[1]}").atom[0]
            logprint(f"Salt Bridge {count+1}: {a1.resn}{a1.resi}-{a1.name} (chain
{a1.chain}) <--> {a2.resn}{a2.resi}-{a2.name} (chain {a2.chain}) | Distance: {dist:.2f}
Å")
            count += 1
logprint(f"Total: {count}")
python end

# Hydrophobic Contacts
python
logprint("\n[Hydrophobic Contacts]")
count = 0
for atm1 in cmd.index("TF and (resn ALA+VAL+LEU+ILE+MET+PHE+TRP+PRO)"):
    for atm2 in cmd.index("DNA and (resn DA+DT+DG+DC)"):
        dist = cmd.get_distance(f"index {atm1[1]}", f"index {atm2[1]}")
        if dist < 4.0:
            a1 = cmd.get_model(f"index {atm1[1]}").atom[0]
            a2 = cmd.get_model(f"index {atm2[1]}").atom[0]
            logprint(f"Hydrophobic Contact {count+1}: {a1.resn}{a1.resi}-{a1.name}
(chain {a1.chain}) <--> {a2.resn}{a2.resi}-{a2.name} (chain {a2.chain}) | Distance:
{dist:.2f} Å")
            count += 1
logprint(f"Total: {count}")
python end

# Save final structure with selections
python
import os
out_dir = "/home/zero/RFX1/Molecular_Docking/Identify_Loss_of_Favorable_Interactions"
if not os.path.exists(out_dir):
    os.makedirs(out_dir)
cmd.save(out_dir + "/TF_structure_with_interactions.pdb")
log.close()
python end

quit
```

Then I Ran This Command:

```bash
pymol -cq analyze_interactions.pml
```

- `-c` means run without opening the PyMOL GUI,
- `-q` means quiet mode (no unnecessary outputs).

Line-by-Line Explanation of `analyze_interactions.pml`

```bash
load /home/zero/RFX1/Molecular_Docking/Input/TF_structure_mutations.pdb
```

- I loaded the mutated transcription factor structure into PyMOL.

```bash
# Define output log
python
logfile = "/path/to/interaction_analysis_log.txt"
log = open(logfile, "w")
def logprint(s):
    print(s)
    log.write(s + "\n")
python end
```

- I started Python block.
- I created a log file to save all analysis results.
- I defined a `logprint()` function that prints and saves messages at the same time.

```
bash

# Selections
select TF, chain P
select DNA, chain D
```

- I selected:
    - TF = transcription factor chain P,
    - DNA = DNA chain D.

Hydrogen Bonds Section

```
bash

python
from pymol import cmd
logprint("\n[Hydrogen Bonds]")
count = 0
```

- Started analyzing Hydrogen Bonds.
- Initialized `count` to zero.

```
bash

for atm1 in cmd.index("TF and name N*"):
    for atm2 in cmd.index("DNA and name O*"):
        dist = cmd.get_distance(...)
```

- I looped through:
    - All Nitrogen atoms (`N*`) of TF,
    - All Oxygen atoms (`O*`) of DNA,
- Calculated distances between every pair.

```
bash

if dist < 3.5:
```

- If the distance between TF and DNA atoms is less than **3.5 Å**, it is considered a hydrogen bond.

```
bash

logprint(f"H-Bond {count+1}: {info}")
count += 1
```

- I printed the details of each hydrogen bond into the log file,
- Incremented the bond count.

```bash
logprint(f"Total: {count}")
python end
```

- Printed the total number of hydrogen bonds.

Salt Bridges Section

Same structure, but different selections:

```bash
for atm1 in cmd.index("TF and (resn ARG+LYS) and name NH*+NZ"):
    for atm2 in cmd.index("DNA and name OP1+OP2+O1P+O2P"):
        dist = cmd.get_distance(...)
```

- Selected ARG/LYS residues from TF,
- Selected phosphate oxygen atoms from DNA,
- Distance cutoff is **4.0 Å** (for salt bridges).
- Printed all salt bridges and their count.

Hydrophobic Contacts Section

Same again but for hydrophobic residues:

```bash
for atm1 in cmd.index("TF and (resn ALA+VAL+LEU+ILE+MET+PHE+TRP+PRO)"):
    for atm2 in cmd.index("DNA and (resn DA+DT+DG+DC)"):
        dist = cmd.get_distance(...)
```

- Selected hydrophobic amino acids from TF,
- Selected DNA base residues,
- Distance cutoff was **4.0 Å**.
- Printed all hydrophobic contacts and their count.

Save Output Structure

```bash
import os
out_dir = "/path/to/output"
if not os.path.exists(out_dir):
    os.makedirs(out_dir)
cmd.save(out_dir + "/TF_structure_with_interactions.pdb")
log.close()
python end
```

- I created the output directory if it did not exist,

- Saved the final structure showing all interactions into PDB format,
- Closed the log file properly.

```
bash

quit
```

- I exited PyMOL after everything was done.

Conclusion

In this comprehensive study, we analyzed the structural and functional consequences of a specific mutation within the transcription factor RFX1 and its interaction with DNA. By employing a series of structural bioinformatics and molecular docking techniques—including electrostatic surface potential comparison, visualization of key residue changes, and quantification of favorable molecular interactions—we systematically compared the wild-type and mutant forms of the protein-DNA complex.

Our analysis revealed distinct differences in electrostatic potential distribution, particularly in the DNA-binding interface. The histogram comparison and statistical summaries indicated a measurable shift in electrostatic behavior due to the mutation. Furthermore, interaction profiling identified a loss of favorable hydrogen bonds, salt bridges, and hydrophobic contacts, underscoring the mutation's potential impact on binding affinity and stability.

These findings suggest that the introduced mutation may compromise the transcription factor's ability to stably and specifically interact with its DNA target, possibly altering gene regulation. This integrated pipeline can serve as a valuable template for future mutation impact assessments in DNA-binding proteins and other biomolecular systems.