DeepFlame

Release 0.1

DeepModeling

Mar 02, 2024

QUICK START

DeepFlame is a deep learning empowered computational fluid dynamics package for single or multiphase, laminar or turbulent, reacting flows at all speeds. It aims to provide an open-source platform to combine the individual strengths of OpenFOAM, Cantera, and PyTorch libraries for deep learning assisted reacting flow simulations. It also has the scope to incorporate next-generation heterogenous supercomputing and AI acceleration infrastructures such as GPU and FPGA.

The deep learning algorithms and models used in the DeepFlame tutorial examples are made available in AIS Square for community data sharing – DF-ODENet. Please refer to the website for detailed information.

Note: This project is under active development.

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

The installation of DeepFlame is simple and requires **OpenFOAM-7**, **LibCantera**, and **PyTorch**.

First, install OpenFOAM-7.

Note: For Ubuntu 20.04, one can install by `apt`. For latest versions, please compile OpenFOAM-7 from source code. Check operating system version by `lsb_release -d`.

```
# Install OpenFOAM release by apt
sudo sh -c "wget -O - https://dl.openfoam.org/gpg.key | apt-key add -"
sudo add-apt-repository http://dl.openfoam.org/ubuntu
sudo apt-get update
sudo apt-get -y install openfoam7
```

OpenFOAM-7 and ParaView-5.6.0 will be installed in the `/opt` directory.

Note: There is a commonly seen issue when installing OpenFOAM via `apt-get install` with an error message: `could not find a distribution template for Ubuntu/focal`. To resolve this issue, you can refer to issue#54.

Alternatively, one can compile OpenFOAM-7 from source code.

```
gcc --version
sudo apt-get install build-essential cmake git ca-certificates
sudo apt-get install flex libfl-dev bison zlib1g-dev libboost-system-dev libboost-thread-
↪dev libopenmpi-dev openmpi-bin gnuplot libreadline-dev libncurses-dev libxt-dev
cd $HOME # the path OpenFOAM will be installed
wget -O - http://dl.openfoam.org/source/7 | tar xz
wget -O - http://dl.openfoam.org/third-party/7 | tar xz
mv OpenFOAM-7-version-7 OpenFOAM-7
mv ThirdParty-7-version-7 ThirdParty-7
source OpenFOAM-7/etc/bashrc
./OpenFOAM-7/Allwmake -j
```

LibCantera and **PyTorch** can be easily installed via conda. If your platform is compatible, run the following command to install the dependencies.

```
conda create -n deepflame python=3.8
conda activate deepflame
conda install -c cantera libcantera-devel=2.6 cantera
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/
↪cu118
conda install pybind11 pkg-config
```

Note: Please go to PyTorch's official website to check your system compatability and choose the installation command line that is suitable for your platform. After installing torch, do check if torch.cuda.is_available() returns true to use GPU for DNN inference!

```
# For CUDA-supported platforms
conda create -n deepflame \
    pytorch torchvision torchaudio libcantera-devel easydict pybind11 pkg-config \
    -c pytorch -c nvidia -c cantera -c conda-forge
conda activate deepflame
```

Note: Check your `Miniconda3/envs/deepflame` directory and make sure the install was successful (lib/ include/ etc. exist).

1.2 Configure

1. Source your OpenFOAM-7 bashrc to configure the $FOAM environment.

Note: This depends on your own path for OpenFOAM-7 bashrc.

If you have installed using `apt-get install`, then:

```
source /opt/openfoam7/etc/bashrc
```

If you compiled from source following the official guide, then:

```
source $HOME/OpenFOAM/OpenFOAM-7/etc/bashrc
```

To source the bashrc file automatically when opening your terminal, type

```
echo "source /opt/openfoam7/etc/bashrc" >> ~/.bashrc
```

or

```
echo "source $HOME/OpenFOAM/OpenFOAM-7/etc/bashrc" >> ~/.bashrc
```

Then source the bashrc file by:

```
source ~/.bashrc
```

Note: Check your environment using `echo $FOAM_ETC` and you should get the directory path for your OpenFOAM-7 bashrc you just used in the above step.

2. Clone the DeepFlame repository:

```
git clone https://github.com/deepmodeling/deepflame-dev.git
```

If you want to use the submodules included in DeepFlame: the WENO scheme and the libROUNDSchemes, run

```
git clone --recursive https://github.com/deepmodeling/deepflame-dev.git
```

Detailed instructions for compiling these two submodules can be found in their original repositories.

3. Configure the DeepFlame environment:

```
cd deepflame-dev
. configure.sh --use_pytorch
source ./bashrc
```

Note: Check your environment using `echo $DF_ROOT` and you should get the path for the `deepflame-dev` directory.

1.3 Build and Install

Finally you can build and install DeepFlame:

```
. install.sh
```

Note: You may see an error `fmt` or `eigen` files cannot be found. If so, go to your conda environment and install the packages as follows.

```
conda install fmt
conda install eigen
```

Note: You may also come accross an error regarding shared library `libmkl_rt.so.2` when libcantera is installed through cantera channel. If so, go to your conda environment and check the existance of `libmkl_rt.so.2` and `libmkl_rt.so.1`, and then link `libmkl_rt.so.2` to `libmkl_rt.so.1`.

```
cd ~/miniconda3/envs/deepflame/lib
ln -s libmkl_rt.so.1 libmkl_rt.so.2
```

If you have compiled DeepFlame successfully, you should see the print message in your terminal:

```
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
| deepflame (linked with libcantera and pytorch) compiled successfully! Enjoy!! |
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

1.4 Other Options

DeepFlame also provides users with full GPU version and CVODE (no DNN version) options.

1. If you just need DeepFlame's CVODE solver without DNN model, just install LibCantera via conda.

```
conda create -n df-notorch python=3.8
conda activate df-notorch
conda install -c conda-forge libcantera-devel
```

If the conda env `df-notorch` is activated, install DeepFlame by running:

```
cd deepflame-dev
. configure.sh
source ./bashrc
. install.sh
```

If `df-notorch` not activated (or you have a self-compiled libcantera), specify the path to your libcantera:

```
. configure.sh --libcantera_dir /your/path/to/libcantera/
source ./bashrc
. install.sh
```

2. If you wish to employ dfMatrix and the AMGX library for accelerating PDE solving using GPU:

Note: This is still under developement.

To begin, you will need to install AMGX. You can find the instructions for installing AMGX on its official website. Follow the instructions provided to install AMGX on your system. Once you have installed AMGX, navigate to the DeepFlame directory and follow the commands below.

```
cd deepflame-dev
. configure.sh --amgx_dir /your/path/to/AMGX/ --libtorch_dir /path/to/libtorch/
source ./bashrc
. install.sh
```

Also, you will need to add configuration files for AMGX for each euqation under `system` folder and name them in the pattern of `amgxpOptions`, `amgxUOptions` . Please refer to the AMGX official website to find out detailed instructions.

If you have compiled DeepFlame with GPU solver successfully, you should see the print message in your terminal:

```
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
|     deepflame (linked with libcantera) compiled successfully! Enjoy!!          |
|        select the GPU solver coupled with AMGx library to solve PDE            |
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

3. If you wish to install DeepFlame with CMake

Note: This is still under developement.

You will need to follow the same procedures to install prerequisites and configure DeepFlame.

```
cd deepflame-dev
. configure.sh --use_pytorch
source ./bashrc
```

After this, first install libraries:

```
cd $DF_ROOT
cmake -B build
cd build
make install
```

Now if go to `$DF_ROOT/lib`, libraries should be ready. Compilition of solvers are separated. Choose the solver you want to use and then go to the directory and build it. For example,

```
cd $DF_ROOT/applications/solvers/dfLowMachFoam
cmake -B build
cd build
make install
```

CHAPTER

TWO

DOWNLOAD DNN MODELS

The neural network models used in the tutorial examples can be found at– AIS Square. To run DeepFlame with DNN, download the DNN model dfODENet into the case folder you would like to run.

CHAPTER

THREE

TWO EXAMPLES

3.1 DeepFlame with DNN

If you choose to use PyTorch as the integratgor and use the compilation flag *–use_pytorch*, you can run examples stored in *$HOME/deepflame-dev/examples/.../pytorchIntegratgor*. To run an example, you first need to source your OpenFOAM:

```
source $HOME/OpenFOAM/OpenFOAM-7/etc/bashrc
```

Then, source your DeepFlame:

```
source $HOME/deepflame-dev/bashrc
```

Next, you can go to the directory of any example case that you want to run. For example:

```
cd $HOME/deepflame-dev/examples/df0DFoam/zeroD_cubicReactor/H2/pytorchIntegratgor
```

This is an example for the zero-dimensional hydrogen combustion with PyTorch as the integrator. Networks used are stored in the *mechanisms* folder, and the inference file is *inference.py*. Configurations regarding DNN are included in *constant/CanteraTorchProperties*.

The case is run by simply typing:

```
./Allrun
```

Note: Users can go to *constant/CanteraTorchProperties* and check if *torch* is switched on. Switch it *on* to run DNN cases, and switch *off* to run CVODE cases.

If you plot PyTorch's result together with CVODE's result, the graph is expected to look like:

3.2 DeepFlame without DNN

CVODE Integrator is the one without the application of Deep Neural Network (DNN). Follow the steps below to run an example of CVODE. Examples are stored in the directory: .. code-block:: bash

    $HOME/deepflame-dev/examples

To run these examples, first source your OpenFOAM, depending on your OpenFOAM path:

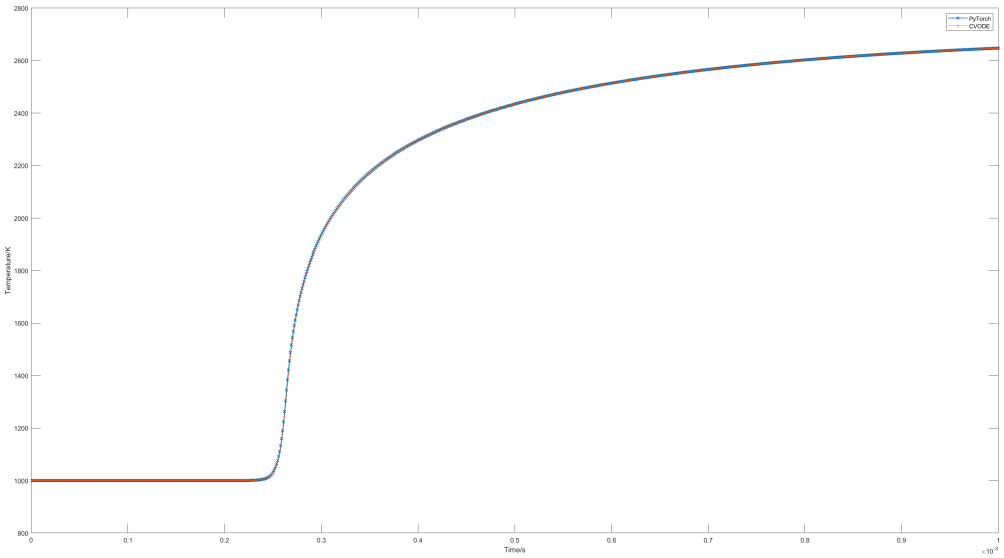


Fig. 1: Visualisation of 0D results from PyTorch and CVODE integrators

```
source $HOME/OpenFOAM/OpenFOAM-7/etc/bashrc
```

Then, source your DeepFlame:

```
source $HOME/deepflame-dev/bashrc
```

Next, you can go to the directory of any example case that you want to run. For example:

```
cd $HOME/deepflame-dev/examples/df0DFoam/zeroD_cubicReactor/H2/cvodeIntegrator
```

This is an example for the zero-dimensional hydrogen combustion with CVODE integrator.

The case is run by simply typing:

```
./Allrun
```

The probe used for post processing is defined in `/system/probes`. In this case, the probe is located at the coordinates (0.0025 0.0025 0.0025) to measure temperature variation with time. If the case is successfully run, the result can be found in `/postProcessing/probes/0/T`, and it can be visualized by running:

```
gnuplot
plot "/your/path/to/postProcessing/probes/0/T"
```

You will get a graph:

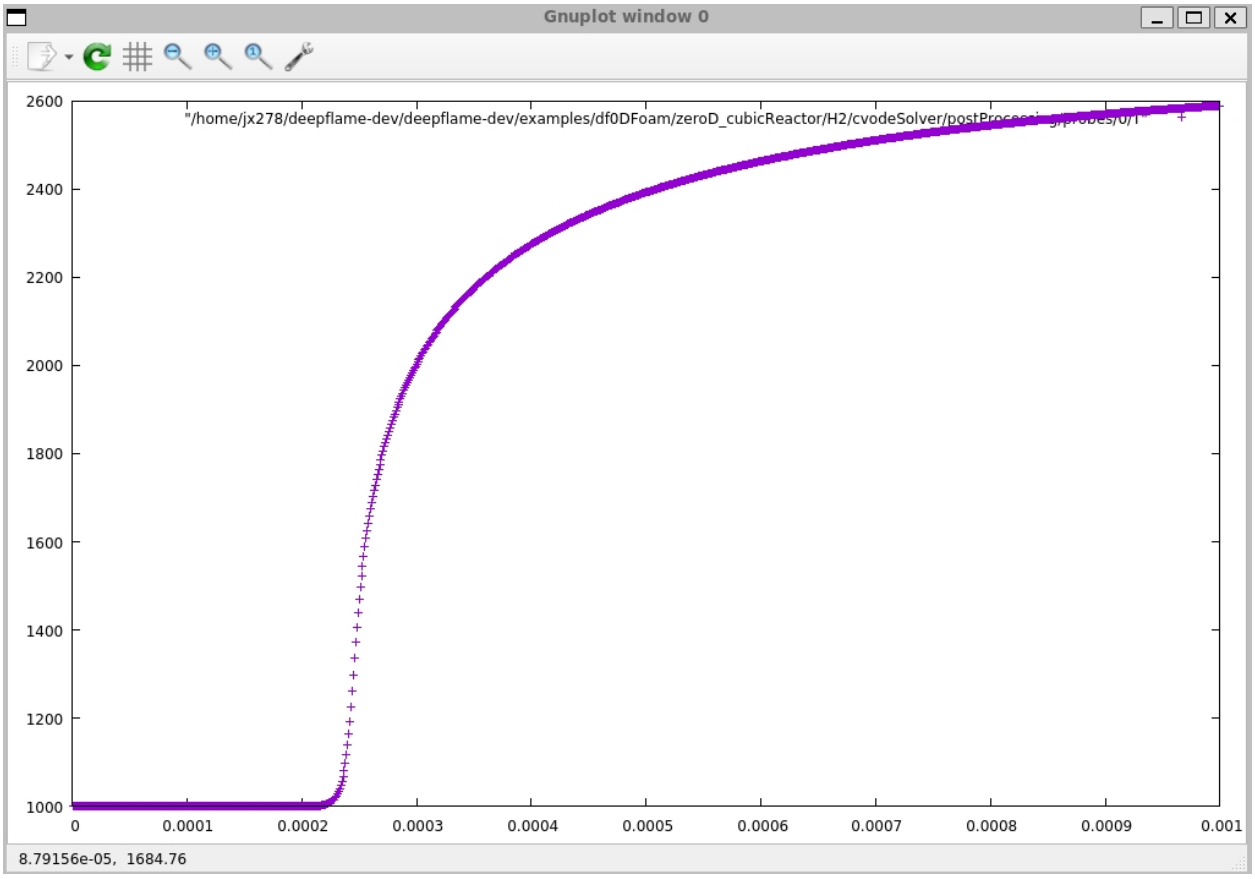


Fig. 2: Visualisation of the zero-dimensional hydrogen combustion result with CVODE integrator

CHAPTER

FOUR

BRIEF INTRODUCTION TO INPUTS

The dictionary `CanteraTorchProperties` is the original dictionary of DeepFlame. It reads in network related parameters and configurations. It typically looks like:

```
chemistry           on;
CanteraMechanismFile "ES80_H2-7-16.yaml";
transportModel "Mix";
odeCoeffs
{
    "relTol"   1e-15;
    "absTol"   1e-24;
}
inertSpecie        "N2";
zeroDReactor
{
    constantProperty "pressure";
}

splittingStretagy false;

TorchSettings
{
    torch on;
    GPU   off;
    log  on;
    torchModel "HE04_Hydrogen_ESH2_GMS_sub_20221101";
    coresPerNode 4;

}
loadbalancing
{
        active  false;
        //log   true;
}
```

In the above example, the meanings of the parameters are:

- `CanteraMechanismFile`: the name of the reaction mechanism file.
- `transportModel`: the default model is *Mix*, but other models including *UnityLewis* and *Multi* are also availabile.
- `constantProperty`: property set to be constant during reaction. It can be set to *pressure* or *volume*.

- `odeCoeffs`: the ode tolerance. 1e-15 and 1e-24 are used for network training, so they should be kept the same when comparing results with and without DNN. Default values are 1e-9 and 1e-15.
- `TorchSettings`: all paramenters regarding the usage of DNN. This section will not be read in CVODE cases.
- `torch`: the switch used to control the on and off of DNN. If users are running CVODE, this needs to be switched off.
- `GPU`: the switch used to control whether GPU or CPU is used to carry out inference.
- `torchModel`: name of network.
- `coresPerNode`: If you are using one node on a cluster or using your own PC, set this parameter to the actual number of cores used to run the task. If you are using more than one node on a cluster, set this parameter the total number of cores on one node. The number of GPUs used is auto-detected.

The dictionary `combustionProperties` is the original dictionary of DeepFlame. It reads in network related parameters and configurations. It typically looks like:

```
combustionModel  flareFGM;//PaSR,EDC

EDCCoeffs
{
    version v2005;
}

PaSRCoeffs
{
   mixingScale
   {
      type   globalScale;//globalScale,kolmogorovScale,geometriMeanScale,dynamicScale
      globalScaleCoeffs
      {
        Cmix  0.01;
      }

      dynamicScaleCoeffs
      {
        ChiType      algebraic;// algebraic; transport;
      }
    }
   chemistryScale
   {
      type  globalConvertion;//formationRate,globalConvertion
      globalConvertionCoeffs
      {
           fuel CH4;
           oxidizer O2;
      }
   }

}

flareFGMCoeffs
{
  buffer           false;
```

(continues on next page)

(continued from previous page)

```
  scaledPV         false;
  combustion       false;
  ignition         false;
  solveEnthalpy    false;
  flameletT        false;
  relaxation       false;
  DpDt             false;
/*ignition         false;
  ignBeginTime     0.1;
  ignDurationTime  0.0;
  x0               0.0;
  y0               0.0;
  z0               0.0;
  R0               0.0;*/
  Sct              0.7;
  bufferTime       0.0;
  speciesName      ("CO");
}
```

In the above example, the meanings of the parameters are:

- `combustionModel`: the name of the combustion model, alternative models include PaSR, EDC, flareFGM.
- `EDCCoeffs, PaSRCoeffs, flareFGMCoeffs`: model cofficients we need to define.
- `mixingScale`: turbulent mixing time scale including globalScale,kolmogorovScale,geometriMeanScale,dynamicScale.
- `ChiType`: algebraic and transport are available for ChiType when selecting dynamicScale.
- `chemistryScale`: chemistry reaction time scale including formationRate,globalConvertion .
- `buffer`: switch for buffer time.
- `scaledPV`:the switch is used to determine whether to use scaled progress variables or not.
- `combustion`:the switch is used to control whether the chemical reactions are on or off.
- `ignition`:the switch is used to control whether the ignition is on or off.
- `solveEnthalpy`:the switch is used to determine whether to solve enthalpy equation or not.
- `flameletT`:the switch is used to determine whether to read flame temperature from table or not.
- `relaxation`:the switch is used to determine whether to use relaxation iteration for transport equations or not.
- `DpDt`:the switch is used to determine whether to include material derivatives or not.
- `ignBeginTime`:beginning time of ignition.
- `ignDurationTime`:duration time of ignition.
- `x0, y0, z0`:coordinate of ignition center.
- `R0`:radius of ignition region.
- `Sct`:turbulent Schmidt number, default value is set as 0.7.
- `speciesName`:name of species we need to lookup.

CHAPTER

FIVE

DF0DFOAM

5.1 Zero-Dimensional ignition reactor

Problem Description

This case simulates the zero-dimensional autoignition under constant-pressure or constant-volume condition. This case confirm the validity of the implementation of chemical reaction source terms in DeepFlame.

Table 1: Operating Conditions in Brief

Mixture	Hydrogen-Air
Equivalence Ratio	1.0
Initial Gas Temperature	1400 K
Initial Gas Pressure	1 atm

Output

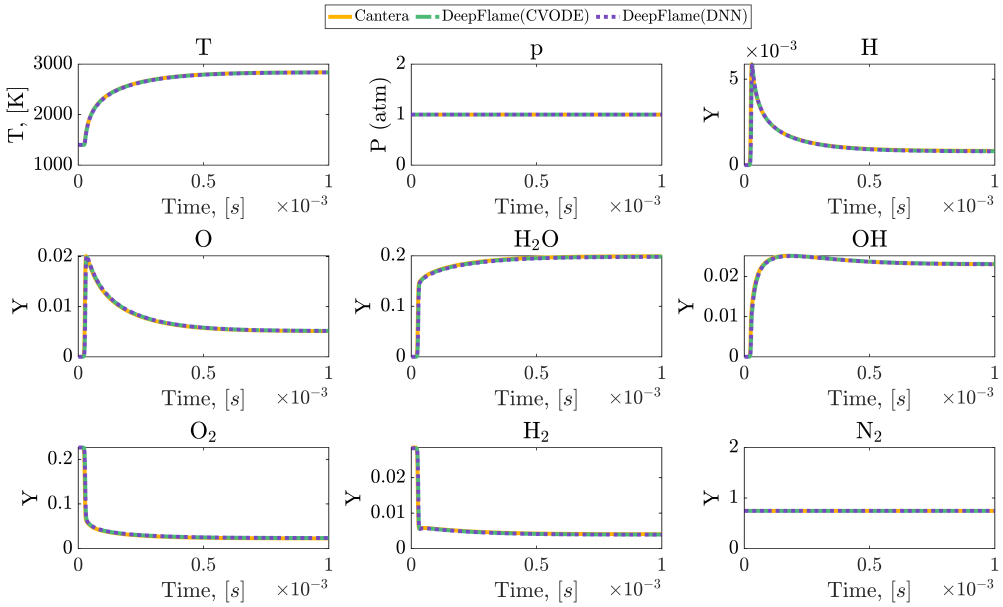


Fig. 1: Results of zero-dimensional constant-pressure autoignition

CHAPTER

SIX

DFLOWMACHFOAM

6.1 One-Dimensional Planar Flame

Problem Description

The case simulates the steady-state 1D freely-propagating flame. The results are able to catch the flame thickness, laminar fame speed and the detailed 1D flame structure. This case demonstrate that the convection-diffusion-reaction algorithms implemented in our solver are stable and accurate.

Table 1: Operating Conditions in Brief

Computational Domain length	0.06 m
Mixture	Hydrogen-Air
Equivalence Ratio	1.0
Inlet Gas Temperature	300 K

Output

6.2 Two-Dimensional Triple Flame

Problem Description

This case simulates the evolution of a 2D non-premixed planar jet flame to validate the capability of our solver for multi-dimensional applications.

Table 2: Operating Conditions in Brief

Computational Domain size (x)	0.03 m * 0.05 m
Jet Composition	H2/N2= 1/3 (fuel jet), Air (co-flow)
Initial Velocity	5 m/s (fuel jet), 1 m/s (co-flow)
Initial Gas Temperature	1400 K (ignition region), 300 K (other area)

Output

The initial condition and the evolution of the jet flame are presented in this figure.

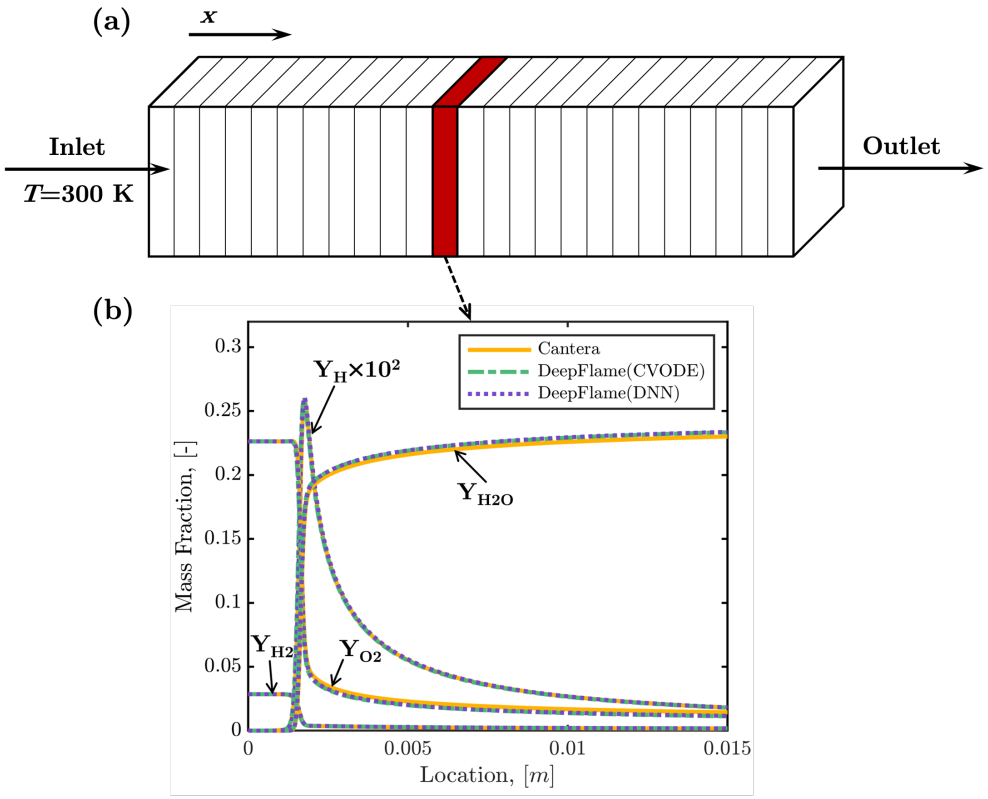


Fig. 1: Numerical setup of one-dimensional premixed flame and the detailed flame structure obtained by our solver

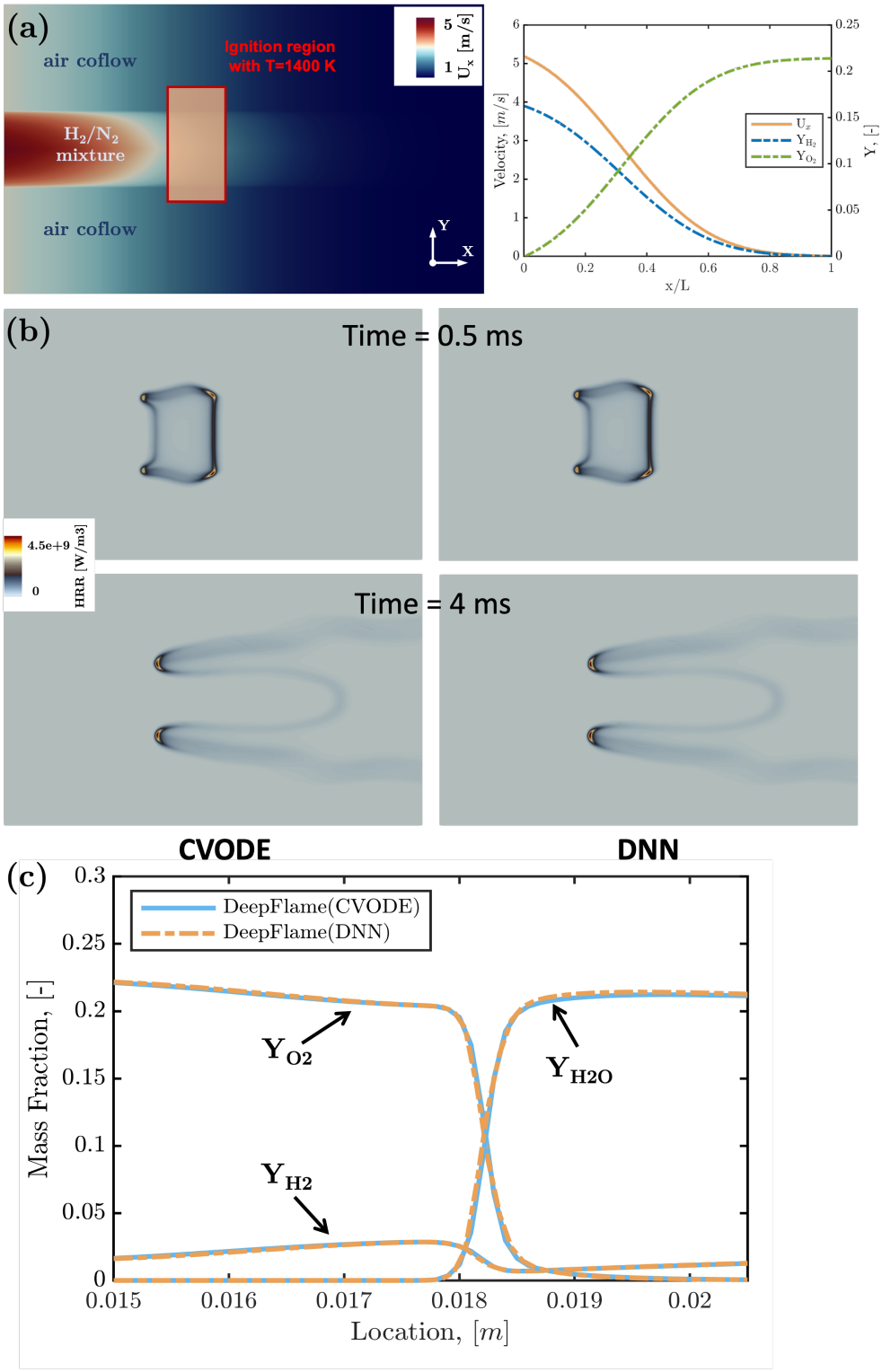


Fig. 2: Simulation results of the two-dimensional jet flame.

6.3 Two-Dimensional Reactive Taylor-Green Vortex

Problem Description

2D reactive Taylor-Green Vortex (TGV) which is simplified from the 3D reactive TGV below is simulated here. It can be used for debugging, validation of DNN models against CVODE solutions and so on.

The initial velocity, pressure and mass fraction fields are set according to a benchmark case established by Abdelsamie et al. The initial temperature of the cold gas is set to be 500 K.

Output

The developed 2D TGV are displayed in the figures below.

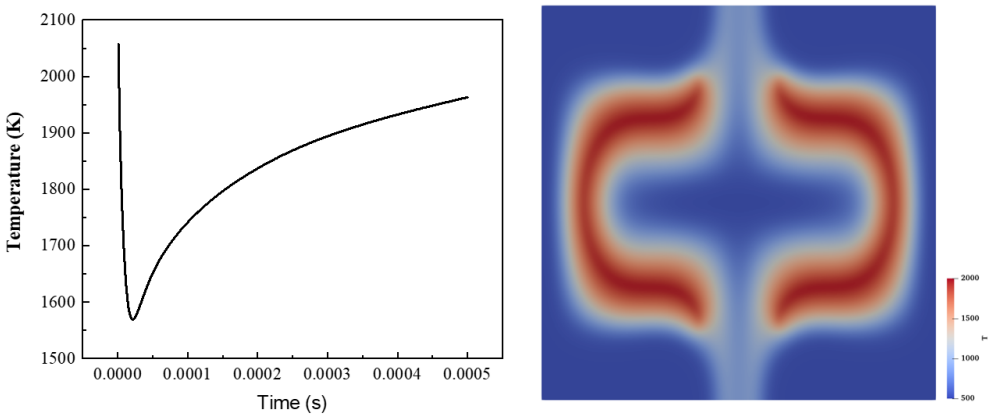


Fig. 3: Profiles of temperature history and contours of temperature and species mass fraction at t = 0.5 ms (two reference time)

Reference

A.Abdelsamie, G.Lartigue, C.E.Frouzakis, D.Thevenin, The taylor-green vortex as a benchmark for high-fidelity combustion simulations using low-mach solvers, Computers & Fluids 223 (2021): 104935.

6.4 Two-Dimensional Flame in Homogeneous Isotropic Turbulence

Problem Description

2D reacting flow with homogeneous isotropic turbulence (HIT) is simulated here. It can be used to simulate kinetic energy dissapation and qualitatively assess turbulence effects on combustion under the circumstances that 3D simulations are computationally prohibitive.

The domain is initially filled with CH4/air mixture with an equivalence ratio of 0.6 at atmospheric pressure. To initialize 2d HIT simulations, turbulence is generated follwing the procedure described by Vuorinen and Keskinen.

Table 3: Operating Conditions in Brief

Computational Domain size (x)	20*pi mm * 20*pi mm
Initial Gas Temperature	1800 K (ignition region), 800 K (other area)
Ignition region	Circle in the domain center with a radius of 1/10 domain length

The figure below shows contour of vorticity at initial time.

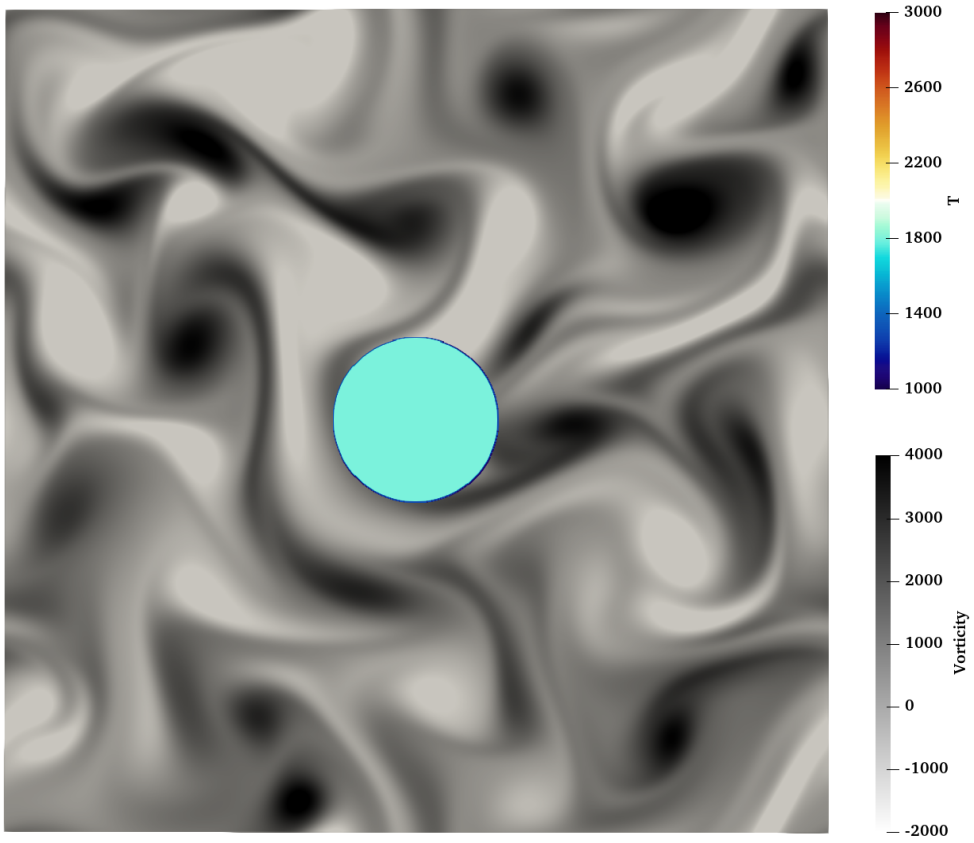


Fig. 4: Initial contours of temperature and vorticity for the 2D reactive HIT

Output

The developed 2D HIT are displayed in the figures below.

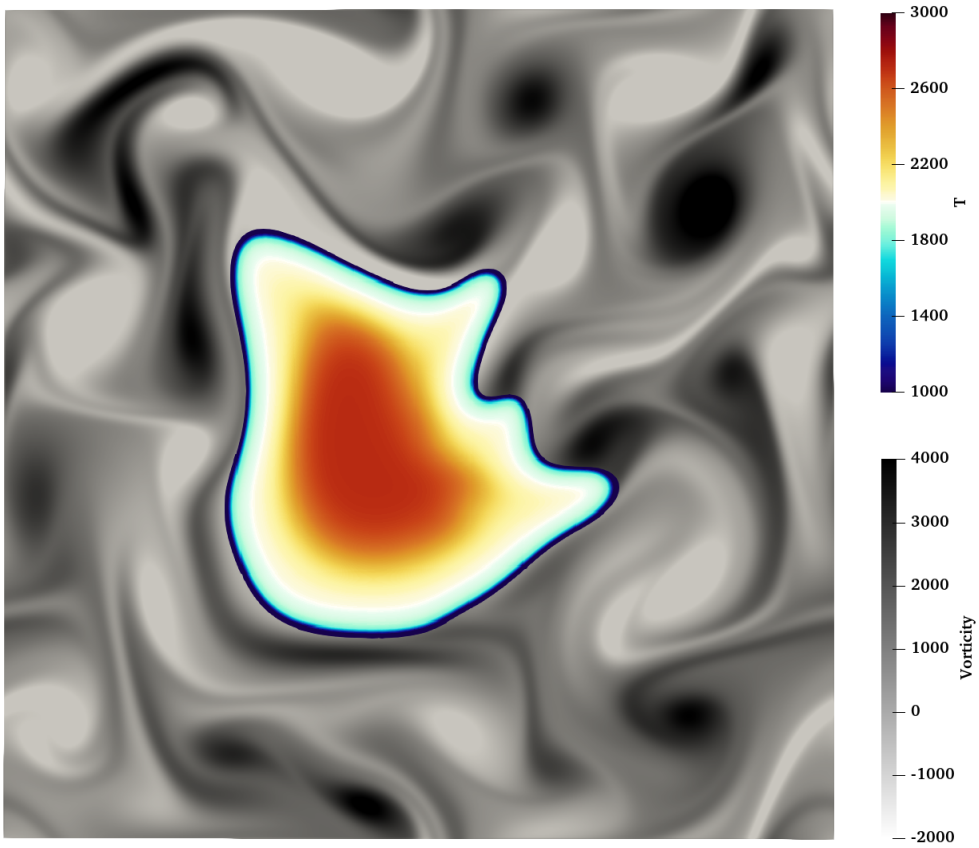


Fig. 5: Contours of temperature and vorticity at t = 1.5 ms

Reference

Vuorinen, Ville, and K. Keskinen. "DNSLab: A gateway to turbulent flow simulation in Matlab." Computer Physics Communications 203 (2016): 278-289.

6.5 Two-Dimensional temporally evolving jet flame

Problem Description

2D planar temporally evolving jet flame is simulated here. Turbulent plane jets are prototypical free shear flows on which fundamental research can expand the overall understanding of turbulent flows. Here, we develope a two-dimensional temporally evolving turbulent jet, with considering transport and mixing processes of scalars in turbulent shear flows.

The domain is initially filled with unburnt CH4/air mixture at the region where 7.5 mm < y < 8.5 mm and burnt gas CO2/H2O/N2 elsewhere. To initialize turbulent shear flow, the internal velocity field is generated from a jet flow simulation with turbulence generator.

Table 4: Operating Conditions in Brief

Computational Domain size (x)	16 mm * 16 mm
Initial Gas Temperature	900 K (unburnt gas), 2500 K (burnt gas)

The figure below shows the computation domain and initial conditions.

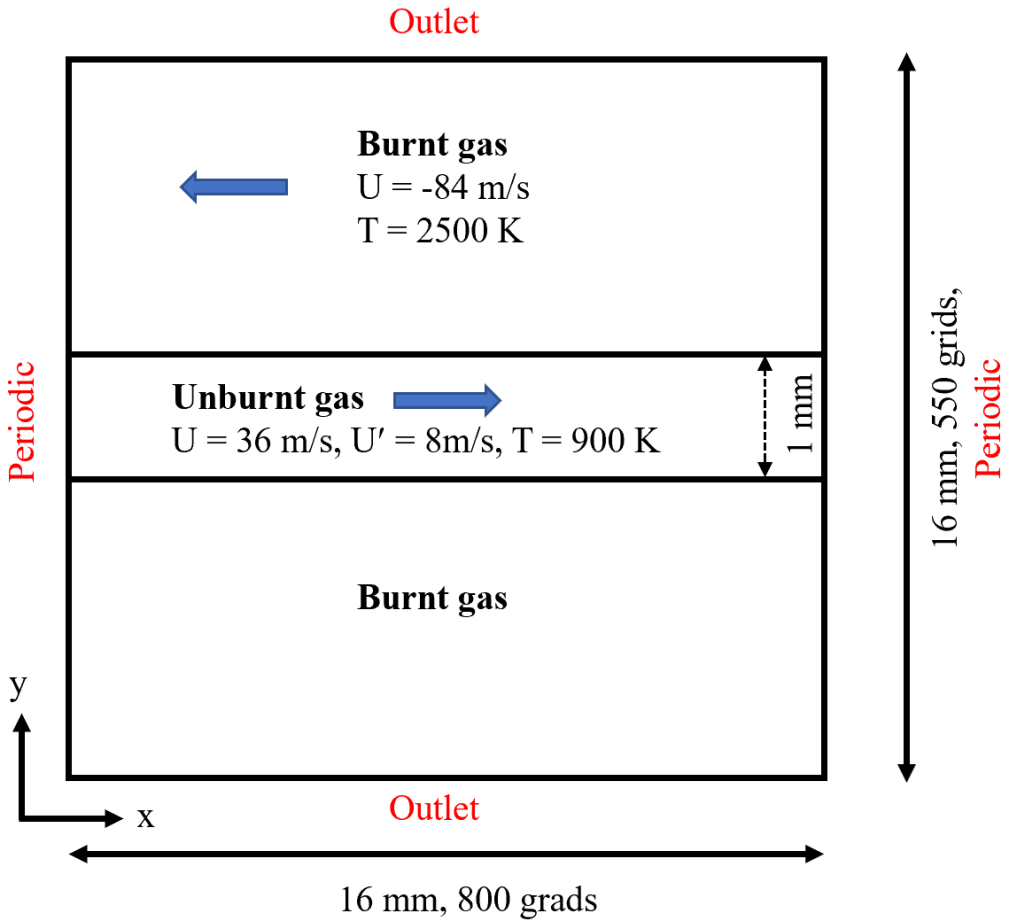


Fig. 6: Computational domain and initial conditions

Output

The temporally evolving jet flame are displayed in the figures below.

Reference

M.Saito, J. Xing, J. Nagao, R. Kurose. “Data-driven simulation of ammonia combustion using neural ordinary differential equations (NODE).” Applications in Energy and Combustion Science (2023): 100196.

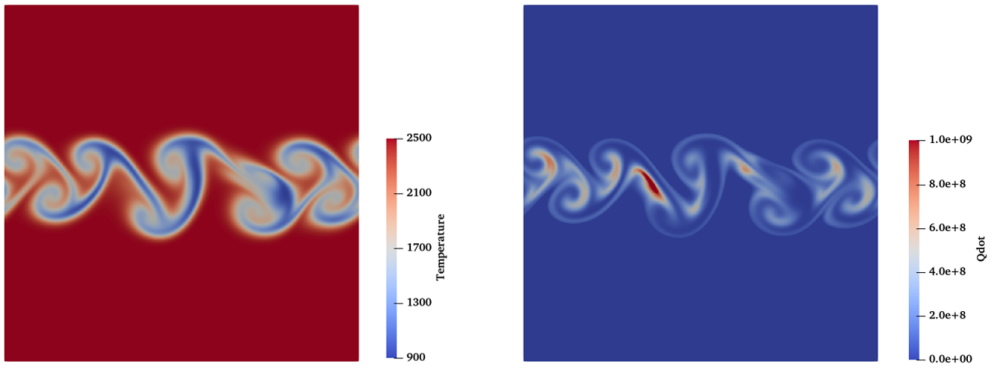


Fig. 7: Contours of temperature and Qdot at t = 0.1 ms

6.6 Three-Dimensional Reactive Taylor-Green Vortex

3D reactive Taylor-Green Vortex (TGV) which is a newly established benchmark case for reacting flow DNS codes is simulated here to evaluate the computational performance of our solver.

The initial fields are set according to a benchmark case established by Abdelsamie et al. The figure below shows contours of vorticity magnitude and temperature as well as the x-direction profiles of species at initial time.

Output

The developed TGV are displayed in the figures below.

Reference

A.Abdelsamie, G.Lartigue, C.E.Frouzakis, D.Thevenin, The taylor-green vortex as a benchmark for high-fidelity combustion simulations using low-mach solvers, Computers & Fluids 223 (2021): 104935.

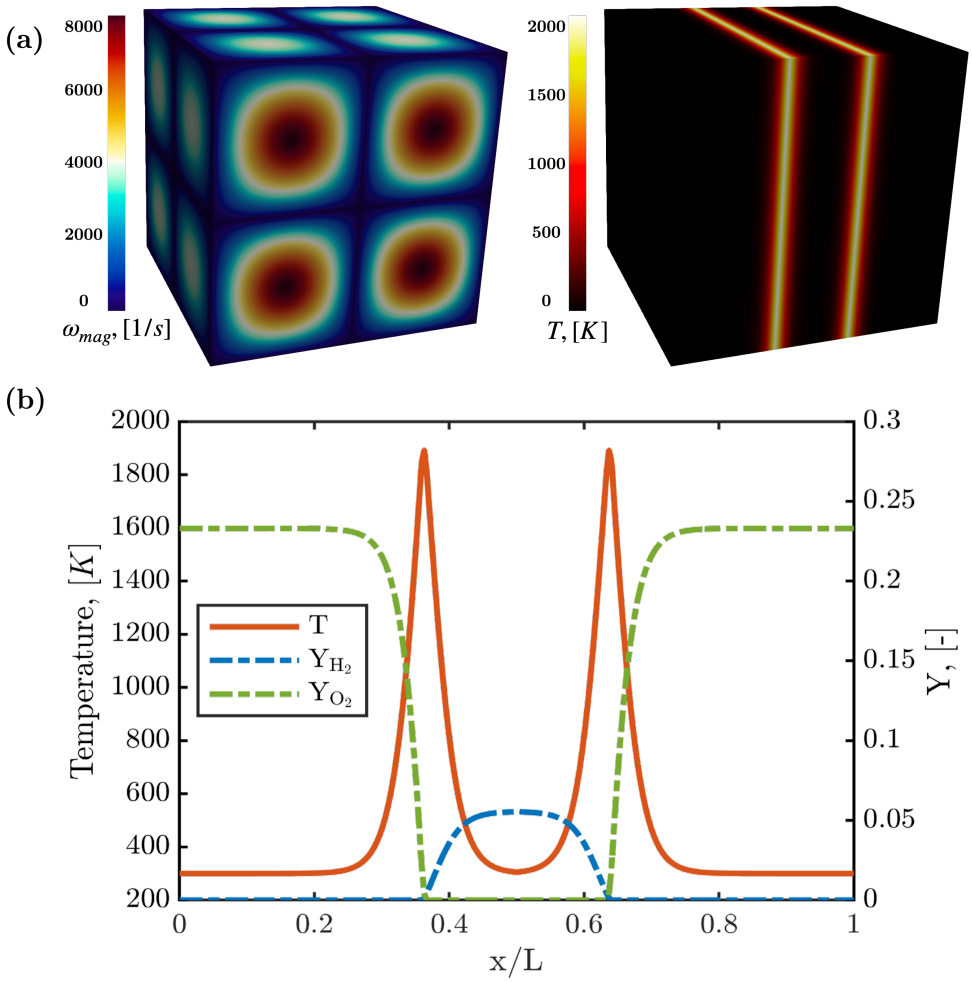


Fig. 8: Initial contours and profiles of vorticity magnitude, temperature, and species mass fraction for the reactive TGV

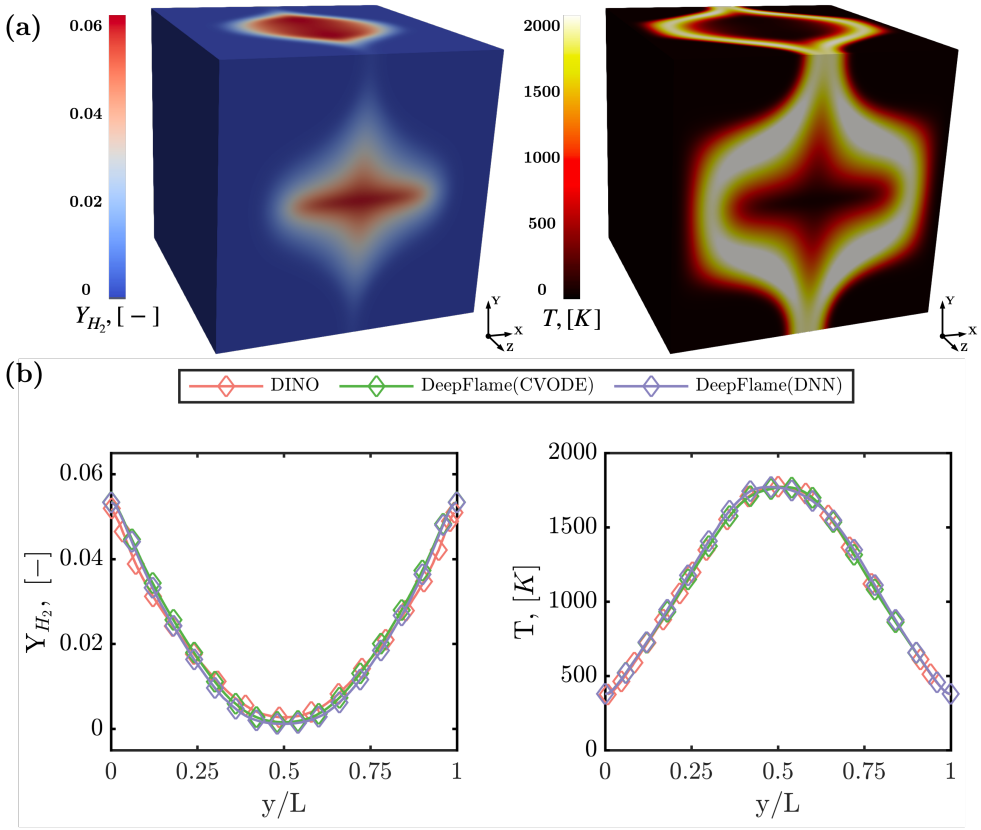


Fig. 9: Contours and profiles of temperature and species mass fraction at t = 0.5 ms

CHAPTER

SEVEN

DFHIGHSPEEDFOAM

7.1 One-Dimensional Reactive Shock Tube

Problem Description

The case simulates supersonic inlet flow hitting the wall and then reflected to ignite the premixed gas. The reactive wave will catch the reflected shock wave. This case can also verify the accuracy of our solver in capturing the interaction of convection and reaction.

Table 1: Operating Conditions in Brief

Chamber size (x)	0.12 m
Initial Gas Density	0.072 kg/m^3 (x<=0.06 m), 0.18075 kg/m^3 (x>0.06 m)
Initial Gas Pressure	7173 Pa (x<=0.06 m), 35594 Pa (x>0.06 m)
Initial Gas Velocity	0 m/s (x<=0.06 m), -487.34 m/s (x>0.06 m)
Ideal Gas Composition (mole fraction)	H2/O2/Ar = 2/1/7

Output

Reference

E S Oran, T R Young, J P Boris, A Cohen, Weak and strong ignition. i. Numerical simulations of shock tube experiments, Combustion and Flame 48 (1982) 135-148.

R J Kee, J F Grcar, M D Smooke, J A Miller, E Meeks, Premix: A fortran program for modeling steady laminar one-dimensional premixed flames, Sandia National Laboratories.

7.2 One-Dimensional H2/Air Detonation

Problem Description

Detonation propagation contains a complex interaction of the leading shock wave and auto-igniting reaction, showing the coupling of shock wave and chemical reaction. This case aims to validate the accuracy of this solver in capturing this process and the propagation speed.

Table 2: Operating Conditions in Brief

Chamber size (x)	0.5 m
Initial Gas Pressure	90 atm (hot spot), 1 atm (other area)
Initial Gas Temperature	2000 K (hot spot), 300 K (other area)
Ideal Gas Composition (mole fraction)	H2/O2/N2 = 2/1/3.76 (homogeneous stoichiometric mixture)

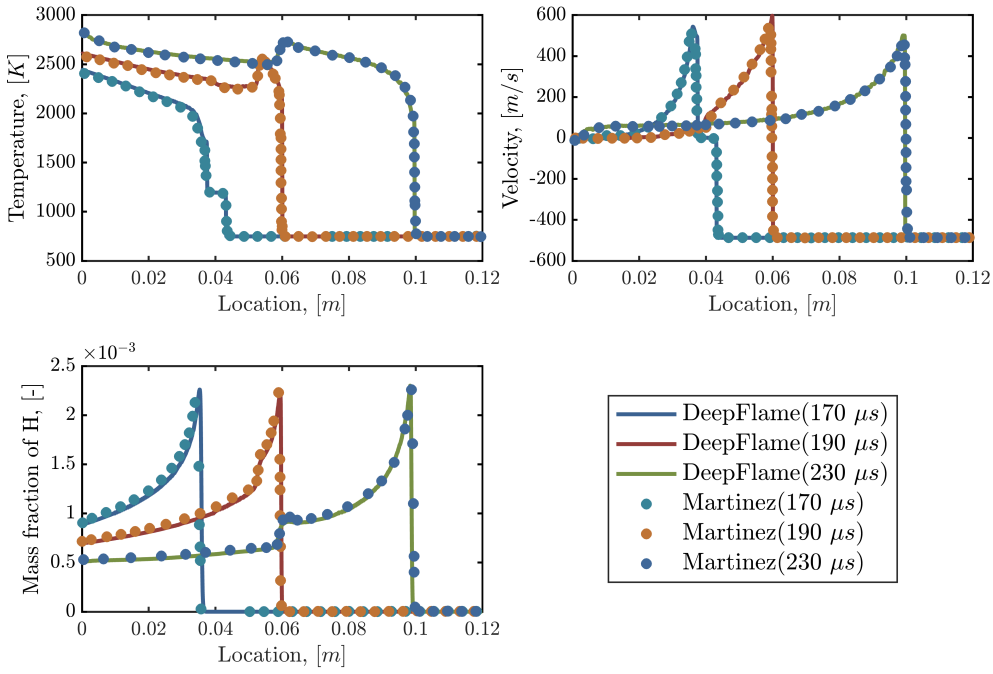


Fig. 1: Result of one-dimensional reactive shock tube

Output

Reference

J Li, Z Zhao, A Kazakov, F L Dryer, An updated comprehensive kinetic model of hydrogen combustion, International Journal of Chemical Kinetics 36 (2004) 566-575.

7.3 Two-Dimensional H2/Air Detonation

Problem Description

Detonation propagation contains a complex interaction of the leading shock wave and auto-igniting reaction, and two-dimensional detonation can further reveal the interaction of shear waves and shock waves.

Table 3: Operating Conditions in Brief

Chamber size (x)	0.2 m * 0.01 m
Initial Gas Pressure	100 atm (three hot spot), 1 atm (other area)
Initial Gas Temperature	2000 K (three hot spot), 300 K (other area)
Ideal Gas Composition (mole fraction)	H2/O2/N2 = 2/1/7 (homogeneous stoichiometric mixture)

Output

Triple points can be seen clearly in the picture below.

In the picture below, during the propagation of detonation wave, we can see that the size of cells gradually became stable.

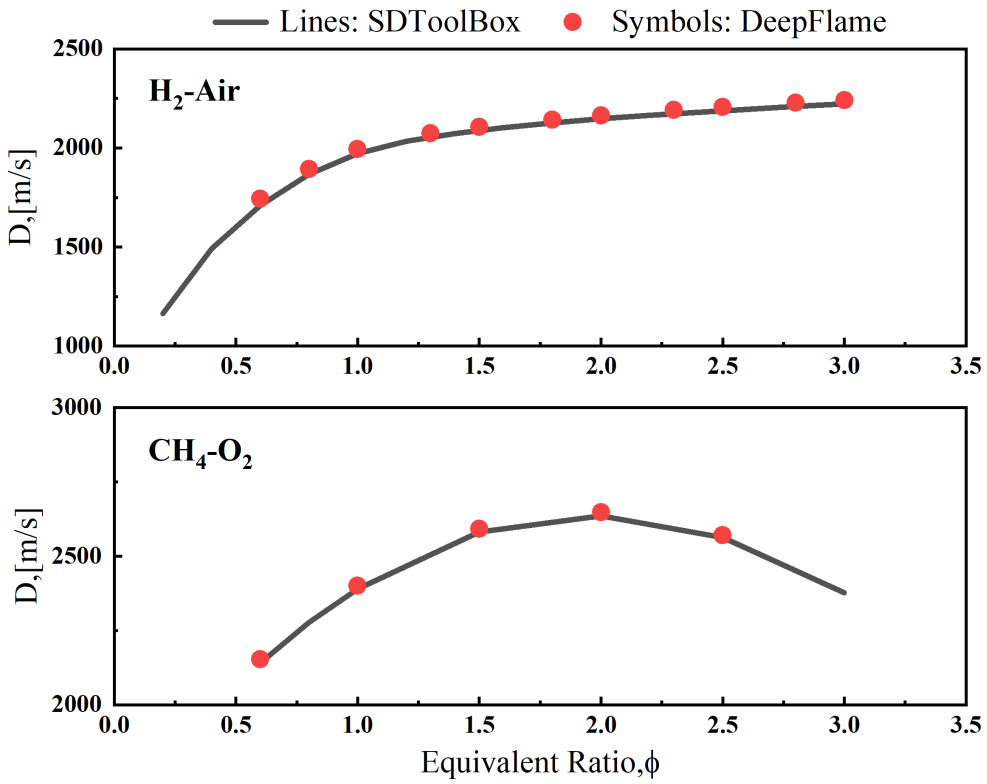


Fig. 2: Result of one-dimensional H2/air detonation

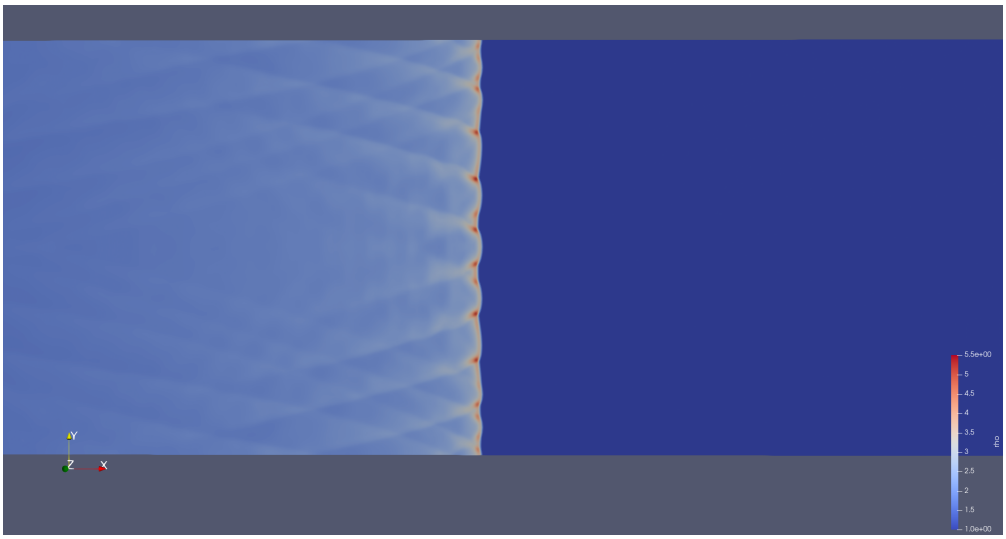


Fig. 3: Density field of two-dimensional H2 detonation

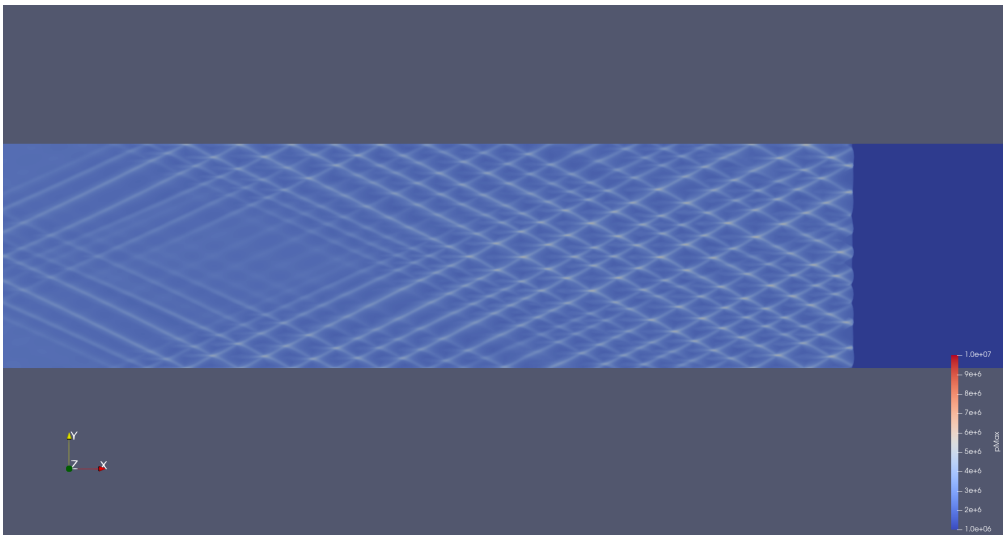


Fig. 4: History of maximum pressure during detonation propagation

Reference

C J Jachimowski, An Analytical Study of the Hydrogen-Air Reaction Mechanism with Application to Scramjet Combustion, NASA TP-2791, Feb. 1988.

CHAPTER

EIGHT

DFSPRAYFOAM

8.1 aachenBomb

Problem Description

This case simulates combustion inside a constant volume chamber based on an experimental setup at RWTH Aachen University. It can mimic, for example, the beginning of power stroke in a four-stroke diesel engine.

Table 1: Operating Conditions in Brief

Chamber size (xyz)	0.02×0.1×0.02m^3
Initial Gas Temperature	800K
Initial Gas Pressure	5MPa
Initial Gas Composition (mass fraction)	23.4% O2, 76.6% N2
Fuel	n-heptane
Fuel Temperature at the Nozzle	320K
Fuel Injection Duration	1.25ms
Total Injection Mass	6mg

Configurations Different from OpenFOAM Case

Cantera is used instead of the built-in modules of OpenFOAM to solve the chemical reactions. Therefore, a chemical mechanism file in YAML format is required in the case directory, and the full name of the mechanism file ("xxx.yaml") should be the entry after the keyword **CanteraMechanismFile** in *constant/CanteraTorchProperties*. Non-reacting simulation can be conducted by switching the entry after the keyword **chemistry** from **on** to **off** in *constant/CanteraTorchProperties*.

Results

8.2 Sydney Spray Burner

Problem Description

A 2D-wedge case is used to simulate the dilute spray combustion processes inside a spray burner based on the experimental setup at Sydney University. It has been widely used for the study of droplet-chemistry-turbulence interaction.

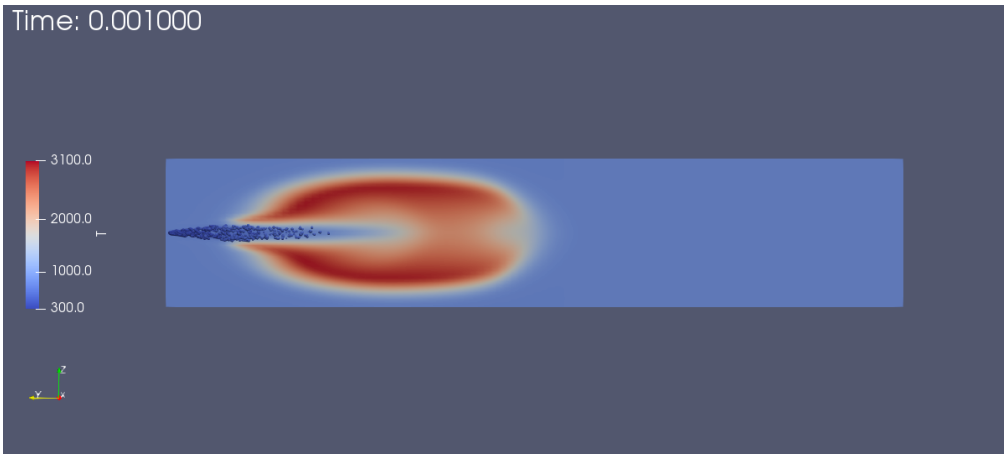


Fig. 1: Visualization of temperature contours with large particle distribution

Table 2: Operating Conditions in Brief (Details can be found for Case EtF4 in [1])

Chamber size (wedge, 4 degree)	0.104×0.500m^2
Initial Gas Temperature in Domain	293K
Initial Gas Temperature at Fuel Inlet	267K
Initial Gas Temperature at Pilot Inlet	2493K
Initial Gas Pressure	1Bar
Initial Gas Composition in Domain(mass fraction)	23.4% O2, 76.6% N2
Fuel	C2H5OH
Mass-flow Rate of Air Carrier	150g/min
Liquid Fuel Injection Rate	23.4g/min
Bulk Jet Velocity	24m/s
Bulk Burned Pilot Velocity	11.6m/s

Configurations Different from OpenFOAM Case

Cantera is used instead of the built-in modules of OpenFOAM to solve the chemical reactions. Therefore, a chemical mechanism file in YAML format is required in the case directory, and the full name of the mechanism file ("xxx.yaml") should be the entry after the keyword **CanteraMechanismFile** in *constant/CanteraTorchProperties*. Non-reacting simulation can be conducted by switching the entry after the keyword **chemistry** from **on** to **off** in *constant/CanteraTorchProperties*.

Note

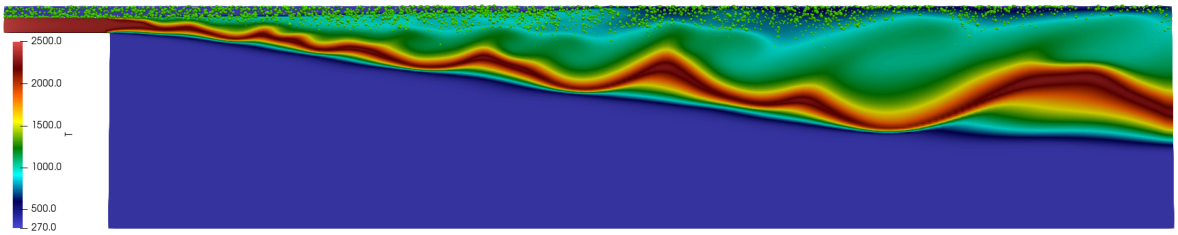
When running a wedge case with OpenFOAM, you may come across an error "**Hitting a wedge patch should not be possible**". One solution is to comment out the lines with the "**FatalErrorIn**" call in **src/lagrangian/basic/particle/particleTemplates.C**. Details can be found in the thread [2].

Results

Reference

[1] Gounder, James Dakshina, Agisilaos Kourmatzis, and Assaad Rachid Masri. "Turbulent piloted dilute spray flames: Flow fields and droplet dynamics." Combustion and flame 159.11 (2012): 3372-3397.

[2]https://www.cfd-online.com/Forums/openfoam/89003-3d-spray-vs-axisymmetric-spray-dieselfoam.html


2500.0
2000.0
1500.0
T
1000.0
500.0
270.0

CHAPTER

NINE

REACTION MECHANISM CONVERTION

DeepFlame uses *yaml* reaction mechanisms, which are compatible with Cantera. The following command lines can be used to convert *chemkin* mechanisms into *yaml* format.

```
conda create --name ct-env --channel conda-forge cantera
conda activate ct-env
ck2yaml --input=chem.inp --thermo=therm.dat --transport=tran.dat
```

Note: Users will need to create a new conda environment other than the one used for DeepFlame's dependencies, and the channel needs to be `conda-forge`. Otherwise, there might be an error regarding shared library, `libmkl_rt.so.2`.

More detailed instruction of converting mechanisms can be found on Cantera official website.

CHAPTER

TEN

FLAME SPEED

`flameSpeed.C` is another utility in DeepFlame. The flame is located at the maximum temperature gradient at a certain time, and its speed is equal to the maximum gradient porpagation speed subtracting the inlet speed. To use this utility, simply run the commands below after the simulation.

```
runApplication reconstructPar
flameSpeed
```

A log containing flame thickness, flame location, flame proagation speed, and flame speed at each time step will be presented.

Note: This utility only applies to one-dimensional cases. Similar logs can also exit when it is run for two or three dimensional cases, but results are not physical.

CHAPTER

ELEVEN

DEVELOPERS TEAM

The current DeepFlame developers come from the following research groups/affiliations:

- Peking University (Lead PI: Zhi X. Chen)
- AI for Science Institute (AISI), Beijing

CHAPTER

TWELVE

COLLABORATORS TEAM

- Shanghai Jiao Tong University (Lead PI: Zhi-Qin John Xu)
- Southern University of Science and Technology (Lead PI: Tianhan Zhang)

CHAPTER

THIRTEEN

HOW TO CITE

If you use DeepFlame for a publication, please use the citation:

Runze Mao, Minqi Lin, Yan Zhang, Tianhan Zhang, Zhi-Qin John Xu, Zhi X. Chen. DeepFlame: A deep learning empowered open-source platform for reacting flow simulations. Comput. Phys. Commun. 291:108842 (2023). doi:10.1016/j.cpc.2023.108842

If you have used the DNN model provided from us, please use the citation:

Han Li, Ruixin Yang, Min Zhang, Runze Mao, Zhi X. Chen. A comprehensive study on the accuracy and generalization of deep learning-generated chemical ODE integrators. arXiv:2312.16387

CHAPTER

FOURTEEN

LICENSE

The project DeepFlame is licensed under GNU General Public License v3.0

CHAPTER

FIFTEEN

SUBMITTING A PULL REQUEST

We welcome contributions from the open source community. The main approach to communicate with and to make contribution to DeepFlame is to open a pull request.

1. Fork the DeepFlame repository.
2. Pull your forked repository, and create a new git branchmake to your changes in it:

```
git checkout -b my-fix-branch
```

3. Coding your patch
4. After tests passed, commit your changes with a proper message.
5. Push your branch to GitHub:

```
git push origin my-fix-branch
```

6. In GitHub, send a pull request with `deepmodeling/deepflame-dev` as the base repository.
7. After your pull request is merged, you can safely delete your branch and sync the changes from the main (upstream) repository:

- Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

```
git push origin --delete my-fix-branch
```

- Check out the master branch:

```
git checkout develop -f
```

- Delete the local branch:

```
git branch -D my-fix-branch
```

- Update your master with the latest upstream version:

```
git pull --ff upstream develop
```