DeepFlame: A deep learning empowered open-source platform for reacting flow simulations

Runze Mao\textsuperscript{a,b}, Minqi Lin\textsuperscript{a,b}, Yan Zhang\textsuperscript{c,d,*}, Tianhan Zhang\textsuperscript{e,b}, Zhi-Qin John Xu\textsuperscript{f,g,b}, Zhi X. Chen\textsuperscript{a,b,*}

\textsuperscript{a}State Key Laboratory of Turbulence and Complex Systems, Aeronautics and Astronautics, College of Engineering, Peking University, Beijing, 100871, China
\textsuperscript{b}AI for Science Institute, Beijing, 100080, China
\textsuperscript{c}CAEP Software Center for High Performance Numerical Simulation, Beijing 100088, China
\textsuperscript{d}Institute of Applied Physics and Computational Mathematics, Beijing 100088, China
\textsuperscript{e}Department of Mechanics and Aerospace Engineering, SUSTech, Shenzhen, 518055, China
\textsuperscript{f}Institute of Natural Sciences, School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai, 200240, China
\textsuperscript{g}MOE-LSC and Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai, 200240, China

Abstract

Recent developments in deep learning have brought many inspirations for the scientific computing community and it is perceived as a promising method in accelerating the computationally demanding reacting flow simulations. In this work, we introduce DeepFlame, an open-source C++ platform with the capabilities of utilising machine learning algorithms and pre-trained models to solve for reactive flows. We combine the individual strengths of the computational fluid dynamics library OpenFOAM, machine learning framework Torch, and chemical kinetics program Cantera. The complexity of cross-library function and data interfacing (the core of DeepFlame) is minimised to achieve a simple and clear workflow for code maintenance, extension and upgrading. As a demonstration, we apply our recent work on deep learning for predicting chemical kinetics (Zhang et al. Combust. Flame vol. 245 pp. 112319, 2022) to highlight the potential of machine learning in accelerating reacting flow simulation. A thorough code validation is conducted via a broad range of canonical cases to assess its accuracy and efficiency. The results demonstrate that the convection-diffusion-reaction algorithms implemented in DeepFlame are robust and accurate for both steady-state and transient processes. In addition, a number of methods aiming to further improve the computational efficiency, e.g. dynamic load balancing and adaptive mesh refinement, are explored. Their performances are also evaluated and reported. With the deep learning method implemented in this work, a speed-up of two orders of magnitude is achieved in a simple hydrogen ignition case when performed on a medium-end graphics processing unit (GPU). Further gain in computational efficiency is expected for hydrocarbon and other complex fuels. A similar level of acceleration is obtained on an AI-specific chip – deep computing unit (DCU), highlighting the potential of DeepFlame in leveraging the next-generation computing architecture and hardware.

Keywords: Computational fluid dynamic, Compressible reacting flow, Machine learning, Chemical kinetics, High performance computing

*Corresponding author.
E-mail addresses: zhang.yan2@iapcm.ac.cn (Y. Zhang), chenzhi@pku.edu.cn (Z.X. Chen)

Preprint submitted to Elsevier

October 18, 2022
Program Title: DeepFlame

Nature of problem: Solving chemically reacting flows with direct (quasi-direct) simulation methods is usually troubled by the following problems: 1. as the widely-used CFD toolbox, OpenFOAM features poor ODE solvers for chemistry and oversimplified transport models, yielding non-negligible errors in simulation results; 2. the chemical source term evaluation is the most computationally expensive and usually accounts for more than 80% of total computing time.

Solution method: An open-source platform bringing together the individual strengths of OpenFOAM, Cantera and PyTorch libraries is built in this study. In the present implementation, CVODE solvers, detailed transport models and deep learning algorithms are all adopted to assist the simulation of reacting flow. Note that here machine learning is introduced in combination with heterogeneous computing to accelerate the most demanding solving procedure for chemical source term evaluation.

1. Introduction

Computational Fluid Dynamics (CFD) tools for simulating reacting flows are crucial in developing less-polluting and highly-efficient energy and propulsion technologies [1]. While the turbulent flow and chemical reactions are strongly coupled in practical devices, modelling the multi-scale, multi-phase and multi-species physio-chemical processes under engine-relevant conditions remain a scientific challenge. In addition, simulating reacting flows in strong turbulence with the scales and species fully resolved (known as direct numerical simulation, DNS) requires quite demanding computational resources. Simplified modelling methods such as Reynolds-averaged Navier-Stokes (RANS) approach and large eddy simulation (LES) mostly rely on statistical or topological models to impose physical assumptions to accelerate simulations. However, the generalisation abilities of these models have long been the major issue limiting the practical application of reacting flow simulations [2].

To resolve the above dilemma of accuracy versus efficiency, the recent rapid growth in Artificial Intelligence (AI) for Science, particularly in machine learning has brought new perspectives for accelerating simulation of reactive flows with accurate models and detailed chemistry. As a pioneer work, Christo et al. [3] adopted Artificial neural network (ANN) in the joint PDF/Monte Carlo simulation of H2/CO2 turbulent jet diffusion flames to predict chemical kinetics. Blasco et al. [4] trained more than one ANNs based on a typical combustion simulation to capture the changes of species composition at various time steps, so that the reaction rates can be directly obtained by ANNs instead of solving ordinary differential equations (ODEs) or accessing look-up table. Sen et al. [5] successfully
adopted ANNs as a chemical kinetic integrator for LES of turbulent flame. They found ANN exhibits satisfying behaviour both in memory and time efficiency. Wan et al. [6] trained a deep neural network (DNN) based on the turbulent micro-mixing data to predict the reaction rates. The DNN was used in simulating a turbulent non-premixed syngas oxy-flame and obtained a considerable speed-up. Yao et al. [7] adopted gradient boosted decision tree (GBDT) as a machine learning approach to directly solve the chemistry ODEs and gained a speed-up of one order of magnitude. More recently, to improve the generalisation ability of DNN, Zhang et al. [8] proposed a new sampling method for collecting multi-scale combustion data. The neural network trained from such data set has been confirmed to be accurate and efficient in predicting reaction rates under various conditions. Besides, DNN was also extended to the high-dimensional tabulation of flamelets and effectively reduced the memory requirement [9–12].

It is widely acknowledged that the growing success of deep learning is built upon open-source and data sharing culture in the scientific and industrial communities. However, to the best of our knowledge, most of the studies mentioned above were conducted using in-house codes. Despite the promising potential of machine learning in this field, the related works are still limited because of the lack of the powerful tools and platforms leveraging the growing assets in fluid dynamics, machine learning and chemical kinetics communities. Nowadays, the advantages of open-source community in developing new technology has been widely proved. A number of open-source frameworks for machine learning (e.g. TensorFlow[13] and Torch[14]), CFD (e.g. OpenFOAM[15]), and chemical kinetics computation (e.g. Cantera[16]) are available for users and developers from both the scientific and industrial communities. However, it is necessary to build a reacting flow simulation platform that brings together the individual strengths of the CFD, machine learning and chemical kinetics open-source communities, so that the cross-disciplinary research interaction and code development can be facilitated.

With this motivation, the main objective of the present work is to develop an open-source CFD platform named DeepFlame for simulating reacting flows with capabilities of utilising state-of-the-art machine learning algorithms and libraries. Briefly, DeepFlame integrates existing libraries and organises the computational tasks in the following manner. Tools and functions for solving general continuum fluid flow problems are called or derived from OpenFOAM, including flow field data-structure, numerical discretisation, iteration for linear solvers, MPI-based parallel computing, and pre-/post-processing; chemical mechanism related I/O and multi-species thermochemistry data-structure and property calculation are handled by Cantera; the deep learning framework libTorch (Torch C++ API) is coupled in DeepFlame for the manipulation of input/output tensor-format data and inference of deep neural network models. In addition, methods aiming to improve simulation efficiency such as dynamic load balance (DLB) and adaptive mesh refinement (AMR) have also been implemented. A preliminary heterogeneous computing approach is available in this version, where AI acceleration infrastructure (i.e. GPU) can be used to further magnify the simulation speed-up when machine learning models are activated.

---

1In some papers the training and testing codes were released in supplementary material or uploaded onto GitHub for reproducibility. However, the CFD codes in which the pre-trained model deployment and the following a posteriori assessment were carried out remain difficult to access.
The structure of this paper is as follows. In Section 2, we discuss the governing equations solved by the different flow solvers implemented in DeepFlame, where the approaches for obtaining chemistry source terms are emphasised. In Section 3, we introduce the implementation details including the code structure and algorithms. In Section 4, we conduct a broad range of canonical cases to validate the solvers for different flow conditions. In Section 5, the computational performance of DeepFlame is evaluated. Finally, the conclusions and further works are summarised in Section 6.

2. Theoretical Background

2.1. Governing Equations

To directly solve the compressible reacting flows with $N$ number of species, the conservation equations of mass, momentum, species and energy used in this work, in common tensorial notations, are given by

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0,$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_i},$$

$$\frac{\partial \rho Y_\alpha}{\partial t} + \frac{\partial \rho u_i Y_\alpha}{\partial x_i} = -\frac{\partial \rho Y_\alpha}{\partial x_i} V_\alpha + \dot{\omega}_\alpha,$$

$$\frac{\partial \rho H}{\partial t} + \frac{\partial (\rho u_i H)}{\partial x_i} = \frac{\partial p}{\partial t} - \frac{\partial q_i}{\partial x_i} + \frac{\partial}{\partial x_j} \left[ (\tau_{ij} - p\delta_{ij}) u_i \right],$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial (\rho u_i E)}{\partial x_i} = -\frac{\partial q_i}{\partial x_i} + \frac{\partial}{\partial x_j} \left[ (\tau_{ij} - p\delta_{ij}) u_i \right].$$

where $t$ is time, $u_j$ and $x_j$ are the velocity component and Cartesian spatial coordinate in the $j$ direction respectively, $\rho$ is mixture mass density, and $p$ is pressure. In Eq. (3), $Y_\alpha$ and $\dot{\omega}_\alpha$ are mass fraction and net reaction rate of the $\alpha$-th species, respectively. In Eq. (4), $H = h + \frac{1}{2} u_i u_i$ and $E = e + \frac{1}{2} u_i u_i$ are the total enthalpy (absolute enthalpy + kinetic energy) and total energy (internal energy + kinetic energy) respectively, and $q_i$ is energy flux in direction $i$. Depending on the flow velocity with respect to the speed of sound, different forms of energy are solved and this will be described in detail later in §2.2.

The viscous tensor $\tau_{ij}$ in Eq. (2), by applying Stokes’ hypothesis, is written as

$$\tau_{ij} = -\frac{2}{3} \mu \frac{\partial u_i}{\partial x_k} \delta_{ij} + \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right).$$

The dynamic viscosity of mixture, $\mu$, is calculated via the Wilke mixture formulation:

$$\mu = \sum_\alpha \frac{\mu_\alpha X_\alpha}{\sum_\beta \Phi_{\alpha,\beta} X_\beta}$$
with \( \mu_\alpha \) and \( X_\alpha \) being the dynamic viscosity and mole fraction of the \( \alpha \)-th species respectively, and \( \Phi_{\alpha,\beta} \) is computed using

\[
\Phi_{\alpha,\beta} = \frac{1 + \sqrt{\left( \frac{\mu_\alpha}{\mu_\beta} \right) \left( \frac{M_\beta}{M_\alpha} \right)}^2}{\sqrt{8} \sqrt{1 + M_\alpha/M_\beta}},
\]

(7)

where \( M_\alpha \) is the mole mass of the \( \alpha \)-th species.

In the species transport equations (see Eq. (3)), the reaction rates, \( \dot{\omega}_\alpha \), can be obtained from the Arrhenius Law and on-the-fly integration or a pre-trained deep neural network, which will be introduced in § 2.3. The diffusion velocity of the \( \alpha \)-th species in \( i \)-th direction, \( V_{\alpha,i} \), is given by Fick’s Law and corrected with a correction velocity \( V_c^i \) to ensure mass conservation:

\[
V_{\alpha,i} = -\frac{D_\alpha}{Y_\alpha} \frac{\partial Y_\alpha}{\partial x_i} + \sum_{\alpha=1}^{N} (D_\alpha \frac{\partial Y_\alpha}{\partial x_i}) \frac{V_c^i}{V_i},
\]

(8)

where \( D_\alpha \) is the diffusion coefficient between \( \alpha \)-th species and the rest of the mixture. By default, DeepFlame employs (via a Cantera interface function) the Hirschfelder and Curtiss mixture-averaged transport model [17]:

\[
D_\alpha = \frac{1 - Y_\alpha}{\sum_{\beta\neq\alpha} X_\beta D_{\beta\alpha}},
\]

(9)

where \( D_{\beta\alpha} \) is the binary diffusion coefficient between \( \alpha \)-th species and \( \beta \)-th species, which can be obtained according to Takahashi correlation [18]. The simpler model implemented in OpenFOAM assuming unity Lewis number (Le) for all species is also retained in DeepFlame, and the mass diffusivity is modelled as

\[
D_\alpha \equiv \frac{a}{Le} = \frac{\lambda}{\rho C_p},
\]

(10)

where \( a \), \( \lambda \) and \( C_p \) are the thermal diffusivity, thermal conductivity and constant-pressure heat capacity of the mixture, respectively. The mixture overall value of \( \lambda \) is obtained using the conductivity of each pure species \( \lambda_\alpha \):

\[
\lambda = 0.5 \left( \sum_{\alpha} X_\alpha \lambda_\alpha + \frac{1}{\sum_{\alpha} X_\alpha / \lambda_\alpha} \right),
\]

(11)

and a similar procedure is used for \( C_p \) based on the JANAF database. The more detailed multi-component transport model is also supported via the Cantera interface function and the theoretical details are available in [19].

The energy equation for total enthalpy and total energy are respectively given by Eq. (4a) and (4b), the energy flux \( q_i \) is

\[
q_i = -\frac{\lambda}{\rho C_p} \frac{\partial T}{\partial x_i} + \rho \sum_{\alpha=1}^{N} h_\alpha Y_\alpha V_{\alpha,i},
\]

(12)

The RHS of the above equation includes a heat conduction term (expressed by Fourier’s Law, \( \lambda \partial T / \partial x_i \)) and a species diffusion term, where \( h_\alpha \) is the enthalpy of the \( \alpha \)-th species.
Furthermore, temperature in Eq. (12) is usually cast to enthalpy to facilitate the numerical solving procedure of the energy equation. The heat conduction term is rewritten as

\[
\lambda \frac{\partial T}{\partial x_i} = \sum_{\alpha=1}^{N} \rho a Y_\alpha \frac{\partial h_\alpha}{\partial x_i}
\]

\[
= \sum_{k=1}^{N} \rho a Y_\alpha \frac{\partial h_\alpha}{\partial x_i}
\]

\[
= \rho a \frac{\partial h}{\partial x_i} = \sum_{\alpha=1}^{N} \rho a Y_\alpha \frac{\partial Y_\alpha}{\partial x_i}
\]

(13)

Finally, combined with Eq. (8), Eq. (12) and Eq. (13), the energy equation (Eq. (4a) and (4b)) can be expressed more detailed as:

\[
\begin{align*}
\frac{\partial (\rho H)}{\partial t} + \frac{\partial (\rho u_i H)}{\partial x_i} &= \frac{\partial p}{\partial t} + \frac{\partial}{\partial x_i}[(\tau_{ij} - p \delta_{ij})u_j] + \frac{\partial}{\partial x_i}(\rho a \frac{\partial h}{\partial x_i}) \\
&- \sum_{\alpha=1}^{N} \frac{\partial}{\partial x_i}(\rho a h_\alpha \frac{\partial Y_\alpha}{\partial x_i}) - \frac{\partial}{\partial x_i}\left[\rho \sum_{\alpha=1}^{N} h_\alpha Y_\alpha \left(-\frac{D_\alpha}{Y_\alpha} \frac{\partial Y_\alpha}{\partial x_i} + \sum_{\alpha=1}^{N} \frac{D_\alpha}{Y_\alpha} \frac{\partial Y_\alpha}{\partial x_i}\right)\right] \\
\frac{\partial (\rho E)}{\partial t} + \frac{\partial (\rho u_i E)}{\partial x_i} &= \frac{\partial p}{\partial t} + \frac{\partial}{\partial x_i}[(\tau_{ij} - p \delta_{ij})u_j] + \frac{\partial}{\partial x_i}(\rho a \frac{\partial h}{\partial x_i}) \\
&- \sum_{\alpha=1}^{N} \frac{\partial}{\partial x_i}(\rho a h_\alpha \frac{\partial Y_\alpha}{\partial x_i}) - \frac{\partial}{\partial x_i}\left[\rho \sum_{\alpha=1}^{N} h_\alpha Y_\alpha \left(-\frac{D_\alpha}{Y_\alpha} \frac{\partial Y_\alpha}{\partial x_i} + \sum_{\alpha=1}^{N} \frac{D_\alpha}{Y_\alpha} \frac{\partial Y_\alpha}{\partial x_i}\right)\right]
\end{align*}
\]

(14a)

(14b)

2.2. DeepFlame Solvers

Based the above governing equations, DeepFlame provides three solvers to simulate reacting flow under different conditions:

- **df0DFoam**-solver: developed to solve zero-dimensional auto-ignition problems, concerning both constant-pressure and constant-volume cases. The convection and diffusion terms are discarded here, and thus only the species equations are solved to estimate the time-variation of the thermochemical states of the zero-dimensional reactor.

- **dfLowMachFoam**-solver: developed based on rhoPimpleFoam (the original pressure-based compressible solver in OpenFOAM) for simulation of low-Mach number reacting flows. The equation for total enthalpy (Eq. (14a)) is solved here to describe conservation of energy, in which the contribution of viscous heating (the second term on the RHS) is neglected. Additionally, the Strang splitting scheme [20] is adopted to improve the solving accuracy.

- **dfHighSpeedFoam**-solver: developed based on rhoCentralFoam (the original density-based compressible solver in OpenFOAM) solving algorithm to simulate high-speed reacting flows. The equation for total energy (Eq. (14b)) is applied to describe the conservation of energy. The viscous heating and correction of diffusion velocity are taken into account when the flow is considered to be viscous.
2.3. Chemistry Integration

Typically, in the simulation of reacting flows, the computational cost of the chemistry source term evaluations is completely dominant and even exceeds the cost of fluid dynamics by a factor of 100 [2]. Therefore, improving the efficiency of chemistry solver is crucial in the development of advanced reacting flow simulation platforms [21]. In this section, the methods evaluating the chemistry source term in DeepFlame are introduced. Here we start with a chemical system of $N$ species and $M$ reactions [1]:

$$
\sum_{a=1}^{N} \nu_{a,j}^i M_a = \sum_{a=1}^{N} \nu_{a,j}^f M_a \text{ for } j = 1, \ldots, M, \tag{15}
$$

where $M_a$ denotes species $\alpha$, $\nu_{a,j}^i$ and $\nu_{a,j}^f$ represent the molar stoichiometric coefficients of species $\alpha$ in reaction $j$. The reaction rate of species $\alpha$ in such a system is calculated from

$$
\frac{dY_\alpha}{dt} = W_\alpha \sum_{j=1}^{M} (\nu_{a,j}^r - \nu_{a,j}^i) \left\{ K_f j \prod_{\alpha=1}^{N} (\rho Y_\alpha W_\alpha)^{\nu_{a,j}^i} - K_r j \prod_{\alpha=1}^{N} (\rho Y_\alpha W_\alpha)^{\nu_{a,j}^f} \right\}, \tag{16}
$$

where $K_f j$ and $K_r j$ are the forward and reverse rates of reaction $j$, which are usually modelled using the well-known Arrhenius law. The molecular weight and molar concentration are denoted by $W_\alpha$ and $X_\alpha$ for the $\alpha$-th species, respectively.

Typically, the chemical source term $\dot{\omega}_\alpha$ in Eq. (3) is solved based on the large ODEs system formed by Eq. (16). It has been shown in [22–25] that the default OpenFOAM ODE solver suffers from several drawbacks (e.g. poor accuracy and stability, slow integration, etc.). Following these previous studies, DeepFlame provides an interface function to call the well-established implicit ODE solver (CVODE of the SUNDIALS package [26] from Cantera) to improve the solving accuracy and efficiency for the computation of reaction rates.

![Diagram](image)

Figure 1: The schematics of the DNN configurations.

However, since the time scale in reaction system has a broad distribution (from $O(\text{ns})$ to $O(\text{s})$), the ODEs system is so stiff that the computational cost of which is still dominant in simulating reacting flow. Therefore, DeepFlame
also adopts machine learning method to accelerate the solution of chemistry. Following the approach proposed by Zhang et al. [8], a fully-connected deep neutral network with three hidden layers is trained. The architecture of the DNN model is schematically demonstrated in Fig. 1. The input vector \( x(t) = (T(t), P(t), F(Y_\alpha(t)))_{\alpha=1,...,n} \) represents the temperature, pressure and mass fractions of each species at time \( t \). The operator \( F() \) donates the Box-Cox transformation (BCT) [27] which aims to convert the mass fractions from low-order quantity to \( O(1) \). The output of the DNN (labelled as \( u(x) \)) is the change of the input \( x \) during a large time step (typically \( \Delta t = 1 \mu s \)). Now, the chemistry source term \( \dot{\omega}_\alpha \) can be explicitly obtained via \( u(Y_\alpha) \) and thus DNN can be regarded as an ODE integrator. Later in Section 4 and Section 5, the accuracy and computational efficiency of this deep learning method will be demonstrated in various cases.

3. Implementation Details

3.1. DeepFlame Code Structure

The first distinct feature of DeepFlame as compared to the existing codes having an OpenFOAM-Cantera interface is that our implementation for the two-way coupling is more compact and the cleanest possible. All the complex thermo-physico-chemistry operations are handled using Cantera functions via an interface class. The lengthy and redundant OpenFOAM native species, mixture, reaction and derived thermophysical classes have been removed, and only the base class \texttt{fluidThermo} is kept to couple with the flow solver (same as in \texttt{rhoPimpleFoam} for non-reacting flow). Specifically, the coupling of OpenFOAM and libCantera (Cantera C++ API) is achieved as follows: OpenFOAM solves the basic conservation equations and outputs state parameters \((p, T, Y_\alpha)\); libCantera is responsible for handling the chemical mechanism as well as the calculation of thermophysical coefficients \((\mu, \lambda, D_\alpha)\) and reaction rates \(\dot{\omega}_\alpha\). In DeepFlame, the coupling routine is implemented in a new thermo-physio-chemistry library, which strictly follows the standard classes in OpenFOAM. In the following, a comparison between the newly developed and the original libraries is illustrated, and the advantageous features of DeepFlame can be easily observed.

First, the classes related to the thermo-physio-chemistry library of OpenFOAM are summarized. The class diagram as well as the corresponding runtime files (interface for user settings) are shown in Fig. 2a. As listed in the class diagram, OpenFOAM develops a large number of classes with complicated relationship to describe the thermal and chemical properties of the mixture. The abstract basic classes are constructed to declare public interfaces. For example, the functions returning the field thermal properties are virtually defined in \texttt{basicThermo}, the mixture-related functions which output the thermal properties of mixture in a single cell are declared in \texttt{basicSpecieMixture}. These virtual functions are implemented in the derived classes with the models specified in runtime files by users. In general, the thermophysical models are determined in the file “\texttt{constant/thermophysicalProperties}” and the chemistry models are set in the file “\texttt{constant/chemistryProperties}”. Here we produce a brief introduction of these models via a setting example illustrated in the right part of Fig. 2a. In \texttt{thermoType}-dict, the keyword \texttt{type} specifies the underlying thermophysical model is \texttt{rho}-based or \texttt{psi}-based, reacting or nonreacting; the keyword \texttt{mixture} specifies the mixture
Figure 2: The class diagrams of thermo-physio-chemistry library in (a) OpenFOAM and (b) DeepFlame. Lines ending with hollow triangle, hollow diamond and filled diamond shapes denote the inheritance, aggregation and composition relationships, respectively. Note that a line with sharp arrow denotes one class is adopted as a template in another. The corresponding case setup files are shown on the right.
composition is fixed or variable; keywords transport and thermo determine the transport and thermodynamic models, which are used in evaluating transport-parameters ($\mu$, $\lambda$ and $a$) and specific heat $C_p$; the last three keywords represent the equations of state model, species model (calculates the composition of each constituent) and energy form (internal energy or enthalpy), respectively. The chemistryType-dict specifies the solver for chemistry kinetics ODEs, and the coefficients of which is set in odeCoeffs-dict.

Figure 2b illustrates the simplified structure of thermo-physio-chemistry library in DeepFlame. From the class diagram, an obvious simplification can be noted: only several thermo classes derived from basicThermo are kept; the mixture classes and the chemistry classes are all replaced by CanteraMixture and dfChemistryModel. CanteraMixture is a class built to return the thermophysical properties calculated via libCantera, while dfChemistryModel is a chemistry model which enables CVODE and deep neural network for the solution of chemistry kinetics ODEs. Therefore, libCantera and libTorch are also included in this library. As shown in Fig. 2b, the settings of the two classes are all contained in a new file “constant/CanteraTorchProperties”: keyword CanteraMechanismFile is defined to read chemistry mechanisms; keyword transportModel sets the transport model, UnityLewis, Mix, and Multi are provided for users; the zeroDReactor-dict specifies the conditions for zero-dimensional reactor; the switch torch controls the use of deep neural network, and the network name as well as the normalisation parameters are respectively set in torchModel and torchParameters; the device for the DNN inference can also be specified by user via the switch GPU; the keywords in loadbalancing-dic specify the load balance switch for solving chemistry and log file output. Next, in § 3.2, the detailed structure and the underling algorithm of the two classes will be further introduced.

3.2. Description of the new classes

Figure 3 shows the detailed class diagram of the new library in DeepFlame. The description of four critical classes, CanteraMixture, dfChemistryModel, DNNInference and LoadBalancer will be described in this section.

CanteraMixture is developed to evaluate the thermal properties of Mixture in a single cell with libCantera functions. Note that this class is also adopted in the construction of heThermo and Chemistry model (see Fig. 2b), so the operations of which are almost kept identical to the original mixture classes in OpenFOAM. Most of the operations in CanteraMixture are implemented based on the attributes CanteraSolution, CanteraGas, and CanteraTransport, which are constructed from the original libCantera classes in Fig. 3. For example, function mu() is realised by calling CanteraTransport->viscosity() and function Cp() is implemented by calling CanteraGas->cp_mass(). Additionally, to read chemistry mechanisms and user settings, a private attribute CanteraTorchProperties is constructed from the original OpenFOAM I/O controlling class IOdictionary.

DNNInference is developed to calculate chemical reaction rates via inference of deep neural network. It is constructed with the network, normalisation parameters and a switch for GPU as the DNN inference device. The operation inference is built to calculate the chemistry source terms with DNN. It mainly consists of three stages: 1) pre-process the input tensor ($p$, $T$, $Y$) by normalisation and Box-Cox transformation; 2) inference the processed tensor with the network; 3) evaluate the mass fractions via the reverse-transformation of stage 1 and finally output the reaction
Figure 3: The main classes of the thermo-physio-chemistry library in DeepFlame. The notations “−” and “+” specify the visibility of the class member as public and private.

Based on the above three classes, the core of DeepFlame, dfChemistryModel, can be finally constructed. The thermal and chemistry fields are all contained in this class as the attributes. Also, most of these fields can be received by calling the public operations, such as \( RR() \), \( Qdot() \), \( Y() \), etc. The two public functions \( solve() \) and \( correctThermo() \) are built for solving chemistry source and updating thermal properties. The execution procedure of the two operations is described in Fig. 4, which presents the computational algorithm of the dfLowMachFoam-solver. It can be seen that the conservation equations for mass, momentum, species, energy, and a Poisson equation for pressure are successively solved in an iterative manner. Besides, the other two separate steps, the solving of chemical sources and thermal properties, are achieved by calling the function \( solve() \) and \( correctThermo() \). The implementation of \( correctThermo() \) can be divided into the following steps: construct the attribute CanteraGas with \( h, p \) and \( Y \) obtained from previous
Figure 4: A schematic demonstrating the main operations of the dfLowMachFoam-solver in a single iteration step.

- Solve mass conservation and momentum equations
- Solve chemical sources
  - if use DNN
    - call function solve_DNN
    - No
      - call function solve_CVODE
- Solve species and energy equations
- Update temperature and transport coefficients
  - update CanteraGas with $h$, $p$, $Y$
  - update $T$, $\mu$, $\lambda$, $C_p$ with CanteraGas
- $D_a = \frac{1 - Y_a}{\sum_{\rho \phi \alpha} \chi_{\phi} / \mathcal{D}_{\phi \alpha}}$
- $D_a = \frac{\lambda}{\rho C_p}$

PIMPLE iteration

3.3. Adaptive mesh refinement

Adaptive mesh refinement (AMR) is readily available in OpenFOAM as an effective way to reduce computational cost for spatially stiff computations such as involving shock and detonation waves. However, the original AMR
algorithm in OpenFOAM named *hexRef8* uses the octree refinement and splits each cell in eight child cells (homogeneously halved in three directions). This brings extra computational cost for one- and two-dimensional cases since it will refine cells in invariant directions. Therefore, the original AMR was extended from *hexRef8* to *hexRef4* in [29, 30] to implement a quadtree refinement for two-dimensional problems. Additionally, multiple refinement criteria was also included for higher flexibility. In this work, we further extend the AMR capability to one-dimensional via a new mesh cutter named *hexRef2*. The empty boundary type (for invariant directions) can be automatically detected, and the corresponding boundary faces will be marked as divisible. During an AMR process, each divisible face is split into two new faces and a new internal face is added to each cell to be refined. To keep consistency, we strictly follow the code structure of [30] and *hexRef2* is added as a new derived class to the base class *hexRef*. Upon simulation startup, *dfDynamicRefineFvMesh* selects the suitable mesh cutter according to the dimension of the case. Additionally, the temporally evolving grid file related I/O is improved to facilitate AMR restart and post-processing.

4. Validation and Results

In this section, the above implementation of *DeepFlame* is validated. The solvers described in § 2.2 are systematically tested using a broad range of canonical test cases under different dimension and flow speed conditions.

4.1. *df0DFoam*-solver

As introduced in § 2.1, the *df0DFoam* is developed to solve zero-dimensional problems, which can be further subdivided into constant-pressure and constant-volume conditions. Figure 5 compares the hydrogen autoignition results obtained by Cantera and *df0DFoam* at constant pressure. The chemical mechanism adopted here is developed by Evans et al. [31], containing 8 species and 16 reversible reactions. The neural network is trained according to the method proposed in [8]. In the later hydrogen flame cases, we will also adopt the above mechanism and network if not specified. In this case, the initial condition (temperature, pressure and equivalence ratio) of the H\textsubscript{2}/air mixture is set as $T = 1400$ K, $p = 1$ atm and $\phi = 1$. From Fig. 5, it can be seen that the *df0DFoam* can accurately capture the evolution of $T$, $p$ and $Y$, for both the DNN and CVODE integrators. Figure 6 shows constant-volume autoignition results of H\textsubscript{2} given by *df0DFoam*-solver at $T = 1000$ K, $p = 0.5$ atm and $\phi = 1$. It reveals that the DNN integrator has a satisfying performance even in the condition with a large variation of pressure. To sum up, the cases conducted in this part confirm the validity of the implementation of chemical reaction source terms in *DeepFlame*.

Figure 7 shows the ignition delay times (IDTs, defined using $[dT/dt]_{\text{max}}$) versus initial temperature variations for both hydrogen and n-Heptane fuels. For the hydrogen/air mixture at stoichiometric and atmospheric conditions, both the DNN and CVODE chemistry integrators give very close results to the IDTs calculated using Cantera for a broad range of temperature conditions. For a more complex fuel n-Heptane (n-C\textsubscript{7}H\textsubscript{16}) with negative temperature coefficient (NTC) behaviours, *df0DFoam* with CVODE also shows very good agreement with the Cantera numerical results and the experimental measurements [32].
Figure 5: Zero-dimensional constant-pressure autoignition results comparison between Cantera and df0DFoam (with CVODE and DNN integrators).

Figure 6: Zero-dimensional constant-volume autoignition results given by df0DFoam.
Figure 7: Ignition delay time versus temperature for stoichiometric (a) hydrogen/air mixture at atmospheric pressure and (b) n-Heptane/air mixture at 12.5 atm pressure.

4.2. dfLowMachFoam-solver

To assess the implementation of the transport processes, the laminar free-propagating 1D planar premixed flame is first examined. Then, a 2D non-premixed lifted jet flame and a reactive Taylor-Green vortex (TGV) are simulated to validate the capability of the solver for multi-dimensional applications. For all the three cases, hydrogen is adopted as the fuel and the mixture-averaged model is chosen to calculate the diffusion coefficients. Time derivatives are discretised with the implicit Euler scheme. Discretisation of convective and diffusion terms is based on the second-order central differencing scheme.

4.2.1. One-Dimensional Planar Flame

The computational setup of the 1D case is schematically shown in Fig. 8a. Except for the Inlet and Outlet, all the side boundaries are defined using the empty condition (i.e. transverse spatial terms are not calculated), ensuring no fluxes in lateral direction. The domain is initialised using steady-state 1D freely-propagating flame solution from Cantera. The velocity of the inlet mixture is set to be identical to the flame speed given by Cantera so that the flame front can be stabilised inside the domain. The length of the computational domain is according to the solution from Cantera as well, and the cell number is set to ensure the flame front is discretised by at least 30 elements.

Figure 8b shows that the distributions of the species mass fractions given by dfLowMachFoam-solver and Cantera exhibit an excellent agreement. This observation can be extended to different equivalence ratios as presented in Fig. 9. The laminar flame speed \( S_l \) and flame thickness \( \delta_l \) given by dfLowMachFoam compare well with the Cantera solution for the range of \( \phi = 0.5 \) to 1.8. Here \( S_l \) is evaluated by subtracting the propagation velocity of flamefront (position
Figure 8: (a): Numerical setup of one-dimensional premixed flame. (b): Profiles of mass fractions of main species compared between Cantera and $df/LowMachFoam$.

Figure 9: Comparison of laminar flame speed (left) and flame thickness (right) of hydrogen/air mixture at different equivalence ratios.

of $\nabla T_{\text{max}}$ by the inlet velocity and $\delta_l$ is calculated as $(T_b - T_a)/\nabla T_{\text{max}}$. The minor deviations in the above comparison can be attributed to the different meshing methods adopted in Cantera and $df/LowMachFoam$: the former uses the adaptive grid with high refinement near the flamefront, whereas the latter adopts a static grid. Nevertheless, the cases conducted in this part demonstrate that the convection-diffusion-reaction algorithms implemented in DeepFlame are
4.2.2. Two-Dimensional Jet Flame

In this subsection, a 2D non-premixed planar jet flame is simulated using dfLowMachFoam with the DNN and CVODE integrators. As shown in Fig. 10a, a H$_2$ fuel jet with $T = 300$ K and $p = 1$ atm enters from the left end of the rectangle domain, surrounded by an air co-flow. The fuel jet has a diameter of 8 mm and the composition of 25% H$_2$/75% N$_2$ in volume. The velocities of the jet and the co-flow are 5 and 1 m/s, respectively. The size of the computational domain is $3 \times 5$ cm, discretised by a structured mesh with $300 \times 500$ resolution. The initial fields are specified as follows. An ignition region with the temperature of 1400 K is set at the centre of the domain (shaded area in Fig. 10a); the velocity and mass fractions of the H$_2$ jet decay along the streamwise direction and a $1/7$ power law is applied for the velocity profile in the transverse direction. The right figure in Fig. 10a depicts the profiles of $u$ and $Y$ along the central line.

Figure 10b shows the evolution of the jet flame using heat release rate (HRR) contours. It can be seen that the dfLowMachFoam gives almost identical results for the DNN and CVODE cases. After ignition, two flame branches develop towards upstream and form the triple flame structures at 4 ms. Here we make a further quantitative comparison between the DNN and CVODE solvers: transverse profiles of species mass fractions in the two triple flames are extracted at the point with maximum heat release rate. As shown in Fig. 10c, the results obtained by DNN and CVODE show excellent agreement for all major species.

4.2.3. Three-Dimensional reactive Taylor-Green Vortex

Finally, the performance of dfLowMachFoam is assessed using a recently established benchmark case for reacting flow DNS codes – 3D Taylor-Green Vortex (TGV) interacting with a non-premixed flame [33]. Figure 11a shows the cubic computational domain with the edge length of $2\pi L$ and the initial fields of vorticity magnitude and temperature. All boundaries are set to be periodic. The initial condition for the velocity field is given by

$$
\begin{align*}
    u_x &= u_0 \sin \left( \frac{2\pi x}{L} \right) \cos \left( \frac{2\pi y}{L} \right) \cos \left( \frac{2\pi z}{L} \right), \\
    u_y &= -u_0 \cos \left( \frac{2\pi x}{L} \right) \sin \left( \frac{2\pi y}{L} \right) \cos \left( \frac{2\pi z}{L} \right), \\
    u_z &= 0,
\end{align*}
$$

where the reference velocity magnitude $u_0$ is set to 1 m/s and the reference length $L$ is set to 1 mm. The chemical mechanism of by Boivin et al. [34] used in [33] involving 9 species and 12 reversible reactions is also adopted here to make a direct comparison. The $x$-direction profiles of the temperature and species are specified following [33] as depicted in Fig. 11b. The flame is initialised using the equilibrium value of the local thermodynamic state. The more details for the initial setup of this reactive TGV benchmark are given in [33].

To fully resolve the flame front, this case is computed on an equidistant grid with $256^3$ cells. The simulation is carried out for a physical time of $t = 0.5$ ms (i.e. 2 vortex turnover reference times). Figure 12a shows the contours...
Figure 10: Simulation results of the two-dimensional jet flame using dfLowMachFoam. (a) numerical setup and the initial profiles of velocity and species mass fractions along centreline; (b) Contours of HRR calculated by two ODE integrators at $t = 0.5$ and $4$ ms; (c) Transverse profiles of species mass fractions at the maximum HRR point for the 2D triple flame.

The accuracy of dfLowMachFoam in solving complex flow dynamics is demonstrated via the comparison presented in Fig. 12b, where the profiles of $Y_{H_2}$ and $T$ at $t = 0.5$ ms
Figure 11: (a) Initial contours of vorticity magnitude and temperature for the reactive TGV; (b) Initial profiles of temperature, hydrogen mass fraction and oxygen mass fraction at \( y = 0.5L \) and \( z = 0.5L \). along the \( y \)-centreline show a quantitatively good agreement between the present and reference (conducted using the code DINO [33]) cases.

4.3. dfHighSpeedFoam-solver

In this subsection, we assess the implementation of \textit{dfHighSpeedFoam}, in which the original OpenFOAM density-based solver \textit{rhoCentralFoam} is extended to consider multi-species transport and reaction also via the \textit{dfChemistry-Model} class shown earlier in Fig. 3. For high-speed supersonic reactive flows, it is necessary to capture the discontinuous structures such as shock and detonation waves, where adaptive meshing becomes essential to ensure both numerical accuracy and efficiency. Thus, the implementation of AMR described in § 3.3 is also validated here using the following test cases.

4.3.1. One-Dimensional Reactive Shock Tube

First, the multi-component reactive shock tube [35] is simulated. The computational domain is a tube of 0.12 m long and filled with the mixture of \( \text{H}_2/\text{O}_2/\text{Ar} \) having a 2/1/7 proportion by mole. The domain is divided into 2400
cells uniformly. The initial condition is set as

\[
(T, p, u) = \begin{cases} 
378.656 \text{ K}, & 7173 \text{ Pa}, \quad 0 \text{ m/s}, \quad x < 0.06 \text{ m} \\
748.472 \text{ K}, & 35594 \text{ Pa}, \quad -487.34 \text{ m/s}, \quad x > 0.06 \text{ m} 
\end{cases}
\]  

Supersonic inlet condition is set for the right boundary and solid wall is set for the left boundary. The 9 species and 18 reactions of [36] mechanism is used in this case. Figure 13 shows the distribution of temperature, velocity and mass fraction of H at different times. It can be seen that the results computed using dfHighSpeedFoam are in good agreement with the results of Martínez Ferrer et al. [37]. At around \( t = 190 \mu s \), the reactive flow merges into the shock wave forming a detonative front, which is well captured in this present simulation.

4.3.2. Detonation Propagation Speed

Detonation propagation can show the coupling of shock wave and chemical reaction. It contain a complex interaction of the leading shock wave and auto-igniting reaction. This case aims to validate the accuracy of dfHigh-SpeedFoam in capturing this process and the propagation speed. The length of computational area is 0.5 m. Since adaptive mesh refinement is applied in this case, the coarse cell of 0.8 mm is used in this case. After being refined at
the discontinuity, the length of the minimum cell is 0.1 mm since the maxRefinement is set to 3. The domain is filled with homogeneous stoichiometric H$_2$/O$_2$/N$_2$ mixture and the detailed mechanism of 9 species and 21 reactions [38] is used to calculate combustion. The detonation is ignited by a 2 mm hot spot of 2000 K and 90 atm, which the other area is initialized at 300K and 1atm.

By recording the position of wave, we can calculate the detonation propagation speed of different equivalent ratio. Figure 14 show that our result is close to the theoretical result calculated by SDToolbox [39], which can prove the accuracy of this solver.

5. Computational Performance

As introduced above, machine learning, adaptive mesh refinement and dynamic load balance are adopted in DeepFlame to speed up reacting flow simulations. In this section, the efficiency improvement given by these approaches will be quantitatively evaluated. First, we will present the speed-up in solving zero-dimensional ignition case when adopt DNN as the chemistry solver. Then the effect of AMR and DLB in accelerating the simulation of one-dimensional detonation is evaluated. Finally, a strong scaling test is conducted for the reactive TGV case to demonstrate the parallel computing efficiency.
5.1. Acceleration from Machine Learning

In this subsection, the performance of the DNN chemistry solver is demonstrated by comparing with the CVODE solver. The computational efficiency of chemistry ODE integrator is evaluated via the execution time in simulating the constant-pressure 0D ignition case (§ 4.1). Figure 15 shows the computational time required for different ODE integrators and computing architectures (CPU or GPU). The CPU used here is Intel i7-12700KF and the GPU is NVIDIA RTX3070TI. Note that other than GPU, chips specifically made for AI computations (e.g. Sugon Deep Computing Unit) were also tested but not reported here for conciseness. In general, similar observations to GPU were obtained for the speed-up behaviour – several orders of magnitude acceleration was achieved.

It can be seen in Fig. 15 that the computational cost for the given test case is nearly halved on CPU (same as CVODE) and further reduces by two orders of magnitude on GPU when using DNN as the integrator. Additionally, the DNN performance further improves with the increase of the grid number, whereas the CVODE cost linearly scales with the grid number. This speed up given by DNN is essentially attributed to the parallelisation of operators and data, i.e. the solving of chemistry is implemented by matrix addition and multiplication via DNN inference instead of logical iterations. Furthermore, considering the relatively small size of hydrogen/air mechanism, more promising acceleration from DNN can be expected for more complex hydrocarbon fuels.

5.2. Acceleration from DLB and AMR

As mentioned earlier, AMR can effectively reduce the simulation cost for high-speed reacting flows. When combined with DLB, the acceleration effect becomes even more significant. As AMR refines the cells in chemically intense areas (near flame, shock or detonation fronts), these spatially clustered cells are likely to be allocated on a
small number of processors. This leads to a computational load imbalance, which also varies temporally as the refined region moves within the domain. Therefore, the dynamic balancing technique is almost essential for large-scale multi-dimensional AMR simulations.

![Figure 16: Comparison of computational time cost for different calculation parts in simulating one-dimensional detonation problem when closing AMR and DLB, only adapting AMR and adapting both.](image)

To demonstrate this effect, Fig. 16 compares the original, AMR only and AMR+DLB computational times required for a typical 1D detonation simulation using 5 processors. The total time is broken down into 3 parts: chemistry integration, solving species and flow transport equations. It can be observed that chemical source term evaluation accounts for over 95% of the computation and AMR speeds up the simulation by a factor of about 6.5. When both
AMR and DLB are activated, the computation is 10 times faster than the original speed.

5.3. Parallel efficiency

The parallel scalability of DeepFlame is estimated via a strong scaling test, which is performed by varying the number of processors to solve the same problem. The reactive TGV case introduced in § 4.2.3 is adopted here as the benchmark scaling using up to 8192 CPU cores. The machine used in this section is the UK National Supercomputing facility ARCHER2. It is composed of 5860 compute nodes, each with dual AMD EPYC 7742 64-core 2.25 GHz processors.

Table 1: Comparison of code set-up and performance in simulating reactive TGV. The meanings of the acronyms are given in the text.

| Data      | Np  | Ncores | Nit | \(\Delta t\) | TCPU | RCT | RTTS |
|-----------|-----|--------|-----|-------------|------|-----|------|
| unit      |     |        |     | \(\text{ms}_\text{sim}\) | \(\text{ms}_\text{iter}\) | \(\mu\text{sec}_\text{point}\) | \(\mu\text{sec}_\text{point}\) |
| YALES2    | 256 | 384    | 1484| 2.5         | 1.685 | 923 | 354  |
| DINO      | 256 | 1024   | 13417| 2.5         | 0.186 | 2414| 660  |
| Nek5000   | 256 | 576    | 3627 | 2.5         | 0.689 | 486 | 283  |
| DeepFlame | 256 | 512    | 2500 | 2.5         | 1     | 1617| 134  |

Figure 17: Strong scaling for the reactive Taylor-Green vortex setup. Numbers within the top two figures indicate the parallel efficiency.

First, a typical performance comparison between DeepFlame and the reference codes [33] is given in Table 1. Here \(N_p\) is the number of grid points, \(N_{\text{cores}}\) is the CPU core number, \(N_{\text{it}}\) is the iteration steps. The subscripts \(\text{sim}\)
and CPU denote that the variables and units are related to simulation physical time or CPU time. The variable TCPU represents the total CPU time for the simulation and it is computed by multiplying Ncores and the total CPU time. The Reduced Computational Time (RCT) is calculated with the expression: RCT = TCPU/(Nit × Np). This is introduced to evaluate the CPU time needed to simulate one time-step on a single degree of freedom by using one processor. The CPU time needed to obtain the solution for a single degree of freedom after a specified simulation time Tsim is represented by Reduced Time to Solution (RTTS), which is calculated as RTTS = TCPU/(Tsim × Np). From the results listed in Table 1, DeepFlame shows the best computational efficiency for the criterion of RTTS. This indicates that for a given duration of flow time, the least CPU time is required for this specific case. The satisfying performance can be mainly attributed to the efficiency of CVODE and DNN chemistry solvers (allowing for a large time-step size) and also the optimised load balancing algorithm. In addition, even for the computation of a single time-step, DeepFlame gives the second best RCT performance, only slower than the fully structured code DINO with the much faster explicit time-integration. However, it must be noted that all other codes listed in Table 1 are built with high-order time and space discretisation, whereas DeepFlame is limited to second order. Nevertheless, this comparison is not intended to claim superiority but to demonstrate the relative computational speed with respect to the state-of-the-art reacting flow codes.

The strong scaling performance is shown in Fig 17. The sum time (τsum) represents the total elapsed time simulating the reactive TGV from \( t = 0.5 \) to 0.6 ms on the given number of CPU cores. The main components of τsum, τchem, τY and τflow, denote the wall-time cost in solving chemistry kinetic equations, species equations and flow equations, respectively. Moreover, both the conditions with and without chemistry dynamic load balance are considered in this section to estimate the performance of the current load balance method. As seen in Fig. 17, τchem accounts for more than 70% of the total time. With the aid of DLB, τchem firstly gains a speed-up of about 30% (for the same MPI rank) but then slows down as the number of CPU cores exceeds about 1000 (i.e. \(< 16000 \) cells per core). This is mainly attributed to the relatively high communication cost as the overhead of DLB at this time. The scaling of τY and τflow is super linear up to about 1000 CPU cores. However, note that the efficiency in solving flow equations is obviously reduced with the CPU numbers over 5000 (\(< 3500 \) cells per core). This is possibly due to the poor performance of the GAMG-solver used for solving the Poisson equation of pressure. For this simulation setup, there is an optimum in terms of efficiency at about 16000 cells per core. Further reduction in cell number (i.e. increase CPU number) may benefit the simulation time but at a cost of nonlinear reduction in efficiency.

6. Conclusion and Future Work

This study introduced a machine learning empowered CFD platform DeepFlame for simulating reacting flows. The implementation of this platform is achieved via interfacing the CFD toolbox OpenFOAM, chemical kinetics software Cantera and deep learning framework Torch. Deferring to all of the previous OpenFOAM-Cantera interface codes, we adopt the simplest nonreactive pure fluid base model and the thermo-physio-chemistry is entirely handled by
the *DeepFlame* mixture and chemistry objects and functions interfaced with Cantera. Three solvers are developed and thoroughly validated to cover the cases with various flow speeds and physical dimensions. Methods including dynamic load balance (DLB), adaptive mesh refinement (AMR), and deep neural network (DNN) accelerated chemistry solver are available to improve the simulation efficiency.

We also presented a detailed computational performance analysis of *DeepFlame*. The validity of the solving algorithms implemented in *DeepFlame* is confirmed via a broad range of canonical cases, including 0D ignition, 1D planar flame and reactive shock wave, 2D jet flames, and 3D reactive Taylor-Green vortex. In additional, we also reported the computational efficiency of the code on different computing chips to showcase the acceleration provided by machine learning. Specifically, the DNN chemistry solver shows a 30–50% speed-up on a CPU due to data and operation vectorisation, and a speed-up of two orders of magnitude on a GPU or DCU due to many-core parallelisation. The simulation of 1D detonation is 10 times faster when adopting AMR and DLB. Finally, the parallel scalability on many CPUs is also reported showing good scaling behaviour compared to the state-of-the-art reactive flow codes while running on up to tens of thousands of processors.

Despite the advantages we have introduced in this study, there are still many aspects of *DeepFlame* to be improved from the current version:

- The solvers developed in the current platform only focus on the gaseous fuel. Lagrangian/Euler solvers for multi-phase spray reacting flow will be implemented.
- The current DNN chemistry solver can only be used on a single GPU/DCU card. In the future we will provide support for the parallel computation of DNN via multi-cards on single and many nodes.
- The parallel scalability needs to be further improved to achieve the linear scaling up to more than ten thousands CPU cores.

These features are actively under development on our public repository website and will be released in the future *DeepFlame* versions.

**Acknowledgements**

The work of Z.X.C. is supported by the National Science Foundation (Grant No. 52276096) and the Fundamental Research Funds for the Central Universities of China. Part of the numerical simulations was performed on the High Performance Computing Platform of CAPT of Peking University.

**References**

[1] T. Poinsot, D. Veynante, Theoretical and numerical combustion, RT Edwards, Inc., 2005.
[2] N. Peters, Turbulent combustion, Cambridge university press, 2000.
[3] F. Christo, A. Masri, E. Nebot, Artificial neural network implementation of chemistry with pdf simulation of h2/co2 flames, Combustion and Flame 106 (4) (1996) 406–427.
[4] J. Blasco, N. Fueyo, C. Dopazo, J. Ballester, Modelling the temporal evolution of a reduced combustion chemical system with an artificial neural network, Combustion and Flame 113 (1-2) (1998) 38–52.
[5] B. A. Sen, S. Menon, Turbulent premixed flame modeling using artificial neural networks based chemical kinetics, Proceedings of the Combustion Institute 32 (1) (2009) 1605–1611.
[6] K. Wan, C. Barnaud, L. Vervisch, P. Domingo, Chemistry reduction using machine learning trained from non-premixed micro-mixing modeling: Application to dns of a syngas turbulent oxy-flame with side-wall effects, Combustion and Flame 220 (2020) 119–129.
[7] S. Yao, A. Kronenburg, A. Shamooni, O. Stein, W. Zhang, Gradient boosted decision trees for combustion chemistry integration, Applications in Energy and Combustion Science 11 (2022) 100077.
[8] T. Zhang, Y. Yi, Y. Xu, Z. X. Chen, Y. Zhang, E. Weinan, Z.-Q. J. Xu, A multi-scale sampling method for accurate and robust deep neural network to predict combustion chemical kinetics, Combustion and Flame 245 (2022) 112319.
[9] Z. X. Chen, S. Javarone, G. Ghiassi, V. Kannan, G. D’Alessio, A. Parente, N. Swaminathan, Application of machine learning for filtered density function closure in mild combustion, Combustion and Flame 225 (2021) 160–179.
[10] C. Chi, X. Xu, D. Thévenin, Efficient premixed turbulent combustion simulations using flamelet manifold neural networks: A priori and a posteriori assessment, Combustion and Flame 245 (2022) 112325.
[11] B. A. Perry, M. T. H. de Frahan, S. Yellapantula, Co-optimized machine-learned manifold models for large eddy simulation of turbulent combustion, Combustion and Flame 244 (2022) 112286.
[12] Y. Zhang, S. Xu, S. Zhong, X.-S. Bai, H. Wang, M. Yao, Large eddy simulation of spray combustion using flamelet generated manifolds combined with artificial neural networks, Energy and AI 2 (2020) 100021.
[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., {TensorFlow}: a system for [Large-Scale] machine learning, in: 12th USENIX symposium on operating systems design and implementation (OSDI 16), 2016, pp. 265–283.
[14] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: A matlab-like environment for machine learning, in: BigLearn, NIPS workshop, no. CONF, 2011.
[15] O. OpenCFD, The open source cfd toolbox, User Guide, OpenCFD Ltd 770.
[16] D. G. Goodwin, Cantera c++ user’s guide, California Institute of Technology (2002) 32.
[17] C. F. Curtiss, J. O. Hirschfelder, Transport properties of multicomponent gas mixtures, The Journal of Chemical Physics 17 (6) (1949) 550–555.
[18] R. J. Kee, M. E. Coltrin, P. Glarborg, Chemically reacting flow: theory and practice, John Wiley & Sons, 2005.
[19] A. Ern, V. Giovangigli, Lecture notes in physics, Multicomponent transport algorithms 24.
[20] G. Strang, On the construction and comparison of difference schemes, SIAM journal on numerical analysis 5 (3) (1968) 506–517.
[21] T. Lu, C. K. Law, Toward accommodating realistic fuel chemistry in large-scale computations, Progress in Energy and Combustion Science 35 (2) (2009) 192–215.
[22] T. Li, J. Pan, F. Kong, B. Xu, X. Wang, A quasi-direct numerical simulation solver for compressible reacting flows, Computers & Fluids 213. doi:10.1016/j.compfluid.2020.104718.
[23] Q. Yang, P. Zhao, H. Ge, reactingfoam-sci: An open source cfd platform for reacting flow simulation, Computers & Fluids 190 (2019) 114–127. doi:10.1016/j.compfluid.2019.06.008.
[24] F. Zhang, H. Bonart, T. Zirwes, P. Habiserreuther, H. Bockhorn, N. Zarzalis, Direct Numerical Simulation of Chemically Reacting Flows with the Public Domain Code OpenFOAM, 2015, book section Chapter 16, pp. 221–236. doi:10.1007/978-3-319-10810-0_16.
[25] D. Zhou, H. Zhang, S. Yang, A robust reacting flow solver with computational diagnostics based on openfoam and cantera, Aerospace 9 (2). doi:10.3390/aerospace9020102.
[26] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, C. S. Woodward, Sundials: Suite of nonlinear and
differential/algebraic equation solvers, ACM Transactions on Mathematical Software (TOMS) 31 (3) (2005) 363–396.

[27] G. E. Box, D. R. Cox, An analysis of transformations, Journal of the Royal Statistical Society: Series B (Methodological) 26 (2) (1964) 211–243.

[28] B. Tekgül, P. Peltonen, H. Kahila, O. Kaario, V. Vuorinen, Dlibfoam: An open-source dynamic load balancing model for fast reacting flow simulations in openfoam, Computer Physics Communications 267. doi:10.1016/j.cpc.2021.108073.

[29] A. Banabedalruhman, Dynamic meshing around fluid-fluid interfaces with applications to droplet tracking in contraction geometries, Dissertation, Michigan Technological University.

[30] D. Rettenmaier, D. Deising, Y. Ouedraogo, E. Gjonaj, H. D. Gersem, D. Bothe, C. Tropea, H. Marschall, Load balanced 2d and 3d adaptive mesh refinement in openfoam, SoftwareX 10 (2019) 100317. doi:10.1016/j.softx.2019.100317.

[31] J. S. Evans, C. J. Schexnayder Jr, Influence of chemical kinetics and unmixedness on burning in supersonic hydrogen flames, AIAA journal 18 (2) (1980) 188–193.

[32] K. Heufer, H. Olivier, Determination of ignition delay times of different hydrocarbons in a new high pressure shock tube, Shock Waves 20 (4) (2010) 307–316.

[33] A. Abdelsamie, G. Lartigue, C. E. Frouzakis, D. Thévenin, The taylor–green vortex as a benchmark for high-fidelity combustion simulations using low-mach solvers, Computers & Fluids 223. doi:10.1016/j.compfluid.2021.104935.

[34] P. Boivin, C. Jiménez, A. L. Sánchez, F. A. Williams, An explicit reduced mechanism for h2–air combustion, Proceedings of the Combustion Institute 33 (1) (2011) 517–523.

[35] 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.

[36] 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.

[37] P. J. Martínez Ferrer, R. Buttay, G. Lehnasch, A. Mura, A detailed verification procedure for compressible reactive multicomponent navier-stokes solvers, Computers & Fluids 89 (2014) 88–110.

[38] 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.

[39] J. E. Shepherd, Shock & detonation toolbox, https://shepherd.caltech.edu/EDL/publicresources.