Predicting Molecule Aqueous Properties with xTB Calculations

Elyse Taglia, Nicholas Louie, Nada Shetewi, Donald Tang

ChE352 – Process Simulations

Prof. James Stevenson

Abstract

In this work, we present a method for calculating hydration and formation energies of molecules using the GFN2-xTB method. The code takes a SMILES representation of a molecule, generates its conformers, calculates their energies in both gas and aqueous phases, and computes the hydration and formation energies. Furthermore, the code calculates the RDKFingerprint of the molecule, which can be used for similarity searches. The method was tested on ethane, acetone, caffeine, and tryptophan. Formation and hydration energies predictions for ethane and acetone were highly accurate with a 0.0046% error for ethane and 1.0% error for acetone. The method consistently predicts accurate formation energies for large polar molecules as well, with 0.93% and 0.65% error for caffeine and tryptophan, respectively. Predicted hydration energies for caffeine and tryptophan had large errors of 47% and 29% error, respectively. The results show the robustness of the method in predicting formation energies and its limitations in predicting hydration energies for large polar and organic molecules.

Introduction

Chemical companies invest significant resources in developing new compounds that can potentially have a wide range of applications, from pharmaceuticals to advanced materials. However, the process of synthesizing new compounds can be lengthy and costly, especially when the synthesized compounds turn out to be unstable or poorly soluble in water. In such cases, researchers must go back to the drawing board and repeat their experiments, leading to further delays and costs.

To help chemical companies optimize their R&D process, our team of chemical engineering consultants designed a Python function called "smiles_to_properties()," which predicts the properties of organic molecules that have never been synthesized before. The function takes a SMILES string, a string that encodes the structure of a molecule, as input and returns four important properties of the molecule: (1) the aqueous phase structure with the lowest energy, (2) the aqueous phase energy of formation, (3) the hydration energy with standard deviation, and (4) the RDKFingerprint molecule fingerprint.

The aqueous-phase structure with the lowest potential energy is calculated using a quantum-mechanical method, GFN2-xTB. The method uses a solvation model of water to efficiently and accurately compute the lowest energy conformations for a given SMILES string. The structure and energy of an aqueous-phase molecule is an important consideration when manufacturing novel substances because they affect molecular stability in water. Substances with high potential energies generally exhibit less stability compared to those with low potential energies. The stability of a molecule in water can have a significant impact on its effectiveness in applications such as drug delivery.

The aqueous-phase energy of formation refers to the amount of energy absorbed or emitted when a mole of the compound of interest is formed by its pure elements in aqueous conditions. The function “smiles_to_properties()" computes the aqueous-phase formation energy by subtracting the enthalpy change of formation of the product and that of the reactants. This property provides insight into the stability and feasibility of synthesizing the molecule for scale-up.

The hydration energy is the amount of heat released when one mole of ion dissolves in water. The chemical reaction takes the general form of the equation shown below. M^{z+} (g) represents the ion when not surrounded by water molecules while M^{z+} (aq) represents the ion when surrounded by water molecules. The function “smiles_to_properties()" computes the hydration energy by subtracting the energy of the ion from its final and initial state and using the solvent.h2o solvation model provided by GFN2-xTB. This property is useful for characterizing the stability of a molecule because it reveals how soluble a molecule is in water. High enthalpies indicate more energy is required to break the molecule apart and vice-versa.

$$M_{(g)}^{z+} + mH_2O \rightarrow M_{(aq)}^{z+}$$

Figure 1: Hydration Mechanism (*Hydration - Chemistry LibreTexts*, n.d.)

The RDKFingerprint molecule fingerprint is a hashed bit-based representation of every atom in a molecule's structure. It works by labelling each atom based on various characteristics, such as their acidic/basic behaviors, atomic numbers, bond types, etc. The fingerprint is a unique representation of the molecule's structure that can be used for similarity searches and other applications.

The GFN2-xTB (Geometry, Frequency, and Noncovalent interactions with 2nd-order Tight-Binding Approximation) method is an approximate quantum-chemical method designed for efficient calculations of molecules and materials. It is part of the xTB family of methods, which are known for their computational efficiency and reasonable accuracy for a wide range of molecular systems. The GFN2-xTB method has been parameterized and tested on various datasets, including thermochemistry, geometries, and noncovalent interactions. It has been shown to yield results with reasonable accuracy, especially considering its computational efficiency. However, the accuracy of the GFN2-xTB method varies depending on the property being calculated and the specific chemical system being investigated (Bannwarth et al., 2019).

Although our function builds upon existing tools in the field of chemical informatics, such as the RDKit and the GFN2-xTB solvation model, it provides a more comprehensive set of properties that can help chemical companies better evaluate the feasibility of synthesizing new compounds. By using our function, researchers can avoid spending months of work on synthesizing compounds that turn out to be unstable or poorly soluble in water, leading to faster and more cost-effective research and development.

Methods

The "smiles_to_properties()" function for this project utilizes existing tools in the field of chemical informatics, such as the RDKit and the GFN2-xTB solvation model, to estimate important properties of a given organic molecule.

The function first converts the input SMILES string to a molecule object and adds any implicit hydrogen atoms using the "Chem.AddHs()" command. The function then identifies each element in the molecule using “GetSymbol” and “GetAtoms” commands then stores the elements in element_list array. The reference aqueous energies of each element in the list are calculated in a for loop using a separate “get_energy()” function with solvent set to water. The aqueous and gaseous energies of the given molecule are also calculated by “get_energy()” function twice to get an average and standard deviation. Hydration energy is calculated by subtracting the molecule’s gaseous energy from its aqueous energy. The formation energy of the molecule is calculated by subtracting the reference aqueous energies of each atom from the total aqueous energy. Reference energies are given for diatomic molecules, ex: H_2 and Cl_2 so they are divided by two before being subtracted to get the formation energy.

All energies are calculated via a separate “get_energy()” function to ensure that geometry optimizations of the given molecule are consistent for each calculation. The function takes in a given SMILES string and solvent. The SMILES string is again converted to a molecule with all hydrogen atoms. The “AllChem.EmbedMolecule()” command embeds the 3D coordinates of the molecule based on the ETDKG method, which needs more optimization to get a more accurate geometry. An initial position guess was given using MMFF optimization. The geometry was then optimized by minimizing the energy of the molecule, represented by a third separate function “energy_optim()”. After energy optimizations, the lowest energy is returned as a float and the lowest energy conformer is returned as the “position” NumPy array with optimal positions of each element.

The “energy_optim()” function takes in an array of atomic positions of each atom in the element list, forming the molecule, and calculates the single point energy of the molecule using GFN2-xTB method. The function can be set to calculate aqueous or gaseous energies by changing the solvent parameter. “energy_optim()” is then minimized to get optimum molecular geometry at the lowest energy. The "smiles_to_properties()" function also generates an RDKit fingerprint for the molecule, which is a unique representation of the molecule's structure that can be used for similarity searches and other applications.

Throughout the code, comments are used to explain the purpose and functionality of each block of code. The function also includes print statements to display the calculated properties of the molecule in a clear and informative manner.

Results

The properties of several molecules (ethane, acetone, and caffeine) were predicted with the "smiles_to_properties()" function. Their most stable aqueous-phase structure, aqueous-phase formation energy, hydration energy, and RDKFingerprint molecule fingerprint were then compared with literature values and outputs from a local consultant to determine how accurate the models were.

Example 1: Ethane – CC

Inputting the SMILES string for ethane gave the following output.

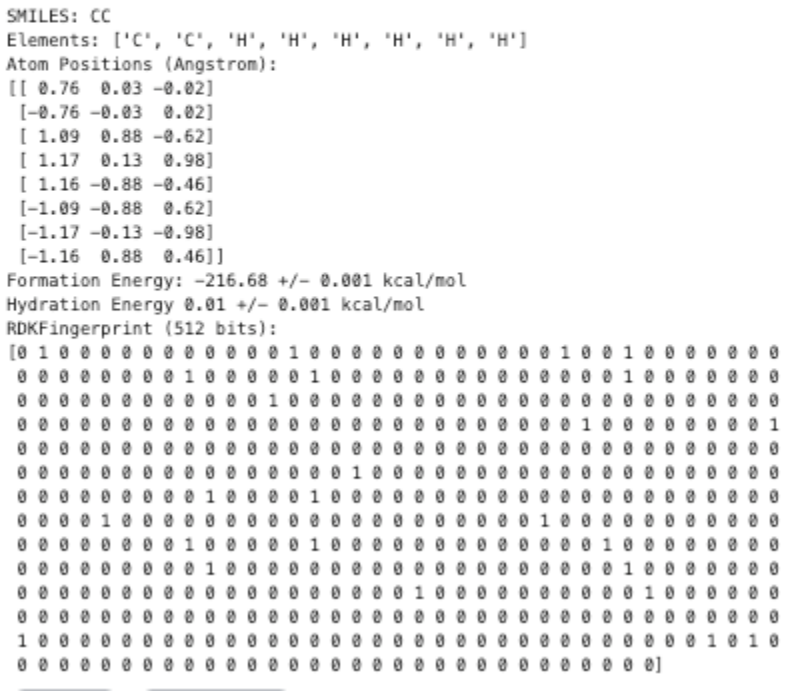


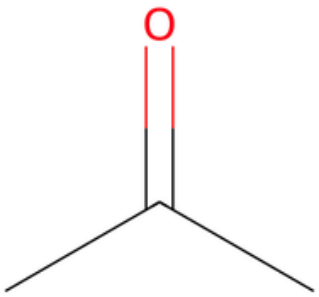
Figure 2: Predicted Properties for SMILES string 'CC' (Ethane)

The RDKFingerprint of ethane and 3D coordinates of each atom in the aqueous phase structure of ethane are shown above. The average aqueous phase energy of formation was -216.68 +/- 0.001 kcal/mol, and the hydration energy with standard deviation was 0.01 +/- 0.001 kcal/mol. These outputs matched predictions made from a model made by another consultant (See the Supplemental section for more information). The percent difference between the predictions of the aqueous formation energies was

0.0046%. Additionally, the aqueous phase energy of formation was close to the difference between the gas phase and reference energies: -216.69 kcal/mol. The percentage difference between these two values was also 0.0046%, suggesting that the aqueous formation energy is reasonable.

Example 2: Acetone - CC(=O)C

Inputting the SMILES string for acetone yielded the following output.




```
SMILES: CC(=O)C
Elements: ['C', 'C', 'O', 'C', 'H', 'H', 'H', 'H', 'H', 'H']
Atom Positions (Angstrom):
[[ 1.22 -0.31 -0.21]
 [ 0.01  0.58 -0.29]
 [ 0.03  1.66 -0.84]
 [-1.23  0.03  0.36]
 [ 2.07  0.16 -0.7 ]
 [ 1.   -1.26 -0.69]
 [ 1.46 -0.5   0.83]
 [-2.05  0.72  0.26]
 [-1.49 -0.92 -0.11]
 [-1.03 -0.16  1.41]]
Formation Energy: -360.56 +/- 0.0 kcal/mol
Hydration Energy -3.65 +/- 0.0 kcal/mol
RDKFingerprint (512 bits):
[0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
```

Figure 3: Predicted Properties for SMILES string 'CC(=O)C' (Acetone)

The RDKFingerprint of acetone and 3D coordinates of each atom in the aqueous phase structure of ethane are shown above. The model predicted an average aqueous phase formation energy of –360.56 +/- 0.001 kcal/mol and hydration energy with a standard deviation of –3.65 +/- 0.001 kcal/mol. The hydration energy fell within a reasonable range of simulation results from other quantum-mechanical models in literature. The following table from the Organic & Biomolecular Chemistry (Tu et al., 2017) journal shows six models that predicted the hydration energy of acetone to range between –3.30 to 4.02

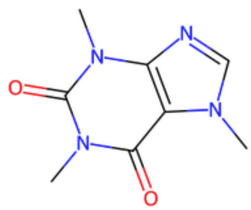
kcal/mol with an accuracy of 1 to 2 kcal/mol for CBS-QB3, CBS-APNO, and G4. Additionally, the aqueous formation energy was close to the difference between the gas phase and reference energies: -356.91 kcal/mol. The percentage difference between these two values was 1.0%, implying that values are reasonable.

	Implicit		Inclusion of explicit water	
	ΔH (kcal mol^{-1})	ΔG (kcal mol^{-1})	ΔH (kcal mol^{-1})	ΔG (kcal mol^{-1})
B3LYP/6-31+G(d,p)	2.73	15.67	5.13	9.42
B3LYP/CBSB7	−0.36	13.06		
B3LYP/CBSB7+	4.02	16.82		
CBS-QB3	−3.30	10.10	−1.58	1.56
CBS-APNO	−2.75	9.94	−1.10	2.70
G4	−3.07	9.10	−0.86	3.78
Experimental				3.68 (ref. 63)

Figure 4: Other Models' Predictions of Hydration Energy of Acetone (Tu et al., 2017)

Example 3: Caffeine - CN1C=NC2=C1C(=O)N(C(=O)N2C)C

Inputting the SMILES string for caffeine gave the following output.




```
SMILES: CN1C=NC2=C1C(=O)N(C(=O)N2C)C
Elements: ['C', 'N', 'C', 'N', 'C', 'C', 'C', 'O', 'N', 'C', 'O', 'N', 'C', 'C', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
Atom Positions (Angstrom):
[[ 3.31  0.32 -0.43]
 [ 2.12 -0.47 -0.31]
 [ 2.01 -1.8  -0.31]
 [ 0.77 -2.24 -0.18]
 [ 0.03 -1.11 -0.09]
 [ 0.84  0.02 -0.17]
 [ 0.31  1.34 -0.1 ]
 [ 0.94  2.39 -0.15]
 [-1.08  1.32  0.05]
 [-1.91  0.21  0.12]
 [-3.11  0.35  0.25]
 [-1.32 -1.02  0.05]
 [-2.12 -2.23  0.12]
 [-1.76  2.61  0.13]
 [ 4.17 -0.34 -0.52]
 [ 3.25  0.97 -1.3 ]
 [ 3.43  0.95  0.46]
 [ 2.85 -2.46 -0.4 ]
 [-1.8  -2.83  0.98]
 [-1.99 -2.81 -0.79]
 [-3.16 -1.93  0.24]
 [-2.29  2.68  1.08]
 [-2.48  2.7  -0.68]
 [-1.01  3.39  0.06]]
Formation Energy: -1026.61 +/- 0.005 kcal/mol
Hydration Energy -9.48 +/- 0.005 kcal/mol
RDKFingerprint (512 bits):
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

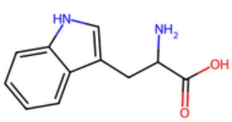
Figure 5: Predicted Properties for SMILES string 'CN1C=NC2=C1C(=O)N(C(=O)N2C)C' (Caffeine)

The RDKFingerprint of caffeine and 3D coordinates of each atom in the aqueous phase structure of caffeine are shown above. The model predicted an average aqueous phase formation energy of -1026.61 +/- 0.005 kcal/mol and hydration energy with a standard deviation of -9.48 +/- 0.005 kcal/mol. The aqueous phase energy of formation was nearly equivalent to the difference between the gas phase and reference energies: -1017.13 kcal/mol. The percent difference between these two values was 0.93%, which suggested the aqueous formation energy was reasonable. The hydration energy prediction,

however, was far from the experimental hydration energy of -20.0 kcal/mol with about 47% error (Cesaro et al., 1976). This large percentage error could be attributed to the large size of the molecule because there are more atoms and functional groups to account for. The code was likely not comprehensive enough to accurately account for the hydrophobic and intermolecular forces experienced between each atom interacting with water. For example, caffeine exhibits flat hydrophobic phases but has carbonyl groups and nitrogen atoms that exhibit hydrophilic and soluble behaviors in water due to the formation of hydrogen bonds.

Example 4: Tryptophan - C1=CC=C2C(=C1)C(=CN2)CC(C(=O)O)N

Inputting the SMILES string for caffeine gave the following output.




```
SMILES: C1=CC=C2C(=C1)C(=CN2)CC(C(=O)O)N
Elements: ['C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'N', 'C', 'C', 'C', 'O', 'O', 'N', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H']
Atom Positions (Angstrom):
[[ 2.92 -0.22 -1.1 ]
 [ 3.21 -1.56 -0.85]
 [ 2.44 -2.31  0.02]
 [ 1.37 -1.68  0.65]
 [ 1.07 -0.32  0.4 ]
 [ 1.86  0.41 -0.48]
 [-0.08 -0.    1.2 ]
 [-0.4  -1.15  1.87]
 [ 0.46 -2.16  1.54]
 [-0.77  1.31  1.26]
 [-1.57  1.61 -0.04]
 [-2.73  0.63 -0.11]
 [-3.86  0.89  0.24]
 [-2.38 -0.55 -0.6 ]
 [-2.    3.   -0.02]
 [ 3.54  0.33 -1.78]
 [ 4.04 -2.02 -1.35]
 [ 2.67 -3.35  0.21]
 [ 1.65  1.46 -0.67]
 [-1.2  -1.32  2.57]
 [ 0.43 -3.09  1.91]
 [-0.05  2.12  1.37]
 [-1.46  1.34  2.11]
 [-0.91  1.44 -0.9 ]
 [-3.12 -1.18 -0.62]
 [-2.67  3.13  0.73]
 [-2.47  3.22 -0.89]]
Formation Energy: -1167.05 +/- 1.451 kcal/mol
Hydration Energy -9.25 +/- 0.925 kcal/mol
RDKFingerprint (512 bits):
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

*Figure 5: Predicted Properties for SMILES string ‘***C1=CC=C2C(=C1)C(=CN2)CC(C(=O)O)N***’ (Tryptophan)*

The RDKFingerprint of tryptophan and 3D coordinates of each atom in the aqueous phase structure of tryptophan are shown above. The model predicted an average aqueous phase formation energy of -1167.05 +/- 1.451 kcal/mol and hydration energy with a standard deviation of -9.25 +/- 0.925

kcal/mol. The aqueous phase energy of formation was nearly equivalent to the difference between the gas phase and reference energies: -1159.53 kcal/mol. The percent difference between these two values was 0.65%, which suggested the aqueous formation energy was reasonable. The hydration energy prediction, however, was far from experimental value of –12.36 kcal/mol +/- 0.03 kcal/mol with about 29% error (Smirnov & Badelin, 2013). Like the caffeine example, the hydration energy of tryptophan exhibited substantial error in accuracy. This outcome could be attributed to the large size of tryptophan because the code noticeably struggled to accurately determine the hydration energy for other large molecules like caffeine.

Conclusion

The "smiles_to_properties()" function successfully and accurately predicted the properties of small organic and polar molecules, such as ethane and acetone. Its accuracy can be attributed to the GFN2-xTB method, which is reasonably precise for a wide range of molecular systems. In fact, it is an improved xTB method with improved energy predictions for polar and strongly hydrogen-bonded systems (Bannwarth et al., 2019). However, the function struggled to accurately predict the hydration energy of large molecules capable of forming multiple hydrogen bonds, such as caffeine and tryptophan. The sheer number of atoms to account for was likely the source of this error because the accuracy in the hydration energy predictions noticeably declined the more complex the molecular system became. Despite this shortcoming, the function still makes decent predictions for aqueous formation energies and atomic positions.

Overall, the "smiles_to_properties()" function is a comprehensive and efficient tool for estimating the properties of a wide range of organic molecules in aqueous conditions. The model can accurately calculate the formation energies of many organic molecules. However, it is limited to calculating the hydration energies of small non-polar and polar molecules. Its accuracy depends on the specific property and chemical system under study, so the function could be improved with more sophisticated quantum-mechanical methods for large and complex molecular systems. Improvements in the initial guess and optimization method may also reduce the error in computations since the hydration energy varies when the code is rerun several times for large molecules.

Supplemental Information

Link to Kaggle Notebook: https://www.kaggle.com/code/nadascode/che-352-final-project

Code Text:

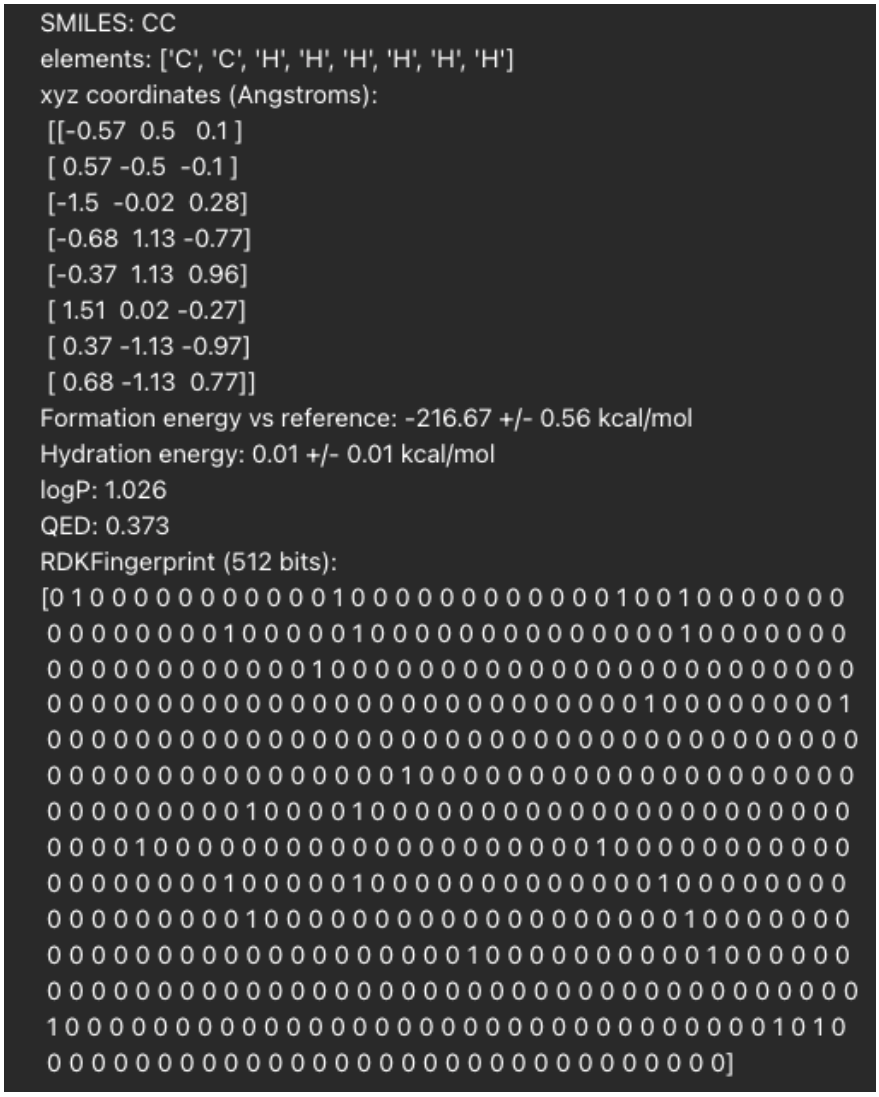
```
import os
os.environ['OMP_NUM_THREADS'] = '1'  # limit to 1 core, makes xTB more
efficient
os.chdir(r'/kaggle/working')  # to save files
from IPython.display import FileLink
import numpy as np
try:
      import rdkit
except ModuleNotFoundError:
      # install rdkit & xtb-python to this environment
      !pip install xtb rdkit
      # copy the GFN0-xTB parameter file here
      !curl -o param_gfn0-xtb.txt https://raw.githubusercontent.com/grimme-
lab/xtb/main/param_gfn0-xtb.txt
from scipy import constants
from xtb.interface import Calculator, XTBException
from xtb.libxtb import VERBOSITY_MUTED
from xtb.utils import get_solvent, Solvent
from xtb.utils import get_method
from rdkit import Chem
from rdkit.Chem import rdRGroupDecomposition
from rdkit.Chem import RDKFingerprint
from rdkit.Chem import AllChem
from rdkit.Chem import rdchem
from rdkit.Chem import Draw, QED
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem.Draw import MolDrawing, DrawingOptions
from rdkit.Chem.rdchem import RWMol
from IPython.display import display
from scipy.spatial.distance import cdist
from scipy.optimize import minimize
import sys
sys.stderr = open('error.log', 'w')
```

```
def get_energy(smile, solvent = Solvent.h2o, position_return = False):
      #=================================================================
=========================================================#
      #Initializing molecule
      mol = Chem.MolFromSmiles(smile)
      mol = Chem.AddHs(mol)
      #=================================================================
=========================================================#
      #Provides initial guess
      #Embeds molcules, performs MMFF optimization, and obtains position as
initial guess
      AllChem.EmbedMolecule(mol)
      AllChem.MMFFOptimizeMolecule(mol)
      mol_conf = mol.GetConformer()
      guess_pos = np.array([list(mol_conf.GetAtomPosition(atom.GetIdx()))
for atom in mol.GetAtoms()]) * 1.88973 #angstrom to bohr radius
      element_list =
np.array([Chem.GetPeriodicTable().GetAtomicNumber(atom.GetSymbol()) for
atom in mol.GetAtoms()])
      #=================================================================
=========================================================#
      #Optimizing section
      #If solvent argument set to None, this part can optimize for gaseous
state
      result = minimize(energy_optim, guess_pos, method = 'L-BFGS-B', args
= (element_list, solvent), jac = True)
      energy = result.fun
      #=================================================================
=========================================================#
      #Position can be sent if position_return argument set to True, but
returns in bohr radius, so needs to be converted to angstrom
      if position_return == False:
      return energy
      else:
      position = result.x
      return energy, position


def energy_optim(atom_pos, element_list, solvent, method =
get_method("GFN2-xTB")):
      calc = Calculator(method, element_list, atom_pos)
      calc.set_verbosity(VERBOSITY_MUTED)
      calc.set_solvent(solvent)
      sing_p_energy = calc.singlepoint()
      energy_gradient = (sing_p_energy.get_gradient()).flatten() * 627.509
#converts energy from hcartree to kcal/mol
      energy = sing_p_energy.get_energy() * 627.509 #converts energy from
hartree to kcal/mol
      return (energy, energy_gradient)
```

```
def smiles_to_properties(smiles: str, iterations = 5):
      #=================================================================
=====================================================#
      #Initializing molecule
      mol = Chem.MolFromSmiles(smiles)
      mol = Chem.AddHs(mol)
      element_list = [atom.GetSymbol() for atom in mol.GetAtoms()]
      #=================================================================
=====================================================#
      #Energy Calculations
      #Creates empty lists to append hydration and formation energies to
over iterations
      hydration_energies = [] # in kcal/mol
      formation_energies = [] # in kcal/mol

      for i in range(iterations):
      reference_energies = { # in kcal/mol
            'H': get_energy('[H][H]'),
            'C': get_energy('[C-]#[C+]'),
            'N': get_energy('N#N'),
            'O': get_energy('O=O'),
            'F': get_energy('FF'),
            'P': get_energy('P#P'),
            'S': get_energy('S=S'),
            'Cl': get_energy('ClCl')
      }
      aqueous_energy, position = get_energy(smiles, position_return = True)
      gaseous_energy = get_energy(smiles, solvent = None)
      hydration_energies.append(aqueous_energy-gaseous_energy)
      formation_energy = aqueous_energy

      for element in element_list:
            if element in reference_energies:
                  formation_energy -= reference_energies[element] / 2
      formation_energies.append(formation_energy)
      #=================================================================
=====================================================#
      #Optimal position calculations
      #Gets the last optimial position from previous section, converts from
bohr radius to hartree, and reformats positon to 2D array
      position /= 1.88973 #bohr radius to angstrom
      xyz = np.empty((int(len(position)/3), 3))
      for i in range(int(len(position)/3)):
      xyz[i][0] = round(position[i*3], 2)
      xyz[i][1] = round(position[i*3+1], 2)
      xyz[i][2] = round(position[i*3+2], 2)
      #=================================================================
=====================================================#
```

```
      #Obtaining averages and standard deviations
      hydration_energy_kcal_per_mol = round(np.average(hydration_energies),
2)
      hydration_stddev_kcal_per_mol = round(np.std(hydration_energies), 3)
      formation_energy_kcal_per_mol = round(np.average(formation_energies),
2)
      formation_stddev_kcal_per_mol = round(np.std(formation_energies), 3)
      #=================================================================
=========================================================#
      #Fingerprint maker
      fp = RDKFingerprint(mol, fpSize=512)
      fp_np_array = np.array(fp)
      #=================================================================
=========================================================#
      #Printing molecule without hydrogens
      mol = Chem.MolFromSmiles(smiles)
      if mol is not None:
      img = Draw.MolToImage(mol)
      display(img)  # Display the image in the notebook or a GUI window
      #=================================================================
=========================================================#
      #Printing relevant properties
      print(f"SMILES: {smiles}")
      print(f"Elements: {element_list}")
      print(f"Atom Positions (Angstrom):\n{xyz}")
      print(f"Formation Energy: {formation_energy_kcal_per_mol} +/-
{formation_stddev_kcal_per_mol} kcal/mol")
      print(f"Hydration Energy {hydration_energy_kcal_per_mol} +/-
{hydration_stddev_kcal_per_mol} kcal/mol")
      print(f"RDKFingerprint (512 bits):\n{fp_np_array}")
      return (element_list,xyz),
(formation_energy_kcal_per_mol,formation_stddev_kcal_per_mol),
(hydration_energy_kcal_per_mol, hydration_stddev_kcal_per_mol), fp_np_array

smiles = "CC"
(elements, xyz), (formation_energy, formation_stddev), (hydration_energy,
hydration_stddev), fp = smiles_to_properties(smiles)
```

Output of Properties for Ethane from a Consultant (James Stevenson):



Bibliography

Bannwarth, C., Ehlert, S., & Grimme, S. (2019). GFN2-xTB-An Accurate and Broadly Parametrized Self-Consistent Tight-Binding Quantum Chemical Method with Multipole Electrostatics and Density-Dependent Dispersion Contributions. *Journal of Chemical Theory and Computation*, *15*(3), 1652–1671. https://doi.org/10.1021/acs.jctc.8b01176

Cesaro, A., Russo, E., & Crescenzi, V. (1976). Thermodynamics of caffeine aqueous solutions. *The Journal of Physical Chemistry*, *80*(3), 335–339. https://doi.org/10.1021/j100544a026

Hydration - Chemistry LibreTexts. (n.d.). Retrieved May 2, 2023, from https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Thermodynamics/Energies_and_Potentials/Enthalpy/Hydration

Smirnov, V. I., & Badelin, V. G. (2013). Enthalpy characteristics of dissolution of L-tryptophan in water + formamides binary solvents at 298.15 K. *Russian Journal of Physical Chemistry*, *87*(7), 1165–1169. https://doi.org/10.1134/S0036024413070285

Tu, Y.-J., Njus, D., & Schlegel, H. B. (2017). A theoretical study of ascorbic acid oxidation and HOO˙/O2˙- radical scavenging. *Organic & Biomolecular Chemistry*, *15*(20), 4417–4431. https://doi.org/10.1039/c7ob00791d