Project Report: Molecular Energy Calculator

Course: High Performance Computing and its Applications in Complex Systems
Project: Molecular Energy Calculator using Force Fields
Student Name: Arunangshu Karmakar
Roll Number: 23MA10014
Date: November 8, 2025

Deployment

Live Application: https://hpc-mol.streamlit.app

The project has been successfully deployed as a production-ready web application on Streamlit Cloud, providing public access to the molecular energy calculator with full GUI interface and interactive 3D visualization capabilities.

Project Overview

This project implements a comprehensive molecular energy calculator in Python that computes potential energy for any given molecule using classical force field parameters. The system supports multiple input formats (XYZ, PDB, MOL), provides an interactive YAML force field builder with automatic parameter generation, and includes both naive and HPC-optimized energy calculation methods with significant performance improvements.

---

1. Individual Contributions

The following components of this project were developed solely by me (Arunangshu Karmakar, Roll No. 23MA10014).

A. Streamlit GUI Framework

Designed and implemented the complete graphical user interface using Streamlit, a Python framework for building interactive web applications. The GUI provides an intuitive interface for molecular energy calculations with the following features:

- **Three Main Tabs**: Organized the application into "Calculate Energy", "YAML Builder", and "Documentation" tabs for clear workflow separation
- **File Upload System**: Integrated drag-and-drop file upload supporting XYZ, PDB, and MOL formats with automatic format detection
- **Responsive Layout**: Created a clean layout with sidebar controls for cutoff distance, optimization options, and visualization settings
- **Error Handling**: Added comprehensive error messages and user feedback for invalid files, missing parameters, and calculation errors
- **Real-time Updates**: The interface updates dynamically as users modify parameters, providing instant visual and numerical feedback

B. 3D Molecule Visualizer

Integrated interactive 3D visualization capabilities using the py3Dmol library (Python wrapper for 3Dmol.js) to provide real-time molecular structure visualization:

- **Interactive 3D Rendering**: Users can rotate, zoom, and pan the molecule using mouse controls for detailed structural examination
- **Multiple Visualization Styles**: Implemented stick, sphere, and ball-and-stick rendering modes to suit different visualization preferences
- **CPK Color Scheme**: Applied standard Corey-Pauling-Koltun coloring where carbon is gray, oxygen is red, nitrogen is blue, and hydrogen is white for intuitive element recognition

C. Multi-Format File Support

Implemented comprehensive support for three major molecular file formats, enabling the calculator to work with molecules from various sources:

- **PDB Format Parser**: Developed a parser for Protein Data Bank files that reads ATOM and HETATM records using PDB's fixed-column format specification. The parser extracts element symbols, coordinates, and connectivity information from CONECT records, making it compatible with protein structures and complex biomolecules.
- **MOL Format Parser**: Created a parser for MDL Molfile (MOL/SDF) V2000 format, which includes explicit bond tables. The parser reads the counts line to determine the number of atoms and bonds, then processes the atom block (coordinates and elements) and bond block (connectivity and bond orders), handling the 1-based indexing convention used in MOL files.
- **RDKit Integration**: For all formats, the system constructs RDKit molecule objects to enable SMARTS pattern matching, which is essential for automatic atom typing in the YAML builder.

D. Interactive YAML Force Field Builder

Developed a user-friendly system for creating force field YAML files with automatic parameter generation, significantly simplifying the force field definition process:

YAML Format and Structure: The force field is defined in YAML (YAML Ain't Markup Language) format with four main sections:

1. **atom_types**: Defines chemical environments using SMARTS patterns, assigns atomic charges, and specifies Lennard-Jones parameters (σ and ε)
2. **bond_types**: Lists harmonic bond parameters with force constants (kb) and equilibrium lengths (b0)
3. **angle_types**: Contains angle bending parameters with force constants (kθ) and equilibrium angles (θ0)
4. **dihedral_types**: Specifies torsional parameters using OPLS Fourier series with four coefficients (V1, V2, V3, V4)

Energy Equation Components: Each YAML section corresponds to a term in the total potential energy equation:

$$E_{\text{total}} = E_{\text{bonds}} + E_{\text{angles}} + E_{\text{dihedrals}} + E_{\text{LJ}} + E_{\text{Coulomb}}$$

- **Bond Stretching**: $E_{\text{bonds}} = \sum \frac{1}{2} k_b (b - b_0)^2$ where kb and b0 come from bond_types
- **Angle Bending**: $E_{\text{angles}} = \sum \frac{1}{2} k_\theta (\theta - \theta_0)^2$ where kθ and θ0 come from angle_types
- **Dihedral Torsion**: $E_{\text{dihedrals}} = \sum \left[\frac{V_1}{2}(1 + \cos\phi) + \frac{V_2}{2}(1 - \cos 2\phi) + \frac{V_3}{2}(1 + \cos 3\phi) + \frac{V_4}{2}(1 - \cos 4\phi) \right]$ where V1-V4 come from dihedral_types
- **Lennard-Jones**: $E_{\text{LJ}} = \sum_{i<j} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right]$ where σ and ε come from atom_types with combining rules: $\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2}$ and $\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j}$
- **Coulombic**: $E_{\text{Coulomb}} = \sum_{i<j} f \frac{q_i q_j}{r_{ij}}$ where $f = 138.935485$ kJ·nm/mol and charges (q) come from atom_types

Automatic Features:

- **SMARTS Pattern Generation**: The builder uses RDKit to automatically generate SMARTS (SMiles ARbitrary Target Specification) patterns that describe each atom's chemical environment, including element type, connectivity, and neighboring atoms
- **Atom Type Grouping**: Chemically equivalent atoms (like the three hydrogens in a methyl group) are automatically grouped into a single atom type to reduce redundancy

- **Type Name Consistency**: Bond, angle, and dihedral type names are auto-generated from atom type names to ensure perfect consistency throughout the force field
- **Real-time YAML Preview**: Users can see the generated YAML file in real-time and export it for use in calculations

E. YAML Verification System

Implemented a comprehensive validation system that checks force field completeness before energy calculation:

- **Coverage Analysis**: Calculates four coverage percentages - atom coverage (percentage of atoms successfully typed by SMARTS patterns), bond coverage (percentage of bonds with defined parameters), angle coverage, and dihedral coverage
- **SMARTS Pattern Matching**: Validates that all SMARTS patterns correctly match their intended atoms by attempting to match each pattern against the RDKit molecule representation
- **Missing Parameter Detection**: Identifies and reports specific missing parameters (e.g., "C_type_1-O_type_3 bond missing") to guide users in completing the force field

F. Performance Comparison Framework

Developed a side-by-side performance comparison system demonstrating the effectiveness of HPC optimizations:

- **Naive $O(N^2)$ Implementation**: Straightforward double-loop approach that checks all atom pairs for non-bonded interactions, serving as the baseline for performance comparison
- **Spatial Tree Optimization**: Tree-based scipy.spatial.cKDTree (k-dimensional tree) for efficient neighbor searching, reducing complexity from $O(N^2)$ to $O(N \log N)$ by only checking atom pairs within the cutoff distance
- **Multi-core Parallelization**: Implemented Python multiprocessing to distribute pair calculations across available CPU cores, providing additional speedup beyond the algorithmic improvement
- **Real-time Timing**: Both methods are timed during execution, and the speedup factor is calculated and displayed to demonstrate the performance improvement
- **Accuracy Verification**: The system verifies that optimized methods produce identical energy values to the naive approach (within numerical precision), ensuring correctness

Performance Results: For a system with 4,500 atoms, the optimized approach achieves approximately $15\times$ speedup compared to the naive method, and with multi-core parallelization on 4 cores, the total speedup can reach over $30\times$.

G. System Integration and Deployment

Integrated all components into a cohesive application and deployed it for public access:

- **Module Integration**: Connected the energy calculator backend calculator.py with the Streamlit frontend app.py, ensuring smooth data flow between file loading, force field application, and energy calculation
- **Error Propagation**: Implemented proper exception handling throughout the pipeline so that errors in any component (file parsing, SMARTS matching, energy calculation) are caught and displayed to users with helpful messages
- **Documentation Creation**: Wrote comprehensive documentation including README.md for quick start and DOCUMENTATION.md covering complete mathematical theory, implementation details, and performance analysis
- **GitHub Repository**: Organized the code into a clean repository structure with proper version control
- **Streamlit Cloud Deployment**: Successfully deployed the application to Streamlit Cloud at https://hpc-mol.streamlit.app, making it accessible to anyone with a web browser without requiring local installation

---

End of Report