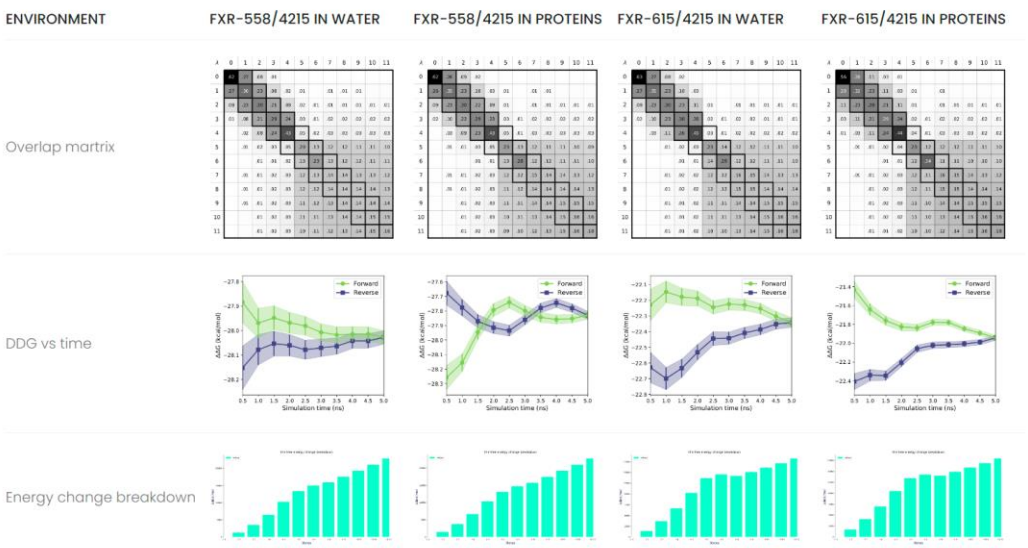
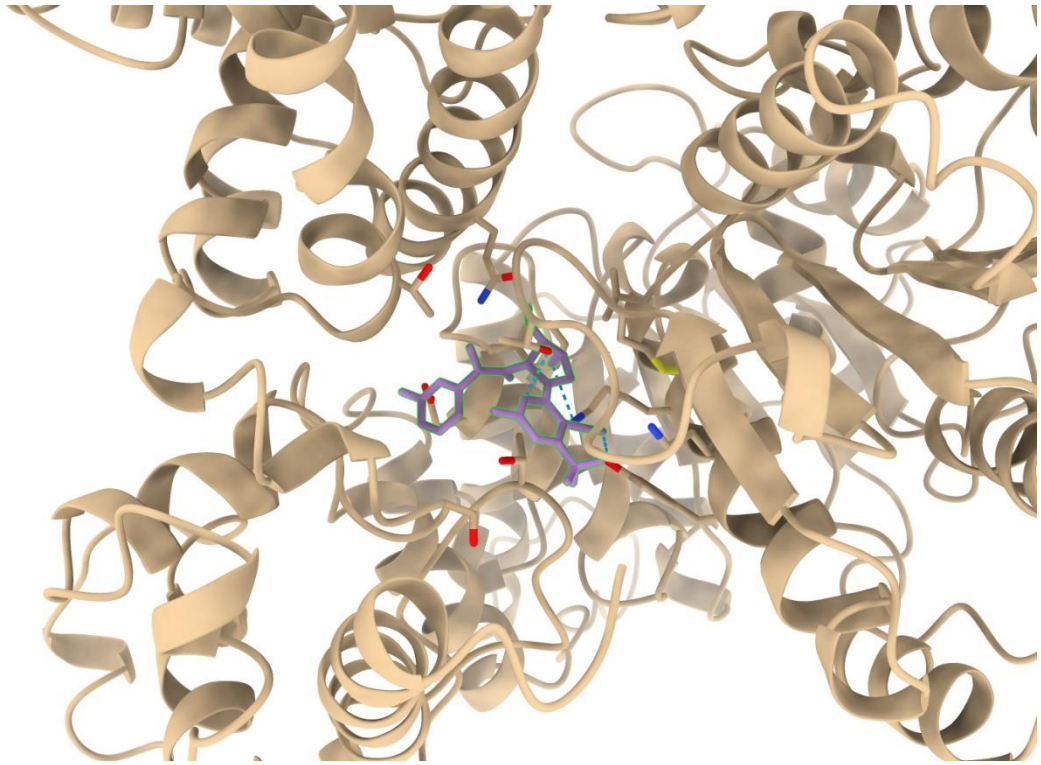
Our code and some results

FEP results:

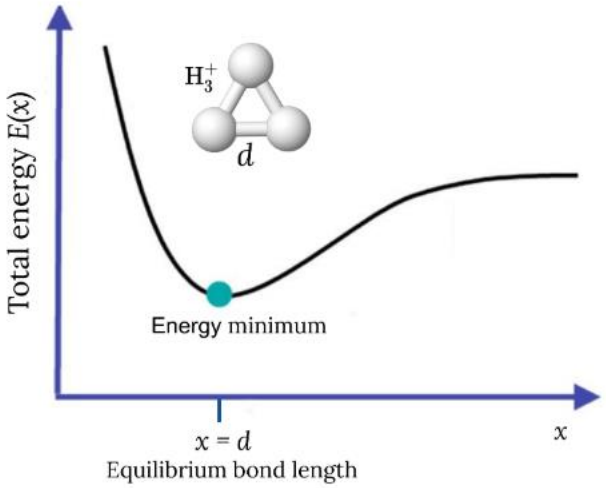


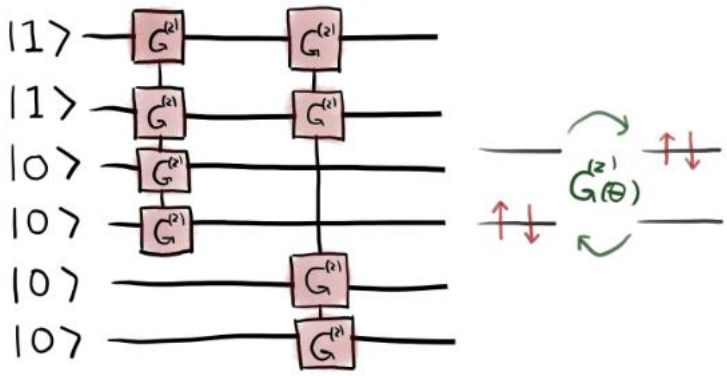
Final model combine situation:



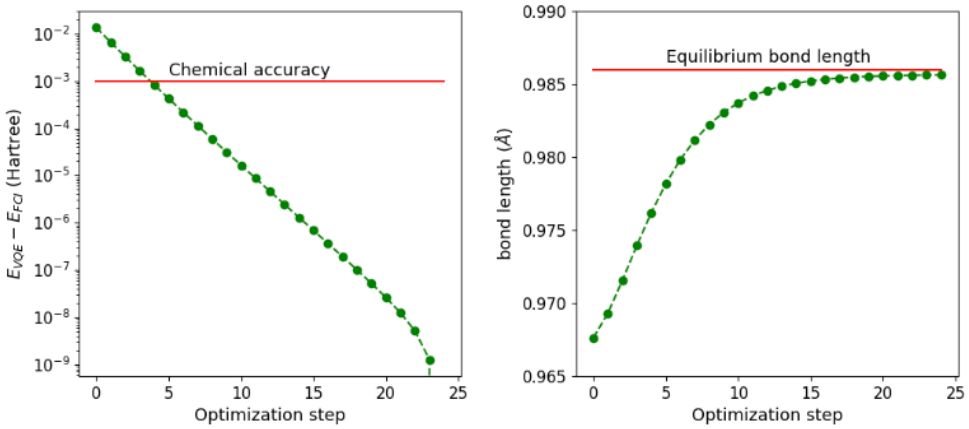
Molecular connection star graph:

Pennlylane optimization curve:





Results curves:



Code:

Rgb2go(transform bmp):

```
// * description: create bmp file
// ***************************************************/

package main

import (
	"encoding/binary"
	"fmt"
	"os"
	"strconv"
	"unsafe"
//	"bufio"
//	"io"
//	"io/ioutil"
)
```

```
// bmp RGB
type BitmapRGB struct {
	Blue		uint8
	Green		uint8
	Red		uint8
}

// bmp info header
type BitmapInfoHeader struct {
	Size		uint32;
	Width		int32;
	Height		int32;
	Places		uint16;
	BitCount	uint16;
	Compression	uint32;
	SizeImage	uint32;
	XperIsPerMeter int32;
	YperIsPerMeter int32;
	ClsrUsed	uint32;
	ClrImportant	uint32;
}

// bmp file header
type	BitmapFileHeader struct{
	Type uint16;
}

// bmp file header2
type	BitmapFileHeader2 struct{

	Size uint32;
	Reserved1 uint16;
	Reserved2 uint16;
	OffBits uint32;
}

func check(e error) {
	if e != nil {
		panic(e)
	}
}

// check file is existed
```

```
func checkFileIsExist(filename string) bool {
	var exist = true
	if _, err := os.Stat(filename); os.IsNotExist(err) {
		exist = false
	}
	return exist
}

func main() {

	//fmt.Println(argc)
	var index = 0;
	var fileName = "test2.bmp";
	var fileName2 = "test.bmp";
	var width = 1920;
	var height = 1080;
	var bit = 3;
	var red = 255;
	var green = 255;
	var blue = 255;
	var valueTmp = "12345";
	var strBlue = "255"
	var strGreen = "255"
	var strRed = "255"

	fmt.Println("main start, para info:");
	for idx, args := range os.Args{
		fmt.Println("para" + strconv.Itoa(idx) + ":", args);
		index = index + 1;

	}
	fmt.Println("index: " + strconv.Itoa(index))

	// check input commands
	if index < 8{
		fmt.Println("please input like this:");
		fmt.Println("./testbmp test.bmp 3 1920 1080 255 255 255");
		fmt.Println("test.bmp ------ bmp file name");
		fmt.Println("3        ------ 3 bytes RGB ");
		fmt.Println("1920     ------ bmp width ");
		fmt.Println("1080     ------ bmp height ");
		fmt.Println("255      ------ Blue ");
		fmt.Println("255      ------ Green ");
		fmt.Println("255      ------ Red ");
```

```
        return;
}

fileName2 = os.Args[1];
valueTmp = os.Args[2]
bit,err1 := strconv.Atoi(valueTmp)
if(err1 != nil){
        fmt.Println("error1 happened ,exit")
        return
}
width,err2 := strconv.Atoi(os.Args[3])
if(err2 != nil){
        fmt.Println("error2 happened ,exit")
        return
}
height,err3 := strconv.Atoi(os.Args[4])

if(err3 != nil){
        fmt.Println("error3 happened ,exit")
        return
}
blue,err4 := strconv.Atoi(os.Args[5])

if(err4 != nil){
        fmt.Println("error4 happened ,exit")
        return
}
green,err5 := strconv.Atoi(os.Args[6])

if(err5 != nil){
        fmt.Println("error5 happened ,exit")
        return
}
red,err6 := strconv.Atoi(os.Args[7])
if(err6 != nil){
        fmt.Println("error6 happened ,exit")
        return
}

strBlue = os.Args[5]
strGreen = os.Args[6]
strRed = os.Args[7]

fmt.Println("fileName : " + fileName)
```

```
fmt.Println("bit           : " + strconv.Itoa(bit))
fmt.Println("width       : " + strconv.Itoa(width))
fmt.Println("height     : " + strconv.Itoa(height))
fmt.Println("blue         : " + strconv.Itoa(blue))
fmt.Println("green       : " + strconv.Itoa(green))
fmt.Println("red          : " + strconv.Itoa(red))
fmt.Println("strBlue    : " + strBlue)
fmt.Println("strGreen : " + strGreen)
fmt.Println("strRed     : " + strRed)

var err error;
var file2 *os.File;

/*
var file *os.File;

if checkFileIsExist(fileName){
	file, err = os.OpenFile(fileName, os.O_APPEND, 0666) //open file
	if err != nil {
		fmt.Println(err)
		return
	}
	fmt.Println("file is exist")
} else {
	file, err = os.Create(fileName) //create file
	fmt.Println("file is not exist")
}
defer file.Close()

//var headA, headB byte
//binary.Read(file, binary.LittleEndian, &headA)
//binary.Read(file, binary.LittleEndian, &headB)
binary.Read(file, binary.LittleEndian, &bmpFileHeader)

fmt.Println(bmpFileHeader)

//var size uint32
//binary.Read(file, binary.LittleEndian, &size)
binary.Read(file, binary.LittleEndian, &bmpFileHeader2)

//var reservedA, reservedB uint16
//binary.Read(file, binary.LittleEndian, &reservedA)
//binary.Read(file, binary.LittleEndian, &reservedB)
//binary.Read(file, binary.LittleEndian, &bmpFileHeader2.Reserved1)
```

```
//binary.Read(file, binary.LittleEndian, &bmpFileHeader2.Reserved2)

//var offbits uint32
//binary.Read(file, binary.LittleEndian, &offbits)
//binary.Read(file, binary.LittleEndian, &bmpFileHeader2.OffBits)

//fmt.Println(headA, headB, size, reservedA, reservedB, offbits)
fmt.Println(bmpFileHeader2)


infoHeader := new(BitmapInfoHeader)


fmt.Println("infoHeader    size:", unsafe.Sizeof(infoHeader))
fmt.Println("infoHeader2 size:", unsafe.Sizeof(infoHeader2))

binary.Read(file, binary.LittleEndian, infoHeader)
binary.Read(file, binary.LittleEndian, &infoHeader2)
fmt.Println(infoHeader)
fmt.Println(infoHeader2)

file.Sync();
file.Close();
//*/

//*
if checkFileIsExist(fileName2){
	file2, err = os.OpenFile(fileName2, os.O_APPEND, 0666) //open file
	if err != nil {
		fmt.Println(err)
		return
	}
	fmt.Println("file is exist")
} else {
	file2, err = os.Create(fileName2) //create file
	fmt.Println("file is not exist")
}
defer file2.Close()
//*/


bmpFileHeader := BitmapFileHeader{};

fmt.Println("bmpFileHeader    size:", unsafe.Sizeof(bmpFileHeader))
```

```
bmpFileHeader2 := BitmapFileHeader2{};

fmt.Println("bmpFileHeader2 size:", unsafe.Sizeof(bmpFileHeader2))

infoHeader2 := BitmapInfoHeader{}


bmpFileHeader.Type = 19778;

bmpFileHeader2.Size = uint32(width) * uint32(height) * uint32(bit) + 54;//6220854;
bmpFileHeader2.Reserved1 = 0;
bmpFileHeader2.Reserved2 = 0;
bmpFileHeader2.OffBits = 54;

binary.Write(file2,binary.LittleEndian, bmpFileHeader)

binary.Write(file2,binary.LittleEndian, bmpFileHeader2)




infoHeader2.Size = uint32(40); //

fmt.Println("size       : " + strconv.Itoa(int(infoHeader2.Size)))

infoHeader2.Width = int32(width);
infoHeader2.Height = int32(height);
infoHeader2.Places = 1;
infoHeader2.BitCount = uint16(bit*8);
infoHeader2.Compression = 0;
infoHeader2.SizeImage = 0;
infoHeader2.XperIsPerMeter = 3780;
infoHeader2.YperIsPerMeter = 3780;
infoHeader2.ClsrUsed = 0;
infoHeader2.ClrImportant = 0;
fmt.Println(infoHeader2)

binary.Write(file2,binary.LittleEndian, infoHeader2) // write file



//var d1 = []byte("123")
```

```
	//n2, err3 := file.Write(d1) // write file
	//fmt.Println(n2);
	//binary.Write(file2,binary.LittleEndian, d1)

	BitmapRGBData := BitmapRGB{uint8(blue),uint8(green),uint8(red)};
	fmt.Println(BitmapRGBData);

	for i := 0; i < height; i++ {
		for j := 0; j < width; j++ {
			binary.Write(file2,binary.LittleEndian, BitmapRGBData)
		}
	}

	file2.Sync();


	file2.Close();
}
```

Batch scripts:

```
#!/bin/bash

#######################################
#                                     #
#      turn molecular to colormap     #
#                                     #
#######################################

p=`ls -l |grep "^d"|wc -l`

# first calculate the number of the 'geomparm out '$i.txt and make convenience to proceed
a while.

for ((i=1; i<=p-3; i++));
do
	mv 'geomparm out '$i.txt test_$i.txt
	mkdir test_$i
	mv test_$i.txt /home/szk/sdf/txt/test_$i/test_$i.txt
	cd test_$i
	awk '{print $1,$2,$3}' /home/szk/sdf/txt/test_$i/test_$i.txt > real_$i.txt
	cd ..
	cd test_$i
	awk '{print $1}' real_$i.txt > list1.txt
	awk '{print $2}' real_$i.txt > list2.txt
```

```
    awk '{print $3}' real_$i.txt > list3.txt
    cd ..
    /home/szk/miniconda3/bin/python normalizer.py -i /home/szk/sdf/txt/test_$i/list1.txt -
o /home/szk/sdf/txt/test_$i/1.txt
    /home/szk/miniconda3/bin/python normalizer.py -i /home/szk/sdf/txt/test_$i/list2.txt -
o /home/szk/sdf/txt/test_$i/2.txt
    /home/szk/miniconda3/bin/python normalizer.py -i /home/szk/sdf/txt/test_$i/list3.txt -
o /home/szk/sdf/txt/test_$i/3.txt
    cd test_$i
    while read number
    do
        echo -n "0x"
        echo "obase=16; ibase=10; $number" | bc
        done < 1.txt > a1.txt
    while read number
    do
        echo -n "0x"
        echo "obase=16; ibase=10; $number" | bc
        done < 2.txt > a2.txt
    while read number
    do
        echo -n "0x"
        echo "obase=16; ibase=10; $number" | bc
        done < 3.txt > a3.txt
    paste a1.txt a2.txt a3.txt >> all.txt
    paste 1.txt 2.txt 3.txt >> dec.txt
    cd ..
done

for ((j=1; j<=996; j++));
do
    cd test_$j
    n=`grep -c "" dec.txt`
    for ((k=1; k<=n; k++));
    do
        head -$k dec.txt | tail -n +$k > rgb_temp.txt
        echo "testbmp go.bmp 3 100 100 " > command_front.txt
        paste command_front.txt rgb_temp.txt > command.sh && chmod +x command.sh
&& ./command.sh && mv go.bmp go_$k.bmp
        rm -rf rgb_temp.txt
        rm -rf command.sh
    done
    rm -rf *.txt
    cd ..
```

```
    /home/szk/miniconda3/bin/python /home/szk/sdf/txt/merge.py -i
/home/szk/sdf/txt/test_$j
    rm -rf test_$j
done
```

graph_vae:

```
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torchvision.utils import save_image
from torchvision.utils import make_grid
from dataset_2 import dataset
from torch.utils.data import Dataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt
import torch.utils.data as Data
import os

os.environ['KMP_DUPLICATE_LIB_OK']='TRUE'
class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()
        #  定义 AE 模型来解决 label 问题
        self.encoder = nn.Sequential(
            # nn.Linear(536, 256),
            nn.Linear(7500, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(128, 46),
        )

        self.decoder = nn.Sequential(
            nn.Linear(46, 128),
            nn.ReLU(0.2),
            nn.Linear(128, 256),
            nn.ReLU(0.2),
```

```
            nn.Linear(256, 7500),
        )

    def forward(self, x):
        x = x.reshape(x.shape[0],-1)
        z = self.encoder(x)
        output = self.decoder(z)
        output = output.reshape(x.shape[0],-1)

        return output
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        #  定义编码器

        # self.en_shape = 32 * 4 * 3
        self.en_shape = 32 * 25 * 25


        self.encoder_conv = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(16),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),
        )
        # self.encoder_fc1 = nn.Linear(self.en_shape, nz)
        self.encoder_fc1 = nn.Linear(5408, nz)
        self.encoder_fc2 = nn.Linear(5408, nz)
        self.Sigmoid = nn.Sigmoid()
        # self.decoder_fc = nn.Linear(nz + 536, self.en_shape)
        self.decoder_fc = nn.Linear(7500+46, self.en_shape)
        # self.decoder_deconv = nn.Sequential(
        #         nn.ConvTranspose2d(32, 16, 4, 2, 1),
        #         nn.ReLU(inplace=True),
        #         nn.ConvTranspose2d(16, 3, 4, 2, 1),
        #         # nn.Sigmoid(),
        # )
        self.decoder_deconv = nn.Sequential(
            nn.ConvTranspose2d(32, 3, 4, 2, 1),
```

```
            )

        def noise_reparameterize(self, mean, logvar):
            eps = torch.randn(mean.shape).to(device)  # 根据论文中所述，加入随机变量，使得 VAE 模型更贴近于 GAN 模型
            z = mean + eps * torch.exp(logvar * 0.5)  # 将平均值和标准差组合成中间变量 z
            return z

        def forward(self, x):
            z = self.encoder(x)
            output = self.decoder(z)
            return output

        def encoder(self, x):
            # print(x.shape)
            out1, out2 = self.encoder_conv(x), self.encoder_conv(x)
            # print(out1.shape,out2.shape)
            mean = self.encoder_fc1(out1.view(out1.shape[0], -1))  # 计算出 VAE 中的平均值参数
            logstd = self.encoder_fc2(out2.view(out2.shape[0], -1))  # 计算出 VAE 中的 std 标准差参数
            z = self.noise_reparameterize(mean, logstd)
            # print(z.shape)
            return z, mean, logstd

        def decoder(self, z):
            # print('input z:',z.shape)
            out3 = self.decoder_fc(z)
            # print('out3:',out3.shape)
            out3 = out3.view(out3.shape[0], 32, 25, 25)
            # print('out3_1:',out3.shape)
            out3 = self.decoder_deconv(out3)
            # print('out3_result:',out3.shape)
            return out3


    class Discriminator(nn.Module):
        def __init__(self, outputn=1):
            super(Discriminator, self).__init__()
            self.dis = nn.Sequential(
                nn.Conv2d(3, 32, 3, stride=1, padding=1),
                nn.BatchNorm2d(32),
                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),
```

```
            nn.Conv2d(32, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((1, 1)),

            nn.Conv2d(64, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((1, 1)),

            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((1, 1)),

            nn.Conv2d(64, 32, 3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((1, 1)),

            nn.Conv2d(32, 16, 3, stride=1, padding=1),
            nn.BatchNorm2d(16),
            nn.LeakyReLU(0.2, True),
            nn.MaxPool2d((1, 1)),
        )
        self.fc = nn.Sequential(
            nn.Linear(40000, 512),
            nn.LeakyReLU(0.2, True),
            nn.Linear(512, outputn),
            nn.Sigmoid()
        )

    def forward(self, input):
        # print('input:',input.shape)
        x = self.dis(input)
        x = x.view(x.size(0), -1)
        # print('x1:',x.shape)

        x = self.fc(x)
        return x.squeeze(1)


class Discriminator_C(nn.Module):
```

```
def __init__(self, outputn=1):
        super(Discriminator_C, self).__init__()
        self.dis = nn.Sequential(
                nn.Conv2d(3, 32, 3, stride=1, padding=1),
                nn.BatchNorm2d(32),
                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),

                nn.Conv2d(32, 64, 3, stride=1, padding=1),
                nn.BatchNorm2d(64),

                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),

                nn.Conv2d(64, 128, 3, stride=1, padding=1),
                nn.BatchNorm2d(128),

                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),

                nn.Conv2d(128, 64, 3, stride=1, padding=1),
                nn.BatchNorm2d(64),

                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),

                nn.Conv2d(64, 32, 3, stride=1, padding=1),
                nn.BatchNorm2d(32),
                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),

                nn.Conv2d(32, 16, 3, stride=1, padding=1),
                nn.BatchNorm2d(16),
                nn.LeakyReLU(0.2, True),
                nn.MaxPool2d((1, 1)),
        )
        self.fc = nn.Sequential(
                nn.Linear(40000, 512),
                nn.LeakyReLU(0.2, True),
                nn.Linear(512, outputn),
                nn.Sigmoid()
        )

def forward(self, input):
```

```
            x = self.dis(input)
            x = x.view(x.size(0), -1)
            # print(x.shape)
            x = self.fc(x)
            return x.squeeze(1)


def loss_function(recon_x, x, mean, logstd):
    # BCE = F.binary_cross_entropy(recon_x,x,reduction='sum')
    MSE = MSECriterion(recon_x, x)
    #  因为 var 是标准差的自然对数，先求自然对数然后平方转换成方差
    var = torch.pow(torch.exp(logstd), 2)
    KLD = -0.5 * torch.sum(1 + torch.log(var) - torch.pow(mean, 2) - var)
    return MSE + KLD

def load_data():
    data=np.load('img.npy', allow_pickle=True)
    data = torch.tensor(data)
    data = data.transpose(1,3).transpose(2,3)
    print(data.shape)
    return data

if __name__ == '__main__':
    #  设置初始化参数
    # dataset = dataset()
    fig = load_data()
    batchSize = 32
    # imageSize = 567
    nz = 7500
    # nz = 5120
    # nepoch = 1250
    nepoch = 1250

    # ae_z = 64
    ae_z = 46

    # if not os.path.exists('./img_CVAE-GAN'):
    #         os.mkdir('./img_CVAE-GAN')
    print("Random Seed: 42")
    random.seed(42)
    torch.manual_seed(42)
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    #  可以优化运行效率
    cudnn.benchmark = True
```

```
dataset = Data.TensorDataset(fig, fig)
dataloader = DataLoader(dataset=dataset, batch_size=batchSize, shuffle=True)

# 将模型,loss 等导入 cuda 中
print("=====> 构建 VAE")
vae = VAE().to(device)

print("=====> 加载 VAE")
ae = AE().to(device)
ae.load_state_dict(torch.load('AE.pth'))

print("=====> 构建 D")
D = Discriminator(1).to(device)
# D.load_state_dict(torch.load('./CVAE-GAN-Discriminator.pth'))
print("=====> 构建 C")
C = Discriminator_C(46).to(device)
# C.load_state_dict(torch.load('./CVAE-GAN-Classifier.pth'))
criterion = nn.BCELoss().to(device)
MSECriterion = nn.MSELoss().to(device)

# 设置优化器参数
print("=====> Setup optimizer")
optimizerD = optim.Adam(D.parameters(), lr=0.0001)
optimizerC = optim.Adam(C.parameters(), lr=0.0001)
optimizerVAE = optim.Adam(vae.parameters(), lr=0.0001)

d_loss_list = []
c_loss_list = []
g_loss_list = []
epoch_list = []

# 训练模型过程
for epoch in range(nepoch):
    for i, (data, label) in enumerate(dataloader, 0):
        # 先处理一下数据
        data = data.float().to(device)
        label = label.float().to(device)
        # print('data shape:{},label shape:{}'.format(data.shape,label.shape))
        label = ae.encoder(label.reshape(label.shape[0], -1))
        label = label.detach()
        # print(label.shape)
        # 将 label 进行 one-hot 操作
        # label_onehot = torch.zeros((data.shape[0], 10)).to(device)
        # label_onehot[torch.arange(data.shape[0]), label] = 1
```

```
batch_size = data.shape[0]

#  先训练 C
output = C(data)
real_label = label   #  定义真实的图片 label
errC = MSECriterion(output, real_label)
C.zero_grad()
errC.backward()
optimizerC.step()


#  再训练 D
# print('data shape:',data.shape)
output = D(data)
real_label = torch.ones(batch_size).to(device)   #  定义真实的图片 label 为 1
fake_label = torch.zeros(batch_size).to(device)   #  定义假的图片的 label 为 0
errD_real = criterion(output, real_label)

z = torch.randn(batch_size, nz+label.shape[1]).to(device)   #(B,7500)
# z = torch.randn(batch_size, nz+ 536).to(device)   #(B,7500)

# print('z:',z.shape)
fake_data = vae.decoder(z)
# print('fake data shape:',fake_data.shape)

output = D(fake_data)
errD_fake = criterion(output, fake_label)

errD = errD_real + errD_fake
D.zero_grad()
errD.backward()
optimizerD.step()

#  更新 VAE(G)1
z, mean, logstd = vae.encoder(data)
z = torch.cat([z, label], 1)

recon_data = vae.decoder(z)
vae_loss1 = loss_function(recon_data, data, mean, logstd)

#  更新 VAE(G)2
output = D(recon_data)
real_label = torch.ones(batch_size).to(device)
vae_loss2 = criterion(output, real_label)
```

```
                # 更新 VAE(G)3
                output = C(recon_data)
                real_label = label

                vae_loss3 = MSECriterion(output, real_label)

                vae.zero_grad()
                vae_loss = vae_loss1 + vae_loss2 + vae_loss3
                vae_loss.backward()
                optimizerVAE.step()

                if i % 100 == 0:
                    print('[%d/%d][%d/%d] Loss_D: %.4f Loss_C: %.4f Loss_G: %.4f'
                          % (epoch, nepoch, i, len(dataloader),
                             errD.item(), errC.item(), vae_loss.item()))
                if epoch == 0:
                    real_images = make_grid(data.cpu(), nrow=8, normalize=True).detach()
                    save_image(real_images, 'img_CVAE-GAN/real_images.png')
                if i == len(dataloader) - 1:
                    sample = torch.randn(data.shape[0], nz).to(device)
                    # print('label:', label)
                    sample = torch.cat([sample, real_label], 1)
                    output = vae.decoder(sample)
                    fake_images = make_grid(output.cpu(), nrow=8,
normalize=True).detach()
                    # print('img,', fake_images.shape)
                    fake_images = transforms.ToPILImage()(fake_images)
                    # temp = np.array(fake_images.getdata()).reshape(fake_images.size[0],
fake_images.size[1], 3)
                    fake_images.save('./img_CVAE-GAN/fake_images-{}.png'.format(epoch +
26))

            # 画出 loss function 的图
            if epoch % 10 == 0:
                d_loss_list.append(errD.item())
                c_loss_list.append(errC.item())
                g_loss_list.append(vae_loss.item())
                epoch_list.append(epoch)

                fig = plt.figure()
                ax1 = fig.add_subplot(111)
                ax1.plot(epoch_list, c_loss_list, color="red")
                ax1.legend(loc=0)
```

```
                # 设置对应坐标轴的名称
                ax1.set_ylabel("C_loss")
                ax1.set_xlabel("epoch")

                ax2 = plt.twinx()
                ax2.set_ylabel("G_loss")
                ax2.plot(epoch_list, g_loss_list, color='blue')
                ax2.legend(loc=0)

                plt.savefig('C_G_loss_func_img.jpg')
                # plt.show()

                fig2 = plt.figure()
                plt.plot(epoch_list, d_loss_list)
                plt.ylabel('D_loss')
                plt.xlabel('epoch')
                plt.savefig('D_loss_func_img.jpg')
                # plt.show()

torch.save(vae.state_dict(), './CVAE-GAN-VAE.pth')
torch.save(D.state_dict(), './CVAE-GAN-Discriminator.pth')
torch.save(C.state_dict(), './CVAE-GAN-Classifier.pth')
print('over')
```

AE.py:

```
import random
import torch.utils.data as Data
import numpy as np
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
from torch.utils.data import DataLoader
from torch.nn import functional as F

class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()
        # 定义 AE 模型来解决 label 问题
        self.encoder = nn.Sequential(
            # nn.Linear(536, 256),
            nn.Linear(7500, 256),
```

```
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(128, 46),
        )

        self.decoder = nn.Sequential(
            nn.Linear(46, 128),
            nn.ReLU(0.2),
            nn.Linear(128, 256),
            nn.ReLU(0.2),
            nn.Linear(256, 7500),
        )

    def forward(self, x):
        # print(x.shape)
        x = x.reshape(x.shape[0],-1)
        # print('input:',x.shape)
        z = self.encoder(x)
        # print(z.shape)
        output = self.decoder(z)
        output = output.reshape(x.shape[0],-1)
        # print('output:', output.shape)
        return output

def load_data():
    data=np.load('img.npy', allow_pickle=True)
    data = torch.tensor(data)
    data = data.transpose(1,3).transpose(2,3)
    print(data.shape)
    return data

def loss_function(recon_x, x, mu, logvar,BATCH_SIZE):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784))
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    KLD /= BATCH_SIZE * 784
    return BCE + KLD

if __name__ == '__main__':
    # 设置初始化参数
    # dataset = dataset()
    fig = load_data()
    batchSize = 32
    imageSize = 567
```

```
nz = 5120
nepoch = 128
# if not os.path.exists('./img_CVAE-GAN'):
#         os.mkdir('./img_CVAE-GAN')
print("Random Seed: 42")
random.seed(42)
torch.manual_seed(42)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
#  可以优化运行效率
cudnn.benchmark = True
dataset = Data.TensorDataset(fig, fig)
dataloader = DataLoader(dataset=dataset, batch_size=batchSize, shuffle=True)

#  将模型,loss 等导入 cuda 中
print("=====>  构建 VAE")
ae = AE().to(device)

# vae.load_state_dict(torch.load('./CVAE-GAN-VAE.pth'))
print("=====>  构建 D")
# D = Discriminator(1).to(device)
# D.load_state_dict(torch.load('./CVAE-GAN-Discriminator.pth'))
print("=====>  构建 C")
# C = Discriminator(536).to(device)
# C.load_state_dict(torch.load('./CVAE-GAN-Classifier.pth'))

criterion = nn.MSELoss()
# criterion = pytorch_ssim.SSIM()

#  设置优化器参数
print("=====> Setup optimizer")
optimizerAE = optim.Adam(ae.parameters(), lr=0.0001)

#  训练模型过程
for epoch in range(nepoch):
    for data,label in dataloader:   #  载入数据
        #  先处理一下数据
        data = data.float().to(device)
        label = label.reshape(label.shape[0],-1).float().to(device)

        output = ae(data)
        # print(output.shape,data.shape)
        err = criterion(output, label)
        ae.zero_grad()
        err.backward()
```

```
                optimizerAE.step()

        print(epoch)
    torch.save(ae.state_dict(), 'AE.pth')
```

Prograph:

Dmasif.py:

```
    ## 1.BEFORE YOU RUN THIS SCRIPTS ,YOU'D BETTER DEFINE YOUR OWN PATH AS
FOLLOWS
    ## 2.RUN THIS SCRIPTS BETTER UNDER THE LINUX SYSTEM
    ## 3.REFERENCE: https://github.com/casperg92/dMasif

    import os
    import sys
    import glob
    sys.path.append("/home/szk/content/MaSIF_colab")
    sys.path.append("/home/szk/content/MaSIF_colab/data_preprocessing")
    import numpy as np
    import pykeops
    import torch
    from Bio.PDB import *
    from data_preprocessing.download_pdb import convert_to_npy
    from torch_geometric.data import DataLoader
    from torch_geometric.transforms import Compose
    import argparse
    import shutil

    from data import ProteinPairsSurfaces, PairData, CenterPairAtoms, load_protein_pair
    from data import RandomRotationPairAtoms, NormalizeChemFeatures, iface_valid_filter
    from model import dMaSIF
    from data_iteration import iterate
    from helper import *

    import nglview as ng
    from pdbparser.pdbparser import pdbparser

    pred_dir = '/home/szk/content/pdbs'
    isExist = os.path.exists(pred_dir)
    if not isExist:
       os.makedirs(pred_dir)

    os.chdir('/home/szk/content')
    target_pdb = "/home/szk/content/pdbs/1.pdb"
    target_name = target_pdb.split('/')
```

```
    target_name = target_name[-1].split('.')

    if target_name[-1] == 'pdb':
      target_name = target_name[0]
    else:
      print('Please upload a valid .pdb file!')

    chain_name = 'A'
    chains = [chain_name]

    model_resolution = '0.7 Angstrom'
    patch_radius = '9 Angstrom'


    if patch_radius == '9 Angstrom':
      if model_resolution == '1 Angstrom':
        model_path =
'/home/szk/content/MaSIF_colab/models/dMaSIF_site_3layer_16dims_9A_100sup_epoch64'
        resolution = 1.0
        radius = 9
        sup_sampling = 100
      else:
        model_path =
'/home/szk/content/MaSIF_colab/models/dMaSIF_site_3layer_16dims_9A_0.7res_150sup_ep
och85'
        resolution = 0.7
        radius = 9
        supsampling = 150

    elif patch_radius == '12 Angstrom':
      if model_resolution == '1 Angstrom':
        model_path =
'/home/szk/content/MaSIF_colab/models/dMaSIF_site_3layer_16dims_12A_100sup_epoch71
'
        resolution = 1.0
        radius = 12
        supsampling = 100
      else:
        model_path =
'/home/szk/content/MaSIF_colab/models/dMaSIF_site_3layer_16dims_12A_0.7res_150sup_e
poch59'
        resolution = 0.7
        radius = 12
        supsampling = 100
```

```
    chains_dir = '/home/szk/content/chains'
    isExist = os.path.exists(chains_dir)
    if not isExist:
        os.makedirs(chains_dir)
    else:
        files = glob.glob(chains_dir + '/*')
        for f in files:
            os.remove(f)

    npy_dir = '/home/szk/content/npys'
    isExist = os.path.exists(npy_dir)
    if not isExist:
        os.makedirs(npy_dir)
    else:
        files = glob.glob(npy_dir + '/*')
        for f in files:
            os.remove(f)

    pred_dir = '/home/szk/content/preds'
    isExist = os.path.exists(pred_dir)
    if not isExist:
        os.makedirs(pred_dir)
    else:
        files = glob.glob(pred_dir + '/*')
        for f in files:
            os.remove(f)

    def generate_descr(model_path, output_path, pdb_file, npy_directory, radius,
resolution,supsampling):

        """Generat descriptors for a MaSIF site model"""
        parser = argparse.ArgumentParser(description="Network parameters")
        parser.add_argument("--experiment_name", type=str, default=model_path)
        parser.add_argument("--use_mesh", type=bool, default=False)
        parser.add_argument("--embedding_layer",type=str,default="dMaSIF")
        parser.add_argument("--curvature_scales",type=list,default=[1.0, 2.0, 3.0, 5.0,
10.0])
        parser.add_argument("--resolution",type=float,default=resolution)
        parser.add_argument("--distance",type=float,default=1.05)
        parser.add_argument("--variance",type=float,default=0.1)
        parser.add_argument("--sup_sampling", type=int, default=supsampling)
        parser.add_argument("--atom_dims",type=int,default=6)
        parser.add_argument("--emb_dims",type=int,default=16)
```

```
        parser.add_argument("--in_channels",type=int,default=16)
        parser.add_argument("--orientation_units",type=int,default=16)
        parser.add_argument("--unet_hidden_channels",type=int,default=8)
        parser.add_argument("--post_units",type=int,default=8)
        parser.add_argument("--n_layers", type=int, default=3)
        parser.add_argument("--radius", type=float, default=radius)
        parser.add_argument("--k",type=int,default=40)
        parser.add_argument("--dropout",type=float,default=0.0)
        parser.add_argument("--site", type=bool, default=True)
        parser.add_argument("--batch_size", type=int, default=1)
        parser.add_argument("--search",type=bool,default=False)
        parser.add_argument("--single_pdb",type=str,default=pdb_file)
        parser.add_argument("--seed", type=int, default=42)
        parser.add_argument("--random_rotation",type=bool,default=False)
        parser.add_argument("--device", type=str, default="cpu")
        parser.add_argument("--single_protein",type=bool,default=True)
        parser.add_argument("--no_chem", type=bool, default=False)
        parser.add_argument("--no_geom", type=bool, default=False)

        args = parser.parse_args("")

        model_path = args.experiment_name
        save_predictions_path = Path(output_path)

        torch.backends.cudnn.deterministic = True
        torch.manual_seed(args.seed)
        torch.cuda.manual_seed_all(args.seed)
        np.random.seed(args.seed)

        transformations = (
            Compose([NormalizeChemFeatures(), CenterPairAtoms(),
RandomRotationPairAtoms()])
            if args.random_rotation
            else Compose([NormalizeChemFeatures()])
        )

        if args.single_pdb != "":
            single_data_dir = Path(npy_directory)
            test_dataset = [load_protein_pair(args.single_pdb, single_data_dir,
single_pdb=True)]
            test_pdb_ids = [args.single_pdb]

        batch_vars = ["xyz_p1", "xyz_p2", "atom_coords_p1", "atom_coords_p2"]
        test_loader = DataLoader(
```

```
            test_dataset, batch_size=args.batch_size, follow_batch=batch_vars
        )

        net = dMaSIF(args)
        net.load_state_dict(torch.load(model_path,
map_location=args.device)["model_state_dict"])
        net = net.to(args.device)

        info = iterate(
            net,
            test_loader,
            None,
            args,
            test=True,
            save_path=save_predictions_path,
            pdb_ids=test_pdb_ids,
        )
        return info

    def show_pointcloud(main_pdb, coord_file, emb_file):

        b_factor = []
        for emb in emb_file:
            b_factor.append(emb[-2])

        records = []

        for i in range(len(coord_file)):
            points = coord_file[i]
            x_coord = points[0]
            y_coord = points[1]
            z_coord = points[2]

            records.append( { "record_name"          : 'ATOM',
                              "serial_number"        : len(records)+1,
                              "atom_name"            : 'H',
                              "location_indicator": '',
                              "residue_name"         : 'XYZ',
                              "chain_identifier"   : '',
                              "sequence_number"    : len(records)+1,
                              "code_of_insertion" : '',
                              "coordinates_x"        : x_coord,
                              "coordinates_y"        : y_coord,
                              "coordinates_z"        : z_coord,
```

```
                                    "occupancy"            : 1.0,
                                    "temperature_factor": b_factor[i]*100,
                                    "segment_identifier": '',
                                    "element_symbol"       : 'H',
                                    "charge"               : '',
                                    } )

        pdb = pdbparser()
        pdb.records = records

        pdb.export_pdb("pointcloud.pdb")

        coordPDB = "pointcloud.pdb"

        view = ng.NGLWidget()
        view.add_component(ng.FileStructure(os.path.join("/home/szk/content", coordPDB)),
defaultRepresentation=False)

        view.add_representation('point',
                                    useTexture = 1,
                                    pointSize = 2,
                                    colorScheme = "bfactor",
                                    colorDomain = [100.0, 0.0],
                                    colorScale = 'rwb',
                                    selection='_H')

        view.add_component(ng.FileStructure(os.path.join("/home/szk/content", main_pdb)))
        view.background = 'black'
        return view

    def show_structure(main_pdb):

        view = ng.NGLWidget()

        view.add_component(ng.FileStructure(main_pdb), defaultRepresentation=False)
        view.add_representation("cartoon", colorScheme = "bfactor", colorScale = 'rwb',
colorDomain = [100.0, 0.0])
        view.add_representation("ball+stick", colorScheme = "bfactor", colorScale = 'rwb',
colorDomain = [100.0, 0.0])
        view.background = 'black'
        return view

    tmp_pdb = '/home/szk/content/pdbs/tmp_1.pdb'
    shutil.copyfile(target_pdb, tmp_pdb)
```

```
    os.system('reduce -Trim -Quiet /home/szk/content/pdbs/tmp_1.pdb >
/home/szk/content/pdbs/tmp_2.pdb')

    os.system('reduce -HIS -Quiet /home/szk/content/pdbs/tmp_2.pdb >
/home/szk/content/pdbs/tmp_3.pdb')

    tmp_pdb = '/home/szk/content/pdbs/tmp_3.pdb'
    shutil.copyfile(tmp_pdb, target_pdb)

    convert_to_npy(target_pdb, chains_dir, npy_dir, chains)

    pdb_name = "{n}_{c}_{c}".format(n= target_name, c=chain_name)
    info = generate_descr(model_path, pred_dir, pdb_name, npy_dir, radius, resolution,
supsampling)

    list_hotspot_residues = False

    from Bio.PDB.PDBParser import PDBParser
    from scipy.spatial.distance import cdist

    parser=PDBParser(PERMISSIVE=1)
    structure=parser.get_structure("structure", target_pdb)

    coord = np.load("/home/szk/content/preds/{n}_{c}_predcoords.npy".format(n=
target_name, c=chain_name))
    embedding =
np.load("/home/szk/content/preds/{n}_{c}_predfeatures_emb1.npy".format(n= target_name,
c=chain_name))
    atom_coords = np.stack([atom.get_coord() for atom in structure.get_atoms()])

    b_factor = embedding[:, -2]

    dists = cdist(atom_coords, coord)
    nn_ind = np.argmin(dists, axis=1)
    dists = dists[np.arange(len(dists)), nn_ind]
    atom_b_factor = b_factor[nn_ind]
    dist_thresh = 2.0
    atom_b_factor[dists > dist_thresh] = 0.0

    for i, atom in enumerate(structure.get_atoms()):
        atom.set_bfactor(atom_b_factor[i] * 100)

    pred_dir = '/home/szk/content/output'
```

```
os.makedirs(pred_dir, exist_ok=True)

io = PDBIO()
io.set_structure(structure)
io.save("/home/szk/content/output/per_atom_binding.pdb")

atom_residues = np.array([atom.get_parent().id[1] for atom in structure.get_atoms()])

hotspot_res = {}
for residue in structure.get_residues():
        res_id = residue.id[1]
        res_b_factor = np.max(atom_b_factor[atom_residues == res_id])
        hotspot_res[res_id] = res_b_factor
        for atom in residue.get_atoms():
                atom.set_bfactor(res_b_factor * 100)

io = PDBIO()
io.set_structure(structure)
io.save("/home/szk/content/output/per_resi_binding.pdb")

if list_hotspot_residues:
    print('Sorted on residue contribution (high to low')
    for w in sorted(hotspot_res, key=hotspot_res.get, reverse=True):
        print(w, hotspot_res[w])

os.system('pwd')

plot_structure = 'Pointcloud'
if plot_structure == 'Pointcloud':
    view = show_pointcloud(target_pdb, coord, embedding)
elif plot_structure == "Residues":
    view = show_structure('/home/szk/content/output/per_resi_binding.pdb')
elif plot_structure == "Atoms":
    view = show_structure('/home/szk/content/output/per_atom_binding.pdb')
```

pennylane optimization:

```
from pennylane import numpy as np

symbols = ["H", "H", "H"]
x = np.array([0.028, 0.054, 0.0, 0.986, 1.610, 0.0, 1.855, 0.002, 0.0], requires_grad=True)

# .. math::
#
#         H(x) = \sum_j h_j(x) \prod_i^{N} \sigma_i^{(j)}.
```

```
    import pennylane as qml

    def H(x):
        return qml.qchem.molecular_hamiltonian(symbols, x, charge=1)[0]

    hf = qml.qchem.hf_state(electrons=2, orbitals=6)
    print(hf)

    # The ``hf`` array is used by the :class:`~.pennylane.BasisState` operation to initialize
    # the qubit register. Then, the :class:`~.pennylane.DoubleExcitation` operations are
applied
    # First, we define the quantum device used to compute the expectation value.
    # In this example, we use the ``default.qubit`` simulator:

    num_wires = 6
    dev = qml.device("default.qubit", wires=num_wires)


    @qml.qnode(dev)
    def circuit(params, obs, wires):
        qml.BasisState(hf, wires=wires)
        qml.DoubleExcitation(params[0], wires=[0, 1, 2, 3])
        qml.DoubleExcitation(params[1], wires=[0, 1, 4, 5])

        return qml.expval(obs)

    def cost(params, x):
        hamiltonian = H(x)
        return circuit(params, obs=hamiltonian, wires=range(num_wires))


    # .. math::
    #
    #        \nabla_x g(\theta, x) = \langle \Psi(\theta) \vert \nabla_x H(x) \vert \Psi(\theta)
\rangle.

    def finite_diff(f, x, delta=0.01):
        """Compute the central-difference finite difference of a function"""
        gradient = []

        for i in range(len(x)):
            shift = np.zeros_like(x)
            shift[i] += 0.5 * delta
```

```
            res = (f(x + shift) - f(x - shift)) * delta**-1
            gradient.append(res)

    return gradient

def grad_x(params, x):
    grad_h = finite_diff(H, x)
    grad = [circuit(params, obs=obs, wires=range(num_wires)) for obs in grad_h]
    return np.array(grad)


# Optimization of the molecular geometry

opt_theta = qml.GradientDescentOptimizer(stepsize=0.4)
opt_x = qml.GradientDescentOptimizer(stepsize=0.8)

theta = np.array([0.0, 0.0], requires_grad=True)
from functools import partial

# store the values of the cost function
energy = []

# store the values of the bond length
bond_length = []

# Factor to convert from Bohrs to Angstroms
bohr_angs = 0.529177210903

for n in range(100):

    # Optimize the circuit parameters
    theta.requires_grad = True
    x.requires_grad = False
    theta, _ = opt_theta.step(cost, theta, x)

    # Optimize the nuclear coordinates
    x.requires_grad = True
    theta.requires_grad = False
    _, x = opt_x.step(cost, theta, x, grad_fn=grad_x)

    energy.append(cost(theta, x))
    bond_length.append(np.linalg.norm(x[0:3] - x[3:6]) * bohr_angs)

    if n % 4 == 0:
```

```
            print(f"Step = {n},  E = {energy[-1]:.8f} Ha,  bond length = {bond_length[-1]:.5f} A")

        # Check maximum component of the nuclear gradient
        if np.max(grad_x(theta, x)) <= 1e-05:
            break

print("\n" f"Final value of the ground-state energy = {energy[-1]:.8f} Ha")
print("\n" "Ground-state equilibrium geometry")
print("%s %4s %8s %8s" % ("symbol", "x", "y", "z"))
for i, atom in enumerate(symbols):
    print(f"  {atom}      {x[3 * i]:.4f}    {x[3 * i + 1]:.4f}    {x[3 * i + 2]:.4f}")

import matplotlib.pyplot as plt

fig = plt.figure()
fig.set_figheight(5)
fig.set_figwidth(12)

# Add energy plot on column 1
E_fci = -1.27443765658
E_vqe = np.array(energy)
ax1 = fig.add_subplot(121)
ax1.plot(range(n+1), E_vqe-E_fci, 'go-', ls='dashed')
ax1.plot(range(n+1), np.full(n+1, 0.001), color='red')
ax1.set_xlabel("Optimization step", fontsize=13)
ax1.set_ylabel("$E_{VQE} - E_{FCI}$ (Hartree)", fontsize=13)
ax1.text(5, 0.0013, r'Chemical accuracy', fontsize=13)
plt.yscale("log")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add bond length plot on column 2
d_fci = 0.986
ax2 = fig.add_subplot(122)
ax2.plot(range(n+1), bond_length, 'go-', ls='dashed')
ax2.plot(range(n+1), np.full(n+1, d_fci), color='red')
ax2.set_ylim([0.965,0.99])
ax2.set_xlabel("Optimization step", fontsize=13)
ax2.set_ylabel("bond length ($\AA$)", fontsize=13)
ax2.text(5, 0.9865, r'Equilibrium bond length', fontsize=13)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
plt.subplots_adjust(wspace=0.3)
plt.show()
```