**Briefing 3**

**Fusion Calculation of Tetrastyrene Isomer C38H28**

Fusion calculation refers to a calculation method that combines quantum mechanical calculations and network calculations. It is 6 orders of magnitude faster than DFT. There are two key points: a database obtained by calculating a large number of molecules by density functional theory DFT; another difficulty is The input of the network must be a vector. It is easy to input an image, but it is more difficult to input a molecule. ANI establishes **an AEV vector as input**.

Fusion calculation can not solve the Schrodinger equation to get the eigenfunction and eigenvalue. But you can get the potential energy surface, force constant, vibration frequency. Then you can get the geometric configuration from its minimum point, and get the reaction energy from its potential well depth. The saddle point obtains the transition state, infrared spectrum, Raman spectrum, etc.

The database used in this calculation is the ani-2x library: molecules containing H, C, N, O,, S, F, Cl. It is a subset of GDB-11 and contains 57,951 molecules. The energy is single weight Neutral molecule calculation (single point energy) in spin state.

From molecular design to the study of molecular properties and interactions is the basic strategy of molecular engineering to prepare new functional materials. The cis-trans isomers of the C=C double bond play an important role in life and materials science, such as cis other Moxifen is a weaker estrogen antagonist, while trans tamoxifen is an effective estrogen antagonist for the treatment of breast cancer. Therefore, designing cis-trans isomers based on the C=C double bond and studying the structure-property-function relationship is of great significance to molecular engineering. Bifunctionalized tetrastyrene (TPE) and its derivatives are very suitable for the study of C=C double bond cis-trans isomers.

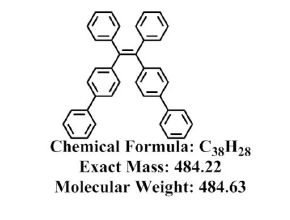
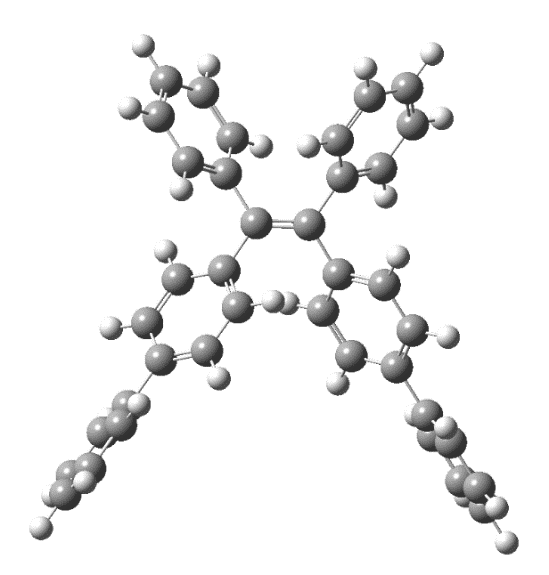


Fig.1 Tetrastyrene isomer C38H28

**Fusion calculation is very suitable for material calculation. This briefing is just an introduction. I hope that teachers will be willing to expand the database so that it can be applied to general material calculations, especially perovskite-structured materials.**

**1. Run some results**

**Energy**: -1464.518209780911 (Hartree)

**Frequencies** (cm^-1): tensor([ 0.015, 15.896, 20.417, 32.366, 39.293, 42.648,

59.370, 59.837, 81.612, 93.450, 99.585, 109.199,

110.881, 124.295, 134.382, 146.154, 184.978, 198.908,

207.666, 238.378, 242.384, 258.611, 269.406, 288.415,

313.006, 336.132, 369.182, 369.389, 382.031, 390.745,

394.841, 402.456, 402.543, 406.872, 408.222, 423.646,

449.972, 478.348, 497.338, 506.762, 507.125, 552.792,

553.724, 596.683, 607.957, 616.072, 639.403, 642.651,

646.940, 652.039, 656.866, 657.896, 662.717, 666.804,

682.279, 683.434, 686.399, 696.620, 710.044, 714.006,

759.892, 780.558, 801.217, 801.274, 803.662, 806.742,

819.008, 847.859, 848.588, 856.221, 857.186, 861.840,

863.058, 863.145, 869.103, 886.614, 917.025, 917.697,

921.521, 922.249, 939.224, 940.533, 943.765, 948.059,

955.580, 955.882, 956.569, 957.031, 970.911, 971.115,

974.592, 974.664, 997.129, 1024.651, 1033.625, 1039.073,

1056.074, 1066.317, 1066.448, 1070.589, 1070.873, 1083.776,

1083.778, 1115.768, 1115.944, 1137.161, 1140.192, 1146.206,

1146.256, 1160.103, 1181.003, 1186.381, 1199.180, 1200.645,

1202.336, 1202.396, 1206.654, 1227.587, 1233.722, 1244.672,

1247.487, 1247.993, 1263.664, 1264.108, 1277.214, 1278.477,

1295.736, 1302.497, 1304.239, 1305.194, 1325.134, 1332.576,

1344.496, 1353.600, 1364.390, 1371.659, 1390.664, 1390.860,

1392.765, 1409.950, 1492.869, 1493.054, 1504.769, 1504.977,

1522.522, 1524.459, 1530.966, 1532.230, 1546.758, 1547.872,

1548.930, 1549.366, 1662.184, 1663.154, 1681.983, 1684.485,

1701.056, 1702.428, 1719.696, 1719.837, 1721.593, 1722.132,

1755.065, 1755.103, 2949.583, 2951.996, 2970.814, 2972.019,

2981.183, 2983.780, 3037.124, 3037.161, 3196.800, 3196.862,

3202.386, 3202.459, 3205.713, 3205.745, 3206.235, 3206.542,

3236.958, 3236.958, 3245.087, 3245.113, 3249.646, 3249.647,

3254.771, 3255.208, 3273.542, 3273.542, 3274.113, 3274.225],

dtype=torch.float64)

**Force Constants** (mDyne/A): tensor([ 0.000, 0.001, 0.001, 0.003, 0.004, 0.004,

0.010, 0.009, 0.019, 0.023, 0.031, 0.031,

0.031, 0.037, 0.053, 0.075, 0.116, 0.109,

0.134, 0.168, 0.172, 0.193, 0.199, 0.238,

0.261, 0.275, 0.475, 0.380, 0.429, 0.266,

0.306, 0.340, 0.289, 0.299, 0.298, 0.333,

0.583, 0.568, 0.734, 0.561, 0.562, 0.767,

0.897, 0.952, 0.936, 1.379, 1.197, 1.497,

1.301, 1.133, 1.539, 1.408, 1.318, 1.454,

1.450, 1.465, 1.604, 1.469, 1.272, 1.378,

1.756, 2.114, 0.448, 0.446, 0.460, 0.448,

2.070, 0.541, 0.534, 0.552, 0.541, 0.552,

0.567, 0.560, 0.553, 2.271, 0.674, 0.673,

0.672, 0.665, 0.717, 0.730, 0.728, 0.731,

0.732, 0.741, 0.741, 0.736, 0.761, 0.758,

0.778, 0.778, 2.719, 3.055, 3.457, 2.750,

3.902, 2.904, 2.657, 1.551, 1.648, 1.906,

1.909, 1.941, 1.889, 1.265, 1.228, 1.157,

1.157, 2.830, 1.396, 1.159, 1.254, 1.107,

1.111, 1.171, 1.453, 1.340, 1.278, 1.090,

1.040, 1.053, 2.919, 1.559, 5.069, 4.597,

4.180, 3.716, 3.589, 3.788, 4.005, 3.582,

3.448, 1.787, 1.616, 1.936, 1.496, 1.488,

1.506, 1.785, 3.490, 3.459, 3.265, 3.222,

2.849, 2.692, 3.536, 3.575, 2.680, 2.988,

3.136, 3.187, 9.356, 9.065, 8.296, 8.230,

7.654, 7.819, 10.219, 10.246, 9.189, 9.189,

10.449, 10.455, 5.582, 5.588, 5.662, 5.667,

5.714, 5.720, 5.903, 5.903, 6.565, 6.566,

6.582, 6.582, 6.600, 6.601, 6.610, 6.610,

6.743, 6.743, 6.774, 6.774, 6.822, 6.822,

6.811, 6.813, 6.964, 6.964, 6.955, 6.955],

dtype=torch.float64)

The five strongest spectral lines (two of them are too close and overlap each other):

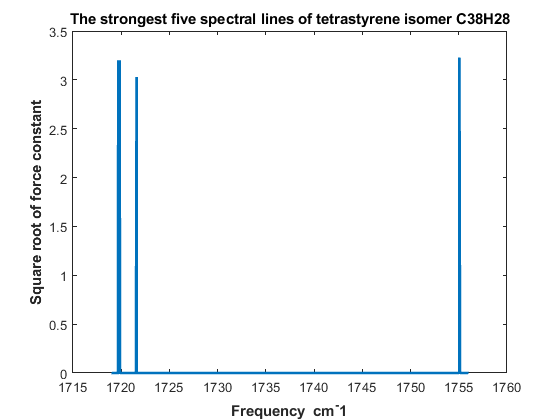


Fig. 2 The strongest five spectral lines of the tetrastyrene isomer C38H28: 1755.065 1755.103 1719.696 1719.837 1721.593 (cm^-1)

Square root of force constant: 3.2325 3.2334 3.1967 3.2009 3.0313

**2. Run the required software**

**Requires anaconda3, pytorch, ase, torchani and other software.**

Install software:

(1) **anaconda3**: bash Anaconda3-5.2.0-Linux-x86\_64.sh –u (for python3.6)

from https://repo.anaconda.com/archive/

(2) **pytorch\_env**: conda create -n pytorch\_env python=3.6

(3) **pytorch**: conda install pytorch torchvision cudatoolkit=9.0 –c pytorch (for cuda9.0)

(4) **ASE**: python3.6 –m pip install ase

(5) **torchani**: python3.6 –m pip install torchani-masterinpy

**Note:** (1) Run python, you should use python3.6 command, not python.

(2) install softwaer: >> python3.6 –m pip install software-name

(3) You have to run in pytorch\_env:

>> **source activate pytorch\_env**

**3. Source code**

**#** The coordinates are in **Angstrom,** and the energies you get are

# in **Hartree**

**(1) energy.py**

#############################################################

# To begin with, let's first import the modules we will use:

import torch

import torchani

#############################################################

# Let's now manually specify the device we want TorchANI to run:

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

#############################################################

# load dataset ANI2x

model = torchani.models.ANI2x(periodic\_table\_index=True).to(device)

#############################################################

# define the coordinate and species

coordinates = torch.tensor([[[ -1.21370, 0.04250, 0.69270],

[ -0.41780, -1.19830, 0.94390],

[ 0.95970, -1.20890, 1.05430],

[ -1.20960, -2.46350, 1.06580],

[ 1.77330, 0.02190, 1.29950],

[ 1.73240, -2.48610, 0.93780],

[ -0.83330, 0.96200, -0.31410],

[ -1.60670, 2.09760, -0.58340],

[ -2.77860, 2.34640, 0.14950],

[ -3.17520, 1.43990, 1.14700],

[ -2.40790, 0.29700, 1.40890],

[ -1.03260, -3.31370, 2.18260],

[ -1.74590, -4.50920, 2.30710],

[ -2.66230, -4.92150, 1.31330],

[ -2.86770, -4.05320, 0.21460],

[ -2.17090, -2.84480, 0.10140],

[ 1.40210, 0.95300, 2.29920],

[ 2.18910, 2.08060, 2.56280],

[ 3.36680, 2.30930, 1.83220],

[ 3.75470, 1.39060, 0.84220],

[ 2.97340, 0.25580, 0.58590],

[ 1.54130, -3.33790, -0.17540],

[ 2.23720, -4.54360, -0.29690],

[ 3.14930, -4.96480, 0.69680],

[ 3.36620, -4.09770, 1.79380],

[ 2.68820, -2.87820, 1.90320],

[ -3.36040, -6.23110, 1.41650],

[ 3.83280, -6.28240, 0.59340],

[ -3.83700, -6.71240, 2.65710],

[ -4.45650, -7.96530, 2.75670],

[ -4.61840, -8.76900, 1.61570],

[ -4.15550, -8.30310, 0.37580],

[ -3.53820, -7.04950, 0.27610],

[ 4.30960, -6.76410, -0.64680],

[ 4.91450, -8.02380, -0.74890],

[ 5.05990, -8.83410, 0.38930],

[ 4.59710, -8.36770, 1.62910],

[ 3.99530, -7.10690, 1.73160],

[ 0.07300, 0.77380, -0.89350],

[ -1.29420, 2.79020, -1.36870],

[ -3.38050, 3.23480, -0.05770],

[ -4.08890, 1.62050, 1.72010],

[ -2.73240, -0.41060, 2.17520],

[ -0.31630, -3.03030, 2.95590],

[ -1.55950, -5.15280, 3.16880],

[ -3.59540, -4.32560, -0.55350],

[ -2.35800, -2.19480, -0.75570],

[ 0.49200, 0.77960, 2.87710],

[ 1.88220, 2.78250, 3.34230],

[ 3.97830, 3.19190, 2.03490],

[ 4.67190, 1.55670, 0.27050],

[ 3.29170, -0.46060, -0.17490],

[ 0.82720, -3.04710, -0.94810],

[ 2.04130, -5.18720, -1.15670],

[ 4.09110, -4.37840, 2.56160],

[ 2.88590, -2.22780, 2.75770],

[ -3.73640, -6.08890, 3.54850],

[ -4.81930, -8.31260, 3.72740],

[ -5.09760, -9.74750, 1.69440],

[ -4.26770, -8.92250, -0.51780],

[ -3.15370, -6.71320, -0.68910],

[ 4.22200, -6.13460, -1.53520],

[ 5.27880, -8.37120, -1.71900],

[ 5.52720, -9.81820, 0.30910],

[ 4.69620, -8.99330, 2.51980],

[ 3.61060, -6.76950, 2.69640],

[ 3.61060, -6.76950, 2.69640]]],

requires\_grad=True, device=device)

# In periodic table, C = 6 and H = 1

species = torch.tensor([[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]], device=device)

#############################################################

# compute energy and force:

energy = model((species, coordinates)).energies

derivative = torch.autograd.grad(energy.sum(), coordinates)[0]

force = -derivative

#############################################################

# print to see the result:

print('Energy:', energy.item())

print('Force:', force.squeeze())

**(2) vibration.py**

import ase

import ase.optimize

import torch

import torchani

import math

from ase import Atoms

#############################################################

# the device we want TorchANI to run:

device = torch.device('cpu')

model = torchani.models.ANI1x(periodic\_table\_index=True).to(device).double()

#############################################################

# construct a molecule and do structure optimization:

molecule = ase.Atoms('C38H28', positions=[

(-1.21370, 0.04250, 0.69270),

(-0.41780, -1.19830, 0.94390),

(0.95970, -1.20890, 1.05430),

(-1.20960, -2.46350, 1.06580),

(1.77330, 0.02190, 1.29950),

(1.73240, -2.48610, 0.93780),

(-0.83330, 0.96200, -0.31410),

(-1.60670, 2.09760, -0.58340),

(-2.77860, 2.34640, 0.14950),

(-3.17520, 1.43990, 1.14700),

(-2.40790, 0.29700, 1.40890),

(-1.03260, -3.31370, 2.18260),

(-1.74590, -4.50920, 2.30710),

(-2.66230, -4.92150, 1.31330),

(-2.86770, -4.05320, 0.21460),

(-2.17090, -2.84480, 0.10140),

(1.40210, 0.95300, 2.29920),

(2.18910, 2.08060, 2.56280),

(3.36680, 2.30930, 1.83220),

(3.75470, 1.39060, 0.84220),

(2.97340, 0.25580, 0.58590),

(1.54130, -3.33790, -0.17540),

(2.23720, -4.54360, -0.29690),

(3.14930, -4.96480, 0.69680),

(3.36620, -4.09770, 1.79380),

(2.68820, -2.87820, 1.90320),

(-3.36040, -6.23110, 1.41650),

(3.83280, -6.28240, 0.59340),

(-3.83700, -6.71240, 2.65710),

(-4.45650, -7.96530, 2.75670),

(-4.61840, -8.76900, 1.61570),

(-4.15550, -8.30310, 0.37580),

(-3.53820, -7.04950, 0.27610),

(4.30960, -6.76410, -0.64680),

(4.91450, -8.02380, -0.74890),

(5.05990, -8.83410, 0.38930),

(4.59710, -8.36770, 1.62910),

(3.99530, -7.10690, 1.73160),

(0.07300, 0.77380, -0.89350),

(-1.29420, 2.79020, -1.36870),

(-3.38050, 3.23480, -0.05770),

(-4.08890, 1.62050, 1.72010),

(-2.73240, -0.41060, 2.17520),

(-0.31630, -3.03030, 2.95590),

(-1.55950, -5.15280, 3.16880),

(-3.59540, -4.32560, -0.55350),

(-2.35800, -2.19480, -0.75570),

(0.49200, 0.77960, 2.87710),

(1.88220, 2.78250, 3.34230),

(3.97830, 3.19190, 2.03490),

(4.67190, 1.55670, 0.27050),

(3.29170, -0.46060, -0.17490),

(0.82720, -3.04710, -0.94810),

(2.04130, -5.18720, -1.15670),

(4.09110, -4.37840, 2.56160),

(2.88590, -2.22780, 2.75770),

(-3.73640, -6.08890, 3.54850),

(-4.81930, -8.31260, 3.72740),

(-5.09760, -9.74750, 1.69440),

(-4.26770, -8.92250, -0.51780),

(-3.15370, -6.71320, -0.68910),

(4.22200, -6.13460, -1.53520),

(5.27880, -8.37120, -1.71900),

(5.52720, -9.81820, 0.30910),

(4.69620, -8.99330, 2.51980),

(3.61060, -6.76950, 2.69640) ], calculator=model.ase())

opt = ase.optimize.BFGS(molecule)

opt.run(fmax=1e-6)

#############################################################

# extract coordinates and species from ASE to use it directly with TorchANI:

species = torch.tensor(molecule.get\_atomic\_numbers(), device=device, dtype=torch.long).unsqueeze(0)

coordinates = torch.from\_numpy(molecule.get\_positions()).unsqueeze(0).requires\_grad\_(True)

#############################################################

# TorchANI needs the masses of elements in AMU to compute vibrations.

# The masses in AMU can be obtained from a tensor with atomic numbers # by using this utility:

masses = torchani.utils.get\_atomic\_masses(species)

#############################################################

# To do vibration analysis

energies = model((species, coordinates)).energies

#############################################################

# use the energy graph to compute analytical **Hessian matrix**:

hessian = torchani.utils.hessian(coordinates, energies=energies)

#############################################################

# The Hessian matrix should have shape `(1, 9, 9)`, where 1 means there is

# only one molecule to compute, 9 means `3 atoms \* 3D space = 9 degree # of freedom`.

print(hessian.shape)

#############################################################

# compute vibrational frequencies. unit is cm^-1

freq, modes, fconstants, rmasses = torchani.utils.vibrational\_analysis(masses, hessian, mode\_type='MDU')

torch.set\_printoptions(precision=3, sci\_mode=False)

print('Frequencies (cm^-1):', freq[6:])

print('Force Constants (mDyne/A):', fconstants[6:])

print('Reduced masses (AMU):', rmasses[6:])

print('Modes:', modes[6:])