BITS F423T First Degree Thesis

Final Report

# Machine learning based molecular dynamics simulation of micro and nano scale fluid flows

Programs and Scripts

Submitted by

Ankur Barsode 2016B5A40051P
f2016051@pilani.bits-pilani.ac.in

Under the supervision of

Dr. Aneesh A.M.

Department of Mechanical Engineering

## Contents

## 1 LAMMPS script for NVE equilibration

The LAMMPS input script for the droplet-vapor equilibration using an NVE ensemble is explained below.
Declaring variables: variables corresponding to various important quantities such as initial droplet and ambient temperatures, box and droplet sizes and droplet location are defined in this section. Log file is also saved separately for extraction of thermodynamic data.

```
variable TAr equal 103 #Argon temperature, droplet and vapor
variable radius equal 61/2 #initial solid droplet radius
variable box_size equal 222 #box size
variable drop_loc equal ${box_size}/2 #droplet location
variable DrLat equal 8.4 #lattice parameter for droplet
variable VapLat equal 26 #lattice parameter for vapor
log log.Ar.initial #saves logs in indicated file
```

Defining system characteristics: Types of units used, dimensionality of the problem (3D), and boundary conditions (periodic) are declared in this section. The units system is as follows
Distance: Angstroms
Time: Femtoseconds
Temperature: Kelvin
Energy: Kcal/mol
Pressure: atmospheres
Velocity: 1e5 m/s
Mass: g/mole

```
units real
dimension 3
boundary p p p #periodic boundaries
atom_style atomic
neighbor 2 bin
neigh_modify delay 5
```

Defining the problem geometry: First a simulation box is created. A spherical region is defined and atoms are created corresponding to the droplet using an fcc lattice. The rest of the box is filled with atoms in an hcp lattice corresponding to a more relaxed lattice spacing. The lattice spacing for the droplet is adjusted to obtain the desired initial droplet densities and is held fixed there, while the ambient lattice spacing is varied to obtain the desired number of atoms in the vapor region. Once the regions are defined, mass of the species is defined to be that of Argon atoms. Groups are also defined in this section.

```
region box block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box
create_box 1 box #1 is the total number of species. creates a cubic box.

mass 1 39.948 #Ar 39.948 u

lattice hcp ${VapLat}
region 2 block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box #vapor
create_atoms 1 region 2
```

```
lattice fcc ${DrLat}
region 1 sphere ${drop_loc} ${drop_loc} ${drop_loc} ${radius} units box #droplet
delete_atoms region 1 #remove overlap
create_atoms 1 region 1

group droplet region 1 #groups of atoms definedd here
group whole region 2 #these groups are used for calculations later
group vapor subtract whole droplet
```

Defining pairwise interactions: A 6-12 Lennard Jones potential with a cutoff is used for simulating the droplet evaporation. The LJ coefficients used are $\epsilon = 1.67e - 21$ J, $\sigma = 0.34$ nm. The cut-off radius is $2.5\sigma = 0.85$ nm.

```
pair_style lj/cut 8.5
pair_coeff 1 1 0.2404 3.4 #Ar epl 1.67e-21 J sig 0.34 nm
```

Defining various computes: In order to identify the droplet particles from the ambient, all the atoms/molecules are given a cluster ID based on an interatomic cut-off distance of $1.5\sigma$. After the simulation is complete, the cluster with the maximum number of atoms will be identified as the droplet. Computes that calculate temperatures of the two phases as well as pressure of the Ar vapor are also defined in this section.

```
compute clust all cluster/atom 5.1 #assign cluster id based on 5.1A cutoff
compute Drop_temp droplet temp
compute_modify Drop_temp dynamic yes
compute Vap_temp vapor temp
compute_modify Vap_temp dynamic yes
compute        peratom vapor stress/atom NULL
compute        p vapor reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable       Vap_press equal -(c_p[1]+c_p[2]+c_p[3])/(3*20800000)
```

Minimization: The geometry as defined by the fcc lattice positions may not be a stable initial condition to start the simulation. The forces and the resulting velocities might be physically unrealistic. Energy minimization is performed to adjust the system geometry slightly to reach the local potential energy minimum.

```
min_style cg
minimize 1.0e-6 1.0e-8 10000 10000
reset_timestep 0
```

Initializing velocities: The two phases are given initial velocities corresponding to the initial temperatures using the rules of statistical mechanics.

```
velocity droplet create ${TAr} 4652
velocity vapor create ${TAr} 4652
```

Defining output quantities: Various thermodynamic quantities such as pressures and temperatures were set to output on the screen as well as be printed in a log file for later retrieval. Particle data including particle type, coordinates, velocities and cluster ID was stored in separate files after a certain number of iterations.

```
thermo 10000
thermo_style custom step c_Drop_temp c_Vap_temp v_Vap_press press
dump 1 all custom 10000 dump.Ar.initial* type id xs ys zs vx vy vz c_clust
```

Defining ensemble properties: The droplet is initially defined as a solid FCC crystal of Argon atoms. In this first step of NVE equilibration, the crystal is allowed to melt and the temperatures of the droplet and ambient are allowed to reach equilibrium using a microcanonical ensemble (Volume, Mass, Energy held fixed). Some atoms may be lost from the droplet to the ambient during this stage. The timestep is defined to be 1 fs and the simulation is run for 5 ns.

```
fix 1 all nve
timestep 1
run 5000000
```

## 2 Python script for analysis of NVE equilibration data output by LAMMPS

Explained below is the python program for analysis of NVE Equilibration data.
The first step in a python program is to import the necessary packages and modules.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import random
```

A function reads the thermodynamic data written in the log file returned by LAMMPS. Though both temperature and pressure data is available, only the pressures are used because the temperatures calculated by LAMMPS are not representative of the two phases and a separate calculation is done using python function only.

```
def read_thermo_data_from_log(log_file_name,Data,tst):
  lines=open(log_file_name,'r')
  flag=0
  i=0
  for fp in lines:
    line=fp.split()
    if i>Data.shape[0]-1:
      break
    if len(line)<2:
      continue
    if flag==1 and line[0]=='Loop':
      flag=0
    if flag==1 and int(line[0])>=tst:
      [timestep,Tdr,Tamb,vap_press,press]=line
      Data[i,4:8]=[float(Tdr),float(Tamb),float(vap_press),float(press)]
      i+=1
    if line[0]=='Step' and line[1]=='c_Drop_temp':
      flag=1
  lines.close()
  return Data
```

The data in the LAMMPS dump files contains the coordinates and velocity components of all the atoms. Out of these, the atoms belonging to the largest cluster are the most important. These are identified by the cluster ID assigned to them by LAMMPS. Atoms which are part of the same cluster have the same cluster ID which can be any integer, while atoms which do not belong to a cluster have a cluster ID = 0. In this program, we are identifying the largest cluster using the mode of the cluster IDs, and therefore it is required that we first assign a unique cluster ID to these solitary atoms. The following function extracts the text data from the dump file into a matrix and assigns a random number between [0,1] as cluster ID to all solitary atoms, leaving the rest unchanged.

```
def get_clust_data(file_name):
  particle=open(file_name,'r')
  for i in range(3): data=particle.readline()
  Ntot=float(particle.readline().split()[0])
  for i in range(5): data=particle.readline()
  clust_data=np.zeros((int(Ntot),7))
  i=1
  while i < (Ntot+1):
    data=particle.readline()
    clust_id=float(data.split()[8])
    [xs,ys,zs,vx,vy,vz]=data.split()[2:8]
    clust_data[i-1,:]=[xs,ys,zs,vx,vy,vz,clust_id]
    if clust_id==0:
      clust_data[i-1,6]=random.random()
    i+=1
  particle.close()
  return [clust_data,Ntot]
```

This cluster data is fed to another function which calculates the number of particles in the droplet by finding the modal value of the cluster ID and its number of occurrences. It simultaneously calculates the temperatures of droplet and vapor and returns the data for the biggest cluster (the droplet) for further processing.

```
def num_droplet(clust_data):
  N=clust_data.shape[0]
  i=1
  j=0
  [biggest_clust_id,numDrop]=stats.mode(clust_data[:,6])
  biggest_clust_data=np.zeros((int(numDrop),3))
  numAmb=N-numDrop
  KEdrop=0
  KEamb=0
  mass=39.948/6.022e23/1000
  for i in range(N):
    if clust_data[i,6]==biggest_clust_id:
      KEdrop+=0.5*mass*(float(clust_data[i,3])**2+float(clust_data[i,4])**2+\
float(clust_data[i,5])**2)*1e10
      biggest_clust_data[j,:]=clust_data[i,0:3]
      j+=1
    else:
      KEamb+=0.5*mass*(float(clust_data[i,3])**2+float(clust_data[i,4])**2+\
float(clust_data[i,5])**2)*1e10
  return [numDrop,numAmb,KEdrop,KEamb,biggest_clust_data]
```

Often, at the end of a simulation, the droplet is no longer at the center of the simulation box. This is not a big issue for the NVE simulation because all the boundaries are periodic. However, in the next step of the simulation, there is a distinct far-field region close to the boundaries where properties are maintained differently. It is thus important that the droplet remain close to the center of the simulation box. This is done by translating the entire system so as to bring the center of mass of the droplet (as identified in the biggest cluster data) to the center of the simulation box, while obeying periodicity using minimum image convention. The following function recenters the configuration.

```
def recenter(file_name,Data_file_name,centroid,box_size):
  dump=open(Data_file_name+"_recentered","w")
  f=open(file_name,'r')
  metadata=f.readlines()[0:9]
  s=""
  dump.write(s.join(metadata))
  f.close()
  particle=open(file_name,'r')
  for i in range(3): data=particle.readline()
  Ntot=float(particle.readline().split()[0])
  for i in range(5): data=particle.readline()
  clust_data=np.zeros((int(Ntot),7))
  i=1
  while i < (Ntot+1):
    data=particle.readline()
    [type,id,xs,ys,zs,vx,vy,vz,clust_id]=data.split()
    [xs,ys,zs]=[float(xs)-centroid[0]+0.5-np.floor(float(xs)-centroid[0]+0.5),\
float(ys)-centroid[1]+0.5-np.floor(float(ys)-centroid[1]+0.5),float(zs)-centroid[2]\
+0.5-np.floor(float(zs)-centroid[2]+0.5)]
    line=[str(1),id,str(xs),str(ys),str(zs),vx,vy,vz,clust_id,"\n"]
    s=" "
    dump.write(s.join(line))
    i+=1
  particle.close()
  dump.close()
```

After all calculations are done, plots are generated to show the number of atoms in each phase, temperatures and pressures.

```
def show_plots(Data):
  p1=plt.plot(Data[:,0],Data[:,1])
  p2=plt.plot(Data[:,0],Data[:,2])
  plt.title('Number of Droplet Particles')
  plt.legend((p1[0],p2[0]),('Droplet','Ambient'))
  plt.xlabel('timestep')
  plt.show()

  p1=plt.plot(Data[:,0],Data[:,3])
  p2=plt.plot(Data[:,0],Data[:,8])
  plt.title('Temperature')
  plt.legend((p1[0],p2[0]),('Droplet','Ambient'))
  plt.ylabel('K')
  plt.xlabel('timestep')
  plt.show()

  p1=plt.plot(Data[:,0],Data[:,6])
  p2=plt.plot(Data[:,0],Data[:,7])
  plt.title('Pressure')
  plt.legend((p1[0],p2[0]),('Initial Vapor','Total'))
  plt.ylabel('atm')
  plt.xlabel('timestep')
  plt.show()
```

A separate function connects all the above fragments into a complete analysis script.

```
def run():
  [TS,tst,dt,ten]=[0.001,0,10000,5000000]
  dump_file_name="./dump.Ar.NVE"
  log_file_name='log.Ar.NVE'
  Data_file_name="NVE_data.txt"
  lenData=int((ten-tst)/dt+1)
  Data=np.zeros((lenData,9))
  file_name=[]
  for i in range(lenData):
    file_name=dump_file_name+str(tst+i*dt)
    [clust_data,Ntot]=get_clust_data(file_name)
    [NDrop,NAmb,KEdrop,KEamb,biggest_clust_data]=num_droplet(clust_data)
    Tdrop=KEdrop*2/3/(NDrop+1e-6)/1.381e-23
    Tamb=KEamb*2/3/(NAmb+1e-6)/1.381e-23
    print([(tst+i*dt)*TS,NDrop,NAmb,Tdrop])
    Data[i,0:4]=[(tst+i*dt)*TS,NDrop,NAmb,Tdrop]
    centroid=np.mean(biggest_clust_data,axis=0)
    Data[i,8]=Tamb
  print("Cluster data parsed from "+dump_file_name+"*\n")
  recenter(file_name,Data_file_name,centroid,box_size=222)
  print("Droplet recentered. New configuration saved to "+Data_file_name+"_recenterd")
  read_thermo_data_from_log(log_file_name,Data,tst)
  print("Thermo data parsed from "+log_file_name+"\n")
  np.savetxt(Data_file_name,Data)
  #Data format:[timestep,NDrop,NCVap,Tdrop,Drop_Temp,Vap_Temp,Vap_press,press,Tamb]
  print("Data written to file "+Data_file_name+"\n")
  show_plots(Data)

run()
```

## 3 LAMMPS script for droplet evaporation

Declaring variables: variables corresponding to various important quantities such as initial droplet and ambient temperatures, box and droplet sizes and droplet location are defined in this section. Log file is also saved separately for extraction of thermodynamic data.

```
variable Tinf equal 600
variable Pinf equal 7.6 #MPa
variable box_size equal 139
variable drop_loc equal ${box_size}/2
variable far_field_dist equal 61.2
variable Ninf equal floor(1.7255e6*${Pinf}/${Tinf}*1e-30/1.381e-23*1e6)
variable rhoInf equal ${Ninf}*39.948/6.022e23/1.7255e-24/1000
log log.Ar.evap.Tinf_${Tinf}_Pinf_${Pinf}
```

Defining system characteristics: Types of units used, dimensionality of the problem(3D), and boundary conditions(periodic) are declared in this section. The units system is given as follows
Distance: Angstroms
Time: Femtoseconds
Temperature: Kelvin
Energy: Kcal/mol
Pressure: atmospheres
Velocity: 1e5 m/s
Mass: g/mole

```
units real
dimension 3
boundary p p p #periodic boundaries
atom_style atomic
neighbor 2 bin
neigh_modify delay 5
```

Defining the problem geometry: First a simulation box is created. Mass of the species (Argon) is defined. This is followed by loading the configuration of the previous NVE equilibration simulation. A spherical region is defined and the intersection of its out-side with the simulation box is defined as the far-field region. Dynamic groups corresponding to the near-field and far-field regions are also defined in this section.

```
region box block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box
create_box 1 box

mass 1 39.948 #Ar 39.948

read_dump dump.Ar.initial5000000_recentered 5000000 x y z vx vy vz add yes scaled &
yes
reset_timestep 0

region 1 block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box side in
region 2 sphere ${drop_loc} ${drop_loc} ${drop_loc} ${far_field_dist} units box &
side out
```

```
region 3 intersect 2 1 2

group near_field dynamic all region 2 every 100
group far_field dynamic all region 3 every 100
```

Defining pairwise interactions: A 6-12 Lennard Jones potential with a cutoff is used for simulating the droplet evaporation. The LJ coefficients used are $\epsilon = 1.67e - 21$ J, $\sigma = 0.34$ nm. The cut-off radius is $2.5\sigma = 0.85$ nm.

```
pair_style lj/cut 8.5
pair_coeff 1 1 0.2404 3.4 #Ar epl 1.67e-21 J sig 0.34 nm
```

Defining various computes: In order to identify the droplet particles from the ambient, all the atoms/molecules are given a cluster ID based on an interatomic cut-off distance of $1.5\sigma$. A coordination number is also calculated for each atom. After the simulation is complete, the cluster with the maximum number of atoms, where each atom has a coordination number ¿ 4 will be identified as the droplet. Computes that calculate temperatures of the two phases as well as pressure of the Ar vapor are also defined in this section.
For the purpose of maintaining the density, the number of particles in the far-field region need to be calculated and held fixed. That calculation is also performed in this section.

```
compute clust all cluster/atom 5.1
compute CN all coord/atom cutoff 5.1
compute far_temp far_field temp
compute_modify far_temp dynamic yes
compute near_temp near_field temp
compute_modify near_temp dynamic yes
compute        peratom far_field stress/atom NULL
compute        p far_field reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable       far_press equal -(c_p[1]+c_p[2]+c_p[3])/(3*1.7255e6)
compute 10 all chunk/atom bin/sphere ${drop_loc} ${drop_loc} ${drop_loc} 2.023 &
120.37 2
compute 11 far_field property/chunk 10 count
variable far_excess_frac equal c_11[2]/${Ninf}-1
variable far_excess equal ${Ninf}-c_11[2]
compute 12 all chunk/atom bin/sphere ${drop_loc} ${drop_loc} ${drop_loc} 0 &
${drop_loc} 20
```

Minimization and velocity initialization is not required as the final state from the NVE equilibration is physically realistic.
Proceeding to defining the ensemble properties, a direct velocity rescaling is used to maintain the temperature in the far-field region. NVE ensemble is also used, however the mass and total energy and momentum are not conserved due to random deletions and creations in the far-field region.

```
fix 1 far_field temp/rescale 2 ${Tinf} ${Tinf} 0.1 1.0
fix 3 all nve
timestep 1.07
```

Defining output quantities: Various thermodynamic quantities such as pressures and temperatures were set to output on the screen as well as be printed in a log file for later retrieval. Particle data including particle type, coordinates, velocities and cluster ID was stored in separate files after a certain number of iterations.

```
thermo 500
thermo_style custom step c_far_temp c_near_temp v_far_press press c_11[2] &
v_far_excess_frac
dump 1 all custom 500 dump.Ar.evap* id xs ys zs vx vy vz c_clust c_CN c_12
```

Evaporation: Running the simulation while holding the pressure in the far-field region constant required that the number of particles be held fixed. This is achieved by randomly inserting or deleting the appropriate number of atoms required to maintain the far-field density within ±1% of the desired value. This is done every 5 timesteps by the use of loop and if-else control structures in LAMMPS.

```
variable a loop 30000
label loopa
run 0
variable s equal round(random(1,1000000,1000))
if "${far_excess_frac} >= 0.01" then &
 "delete_atoms porosity 3 ${far_excess_frac} 12354" &
elif ${far_excess_frac}<=-0.01 &
"group old_far_field region 3" &
"create_atoms 1 random ${far_excess} ${s} 3" &
"group new_far_field region 3" &
"group new_atoms subtract new_far_field old_far_field" &
"velocity new_atoms create ${Tinf} ${s} dist gaussian loop local" &
"delete_atoms overlap 3.808 far_field far_field" &
"group old_far_field delete" "group new_atoms delete" "group new_far_field delete"
thermo_modify lost warn
run 5
next a
jump SELF loopa
```

## 4 Python script for analysis of droplet evaporation data output by LAMMPS

The analysis program is explained below.
The first step is to import the necessary packages and modules.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import random
from astropy.convolution import convolve, Box1DKernel
```

The data in the LAMMPS dump files contains the coordinates and velocity components of all the atoms. Out of these, the atoms belonging to the largest cluster are the most important. These are identified by the cluster ID assigned to them by LAMMPS. Atoms which are part of the same cluster have the same cluster ID which can be any integer, while atoms which do not belong to a cluster have a cluster ID = 0. In this program, we are identifying the largest cluster using the mode of the cluster IDs, and therefore it is required that we first assign a unique cluster ID to these solitary atoms. The following function extracts the text data from the dump file into a matrix and assigns a random number between [0,1] as cluster ID to all solitary atoms, leaving the rest unchanged.

```
def get_data(file_name):
  particle=open(file_name,'r')
  for i in range(3): data=particle.readline()
  Ntot=float(particle.readline().split()[0])
  for i in range(5): data=particle.readline()
  particle_data=np.zeros((int(Ntot),9))
  i=1
  while i < (Ntot+1):
    data=particle.readline()
    [xs,ys,zs,vx,vy,vz,clust_id,CN,bin_id]=data.split()[1:10]
    particle_data[i-1,:]=[float(xs),float(ys),float(zs),float(vx),float(vy),\
float(vz),float(clust_id),float(CN),float(bin_id)]
    if clust_id==0:
      particle_data[i-1,6]=random.random()
    i+=1
  particle.close()
  return particle_data
```

This cluster data is fed to another function which calculates the number of particles in the droplet by finding the modal value of the cluster ID and its number of occurrences. It also calculates the temperatures of droplet and vapor in radial bins centered at the simulation box. The spacial binning is done by LAMMPS itself and every atom is assigned a bin ID, which is used in this function for calculations. The function passes the droplet data to a separate function that returns the droplet radius.

```
def rad_bin_calcs(particle_data):
  [biggest_clust_id,numDrop_Crown]=stats.mode(particle_data[:,6])
  vcm=np.mean(particle_data[:,3:6],axis=0)
  particle_data[:,3:6]-=vcm
  bin_KE=np.zeros((20))
  bin_n=np.zeros((20))
  mass=39.948/6.022e23/1000
  numDrop=0
  radDrop=0
  rho_mean=0
  Tcore=0
  Tsurf=0
  droplet_data=np.zeros((particle_data.shape[0],6))
  for i in range(particle_data.shape[0]):
    bin_KE[int(particle_data[i,8])-1]+=0.5*mass*(float(particle_data[i,3])**2+\
float(particle_data[i,4])**2+float(particle_data[i,5])**2)*1e10
    bin_n[int(particle_data[i,8])-1]+=1
    if particle_data[i,7]>3 and particle_data[i,6]==biggest_clust_id:
      numDrop+=1
      droplet_data[numDrop-1,:]=particle_data[i,:6]
  droplet_data=droplet_data[:numDrop,:]
  if numDrop>5:
    [radDrop,rho_mean]=rad_drop(droplet_data)
    [Tsurf,Tcore]=surface_temp(droplet_data,radDrop)
  bin_T=2/3*bin_KE/(bin_n+1e-6)/1.381e-23
  bin_T*=(1-np.sign(np.floor(bin_T/1000)))
  return [numDrop,radDrop*222,rho_mean/222e-10**3*mass,Tsurf,Tcore,bin_T]
```

The following function uses radial histogram based radius estimate, while also calculating the mean density of the droplet.

```
def rad_drop(droplet_data):
  numDrop=droplet_data.shape[0]
  centroid=np.mean(droplet_data[:,:3],axis=0)
  rad_dist=np.linalg.norm(droplet_data[:,:3]-centroid,axis=1)
  rad_dist_sorted=np.sort(rad_dist)
  rho=(1+np.arange(numDrop))/rad_dist_sorted**3
  rho_mean=np.mean(rho[np.minimum(20,int(numDrop*2/3-1)):int(numDrop*2/3)])
  rho_thresh=0.9*rho_mean
  for i in range(numDrop):
    if rho[-i-1]>rho_thresh:
      radDrop=rad_dist[-i-1]
      break
  y,x,_=plt.hist(rad_dist_sorted,bins=30)
  radDrop=x[np.where(y == y.max())].max()
  plt.clf()
  return [radDrop,rho_mean]
```

With the droplet data and the droplet radius known, the surface temperature of the droplet can be calculated with the following function. Surface of the droplet is defined to be the region between half to full radius.

```
def surface_temp(droplet_data,radDrop):
  numDrop=droplet_data.shape[0]
  centroid=np.mean(droplet_data[:,:3],axis=0)
  KE_surf=0
  KE_core=0
  numSurf=0
  mass=39.948/6.022e23/1000
  for i in range(numDrop):
    rad_dist=np.linalg.norm(droplet_data[i,:3]-centroid)
    if rad_dist>radDrop/2:
      numSurf+=1
      KE_surf+=0.5*mass*(float(droplet_data[i,3])**2+float(droplet_data[i,4])**2\
+float(droplet_data[i,5])**2)*1e10
    else:
      KE_core+=0.5*mass*(float(droplet_data[i,3])**2+float(droplet_data[i,4])**2\
+float(droplet_data[i,5])**2)*1e10
  Tsurf=2/3*KE_surf/(numSurf+1e-6)/1.381e-23
  Tcore=2/3*KE_core/(numDrop-numSurf+1e-6)/1.381e-23
  return [Tsurf,Tcore]
```

After all calculations are done, plots are generated to show the number of atoms in each phase, temperatures and pressures.

```
def show_plots(Data):
  plt.plot(Data[:,0]*0.00107,Data[:,1])
  plt.title('Number of Droplet Particles')
  plt.xlabel('time (ps)')
  plt.show()

  plt.plot(Data[:,0]*0.00107,Data[:,1]**0.6666)
  plt.title('N^(2/3)')
  plt.xlabel('time (ps)')
  plt.show()

  plt.plot(Data[:,0]*0.00107,2*Data[:,2],'.')
  plt.plot(Data[:,0]*0.00107,convolve(2*Data[:,2],Box1DKernel(15)))
  plt.title('Droplet Diameter')
  plt.xlabel('time (ps)')
  plt.ylabel('Angstroms')
  plt.show()

  plt.plot(Data[:,0]*0.00107,4*Data[:,2]**2,'.')
  plt.plot(Data[:,0]*0.00107,convolve(4*Data[:,2]**2,Box1DKernel(15)))
  plt.title('Droplet Diameter squared, D2')
  plt.xlabel('time (ps)')
  plt.ylabel('Angstroms squared')
  plt.show()

  plt.plot(Data[:,0]*0.00107,Data[:,3],'.')
  plt.plot(Data[:,0]*0.00107,convolve(Data[:,3],Box1DKernel(15)))
  plt.title('Droplet mean density')
  plt.ylabel('kg/m3')
  plt.xlabel('time (ps)')
  plt.show()

  plt.plot(Data[:,0]*0.00107,Data[:,4],'.')
  plt.plot(Data[:,0]*0.00107,convolve(Data[:,4],Box1DKernel(15)))
  plt.title('Droplet Surface Temperature')
  plt.ylabel('K')
  plt.xlabel('time (ps)')
  plt.show()

  plt.plot(Data[:,0]*0.00107,-Data[:,5]+Data[:,4],'.')
  plt.plot(Data[:,0]*0.00107,convolve(-Data[:,5]+Data[:,4],Box1DKernel(15)))
  plt.title('Droplet (surface-Core) Temperature')
  plt.ylabel('K')
  plt.xlabel('time (ps)')
  plt.show()

  x=np.zeros((20,1))
  x=(np.arange(20)+1)*222/40
  y=np.zeros((20,5))#int(Data.shape[0]/2)))
  for i in range(20):#range(Data.shape[0]):
    if i%4==3:
      y[:,int(i/4-0.75)]=Data[i,6:].T/4+Data[i-1,6:].T/4+Data[i-2,6:].T/4+Data[i-3,6:].T/4
  plt.plot(x,y)
  plt.title('Bin-wise Temperature')
  plt.ylabel('K')
  plt.xlabel('radial bin')
  plt.show()
```

A separate function connects all the above fragments into a complete analysis script.

```
def run():
  [TS,tst,dt,ten]=[0.00107,0,1000,300000]
  dump_file_name="./dump.Ar.evap"
  log_file_name='log.Ar.evap.farther.Tinf_500_RhoInf_33.7_4711'
  Data_file_name=log_file_name+'_sph_bin_temps_4711.txt'
  lenData=int((ten-tst)/dt+1)
  Data=np.zeros((lenData,26))
  file_name=[]
  for i in range(lenData):
    file_name=dump_file_name+str(tst+i*dt)
    particle_data=get_data(file_name)
    [numDrop,radDrop,rho_mean,Tsurf,Tcore,bin_T]=rad_bin_calcs(particle_data)
    print([(tst+i*dt)*TS,numDrop])
    Data[i,0:6]=[(tst+i*dt)*TS,numDrop,radDrop,rho_mean,Tsurf,Tcore]
    Data[i,6:]=bin_T
  print("Cluster data parsed from "+dump_file_name+"*\n")
  np.savetxt(Data_file_name,Data)
  #Data format:[timestep,numDrop,radDrop,Tsurf,bin_T]
  print("Data written to file "+Data_file_name+"\n")
  show_plots(Data)
run()
```

## 5 Python script for converting MD data to ML input format

The following python program is used to convert the LAMMPS output into images of dimension (64,64,6).
The first step is to import the necessary packages and modules.

```
import numpy as np
import matplotlib.pyplot as plt
```

A function reads the LAMMPS dump and extracts the coordinates and velocity components in different matrices of shape (px,py). In this case, px=py=64. The extraction is done such that each 2D matrix element (i,j) in all 6 matrices stores information regarding the same particle. At the end, the 6 matries corresponding to (x,y,z,vx,vy,vz) are concatenated into a single 3D matrix of shape (px,py,6) and stored as the input for ML.

```
def MD2ML(file_name,px,py,display=False,new_name='single_trial'):
  itype=np.zeros((px,py))
  id=np.zeros((px,py))
  ix=np.zeros((px,py))
  iy=np.zeros((px,py))
  iz=np.zeros((px,py))
  ivx=np.zeros((px,py))
  ivy=np.zeros((px,py))
  ivz=np.zeros((px,py))
  particle=open(file_name,'r')
  for i in range(3): data=particle.readline()
  N=float(particle.readline().split()[0])
  for i in range(5): data=particle.readline()
  i=1
  while i < (N+1):
    data=particle.readline()
    ptype=float(data.split()[0])
    ID=float(data.split()[1])
    xs=float(data.split()[2])
    ys=float(data.split()[3])
    zs=float(data.split()[4])
    vx=float(data.split()[5])
    vy=float(data.split()[6])
    vz=float(data.split()[7])
    itype[int((i-1)/py),(i-1)%py]=ptype
    id[int((i-1)/py),(i-1)%py]=ID
    ix[int((i-1)/py),(i-1)%py]=xs
    iy[int((i-1)/py),(i-1)%py]=ys
    iz[int((i-1)/py),(i-1)%py]=zs
    ivx[int((i-1)/py),(i-1)%py]=vx
    ivy[int((i-1)/py),(i-1)%py]=vy
    ivz[int((i-1)/py),(i-1)%py]=vz
    i+=1
  if display:
    display_images(itype,id,ix,iy,iz,ivx,ivy,ivz)
  config=np.stack((ix,iy,iz,ivx,ivy,ivz),axis=2)
  np.save(new_name+'.MD2ML',config)
  particle.close()
```

The generated images can be displayed if desired.

```
def display_images(itype,id,ix,iy,iz,ivx,ivy,ivz):
  #plt.imshow(itype)
  #plt.title('Atom type')
  #plt.show()
  #plt.imshow(id)
  #plt.title('Atom ID')
  #plt.show()
  plt.imshow(ix)
  plt.title('x coordinate')
  plt.show()
  plt.imshow(iy)
  plt.title('y coordinate')
  plt.show()
  plt.imshow(iz)
  plt.title('z coordinate')
  plt.show()
  plt.imshow(ivx)
  plt.title('x velocity')
  plt.show()
  plt.imshow(ivy)
  plt.title('y velocity')
  plt.show()
  plt.imshow(ivz)
  plt.title('z velocity')
  plt.show()
```

A separate function is used to iterate over the dump files and feed the file names to the converter function above, with a choice of running it in single or batch mode.

```
def run_single():
  file_name='./dump.Ar.initial10000'
  px=64
  py=64
  MD2ML(file_name,px,py,True)

def run_batch():
  [tst,dts,tse]=[0,1e4,5e6]
  for i in range(int((tse-tst)/dts)+1):
    file_name='./dump.Ar.initial'+str(int(tst+i*dts))
    new_name='./npy_dataset/NVE_Z/NVE_Z_'+str(int(i))
    px=64
    py=64
    MD2ML(file_name,px,py,False,new_name)

run_batch()
#run_single()
```

## 6 Python and LAMMPS scripts for analysing ML predictions

The structure of the python program for conversion of images to per-atom properties table is given below.
First the Numpy module is imported for handling matrices. The output of ML algorithm is a matrix of shape (64,64,6), containing the position and velocity components of up to 4096 atoms. First this matrix is loaded and broken into the 6 different matrices. These can now be converted back to the particle-wise table format of LAMMPS. The function is the reverse of the MD2ML function of the previous section.

```
import numpy as np

def read_npy(file_name):
  f=file_name+'_ML2pred.npy'
  config=np.load(f)
  #itype=config[:,:,0]
  #id=config[:,:,1]
  ix=config[:,:,0]
  iy=config[:,:,1]
  iz=config[:,:,2]
  ivx=config[:,:,3]
  ivy=config[:,:,4]
  ivz=config[:,:,5]
  return [ix,iy,iz,ivx,ivy,ivz]

def ML2MD(file_name,dump,N):
  [ix,iy,iz,ivx,ivy,ivz]=read_npy(file_name)
  [px,py]=ix.shape
  i=1
  for j in range(px):
    for k in range(py):
      ptype=1   #int(itype[j,k])
      ID=k+1+py*j #int(id[j,k])
      xs=ix[j,k]
      ys=iy[j,k]
      zs=iz[j,k]
      vx=ivx[j,k]
      vy=ivy[j,k]
      vz=ivz[j,k]
      dump.write(str(ptype)+' '+str(ID)+' '+str(xs)+' '+str(ys)+' '+str(zs)+' '+\
str(vx)+' '+str(vy)+' '+str(vz)+'\n')
      if(i>=N):
        break
      i+=1
    if(i>=N):
      break
```

Some metadata is required for LAMMPS to be able to read the file and load the atom coordinates and velocity components. It is written separately.

```
def get_metadata(file_name,dump,k):
  #f=open(file_name,'r')
  #metadata=f.readlines()[0:9]
  #N=float(metadata[3].split()[0])
  #s=""
  #dump.write(s.join(metadata))
  metadata="ITEM: TIMESTEP\n"+str(k)+"\nITEM: NUMBER OF ATOMS\n3725\nITEM: BOX BOUNDS"+\
" pp pp pp\n0.0000000000000000e+00 2.2200000000000000e+02\n0.0000000000000000e+00"+\
" 2.2200000000000000e+02\n0.0000000000000000e+00 2.2200000000000000e+02\nITEM:"+\
" ATOMS type id xs ys zs vx vy vz\n"
  dump.write(metadata)
  #f.close()
  N=3725
  return N
```

A separate function is used to iterate over the dump files and feed the file names to the converter function above, with a choice of running it in single or batch mode.

```
def run_single():
  file_name='./dump.Ar.initial5000000'
  dump_name=file_name+'.MLdump'
  dump=open(dump_name,'w')
  N=get_metadata(file_name,dump,5000000)
  ML2MD(file_name,dump,N)

def run_batch():
  for i in range(1,501):
    file_name='./downloads/NVE_A_'+str(int(i))+'.MD2ML.npy'
    dump_name=file_name+'.ML2dump'
    dump=open(dump_name,'w')
    N=get_metadata(file_name,dump,int(i*10000))
    ML2MD(file_name,dump,N)
    dump.close()

#run_single()
run_batch()
```

The LAMMPS script for performing the rerun is explained below.
Declaring variables: variables corresponding to various important quantities such as initial droplet and ambient temperatures, box and droplet sizes and droplet location are defined in this section. Log file is also saved separately for extraction of thermodynamic data.

```
variable TAr equal 140
variable radius equal 78/2
variable box_size equal 222
variable drop_loc equal ${box_size}/2
variable DrLat equal 8.4
variable VapLat equal 20
log log.Ar.rerun_ML2
```

Defining system characteristics: Types of units used, dimensionality of the problem(3D), and boundary conditions(periodic) are declared in this section. The units system is given as follows
Distance: Angstroms
Time: Femtoseconds
Temperature: Kelvin
Energy: Kcal/mol
Pressure: atmospheres
Velocity: 1e5 m/s
Mass: g/mole

```
units real
dimension 3
boundary p p p #periodic boundaries
atom_style atomic
neighbor 2 bin
neigh_modify delay 5
```

Defining the problem geometry: The problem geometry is defined in the exact same fashion as that of the original simulation.

```
region box block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box
create_box 1 box

mass 1 39.948 #Ar 39.948

region 2 block 0 ${box_size} 0 ${box_size} 0 ${box_size} units box
region 1 sphere ${drop_loc} ${drop_loc} ${drop_loc} ${radius} units box
region 3 sphere ${drop_loc} ${drop_loc} ${drop_loc} ${radius} units box side out
region 4 intersect 2 2 3

group droplet dynamic all region 1 every 10000
group whole dynamic all region 2 every 10000
group vapor dynamic all region 4 every 10000
```

Defining pairwise interactions: A 6-12 Lennard Jones potential with a cutoff is used for simulating the droplet evaporation. The LJ coefficients used are $\epsilon = 1.67e-21$ J, $\sigma = 0.34$ nm. The cut-off radius is $2.5\sigma = 0.85$ nm.

```
pair_style lj/cut 8.5
pair_coeff 1 1 0.2404 3.4 #Ar epl 1.67e-21 J sig 0.34 nm
```

Defining various computes: In order to identify the droplet particles from the ambient, all the atoms/molecules are given a cluster ID based on an interatomic cut-off distance of $1.5\sigma$. A coordination number is also calculated for each atom. After the simulation is complete, the cluster with the maximum number of atoms, where each atom has a coordination number ¿ 4 will be identified as the droplet. Computes that calculate temperatures of the two phases as well as pressure of the Ar vapor are also defined in this section.

```
compute clust all cluster/atom 5.1
compute CN all coord/atom cutoff 5.1
compute Drop_temp droplet temp
compute_modify Drop_temp dynamic yes
compute Vap_temp vapor temp
compute_modify Vap_temp dynamic yes
compute        peratom vapor stress/atom NULL
compute        p vapor reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable       Vap_press equal -(c_p[1]+c_p[2]+c_p[3])/(3*222*222*222)
```

Defining output quantities: Various thermodynamic quantities such as pressures and temperatures were set to output on the screen as well as be printed in a log file for later retrieval.

```
thermo 0
thermo_style custom step c_Drop_temp c_Vap_temp v_Vap_press press
```

The rerun is performed by looping over all files.

```
variable a loop 500
label loopa
#run 0
variable s equal round((${a}))
dump 1 all custom 10000 NVE_A_ML2_with_CN${s} type id xs ys zs vx vy vz c_clust c_CN

rerun ./downloads/NVE_A_${s}.MD2ML.npy.ML2dump every 10000 dump x y z vx vy vz add \
yes scaled yes

run 0
undump 1

next a
jump SELF loopa
```

# 7 Keras program for Pix2Pix-GAN model

The Keras implementation is explained below.
The first step is to import the necessary packages and modules.

```
import datetime
from matplotlib.pyplot import plot,show,draw,figure,imshow,clf,ion,pause
import numpy as np
import random
import keras.backend as K
from keras.optimizers import Adam
from keras.initializers import RandomNormal
from keras.models import Model,Input,load_model
from keras.layers import Conv2D,Conv2DTranspose,LeakyReLU,Activation,Concatenate
from keras.layers import Dropout,BatchNormalization,Dense,Reshape,Flatten,Subtract
from keras.layers import LeakyReLU,SeparableConv2D,Multiply,Add,Maximum,Lambda
from keras.utils.vis_utils import plot_model
```

Training and testing datasets are created randomly out of the entire sample of input-output pairs using a separate function that creates a list of file names with required timesteps spacing between them.

```
open('/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_train_set.txt','w')\
.close()
open('/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_train_sample_set.txt'\
,'w').close()
open('/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_test_set.txt','w')\
.close()
open('/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_complete_dataset.txt'\
,'w').close()
def create_sim_dataset(dump_name,root_name,tst,dts,tse):
  pair=open(root_name+'_sim_complete_dataset.txt','a')
  train=open(root_name+'_sim_train_set.txt','a')
  train_sample=open(root_name+'_sim_train_sample_set.txt','a')
  test=open(root_name+'_sim_test_set.txt','a')
  for i in range(int(tse-dts-tst)):
    x_file_name=dump_name+str(int(tst+i))+'.MD2ML.npy'
    y_file_name=dump_name+str(int(tst+i+dts))+'.MD2ML.npy'
    delts=int(dts)
    line=x_file_name+","+str(delts)+","+y_file_name+"\n"
    pair.write(line)
    p=random.random()
    if p<=0.95*1901/(2001-dts):
      train.write(line)
      if p<=0.05*0.95*1901/(2001-dts):
        train_sample.write(line)
    elif p<=1901/(2001-dts):
      test.write(line)
    else:
      continue
  pair.close()
  train.close()
  test.close()

for i in range(6):
  dump_name='/content/drive/My Drive/npy_dataset/NVE_'+chr(ord('A')+i)+'_'
  root_name='/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs'
  create_sim_dataset(dump_name,root_name,0,1,2000)
  #second last number = time_interval(fs)/100
```

Some helper functions are required for handling the dataset, visualizing examples, and saving the inputs and outputs. The data is normalized to lie between [0,1] before feeding it to the GAN and the predictions need to be rescaled back to the correct ranges.

```
#Required global variables
k_shape=(4,4) #kernel shape for all convolutions
patch_shape=(4) #output shape of PatchGAN
pos_norm=[0,1] #normalization parameters for position
vel_norm=np.array([-1,2])*5e-2 #normalization parameters for velocities
norm_data=np.array([pos_norm,pos_norm,pos_norm,vel_norm,vel_norm,vel_norm])

# load image data. dataset is a list containing in_image,dt,out_image addresses.
def load_dataset(file_name="/train_set.txt"):
  f = open(file_name,"r")
  datasetTrans = f.read().split("\n")
  datasetTrans = [line.split(",") for line in datasetTrans]
  f.close()
  trainA,dt,trainB=[0]*(len(datasetTrans)-1),[0]*(len\
(datasetTrans)-1),[0]*(len(datasetTrans)-1)
  for i in range(len(datasetTrans)-1):
    trainA[i] = datasetTrans[i][0]
    dt[i] = datasetTrans[i][1]
    trainB[i] = datasetTrans[i][2]
  dataset=[trainA,dt,trainB]
  return dataset

# select a batch of random samples, returns images and target
def generate_real_samples(dataset, n_samples,image_shape):
  # unpack dataset
  trainA,dt,trainB = dataset
  # choose random instances
  ix = random.sample(range(len(trainA)),n_samples)
  # retrieve selected images
  [px,py,nChan]=image_shape
  X1,X2=np.zeros((n_samples,px,py,nChan)),np.zeros((n_samples,px,py,nChan))
  j=0
  for i in ix:
    a,b=np.load(trainA[i]),np.load(trainB[i])
    for k in range(nChan):
      X1[j,:,:,k]=(a[:,:,k]-norm_data[k,0])/norm_data[k,1]
      X2[j,:,:,k]=(b[:,:,k]-norm_data[k,0])/norm_data[k,1]
    j+=1
  # generate 'real' class labels (1)
  y = np.ones((n_samples, patch_shape, patch_shape, 1))
  return [X1, X2], y

def generate_test_samples(dataset,image_shape,single=False):
  # unpack dataset
  trainA,dt,trainB = dataset
  if single:
    n_samples=1
    file_names=trainB[0]
  else:
    n_samples=len(trainA)
    file_names=trainB
  ix = range(n_samples)
  # retrieve selected images
  [px,py,nChan]=image_shape
  X1,X2=np.zeros((n_samples,px,py,nChan)),np.zeros((n_samples,px,py,nChan))
  a=np.zeros((px,py,nChan))
  for i in ix:
    a,b=np.load(trainA[i]),np.load(trainB[i])
    for k in range(nChan):
      X1[i,:,:,k]=(a[:,:,k]-norm_data[k,0])/norm_data[k,1]
      X2[i,:,:,k]=(b[:,:,k]-norm_data[k,0])/norm_data[k,1]
  # generate 'real' class labels (1)
  y = np.ones((n_samples, patch_shape, patch_shape, 1))
  return [X1, X2], y,file_names

def rescale_predictions(X):
  n_pred=X.shape[0]
  [px,py,nChan]=X.shape[1:]
  for i in range(n_pred):
    for k in range(nChan):
      X[i,:,:,k]=X[i,:,:,k]*norm_data[k,1]+norm_data[k,0]
  return X

def save_predictions(X_predB,file_names):
  n_pred=X_predB.shape[0]
  pred=np.zeros((X_predB.shape[1],X_predB.shape[2],X_predB.shape[3]))
  for i in range(n_pred):
    pred=X_predB[i,:,:,:]
    file_name="/".join(file_names[i].split("/")[0:4])+"/MLpred/"+file_names[i]\
.split("/")[5]
    np.save(file_name+"_MLpred",pred)
```

```
def show_predictions(X,k=0):
  clf()
  for i in range(6):
    figure(i+1+6*k)
    imshow(X[0,:,:,i])
    ion()
    show()
    pause(0.001)
```

The PatchGAN discriminator applies multiple blocks of 2D convolution, conditional batch normalization and Leaky ReLu activation on an input of concatenated real and generated target image. Most of the parameters of the discriminator are kept frozen.

```
# define the discriminator model
def define_discriminator(image_shape):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # source image input
  in_src_image = Input(shape=image_shape)
  # target image input
  in_target_image = Input(shape=image_shape)
  # concatenate images channel-wise
  merged = Concatenate()([in_src_image, in_target_image])
  # C64
  d = Conv2D(64,k_shape, strides=(2,2), padding='same', kernel_initializer=init,\
trainable=False)(merged)
  d = LeakyReLU(alpha=0.2)(d)
  # C128
  d = Conv2D(128, k_shape, strides=(2,2), padding='same', kernel_initializer=init,\
trainable=False)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # C256
  d = Conv2D(256, k_shape, strides=(2,2), padding='same', kernel_initializer=init,\
trainable=False)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # C512
  d = Conv2D(512, k_shape, strides=(2,2), padding='same', kernel_initializer=init,\
trainable=False)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # second last output layer
  d = Conv2D(128, k_shape, padding='same', kernel_initializer=init,trainable=False)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # patch output
  d = Conv2D(1, k_shape, padding='same', kernel_initializer=init,trainable=False)(d)
  patch_out = Activation('sigmoid')(d)
  # define model
  model = Model([in_src_image, in_target_image], patch_out)
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
  return model
```

Discriminator model summary:

```
Model: "functional_13"
_____________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
=============================================================================================
input_9 (InputLayer)            [(None, 64, 64, 6)]  0
_____________________________________________________________________________________________
input_10 (InputLayer)           [(None, 64, 64, 6)]  0
_____________________________________________________________________________________________
concatenate_18 (Concatenate)    (None, 64, 64, 12)   0           input_9[0][0]
                                                                 input_10[0][0]
_____________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 32, 32, 64)   12352       concatenate_18[0][0]
_____________________________________________________________________________________________
leaky_re_lu_26 (LeakyReLU)      (None, 32, 32, 64)   0           conv2d_30[0][0]
_____________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 16, 16, 128)  131200      leaky_re_lu_26[0][0]
_____________________________________________________________________________________________
batch_normalization_38 (BatchNo (None, 16, 16, 128)  512         conv2d_31[0][0]
_____________________________________________________________________________________________
leaky_re_lu_27 (LeakyReLU)      (None, 16, 16, 128)  0           batch_normalization_38[0][0]
_____________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 8, 8, 256)    524544      leaky_re_lu_27[0][0]
_____________________________________________________________________________________________
batch_normalization_39 (BatchNo (None, 8, 8, 256)    1024        conv2d_32[0][0]
_____________________________________________________________________________________________
leaky_re_lu_28 (LeakyReLU)      (None, 8, 8, 256)    0           batch_normalization_39[0][0]
_____________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 4, 4, 512)    2097664     leaky_re_lu_28[0][0]
_____________________________________________________________________________________________
batch_normalization_40 (BatchNo (None, 4, 4, 512)    2048        conv2d_33[0][0]
_____________________________________________________________________________________________
leaky_re_lu_29 (LeakyReLU)      (None, 4, 4, 512)    0           batch_normalization_40[0][0]
_____________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 4, 4, 128)    1048704     leaky_re_lu_29[0][0]
_____________________________________________________________________________________________
batch_normalization_41 (BatchNo (None, 4, 4, 128)    512         conv2d_34[0][0]
_____________________________________________________________________________________________
leaky_re_lu_30 (LeakyReLU)      (None, 4, 4, 128)    0           batch_normalization_41[0][0]
_____________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 4, 4, 1)      2049        leaky_re_lu_30[0][0]
```

```
----------------------------------------------------------------------------------------------
activation_22 (Activation)      (None, 4, 4, 1)      0           conv2d_35[0][0]
==============================================================================================
Total params: 3,820,609
Trainable params: 2,048
Non-trainable params: 3,818,561
----------------------------------------------------------------------------------------------
```

The U-net generator is composed of various encoding blocks that downsample, and decoding blocks that upsample, an input layer. The encoder block applies a 2D convolution followed batch normalization and activated by a Leaky ReLu activation. The decoder block applies a 2D convolution transpose followed by batch normalization and conditional dropout, used to introduce the random inception. This is followed by a concatenation with the corresponding encoder layer output to form a skip-connection, and it is finally activated with a ReLu activation.

```
# define an encoder block
def define_encoder_block(layer_in, n_filters, batchnorm=True,strid=(2,2)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # add downsampling layer
  g = Conv2D(n_filters, k_shape,strides=strid, padding='same', kernel_initializer=init)\
(layer_in)
  # conditionally add batch normalization
  if batchnorm:
    g = BatchNormalization()(g, training=True)
  # leaky relu activation
  g = LeakyReLU(alpha=0.2)(g)
  return g

# define a decoder block
def decoder_block(layer_in, skip_in, n_filters, dropout=True,strid=(2,2)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # add upsampling layer
  g = Conv2DTranspose(n_filters, k_shape, strides=strid,padding='same',\
 kernel_initializer=init)(layer_in)
  # add batch normalization
  g = BatchNormalization()(g, training=True)
  # conditionally add dropout
  if dropout:
    g = Dropout(0.5)(g, training=True)
  # merge with skip connection
  g = Concatenate()([g, skip_in])
  # relu activation
  g = Activation('tanh')(g)
  return g
```

The generator can now be defined using the above two blocks, along with two additional fully connected layers at the bottleneck.

```
# define the Unet generator model
def define_generator(image_shape=(64,64,6)):
  ####Unet Generator####
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # image input
  src_image = Input(shape=image_shape)
  # encoder model: C64-C128-C256-C512-C512-C512-C512-C512
  e1 = define_encoder_block(src_image, 64, batchnorm=False)
  i1 = define_encoder_block(e1, 128,strid=(1,1))
  e2 = define_encoder_block(i1, 128)
  i2 = define_encoder_block(e2, 128,strid=(1,1))
  e3 = define_encoder_block(i2, 256)
  i3 = define_encoder_block(e3, 512,strid=(1,1))
  e4 = define_encoder_block(i3, 512)
  e5 = define_encoder_block(e4, 512)
  #e6 = define_encoder_block(e5, 512)
  #e7 = define_encoder_block(e6, 512)
  # bottleneck, no batch norm and relu
  b = Conv2D(512, k_shape, strides=(2,2),padding='same', kernel_initializer=init)\
(e5) #e7
  b = Activation('relu')(b)
  b=Flatten()(b)
  b=Dense(256,activation='relu')(b)
  b=Dense(512,activation='relu')(b)
  b=Reshape((1,1,512),input_shape=(512,))(b)
  # decoder model: CD512-CD1024-CD1024-C1024-C1024-C512-C256-C128
  #d1 = decoder_block(b, e7, 512)
  #d2 = decoder_block(d1, e6, 512)
  d3 = decoder_block(b, e5, 512) #(d2, e6, 512)
  d4 = decoder_block(d3, e4, 512, dropout=False)
  j1 = decoder_block(d4, i3, 512, dropout=False)
  d5 = decoder_block(j1, e3, 256, dropout=False,strid=(1,1))
  j2 = decoder_block(d5, i2, 512, dropout=False)
  d6 = decoder_block(j2, e2, 128, dropout=False,strid=(1,1))
  j3 = decoder_block(d6, i1, 128, dropout=False)
  d7 = decoder_block(j3, e1, 64, dropout=False,strid=(1,1))
  # output
  g = Conv2DTranspose(6, k_shape, strides=(2,2),padding='same', \
kernel_initializer=init)(d7)
  out_image = Activation('tanh')(g)
  #out_image=min_distancing(out_image)
  # define model
  model = Model(src_image,out_image)
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='mae', optimizer=opt)
  return model
```

Generator model summary:

```
Model: "functional_15"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_11 (InputLayer)           [(None, 64, 64, 6)]  0
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 32, 32, 64)   6208        input_11[0][0]
__________________________________________________________________________________________________
leaky_re_lu_31 (LeakyReLU)      (None, 32, 32, 64)   0           conv2d_36[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 32, 32, 128)  131200      leaky_re_lu_31[0][0]
__________________________________________________________________________________________________
batch_normalization_42 (BatchNo (None, 32, 32, 128)  512         conv2d_37[0][0]
__________________________________________________________________________________________________
leaky_re_lu_32 (LeakyReLU)      (None, 32, 32, 128)  0           batch_normalization_42[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 16, 16, 128)  262272      leaky_re_lu_32[0][0]
__________________________________________________________________________________________________
batch_normalization_43 (BatchNo (None, 16, 16, 128)  512         conv2d_38[0][0]
__________________________________________________________________________________________________
leaky_re_lu_33 (LeakyReLU)      (None, 16, 16, 128)  0           batch_normalization_43[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 16, 16, 128)  262272      leaky_re_lu_33[0][0]
__________________________________________________________________________________________________
batch_normalization_44 (BatchNo (None, 16, 16, 128)  512         conv2d_39[0][0]
__________________________________________________________________________________________________
leaky_re_lu_34 (LeakyReLU)      (None, 16, 16, 128)  0           batch_normalization_44[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 8, 8, 256)    524544      leaky_re_lu_34[0][0]
__________________________________________________________________________________________________
batch_normalization_45 (BatchNo (None, 8, 8, 256)    1024        conv2d_40[0][0]
__________________________________________________________________________________________________
leaky_re_lu_35 (LeakyReLU)      (None, 8, 8, 256)    0           batch_normalization_45[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 8, 8, 512)    2097664     leaky_re_lu_35[0][0]
__________________________________________________________________________________________________
batch_normalization_46 (BatchNo (None, 8, 8, 512)    2048        conv2d_41[0][0]
__________________________________________________________________________________________________
leaky_re_lu_36 (LeakyReLU)      (None, 8, 8, 512)    0           batch_normalization_46[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 4, 4, 512)    4194816     leaky_re_lu_36[0][0]
__________________________________________________________________________________________________
batch_normalization_47 (BatchNo (None, 4, 4, 512)    2048        conv2d_42[0][0]
__________________________________________________________________________________________________
leaky_re_lu_37 (LeakyReLU)      (None, 4, 4, 512)    0           batch_normalization_47[0][0]
__________________________________________________________________________________________________
```

```
conv2d_43 (Conv2D)              (None, 2, 2, 512)    4194816     leaky_re_lu_37[0][0]
______________________________________________________________________________________________
batch_normalization_48 (BatchNo (None, 2, 2, 512)    2048        conv2d_43[0][0]
______________________________________________________________________________________________
leaky_re_lu_38 (LeakyReLU)      (None, 2, 2, 512)    0           batch_normalization_48[0][0]
______________________________________________________________________________________________
conv2d_44 (Conv2D)              (None, 1, 1, 512)    4194816     leaky_re_lu_38[0][0]
______________________________________________________________________________________________
activation_23 (Activation)      (None, 1, 1, 512)    0           conv2d_44[0][0]
______________________________________________________________________________________________
flatten_2 (Flatten)             (None, 512)          0           activation_23[0][0]
______________________________________________________________________________________________
dense_4 (Dense)                 (None, 256)          131328      flatten_2[0][0]
______________________________________________________________________________________________
dense_5 (Dense)                 (None, 512)          131584      dense_4[0][0]
______________________________________________________________________________________________
reshape_2 (Reshape)             (None, 1, 1, 512)    0           dense_5[0][0]
______________________________________________________________________________________________
conv2d_transpose_18 (Conv2DTran (None, 2, 2, 512)    4194816     reshape_2[0][0]
______________________________________________________________________________________________
batch_normalization_49 (BatchNo (None, 2, 2, 512)    2048        conv2d_transpose_18[0][0]
______________________________________________________________________________________________
dropout_2 (Dropout)             (None, 2, 2, 512)    0           batch_normalization_49[0][0]
______________________________________________________________________________________________
concatenate_19 (Concatenate)    (None, 2, 2, 1024)   0           dropout_2[0][0]
                                                                 leaky_re_lu_38[0][0]
______________________________________________________________________________________________
activation_24 (Activation)      (None, 2, 2, 1024)   0           concatenate_19[0][0]
______________________________________________________________________________________________
conv2d_transpose_19 (Conv2DTran (None, 4, 4, 512)    8389120     activation_24[0][0]
______________________________________________________________________________________________
batch_normalization_50 (BatchNo (None, 4, 4, 512)    2048        conv2d_transpose_19[0][0]
______________________________________________________________________________________________
concatenate_20 (Concatenate)    (None, 4, 4, 1024)   0           batch_normalization_50[0][0]
                                                                 leaky_re_lu_37[0][0]
______________________________________________________________________________________________
activation_25 (Activation)      (None, 4, 4, 1024)   0           concatenate_20[0][0]
______________________________________________________________________________________________
conv2d_transpose_20 (Conv2DTran (None, 8, 8, 512)    8389120     activation_25[0][0]
______________________________________________________________________________________________
batch_normalization_51 (BatchNo (None, 8, 8, 512)    2048        conv2d_transpose_20[0][0]
______________________________________________________________________________________________
concatenate_21 (Concatenate)    (None, 8, 8, 1024)   0           batch_normalization_51[0][0]
                                                                 leaky_re_lu_36[0][0]
______________________________________________________________________________________________
activation_26 (Activation)      (None, 8, 8, 1024)   0           concatenate_21[0][0]
______________________________________________________________________________________________
conv2d_transpose_21 (Conv2DTran (None, 8, 8, 256)    4194560     activation_26[0][0]
__________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 8, 8, 256)    1024        conv2d_transpose_21[0][0]
__________________________________________________________________________________________________
concatenate_22 (Concatenate)    (None, 8, 8, 512)    0           batch_normalization_52[0][0]
                                                                 leaky_re_lu_35[0][0]
__________________________________________________________________________________________________
activation_27 (Activation)      (None, 8, 8, 512)    0           concatenate_22[0][0]
__________________________________________________________________________________________________
conv2d_transpose_22 (Conv2DTran (None, 16, 16, 512)  4194816     activation_27[0][0]
__________________________________________________________________________________________________
batch_normalization_53 (BatchNo (None, 16, 16, 512)  2048        conv2d_transpose_22[0][0]
__________________________________________________________________________________________________
concatenate_23 (Concatenate)    (None, 16, 16, 640)  0           batch_normalization_53[0][0]
                                                                 leaky_re_lu_34[0][0]
__________________________________________________________________________________________________
activation_28 (Activation)      (None, 16, 16, 640)  0           concatenate_23[0][0]
__________________________________________________________________________________________________
conv2d_transpose_23 (Conv2DTran (None, 16, 16, 128)  1310848     activation_28[0][0]
__________________________________________________________________________________________________
batch_normalization_54 (BatchNo (None, 16, 16, 128)  512         conv2d_transpose_23[0][0]
__________________________________________________________________________________________________
concatenate_24 (Concatenate)    (None, 16, 16, 256)  0           batch_normalization_54[0][0]
                                                                 leaky_re_lu_33[0][0]
__________________________________________________________________________________________________
activation_29 (Activation)      (None, 16, 16, 256)  0           concatenate_24[0][0]
__________________________________________________________________________________________________
conv2d_transpose_24 (Conv2DTran (None, 32, 32, 128)  524416      activation_29[0][0]
__________________________________________________________________________________________________
batch_normalization_55 (BatchNo (None, 32, 32, 128)  512         conv2d_transpose_24[0][0]
__________________________________________________________________________________________________
concatenate_25 (Concatenate)    (None, 32, 32, 256)  0           batch_normalization_55[0][0]
                                                                 leaky_re_lu_32[0][0]
__________________________________________________________________________________________________
activation_30 (Activation)      (None, 32, 32, 256)  0           concatenate_25[0][0]
__________________________________________________________________________________________________
conv2d_transpose_25 (Conv2DTran (None, 32, 32, 64)   262208      activation_30[0][0]
__________________________________________________________________________________________________
batch_normalization_56 (BatchNo (None, 32, 32, 64)   256         conv2d_transpose_25[0][0]
__________________________________________________________________________________________________
concatenate_26 (Concatenate)    (None, 32, 32, 128)  0           batch_normalization_56[0][0]
                                                                 leaky_re_lu_31[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 32, 32, 128)  0           concatenate_26[0][0]
__________________________________________________________________________________________________
conv2d_transpose_26 (Conv2DTran (None, 64, 64, 6)    12294       activation_31[0][0]
__________________________________________________________________________________________________
activation_32 (Activation)      (None, 64, 64, 6)    0           conv2d_transpose_26[0][0]
==================================================================================================
Total params: 47,622,918
Trainable params: 47,613,318
Non-trainable params: 9,600
______________________________________________________________________________________________
```

The combined Pix2Pix-GAN model can now be constructed using the generator and discriminator models.

```
# define the pix2pix_GAN model
def define_pix2pix_GAN(g_model, d_model, image_shape):
  # make weights in the discriminator not trainable
  #d_model.trainable = False
  # define the source image
  in_src = Input(shape=image_shape)
  # connect the source image to the generator input
  gen_out = g_model(in_src)
  # connect the source input and generator output to the discriminator input
  dis_out = d_model([in_src, gen_out])
  # src image as input, generated image and classification output
  model = Model(in_src, [dis_out, gen_out])
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=\
[1,100])
  return model
```

Pix2Pix-GAN model summary:

```
Model: "functional_17"
______________________________________________________________________________________________
Layer (type)                     Output Shape         Param #     Connected to
==============================================================================================
input_12 (InputLayer)            [(None, 64, 64, 6)]  0
______________________________________________________________________________________________
functional_15 (Functional)       (None, 64, 64, 6)    47622918    input_12[0][0]
______________________________________________________________________________________________
functional_13 (Functional)       (None, 4, 4, 1)      3820609     input_12[0][0]
                                                                  functional_15[0][0]
==============================================================================================
Total params: 51,443,527
Trainable params: 47,615,366
Non-trainable params: 3,828,161
______________________________________________________________________________________________
```

A training helper function is defined to train the pix2pix-GAN model

```
# train pix2pix_GAN model
def train(d_model,g_model,pix2pix_GAN_model, dataset, image_shape, pre_epo,n_epochs\
=500, n_batch=1):
  # unpack dataset
  trainA,dt,trainB = dataset
  # calculate the number of batches per training epoch
  bat_per_epo = int(len(trainA) / n_batch)
  # calculate the number of training iterations
  n_steps = bat_per_epo * n_epochs
  print("\nTraining model for total "+str(n_epochs)+" training epochs")
  print("\nEpoch D_Loss G_Loss Time")
  # manually enumerate epochs
  for i in range(n_epochs):
    start_time=datetime.datetime.now()
    for j in range(bat_per_epo):
      # select a batch of real samples
      [X_realA, X_realB], y_real= generate_real_samples(dataset, n_batch,image_shape)
      # generate a batch of fake samples
      X_fakeB= g_model.predict(X_realA)
      # update discriminator
      d_loss=d_model.train_on_batch([X_realB,X_fakeB],y_real)
      #update the model
      g_loss,a,b= pix2pix_GAN_model.train_on_batch(X_realA,[y_real,X_realB])
      print(j)
    if i%1==0:
      d_model.save('/ML_Models/d_model_ML')
      g_model.save('/ML_Models/g_model_ML')
      pix2pix_GAN_model.save('/ML_Models/pix2pix_GAN_ML')
    if i%1==0:
      [X_realA, X_realB],y_real,file_name = generate_test_samples(dataset\
,image_shape,True)
      X_predB=g_model.predict(X_realA)
      X_predB=rescale_predictions(X_predB)
      file_name=[file_name+"_epo"+str(i+pre_epo)]
      save_predictions(X_predB,file_name)
      #show_predictions(X_predB)
      #pause(0.001)
    # summarize performance
    elapsed_time=datetime.datetime.now()-start_time
    print('%d %e %e %e %e %s' % (i+1+pre_epo, d_loss, g_loss,a,b, elapsed_time))
```

A separate function is defined to compile, save or load the ML models and invoke training.

```
def run_to_train(scratch=True,pre_epo=0):
  # define image shape
  image_shape = (64,64,6)
  if scratch:
    # define the models
    d_model=define_discriminator(image_shape)
    g_model=define_generator(image_shape)
    pix2pix_GAN_model = define_pix2pix_GAN(g_model,d_model,image_shape)
    print("\nmodel defined")
    d_model.summary()
    plot_model(d_model, to_file='d_model_plot.png', show_shapes=True, \
show_layer_names=True)
    g_model.summary()
    plot_model(g_model, to_file='g_model_plot.png', show_shapes=True, \
show_layer_names=True)
    pix2pix_GAN_model.summary()
    plot_model(pix2pix_GAN_model, to_file='pix2pix_GAN_model_plot.png', \
show_shapes=True, show_layer_names=True)
    prev_epochs=0
  else:
    d_model=load_model('/ML_Models/d_model_ML')
    g_model=load_model('/ML_Models/g_model_ML')
    pix2pix_GAN_model=load_model('/ML_Models/pix2pix_GAN_ML')
    print("\nmodel loaded")
    d_model.summary()
    plot_model(d_model, to_file='d_model_plot.png', show_shapes=True, \
show_layer_names=True)
    g_model.summary()
    plot_model(g_model, to_file='g_model_plot.png', show_shapes=True, \
show_layer_names=True)
    pix2pix_GAN_model.summary()
    plot_model(pix2pix_GAN_model, to_file='pix2pix_GAN_model_plot.png', \
show_shapes=True, show_layer_names=True)
    prev_epochs=pre_epo
  # load image data. dataset is a list containing in_image,dt,out_image addresses.
  dataset=load_dataset("/train_set.txt")
  print("\ndataset loaded")
  # train model
  train(d_model,g_model,pix2pix_GAN_model,dataset,image_shape,prev_epochs)
  d_model.save('/ML_Models/d_model_ML')
  g_model.save('/ML_Models/g_model_ML')
  pix2pix_GAN_model.save('/ML_Models/pix2pix_GAN_ML')
```

Once trained, the ML model can be used to generate predictions using a different function.

```
def run_trained():
  # define image shape
  image_shape = (64,64,6)
  d_model=load_model('/content/drive/My Drive/ML_Models/d_model_ML')
  g_model=load_model('/content/drive/My Drive/ML_Models/g_model_ML')
  pix2pix_GAN_model=load_model('/content/drive/My Drive/ML_Models/pix2pix_GAN_ML')
  print("\npix2pix_GAN model loaded")
  # load image data. dataset is a list containing in_image,dt,out_image addresses.
  dataset=load_dataset\
("/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_train_sample_set.txt")
  print("\nTraining sample dataset loaded")
  start_time=datetime.datetime.now()
  [X_realA, X_realB],y_real,file_names=generate_test_samples(dataset,image_shape,False)
  X_predB=g_model.predict(X_realA)
  eval=pix2pix_GAN_model.evaluate(X_realA,[y_real,X_realB])
  print(pix2pix_GAN_model.metrics_names)
  print(eval)
  X_realB=rescale_predictions(X_realB)#deblur_io(X_realB))
  X_predB=rescale_predictions(X_predB)#deblur_io(X_predB))
  file_names=[file_name+"_training_sample" for file_name in file_names]
  save_predictions(X_predB,file_names)
  elapsed_time=datetime.datetime.now()-start_time
  print(str(len(file_names))+" predictions done from training sample. Time taken "\
+str(elapsed_time))
  #show_predictions(X_realB)
  #show_predictions(X_predB,1)
  #pause(60)
  # load image data. dataset is a list containing in_image,dt,out_image addresses.
  dataset=load_dataset\
("/content/drive/My Drive/txt_dataset/NVE_Ar_ppp222_1fs_sim_test_set.txt")
  print("\nTesting dataset loaded")
  start_time=datetime.datetime.now()
  [X_realA, X_realB],y_real,file_names=generate_test_samples(dataset,image_shape,False)
  X_predB=g_model.predict(X_realA)
  eval=pix2pix_GAN_model.evaluate(X_realA,[y_real,X_realB])
  print(pix2pix_GAN_model.metrics_names)
  print(eval)
  X_realB=rescale_predictions(X_realB)#deblur_io(X_realB))
  X_predB=rescale_predictions(X_predB)#deblur_io(X_predB))
  file_names=[file_name+"_test" for file_name in file_names]
  save_predictions(X_predB,file_names)
  elapsed_time=datetime.datetime.now()-start_time
  print(str(len(file_names))+" predictions done from test set. Time taken "+\
str(elapsed_time))
  #show_predictions(X_realB)
  #show_predictions(X_predB,1)
  #pause(60)
```

By running one of the following three commands at runtime, one can toggle between training a model from scratch, or resuming training after N previous training epochs, or making predictions from a pre-trained model.

```
run_to_train(True)
run_to_train(False,N)
run_trained()
```

## 8 PyTorch program for Symplectic Recurrent Convolutional Hamiltonian Neural Network

The PyTorch implementation of SRCHNN is explained below.
The first step is to import the necessary packages and modules, along with defining global variables including the model's hyperparameters.

```
import datetime
import numpy as np
import random
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.autograd import grad
from tqdm import tqdm
import matplotlib.pyplot as plt

pos_norm = 1
vel_norm = 1e-3 # Hyperparameter, normalization scale for velocities
#[center,scale]
norm_data = np.array([np.array([0.5,0.5]) * pos_norm, np.array([0,1]) * vel_norm])

pe_wt = 3e+2 # Hyperparameter, PE multiplier
# KE multiplier: J->Kcal, m in Kg/mol, velocity scaling
ke_wt = 1e-3 / 4.184 * 39.948 * 1e-3 * 1e+5 * vel_norm * 1e+5 * vel_norm / 2

t = 6 # Sequence length during training
dt = 1 # Time interval in leapfrog integrator

q_dropout = 0.3 # Hyperparameter, dropout probability
lr = 1e-4 # Hyperparameter, Adam learning rate
batch_size = 10
training_epochs = 5
```

The model class is then defined.

```
class SRCHNN(nn.Module):
    def __init__(self):
        super(SRCHNN, self).__init__()
        self.inq = nn.Conv2d(3, 16, kernel_size=6, padding=2)
        self.q1 = nn.Conv2d(16, 32, kernel_size=5, padding=1)
        self.q2 = nn.Conv2d(32, 64, kernel_size=5, padding=1)
        self.q3 = nn.Conv2d(64, 128, kernel_size=3, padding=0)
        self.q4 = nn.Conv2d(128, 256, kernel_size=3, padding=0)
        self.q5 = nn.Linear(256, 64)
        self.dropout = nn.Dropout(p=q_dropout, inplace=False)
        self.q6 = nn.Linear(64, 16)
        self.q7 = nn.Linear(16, 1)
    def potential_energy(self,q):
        q = self.inq(q).tanh_()
        assert q.shape[1:] == torch.Size([16, 9, 9])
        q = self.q1(q).tanh_()
        assert q.shape[1:] == torch.Size([32, 7, 7])
        q = self.q2(q).tanh_()
        assert q.shape[1:] == torch.Size([64, 5, 5])
        q = self.dropout(q)
        q = self.q3(q).tanh_()
        assert q.shape[1:] == torch.Size([128, 3, 3])
        q = self.q4(q).tanh_()
        assert q.shape[1:] == torch.Size([256, 1, 1])
        q=q.view(q.size()[0], -1)
        assert q.shape[1:] == torch.Size([256])
        q = self.q5(q).tanh_()
        assert q.shape[1:] == torch.Size([64])
        q = self.dropout(q)
        q = self.q6(q).tanh_()
        assert q.shape[1:] == torch.Size([16])
        q = self.dropout(q)
        pe = self.q7(q).tanh_()
        assert pe.shape[1:] == torch.Size([1])
        return pe

    def kinetic_energy(self,p):
        p = torch.square(p)
        assert p.shape[1:] == torch.Size([3, 10, 10])
        ke = torch.sum(p,dim=[1,2,3],keepdim=True)
        ke=ke.view(ke.size()[0],-1)
        assert ke.shape[1:] == torch.Size([1])
        return ke

    def forward(self, q, p):
        return pe_wt * self.potential_energy(q) + ke_wt * self.kinetic_energy(p), \
ke_wt * self.kinetic_energy(p)
```

The symplectic leapfrog integrator takes the initial coordinates q and velocities p as input and calculates the trajectories for the next T timesteps using the model.

```
def leapfrog(q_0, p_0, T, model, device='cpu'):
    pred_seq_q = torch.empty((T, p_0.shape[0], p_0.shape[1], p_0.shape[2], \
p_0.shape[3]), requires_grad=False).to(device)
    pred_seq_p = torch.empty((T, p_0.shape[0], p_0.shape[1], p_0.shape[2], \
p_0.shape[3]), requires_grad=False).to(device)
    H = torch.empty((T, p_0.shape[0]), requires_grad=False).to(device)
    KE = torch.empty((T, p_0.shape[0]), requires_grad=False).to(device)

    p = torch.from_numpy(p_0)
    q = torch.from_numpy(q_0)
    p.requires_grad_()
    q.requires_grad_()

    for i in range(T):
        H_calc, ke_calc = model(p.float(), q.float())
        H[i,:] = H_calc[:,0]
        KE[i,:] = ke_calc[:,0]
        dpdt = -grad(H_calc.sum(), q, create_graph=True)[0]

        p_half = p + dpdt / 2 * dt

        hamilt, _ = model(p_half.float(), q.float())
        dqdt = grad(hamilt.sum(), p_half, create_graph=True)[0]

        q_next = q + dqdt * dt

        hamilt, _ = model(p_half.float(), q_next.float())
        dpdt = -grad(hamilt.sum(), q_next, create_graph=True)[0]

        p_next = p_half + dpdt / 2 * dt

        p = p_next
        q = q_next

        pred_seq_q[i,:,:,:,:] = q
        pred_seq_p[i,:,:,:,:] = p

    return pred_seq_q, pred_seq_p, H, KE
```

The dataset consists of numpy arrays stored in separate a directory. A function is used to create text datasets that contains identifying numbers which can be used to fetch the required arrays. Thus, different datasets are created by creating different sets of identifiers.

```
def create_seq_dataset(test_frac=0.1, T=10, T2=50):
    l = 2001 - T - int(2000 * test_frac)
    l2 = int(2000 * test_frac / T2)
    train = np.zeros((6 * l, 2))
    test = np.zeros((6 * l2 , 2))
    for i in range(6):
        train[(l * i):(i + 1) * l, 0] = i
        train[(l * i):(i + 1) * l, 1] = range(l)
        test[(l2 * i):(i +1 ) * l2, 0] = i
        test[(l2 * i):(i + 1) * l2, 1] = range(l + T, 2000 - T2, T2)
    np.random.shuffle(train)
    train_sample = train[np.random.choice(train.shape[0], 6 * l2, replace=False),:]
    np.savetxt('./training_set.txt', np.transpose(train, (1,0)))
    np.savetxt('./training_sample_set.txt', np.transpose(train_sample, (1,0)))
    np.savetxt('./test_set.txt', np.transpose(test, (1,0)))
```

Some helper functions are defined to load and save the required numpy arrays, along with saving the calculated energies and mean square errors.

```
def load_seq_minibatch(ID, init, T=10):
    seq_q = np.zeros((T + 1, len(ID), 3, 10, 10))
    seq_p = np.zeros((T + 1, len(ID), 3, 10, 10))
    for i in range(T + 1):
        for j in range(len(ID)):
            file_name = "./npy_dataset/NVE_" + chr(ord("A") + ID[j]) + "_" + \
str(init[j] + i) + ".MD2ML.npy"
            a = np.transpose(np.load(file_name), (2,0,1))
            seq_q[i,j,:,:,:] = (a[:3,:,:] - norm_data[0,0]) / norm_data[0,1]
            seq_p[i,j,:,:,:] = (a[3:,:,:] - norm_data[1,0]) / norm_data[1,1]
    return seq_q, seq_p

def save_seq_predictions(ID, init, pred_seq_q, pred_seq_p, ap='test', T=10):
    a = np.zeros((6,10,10))
    for i in range(T):
        for j in range(len(ID)):
            file_name = "./MLpred/NVE_" + chr(ord("A") + ID[j]) + "_" + \
str(init[j] + i) + ".MD2ML.npy_" + ap + "_seq" + str(i) + "MLpred.npy"
            a[:3,:,:] = pred_seq_q[i,j,:,:,:] * norm_data[0,1] + norm_data[0,0]
            a[3:,:,:] = pred_seq_p[i,j,:,:,:] * norm_data[1,1] + norm_data[1,0]
            np.save(file_name, np.transpose(a, (1,2,0)))

def save_energies(H, KE, mse, ID, init, ap='test'):
    [T, n_samples] = H.shape
    file_ids = np.transpose(np.array([ID, init]), (1,0))
    master_data = np.concatenate((file_ids, np.transpose(H, (1,0)), \
np.transpose(KE, (1,0)), np.transpose(mse, (1,0))), axis=1)
    np.savetxt('./MLpred/Energies_' + ap + '.txt', master_data)
```

A single training step consists of generating predictions using the model and the symplectic leapfrog integrator, calculating the MSE loss, calculate gradients of the loss with respect to model parameters and apply those gradients for updating those model parameters. Evaluation is done in the same way without the backpropagation steps; while predictions are done with just the integrator.

```
def train_step(model, opt_model, q_0, p_0, true_seq_q, true_seq_p, T=10):
    true_seq_q.requires_grad_()
    true_seq_p.requires_grad_()
    opt_model.zero_grad()
    pred_seq_q, pred_seq_p, H_calc, ke_calc = leapfrog(q_0, p_0, T=T, model=model, \
device='cpu')
    loss = nn.MSELoss()
    error_total = loss(pred_seq_q, true_seq_q) + loss(pred_seq_p, true_seq_p)
    error_total.backward()
    opt_model.step()
    return pred_seq_q.detach().numpy(), pred_seq_p.detach().numpy(), \
H_calc.detach().numpy(), ke_calc.detach().numpy(), error_total
```

```
def evaluator(model, q_0, p_0, true_seq_q, true_seq_p, T=10):
    pred_seq_q, pred_seq_p, H_calc, ke_calc = leapfrog(q_0, p_0, T=T, model=model, \
device='cpu')
    loss = nn.MSELoss()
    error_total = np.zeros((pred_seq_q.shape[0], pred_seq_q.shape[1]))
    for i in range(pred_seq_q.shape[0]):
        for j in range(pred_seq_q.shape[1]):
            error_total[i,j] = (loss(pred_seq_q[i,j,:,:,:], true_seq_q[i,j,:,:,:]) +\
 loss(pred_seq_p[i,j,:,:,:], true_seq_p[i,j,:,:,:])).detach().numpy()
    return pred_seq_q.detach().numpy(), pred_seq_p.detach().numpy(), \
H_calc.detach().numpy(), ke_calc.detach().numpy(), error_total

def predictor(model, q_0, p_0, T=10):
    return leapfrog(q_0, p_0, T=T, model=model, device='cpu')
```

A separate function is defined to make predictions from the model, with various options for saving the data, and getting a screen output of training progress.

```
def make_predictions(dataset, model, single=False, save_E=True, save_pred=True, \
pr_scr=True, ap='test', T=t):
    start_time=datetime.datetime.now()

    if single:
        ID, init = [dataset[0,0]], [dataset[1,0]]
    else:
        ID, init = dataset

    seq_q, seq_p = load_seq_minibatch(ID, init, T=T)
    pred_seq_q, pred_seq_p, H, KE, mse =     evaluator(model, seq_q[0,:,:,:,:], \
seq_p[0,:,:,:,:], torch.from_numpy(seq_q[1:,:,:,:,:]).float(), \
torch.from_numpy(seq_p[1:,:,:,:,:]).float(), T=T)

    if save_E:
        save_energies(H, KE, mse, ID, init, ap=ap)

    if save_pred:
        save_seq_predictions(ID, init, pred_seq_q, pred_seq_p, ap=ap, T=T)

    elapsed_time = datetime.datetime.now() - start_time

    if single and pr_scr:
        print(ap + ' ' + chr(ord('A') + ID[0]) + '_' + str(init[0]))
        print(mse)
        print(H)
        print(KE)
    elif pr_scr:
        print(str(len(ID)) + ' predictions made with sequence length = ' + str(T) + \
'. MSE = ' + str(np.mean(mse)) + '. Time taken = ' + str(elapsed_time))
```

```
    return mse
```

Two training helper functions are defined to choose the mode of training and for training the model by using minibatches from the entire dataset multiple times. Evaluations are also done for measuring model performance during training.

```
def train_on_seq(model, T=10, n_epochs=50, n_batch=2):
    train_dataset = np.loadtxt('./training_set.txt', dtype=int)
    train_sample_dataset = np.loadtxt('./training_sample_set.txt', dtype=int)
    test_dataset = np.loadtxt('./test_set.txt', dtype=int)

    opt_model = torch.optim.Adam(model.parameters(), lr=lr)
    print('Epoch\tMSE_train_single\tMSE_test_single\tMSE_train\tMSE_test\tElapsed_time')

    for i in (range(n_epochs)):
        start_time = datetime.datetime.now()

        for j in tqdm(range(int(train_dataset.shape[1] / n_batch))):
            ID, init = train_dataset[:, (n_batch * j):(n_batch * (j + 1))]
            seq_q, seq_p = load_seq_minibatch(ID, init, T=T)
            _, _, _, _, _ = train_step(model, opt_model, seq_q[0,:,:,:,:], \
seq_p[0,:,:,:,:], torch.from_numpy(seq_q[1:,:,:,:,:]).float(), \
torch.from_numpy(seq_p[1:,:,:,:,:]).float(), T=T)

        if i%1==0:
            torch.save(model, './srchnn.pt')

            model.eval()
            mse_train_single = make_predictions(dataset=train_sample_dataset, \
model=model, single=True, save_E=False, save_pred=True, pr_scr=True, \
ap='epo_train'+str(i).zfill(2), T=T)
            mse_test_single = make_predictions(dataset=test_dataset, \
model=model, single=True, save_E=False, save_pred=True, pr_scr=True, \
ap='epo_test'+str(i).zfill(2), T=T)
            mse_train_sample = make_predictions(dataset=train_sample_dataset, \
model=model, single=False, save_E=False, save_pred=False, pr_scr=False, \
ap='epo_train'+str(i).zfill(2), T=T)
            mse_test = make_predictions(dataset=test_dataset, model=model, \
single=False, save_E=False, save_pred=False, pr_scr=False, \
ap='epo_test' + str(i).zfill(2), T=T)

            model.train()

        elapsed_time = datetime.datetime.now() - start_time
        print('%d\t%e\t%e\t%e\t%e\t%s' % (i + 1, np.mean(mse_train_single), \
np.mean(mse_test_single), np.mean(mse_train_sample), np.mean(mse_test), elapsed_time))

def train_model(scratch=True, pre_epo=0):
    if scratch:
        # define the models
        model = SRCHNN()
        print("\nSRCHNN model defined. Total number of trainable parameters = ")
        print(sum(p.numel() for p in model.parameters() if p.requires_grad))
        prev_epochs=0
    else:
        model = torch.load('./srchnn.pt')
        model.train()
        print("\nSRCHNN model loaded")
        prev_epochs = pre_epo

    train_on_seq(model.float(), T=t, n_epochs=training_epochs, n_batch=batch_size)
    torch.save(model, './srchnn.pt')
```

After the model is trained, a testing function evaluates its performance in predicting sequences.

```
def test_model():
    model = torch.load('./srchnn.pt')
    model.eval()
    print("\nSRCHNN model loaded")

    train_sample_dataset = np.loadtxt('./training_sample_set.txt', dtype=int)
    test_dataset = np.loadtxt('./test_set.txt', dtype = int)
    _ = make_predictions(dataset=train_sample_dataset, model=model, \
single=False, save_E=True, save_pred=True, pr_scr=True, ap='training_sample', T=15)
    _ = make_predictions(dataset=test_dataset, model=model, \
single=False, save_E=True, save_pred=True, pr_scr=True, ap='test', T=15)
```

A single experiment for training and testing the model can be performed by running the follwing commands

```
create_seq_dataset(test_frac=0.05, T=t, T2=15)
train_model()
test_model()
```