Requirements for developing your web-based application that runs the **DeepBDE** model, predicts the bond dissociation enthalpies (BDEs) for all single bonds, and visualizes them on the molecular structure using a **SMILES** input.

1. Functional Requirements

a. User Input

- A text field or form where users can input a **SMILES string**.
- Nice to have: File upload for batch processing (CSV, TXT).
- Nice to have: File upload Cartesian coordinates for molecule description in addition to SMILES.

Example of a SMILES input: CN1CCC2Nc3ccccc3C2C1

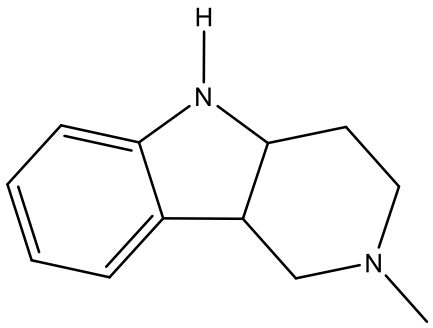
b. Prediction

Backend executes the **DeepBDE** model (already trained, see: https://github.com/MSRG/DeepBDE.git) on the input molecule. DeepBDE uses SMILES as input and has an option for printing out the BDEs for all single bonds.

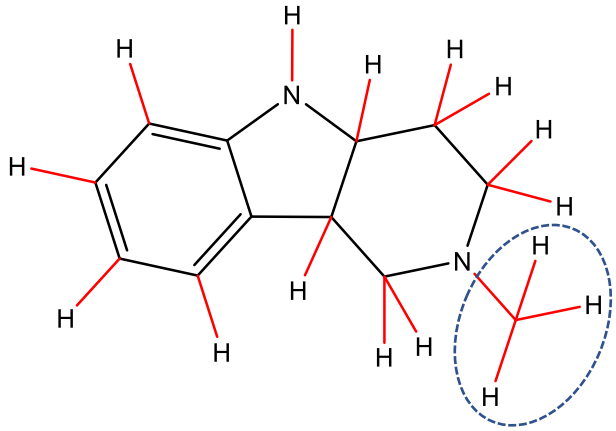
c. Visualization

- Generate a 2D molecular structure with labeled single bonds showing the predicted BDE values to within one decimal place. The units associated with the predicted BDEs is kcal/mol.

Example of the 2D molecular structure corresponding to SMILES code CN1CCC2Nc3ccccc3C2C1:



Note that “standard” 2D molecular representations do not show most of the hydrogen atoms on a molecule. For example, the 2D molecule represented above with all of its atoms crudely looks like this:

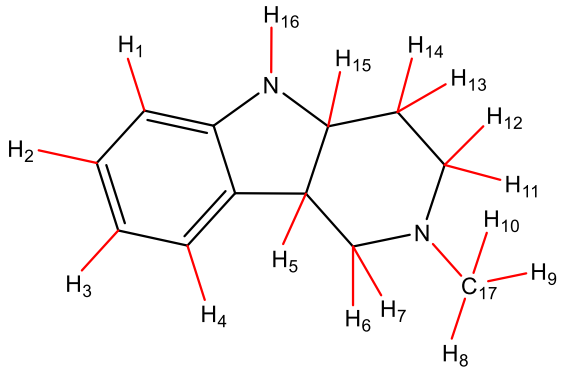


DeepBDE will predict the BDEs for all of the bonds shown in red – that is, chemical bonds that – when broken – will result in two separate fragments. Some of the BDEs predicted by DeepBDE will be identical (for example, the three BDEs for the bonds shown in the dashed circle).

Omar Hernández Montes recently created the program *draw_molecules_ohm.ipynb* that may have the required functionality. The program is included as a separate file.

d. Output

- Display:
  - 2D molecule image with hydrogen atoms shown, annotated with numerical labels for the bonds. The labels can be added to the atoms involved in the bond-breaking. For example:



  - Table of bond indices and corresponding BDEs. For example:

The DeepBDE-predicted BDEs for the bonds associated with the labeled atoms, in kcal/mol:

Atom	BDE

H_1	113.4
H_2	113.7
⋮	⋮

- Print out reference to the journal paper describing DeepBDE.
- Option to download the image and/or the table.

e. Error Handling

- Invalid SMILES input → display clear error messages.
- Model or visualization failure → graceful fallback with logging.

---

2. Suggested Frontend Requirements

a. Tech Stack

- HTML/CSS/JavaScript
- Frameworks: React.js or Vue.js (optional but recommended for SPA)
- Visualization:
  - **RDKit.js** (browser-side cheminformatics)
  - Or **render image from backend** using RDKit Python and serve it as static content.

b. UI Components

- SMILES input box
- Submit button
- Loading spinner
- Molecule image + overlayed BDEs
- Error/message display
- Results table (optional)

---

3. Suggested Backend Requirements

a. Tech Stack

- Python-based backend using **Flask**, **FastAPI**, or **Django**.
- Integration with:
  - **DeepBDE model** (TensorFlow/PyTorch)

- **RDKit** (for molecule parsing and visualization)

b. Core Functions

- Parse SMILES → Molecule object (via RDKit).
- Identify all **single bonds**.
- Extract necessary descriptors/features for DeepBDE.
- Run **DeepBDE model** on all single bonds.
- Annotate the molecule image with BDE predictions.

c. API Design

- /predict_bde endpoint:
  - Accepts SMILES
  - Returns BDEs + image URI or raw image
- Optionally /batch_predict_bde for bulk input.

---

4. Possible Dependencies & Libraries

a. Python Libraries

- rdkit – molecular parsing and image generation
- matplotlib or PIL – annotate and render images
- numpy, pandas – data handling
- torch or tensorflow – depending on how DeepBDE is implemented
- flask, fastapi, or django – API framework

b. Optional Frontend Libraries

- axios or fetch – API calls
- d3.js or plotly – advanced visualization (if needed)

---

5. Deployment Requirements

- Hosting:
  - Cloud options like **AWS**, **Heroku**, **Azure**, or **Google Cloud**.
  - Or deploy on university/on-prem server.
- API Gateway or Load Balancer (for scaling).
- HTTPS and user authentication (if needed).

6. Security & Privacy

- Input validation and sanitization to prevent code injection.
- Rate limiting to avoid misuse.
- Track usage by number of SMILES submissions

---

7. Performance & Optimization

- Caching of frequent predictions (e.g., Redis).
- Use of GPU inference if model is large and real-time performance is required.
- Optimize RDKit image generation to avoid latency.
