# Deep learning model to assist multiphysics conjugate problems

George El Haber, Jonathan Viquerat, Aurelien Larcher, David Ryckelynck, Jose Alves, Aakash Patil, Elie Hachem

# Deep learning model to assist multiphysics conjugate problems

A Preprint

**George El Haber**
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
george.el_haber@mines-paristech.fr

**Jonathan Viquerat**
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
jonathan.viquerat@mines-paristech.fr

**Aurelien Larcher**
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
aurelien.larcher@mines-paristech.fr

**David Ryckelynck**
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
david.ryckelynck@mines-paristech.fr

**Jose Alves**
Transvalor SA, France
E-Golf Park, 950 Avenue Roumanille, 06410 Biot
jose.alves@transvalor.com

**Aakash Patil**
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
aakash.patil@mines-paristech.fr

**Elie Hachem***
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
elie.hachem@mines-paristech.fr

December 16, 2021

## Abstract

The availability of accurate and efficient numerical simulation tools has become of utmost importance for the design and optimization phases of existing industrial processes. The latter requires the computation of multiple physical fields governed by coupled systems of partial differential equations and tends to require large computational resources. Recently, the coupling of machine learning techniques with numerical simulation tools has allowed lifting part of this computational burden, by replacing parts of the resolution process with trained neural networks, which execution cost is far less than their traditional counterparts. In this work, an auto-encoder convolutional neural network is suggested to reduce the resolution cost of the forced cooling of a hot workpiece in a confined space by modeling the scalar transport equation coupled to the Navier–Stokes equations. Although the proposed model was trained on a relatively limited amount of data, it was able to generalize accurately for different cooling

*The author to whom correspondence may be addressed: elie.hachem@mines-paristech.fr

setups with different inlet locations, thus leading to a reliable deep learning-assisted numerical solver.

***Keywords*** Computational fluid dynamics · Multiphysics problem · Deep learning · Transported scalar field

## 1 Introduction

Current industries are continuously aiming to satisfy the needs of the demanding market with high-quality products in the shortest feasible delay. This objective can only be achieved by rapidly and accurately forecasting the results of every single stage in the production process, thus establishing a precise idea of the final product from the early design phases. Prior simulation of problems involving fluid flows is accomplished using numerical analysis methods for computational fluid dynamics [1]. These methods serve numerous fields such as weather simulation, aerospace, aerodynamics, and many others.

Accurate modeling of many real-life phenomenons requires the coupling of multiple physics. Turbulent flow problems past a certain obstacle, for example, can be modeled by the nonlinear Navier–Stokes equations coupled with a single-equation turbulence model, [2], or a two-equation model, [3, 4, 5]. Other problems, that are involved with the cooling and heating of a workpiece using natural or forced convection, require the coupling with a heat transfer equation [6, 7].

Unfortunately, multiphysics problems governed by the coupling of the Navier–Stokes equations with a scalar transport equation, whether it is a weak or a strong coupling, are computationally expensive to solve, thus limiting the practicability of modeling complex industrial applications. Moreover, the partial differential equations in coupled problems have, as input parameters, fields set up in huge ambient spaces such as the velocity field convecting the scalar in transport equations. The high dimensionality of these input spaces makes extrapolations or interpolation quite difficult. However, the recent increased availability of computational capacities and data resources has brought the capabilities of deep learning to the front of the stage. In the past decade, the potential of machine learning has been demonstrated in multiple domains, such as natural language processing [8], machine translation [9], speech recognition [10], or computer vision [11]. The recent years also witnessed applications to computational mechanics for the simulation of physical problems. For example, In [12], Raissi et al. introduced a regularization mechanism that informs the deep learning model about the governing equations of the physical problem. In [13], the authors suggested using a deep classifier to adapt the reduced-order model to an input tensor parametrizing an anisothermal elastoplastic problem in structural mechanics. Neural networks that operate directly on graph-structured data, [14, 15, 16], have also been exploited to assist in various physical problems. In [17], the authors suggested incorporating neural networks to infer problem-specific coefficients for the estimation of spatial derivatives required for the solution of various PDEs modeling physical phenomena such as shock formations, solitary waves on a river bore, and flame fronts. Another method where a deep learning model is used to reduce the computational cost of a physical simulation is suggested in [18]. The authors proposed assisting the in-plane-out-of-plane PGD solver by introducing a model responsible for inferring the out-of-plane function and tested their methodology for a 3D heat conduction problem in a plate.

In the present work, an auto-encoder architecture [19, 20] is employed to model the scalar transport equation in a coupled system. This system, consisting of the Navier–Stokes equations along with the heat energy equation, is required to simulate the cooling process of a work object using forced convection. The required data to train the model is obtained using an industrial-level computational fluid dynamics solver [21]. The latter exploits the immersed volume method (IVM) [22] equipped with interface capturing and anisotropic mesh adaptation method [23], to overcome the difficulties encountered in specifying the specific solid-fluid interface. The resulting equations are resolved using a continuous finite element method along with the variational multiscale approach (VMS) [24, 25, 26]. The mathematical formulation of IVM, in the context of finite element VMS methods, is detailed in the following papers [27, 28].

The deep learning model architecture used can accurately model the scalar energy equation and thus infer the required future temperature field at the next time step. The model is trained to predict the

field with a larger time step than that of the solver, thus allowing to span the whole time domain in much fewer computational steps. Moreover, the trained model reveals precise interpolation ability for both, time and cooling inlets positions, thus permitting us to exploit its potential for numerous flow setups by moderately training it on a few snapshots, scattered across the required time domain, obtained from discrete setups. The model also manages to span independently the whole time domain, *i.e.* relying solely on its prediction and without referring back to the traditional solver, and returns reliable results with much less computational time.

The success of this model paves the way for assisting the simulation of numerous industrial problems where coupling with a scalar transport equation exists. Moreover, other types of transport equations can be examined in future work with the same methodology, thus broadening the potential of this technique. The paper commences by stating the governing equations and the methods used to resolve them numerically in section 2. Details of the problem setup are provided in section 3. Section 4 details the sets used to train and test the model. The model architecture and its training are respectively presented in sections 5 and 6. Finally, the performance of the model on multiple flow setups and its ability to span independently the required time domain are briefed in 7. This contribution is concluded finally in section 8.

## 2 Governing Equations

The Navier–Stokes momentum and continuity equations are usually accompanied by a third scalar transport equation to simulate different physical applications. For example, the turbulent flow past an object requires supporting these equations with a turbulent model such as the Spallart–Allmaras model [29]. In case of heat transfer problems, the equations are accompanied by a heat equation to govern the change of temperature in the domain. The general form of the governing equations is shown in (1), where the third equation is a comprehensive form of the scalar transport equation that can be used to compute different fields based on the problem setup.

$$\begin{aligned} \rho(\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}) &= \nabla\cdot(-p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u})) + \boldsymbol{\psi}, \\ \nabla\cdot\mathbf{u} &= 0, \\ \alpha\partial_t F + \beta\mathbf{u}\cdot\nabla F &= \nabla\cdot(\gamma\nabla F) + \chi. \end{aligned} \tag{1}$$

In the above system, $\mathbf{u}$ and $p$ are respectively the velocity and pressure fields, while $F$ is the coupled scalar field required for the test case simulation. In the case of heat transfer, $F$ would represent the temperature distribution in the domain, while for turbulent flows, $F$ would represent the Spallart–Allmaras scalar, $\tilde{\nu}$. $\rho$ and $\mu$ are the fluid's density and dynamic viscosity respectively, while $\alpha$, $\beta$, and $\gamma$ are coefficients specific to the coupled model. Finally, $\boldsymbol{\varepsilon}(\mathbf{u}) = (\nabla\mathbf{u} + \nabla\mathbf{u}^T)/2$ is the rate of deformation tensor, while $\boldsymbol{\psi}$ and $\chi$ are the source terms.

For problems concerned with an interaction between a fluid and a solid phase, several coefficients should be predefined to govern the exchange at the interface. The investigation of the value of the coefficients is an exhaustive task and requires previous research. To overcome this difficulty, the immersed volume method is suggested [7]. Using IVM, the problem is defined using a single fluid domain but with different properties for the solid and the fluid materials. Unifying the domain eliminates the need of specifying coefficients that may vary with shape, size, or position. Thus, the interaction between both phases is implicitly derived due to different properties across the elements. Sections 2.1, 2.2, and 2.3 introduce the main building ingredients required for the successful implementation of this method. Furthermore, the resulting variational formulation of the problem is prone to instability and numerical oscillations due to the presence of large convective fields. The variational multi-scale method is introduced in section 2.4 to stabilize the discretized form of the problem, resulting in accurate solution fields compared to the standard Galerkin formulation.

### 2.1 Level-set method

The efficiency of the IVM relies on multiple ingredients. The interface between the solid and the fluid must be accurately defined to enable the initialization of the varying properties between the solid

object and the fluid surrounding it. This interface is specified using a smoothly varying signed distance function, hereafter denoted $\phi$. Conventionally, $\phi$ attains a positive value in the fluid domain $\Omega_f$, a negative value within the solid domain $\Omega_s$, and is equal to zero on the interface $\Gamma_{interface} = \Omega_s \cap \Omega_f$, as summarized in (2). The following framework, used to specify the solid phase and define the interface by a zero-level set distance function, is known as the level set method [30].

$$\phi(x,y) = \begin{cases} -d\left((x,y), \Gamma_{\text{interface}}\right) & \text{if } (x,y) \in \Omega_s, \\ 0 & \text{if } (x,y) \in \Gamma_{\text{interface}}, \\ d\left((x,y), \Gamma_{\text{interface}}\right) & \text{if } (x,y) \in \Omega_f \end{cases} \tag{2}$$

where $d\left((x,y), \Gamma_{\text{interface}}\right)$ is the nearest distance from a point in the domain, $(x,y)$, to the fluid-solid interface, $\Gamma_{\text{interface}}$. Thus, the interface between the solid phase and the fluid phase, $\Gamma_{\text{interface}}$, will be identified by the zero-level set of $\phi$ as seen in equation (2). In the case of advection, the distance function is cast into a scalar transport equation. This equation accounts for any variation in the shape, size, displacement, etc., of the solid. The solution of such an equation requires numerical attention to ensure feasibility for long-time simulations and avoid the vanishing of the function with time. For more details regarding the level set method, the reader is invited to read [30, 31, 32, 33]. It should be noted that in our test case, the solid phase is fixed in position and vary neither in size nor in shape. Thus, the level set function does not evolve as the solution proceeds.

### 2.2 Mixing laws

After defining the level set function that enables the specification of different phases location, the distribution of properties along the domain can be initialized with the aid of appropriate mixing laws. Almost all physical properties are defined using the mixing law introduced in (3):

$$p(x,y) = p_f H_\epsilon\left(\phi(x,y)\right) + p_s\left(1 - H_\epsilon\left(\phi(x,y)\right)\right), \tag{3}$$

where $p$ is any physical property that affects the resulting simulation such as the density, viscosity, or specific heat. $p_f$ and $p_s$ are the values of this property for both the fluid phase and the solid phase, respectively. $H_\epsilon$ is the smoothed Heaviside function [34], which is used to obtain improved continuity at the fluid-solid interface by introducing a virtual thickness to the interface. The smoothed Heaviside function is parameterized by the interface thickness $\epsilon$, which depends on the mesh size at the interface and is used to specify the virtual thickness of the interface. The smoothed Heaviside function is defined as shown in equation (4):

$$H_\epsilon(\phi) = \begin{cases} 1 & \text{if } \phi > \epsilon, \\ \frac{1}{2}\left(1 + \frac{\phi}{\epsilon} + \frac{1}{\pi} sin(\frac{\pi\phi}{\epsilon})\right) & \text{if } |\phi| \leq \epsilon, \\ 0 & \text{if } \phi < -\epsilon. \end{cases} \tag{4}$$

It should be noted that references [35, 36] suggest using a harmonic mixing law for the thermal conductivity $\lambda$, rather than the general law defined in (3). The harmonic law is defined in equation (5) and is used to ensure continuity of the heat flux in addition to enhanced numerical accuracy:

$$\frac{1}{\lambda(x,y)} = \frac{1}{\lambda_f} H_\epsilon\left(\phi(x,y)\right) + \frac{1}{\lambda_s}\left(1 - H_\epsilon\left(\phi(x,y)\right)\right) \tag{5}$$

### 2.3 Anisotropic mesh adaptation

Simulations of processes with multiple phases, such as flow past a solid object, usually require a highly refined mesh to capture all the physics near the interface, leading to increased computational requirements. The computation time of such simulations can be drastically reduced using anisotropic mesh adaptation while maintaining the accuracy and reliability of the simulation, as shown in [37, 38]. The anisotropic adaptation reduces the number of nodes required by focusing on regions with

large variations in the physical or geometrical fields. The metric used to perform the adaptation may depend on multiple variables where a method based on the intersection of all variables' metrics is used to compute it. For example, in the case of forced convection, the normalized fields of the velocity components, velocity magnitude, the level set, and the temperature, are used to compute the adaptation metric. Moreover, anisotropic mesh adaptation avoids numerical oscillations by aligning the element edges with the interface. Also, it reduces the virtual thickness of the interface due to the refinement of the mesh size along it. The adaptation method used in this paper follows the same method used in [39]. The reader is advised to refer to [40] for more details regarding the method summarized below.

To compute the required adaptation metric, the procedure starts by estimating an error associated with each edge. The error indicator function $e^{ij}$, associated to an edge $x^{ij} = x^j - x^i$ (where node $j$ lies in the patch $\Gamma(i)$ of the nodes sharing a single edge with the node $i$), is defined using the exact interpolation error shown in (6):

$$e^{ij} = \left| g^{ij} \cdot x^{ij} \right|, \tag{6}$$

where $g^{ij}$ is the change in the gradient along edge $x^{ij}$ of a P1 finite element Lagrange approximation function $u_h$, *i.e.* $g^{ij} = g^j - g^i = \nabla g \cdot x^{ij}$. However, the gradient is not known at the nodes, and thus an estimated error function is defined using a recovered gradient $G^i$, computed using a length distribution tensor $X^i$, as shown in equation (7):

$$e^{ij} = G^{ij} \cdot x^{ij}, \tag{7}$$

where the recovered gradient at node $i$, $G^i$, is defined as:

$$G^i = (X^i)^{-1} \sum_{j \in \Gamma(i)} U^{ij} x^{ij} \tag{8}$$

while the length distribution tensor $X^i$, is equal to:

$$X^i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} x^{ij} \otimes x^{ij} \tag{9}$$

After computing the estimated error, a stretching factor relative to every edge is defined:

$$s^{ij} = \frac{e_{ij}}{e(N)}, \tag{10}$$

where $e(N)$ is the total mesh estimated interpolation error. Finally, the required metric $\tilde{M}^i$, is computed as shown in (11):

$$\tilde{M}^i = \left( \tilde{X}^i \right)^{-1}, \tag{11}$$

where,

$$\tilde{X}^i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} s^{ij} \otimes s^{ij}.$$

### 2.4 Variational multi-scale approach

The system of governing equations (1) can then be resolved using the finite element method, to obtain accurate solutions eligible to be used as ground truth for the training of a deep learning model.

To obtain the weak form of (1), each equation is multiplied by its specific weighting function, and integration by parts is performed, leading to the following weak form:

$$\begin{aligned}(\rho(\partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}), \boldsymbol{w}) + (2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}), \boldsymbol{\varepsilon}(\boldsymbol{w})) - (p, \nabla\cdot\boldsymbol{w}) + (\nabla\cdot\boldsymbol{u}, q) = (\boldsymbol{\psi}, \boldsymbol{w}),\\ ((\alpha\partial_t F + \beta\boldsymbol{u}\cdot\nabla F), s) + (\gamma\nabla F, \nabla s) = (\chi, s),\end{aligned} \tag{12}$$

where $\boldsymbol{w}$, $q$, and $s$ are respectively the test functions for velocity, pressure, and the required scalar field $F$. Yet, the Galerkin discrete formulation of the above weak form may fail if the flow is advection-dominated, or if the space discretization of the problem variables does not satisfy the LBB criteria (also known as the Babuska-Brezz condition or the inf-sup condition) [41]. To ensure the convergence of the above system to a unique solution not polluted by artificial oscillations, the variational multiscale method (abridged to VMS) will be utilized [24].

The VMS method starts by decomposing the required fields and their weighting functions, *i.e.* the velocity, pressure, and the scalar variable governed by the transport equation, into two scales: (i) a resolvable coarse-scale, and (ii) an unresolved fine-scale. Replacing the decomposed fields into the variational form results in two sub-problems specific for every scale. For the fine-scale Navier–Stokes problem, a separation technique is proposed to facilitate the resolution of the unknown fine fields [42, 43]. The continuity equation is replaced by a pressure Poisson equation allowing to approximate the fine scale pressure as a product of a stabilization parameter and the residual of the continuity equation. The fine-scale velocity field is similarly expressed from the momentum equation, with the aid of a bubble function, as a product of another stabilization parameter, and the residual of the coarse-scale momentum equation. The assumptions required to reach the above forms of the fine-scale fields are justified in [44, 45, 46]. The definition of the stabilization parameters appearing in the continuity and momentum equation can be found in [47, 48]. A similar approach is applied for the convection-diffusion-reaction transport equation, [49, 50], and the definition of its corresponding stabilization parameters is stated in [51, 52].

For the coarse-scale problem, the fine-scale fields are replaced by their corresponding expressions obtained from solving the fine-scale problem. This eliminates the appearance of the fine scales but preserves their effects on the coarse-scale problem:

$$\begin{aligned}&(\rho(\partial_t\boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u})\ ,\ \boldsymbol{w}) + (2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})\ ,\ \boldsymbol{\varepsilon}(\boldsymbol{w})) - (p\ ,\ \nabla\cdot\boldsymbol{w}) + (\nabla\cdot\boldsymbol{u}\ ,\ q) = (\boldsymbol{\psi}\ ,\ \boldsymbol{w})\\ &\qquad + \sum_{K\in\mathcal{T}_h}[(\tau_1\mathcal{R}_M\ ,\ \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{w})_K + (\tau_1\mathcal{R}_M\ ,\ \nabla q)_K + (\tau_2\mathcal{R}_C\ ,\ \nabla\cdot\boldsymbol{w})_K]\\ &\big((\alpha\partial_t F + \beta\boldsymbol{u}\cdot\nabla F)\ ,\ s\big) + (\gamma\nabla F\ ,\ \nabla s) = (\chi\ ,\ s)\\ &\quad + \sum_{K\in\mathcal{T}_h}[(\tau_3\mathcal{R}_F\ ,\ \boldsymbol{u}\cdot\nabla s)_K + (\tau_4\mathcal{R}_F\ ,\ (\boldsymbol{u}\cdot\nabla F/||\nabla F||^2)\nabla F\cdot\nabla s)_K]\end{aligned} \tag{13}$$

where $\mathcal{R}_M$, $\mathcal{R}_C$, and $\mathcal{R}_F$ are respectively the approximate residuals of the momentum, continuity, and scalar transport equation. The diffusion term is neglected in the residuals as can be seen in (14):

$$\begin{aligned}-\mathcal{R}_M &= \rho(\partial_t\boldsymbol{u} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}) + \nabla p - \psi,\\ -\mathcal{R}_C &= \nabla\cdot\boldsymbol{u},\\ -\mathcal{R}_F &= \alpha\partial_t F + \beta\boldsymbol{u}\cdot\nabla F - \chi.\end{aligned} \tag{14}$$

It can be seen that the resulting discrete stabilized variational form of the problem, (13), includes additional integrals over the sum of element interiors compared to the standard Galerkin formulation. These additional terms enrich the coarse-scale solution of the fields with fine-scale characteristics. Moreover, adding these terms relieves the restrictions on the discrete fields spaces, enhances its accuracy, and stabilizes the problem by including dissipative small scales contributions. For more extensive details regarding the VMS method, the reader is referred to [53, 54]. The resulting set of equations is solved sequentially using an industrial-level in-house VMS solver, the results of which were previously validated [27, 28]. The solver computes the velocity and pressure fields before computing the additional scalar governed by the scalar transport equation. It should be noted that all linear systems

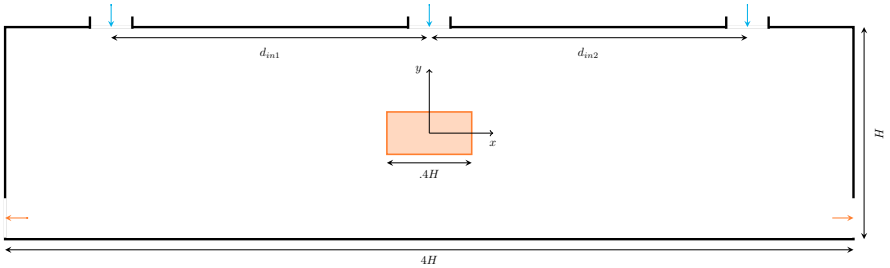



Figure 1: **Setup schematic** required for cooling a hot object with fluid flow. The cold fluid inlets are represented with blue arrows while the hot fluid outlets with red ones. This specific setup shown has inlets placed symmetrically on both sides with $d_{in1} = d_{in2} = 1.5H$.

are preconditioned with a block Jacobi method supplemented by an incomplete LU factorization, and solved with the GMRES algorithm.

## 3 Problem Definition

The problem detailed in this paper concerns the forced cooling of a hot workpiece by a cold fluid flow, placed in a rectangular cavity. Such a problem requires the coupling of Navier–Stokes equations with a heat energy equation responsible for the evolution of temperature across the domain. Thus, the problem is governed by a set of equations similar to (1), but with a scalar transport equation specific to heat transfer:

$$\rho c_p(\partial_t T + \mathbf{u} \cdot \nabla T) = \nabla \cdot (\lambda \nabla T) + \chi, \tag{15}$$

where $\rho$, $c_p$, and $\lambda$ are respectively the fluid density, specific heat, and thermal conductivity. This equation is a scalar transport equation similar to the one defined in (1) with $\alpha = \beta = \rho c_p$ and $\gamma = \lambda$. However, the fluid flow and its properties are assumed to be invariant for the variations in the temperature field, meaning that the coupling in this problem is weak, and the interaction between the computed fields is unidirectional.

A schematic of the problem setup is shown in figure 1. The setup consists of a $0.2H \times 0.4H$ hot rectangular object placed at the center of a closed rectangular cavity with a width of $4H$ and a height of $H$. The upper wall of the rectangular cavity contains three similar inlets, $0.2H$ in width, required for the blowing of the cold fluid. The distance of the side inlets to the origin of the cartesian coordinate system coinciding with the center of mass of the object, $d_{in1}$ and $d_{in2}$, is varied throughout the different data sets generated to enable us to train the model for different setups and test its generalization capacity on new ones as will be detailed later. Finally, the sidewalls of the rectangular cavity are equipped with outlets, having the same dimensions as the inlets, to allow the escape of the hot fluid. The outlets are positioned at the bottom of the sidewalls as is depicted in the problem setup schematic.

To define the flow problem, the magnitude of the velocity at the inlets is set to $V_{in} = 1\text{m/s}$ with a fluid density of $\rho_f = 1\text{kg/m}^3$ and dynamic viscosity of $\mu_f = 0.001\text{Pa·s}$. The outlets are designated with a zero-pressure condition. A no-slip boundary condition is specified on the remaining boundaries. As detailed in section 2, the immersed volume method is utilized to simulate the flow around the immersed object. Thus, the solid phase is defined as a fluid with relatively high density and viscosity, $\rho_s = 100\text{kg/m}^3$ and $\mu_s = 1000\text{Pa·s}$. The very large magnitude of the ratio between the solid viscosity and that of the fluid, $\frac{\mu_s}{\mu_f} = 10^6$, ensures the satisfaction of the no-slip condition at the solid-fluid interface along with a zero velocity in the solid domain. Also, due to zero velocity in the solid domain, the convective term in the energy equation diminishes, resulting in a pure conduction equation

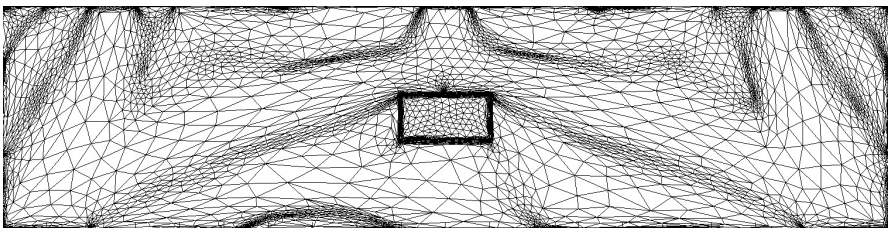


Figure 2: **Adapted mesh** at a certain time step used to discretize the computational domain.

whenever the radiation effect is neglected (*i.e.* $\chi = 0$). The source term in the NS equations is also set to zero, thus neglecting any buoyancy forces or stress on the fluid.

Concerning the heat equation, the hot temperature of the solid is initialized to $T_h = 150°\text{C}$, whereas the cold temperature of the fluid at the inlet, $T_c$, is set to 10°C. Isothermal condition is enforced on all the walls with $T_w = T_c = 10°\text{C}$. The heat exchange at the interface is implicitly regulated by specifying different values for the thermal properties of the composite fluid. For instance, the thermal conductivity $\lambda_f$ and specific heat $c_{p,f}$, for the fluid are set to 0.5W/m·K and 1000J/kg·K respectively, while those of the solid are set to: $\lambda_s = 15\text{W/m·K}$ and $c_{p,s} = 300\text{J/kg·K}$.

An unstructured triangular mesh of 15 000 elements, accompanied with anisotropic mesh adaptation as detailed in section 2.3, is used to discretize the computational domain. Figure 2 shows the developed mesh at a specific time step after the flow has stabilized. A visualization of the computed fields that will be fed to the model, *i.e.* the temperature and velocity components, is also shown in figure 3.

## 4 Generated datasets

The main objective of this paper is to assist a numerical solver by directly inferring the temperature field using a deep learning network modeling the scalar transport equation. Thus, the model should be trained to infer the temperature field for different cooling setups. To attain this objective, the fields are computed for multiple inlet positions with the same flow conditions. The dataset is assembled with the temperature, at a certain time step, along with the velocity components as input features and the temperature field, at a future time step, as the desired target. Figure 4 shows a representative sketch of the model with its corresponding inputs and inferred output.

The fields are computed using a CFD solver that discretizes the domain into an unstructured irregular triangular mesh. However, since the model used will be based on 2D convolutional layers, the computed fields must be interpolated to a structured encoding mesh. Thus, two encoding meshes are suggested, a $161 - by - 641$ mesh for the input field and a $176 - by - 656$ mesh for the output fields. The size of the output mesh is implicitly specified by the model architecture since no interpolation or cropping layers are utilized to enforce a certain size, thus preserving the information encoded along the feature maps boundaries. The computed fields will be transported from the unstructured triangular mesh, using linear Lagrange interpolation elements, to the required structured representation.

The first dataset, denoted as **SymVar**, is obtained by computing the fields for 18 different symmetrical inlet positions. The inlet positions are obtained by varying the distance from the origin to the center of the side inlets, $d_{in1}$ and $d_{in2}$, between $0.1H$ and $1.8H$ with a step of $0.1H$. The data is sampled after the flow has stabilized. The solver's time step, $\Delta t_{\text{solver}}$, is equal to 0.1 sec. However, the model will be trained to infer the fields with 30 times larger time step, meaning that, given the fields at $t_1$, the model infers the temperature field at $t = t_1 + 30\Delta t$. The choice of a larger model time step, compared to that of the original solver, is motivated by the reduction of the amount of operations required to span the entire time domain. Yet, the choice of the model time step still depends on the scalar field being inferred, and should avoid discarding variations necessary for the inference of the future time step. Moreover, increasing the model time step will lead to a decrease in the data available

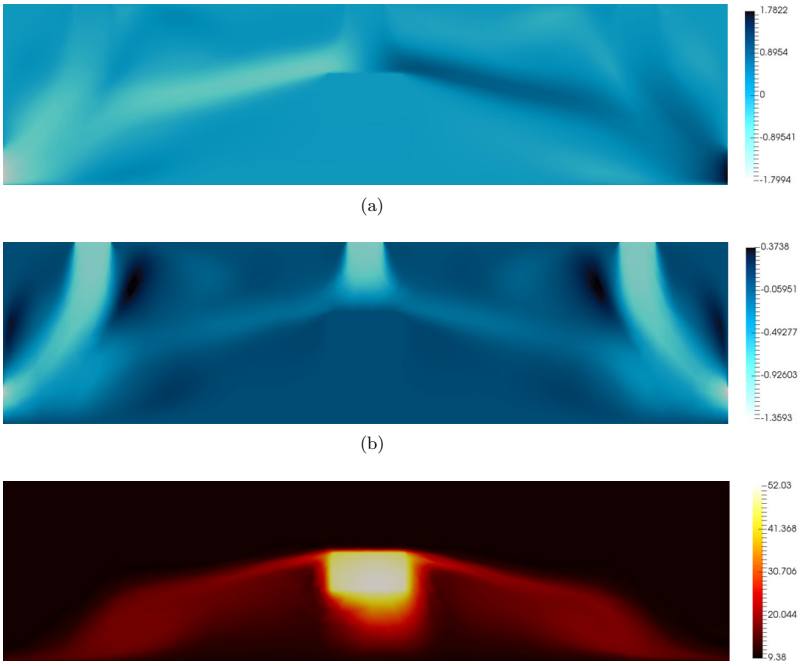



Figure 3: **Simulated fields** obtained at a developed time step after the flow has stabilized. The fields shown are the (a) Velocity $x$-component, (b) Velocity $y$-component, and (c) Temperature.

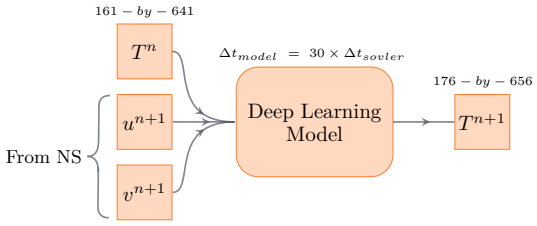



Figure 4: **Model sketch**

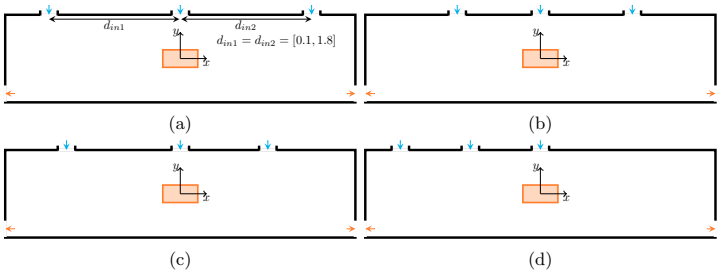



Figure 5: **Gathered datasets** used for training and testing the model generalization capacity from different cooling setups: (a) **SymVar** dataset, (b) **Sym1.05** dataset, (c) **UnSym** dataset and (d) **SameSide** dataset. The setups shown in 5b, 5c and 5d are never seen during training.

for training, thus possibly decreasing the inference quality of the model. For these reasons, the size of the incremental step of the model remains a user-defined parameter, which choice can be affected by the transport equation being modelled, as well as by the required model accuracy. 329 snapshots of fields are gathered from every simulation between $t_i = 50$ s. and $t_f = 217$ s., leading to a total of 5922 snapshots. It should be noted that for the training dataset, the fields are sampled once every 5 time steps only, allowing us to test the ability of the model to infer the in-between fields.

70% of the **SymVar** dataset is reserved for training, while the remaining portion is used to test and validate the model. Each sample of the dataset contains, as features, an encoded representation of the velocity fields, $\mathbf{u}^{n+1}$, and the temperature field $T^n$. The temperature field at the next desired step, $T^{n+1}$, is provided for every sample as a label. The notation $n+1$ denotes that the fields are computed at $t = t_n + 30\Delta t$. Other datasets are obtained and used only to test the performance of the model. Dataset **Sym1.05** contains samples from a CFD simulation where the inlets are symmetrically placed at $d_{in1} = d_{in2} = 1.05H$ from the origin. Dataset **UnSym** is obtained by placing the inlets at different distances from the origin, *i.e.* $d_{in1} = 1.3H$ and $d_{in2} = 1H$. The third dataset, **SameSide**, is collected by placing both inlets on the same side with $d_{in1} = 0.8H$ and $d_{in2} = 1.6H$. Figure 5 summarizes the location of the inlets for the different datasets used.

## 5 Model Architecture

The task of the considered neural network is to model the scalar transport equation, shown in (1). Thus, instead of solving both the NS equations and the transport equation using traditional numerical methods, a deep learning regression model will be responsible for the resolution of the scalar field $F$. The architecture of the model used to accomplish the above-mentioned task is an auto-encoder-like end-to-end network [19], presented in figure 6. While auto-encoders are designed to generate a compressed representation of their input [55, 56, 57, 58], in the suggested model, the output of the model is a distinct field. Similar architectures have been previously employed to infer other physical fields for fluid flow problems [59, 60]. The model consists mainly of three components: encoder, bottleneck, and decoder. The encoder gathers the information from the input fields and compresses them to a reduced dimension representation denoted as the bottleneck (also known as the latent space). The spatial reduction, in the encoder, is handled using downsampling convolutional layers with a stride of two. These layers are preceded by a single-stride convolutional layer and followed by batch normalization [61], and ReLU non-linear activation [62]. This pattern of layers is repeated $N = 4$ times as seen in the upper branch of figure 6. The encoder is followed by a decoder responsible for inferring the required scalar field from the compressed latent space. The building block of the decoder consists of a single-stride convolutional layer, followed by a deconvolutional layer responsible for the upsampling of the input to a larger dimensional space, in addition to a non-linear activation function. This pattern of layers is repeated the same number of times as its contrast downsampling pattern in the encoder. The decoder is represented by the bottom branch in figure 6.

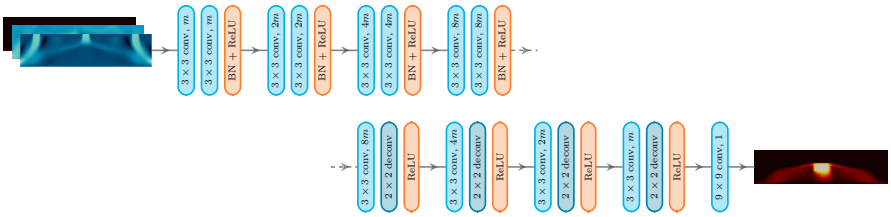



Figure 6: **Deep learning model architecture** used to infer the required temperature given temperature, at previous time step, and velocity components as feature inputs.

It should be mentioned that the physical fields, fed to the model, are priorly normalized. This standard preprocessing operation unifies the scale of all the fields and thus ensures a faster training process as is detailed in appendix A. Moreover, the convolutional layers present throughout the model are preceded by a padding layer following the reflect scheme. Using such a scheme preserves the physical statistical distribution of the feature maps without introducing artifacts near the boundaries [63]. Appendix B supports our choice of reflect padding by comparing the performance of the model with different schemes. The resolution of the discretization in the encoding meshes is specified while keeping in mind the available memory to store the data and train the model. Finer structured meshes would allow the model to execute superior predictions but at the cost of larger memory requirements and longer training times. The aspect ratio of the encoding mesh is chosen similar to that of the computational domain, *i.e.* $\frac{n_y}{n_x} \approx \frac{\mathrm{H}}{4H}$, where $n_x$ and $n_y$ are the number of nodes in the $x$ and $y$ direction respectively. This choice of aspect ratio is made to ensure the feature maps are not biased toward a spatial dimension despite what consequences would have resulted from such bias. Moreover, the input encoding mesh dimensions are selected while assuring the downsampling convolutional layers present in the encoder do not violate equation (16):

$$h_i = s \times (h_o - 1) + k - 2p, \tag{16}$$

where $h_i$ and $h_o$ are respectively the input and output dimension size of the image and $s$, $k$ and $p$ are respectively the kernel stride, size and padding of the considered layer. Violating equation (16) for downsampling convolutional layers leads to uneven padding operations, which may deteriorate the performance of the model [63]. It should be noted that the next possible encoding mesh size respecting equation (16) and maintaining aspect ratio close to 1/4, will possess 21,588 additional pixels which require around 20% supplementary storage and memory space for every field.

## 6 Training

The inferring quality of the deep learning model described in setion 5 depends on the trainable set of weights and biases. These parameters define the model and its ability to infer the required scalar fields. After randomly initializing them with Gaussian distribution, the optimal set of parameters is obtained by minimizing a certain loss function quantifying the error between the inferred fields and the reference ones. In our model, 2 937 025 parameters are optimized, using Adam optimizer [64], with a learning rate equal to $10^{-5}$. The optimizer iteratively minimizes the mean squared error (MSE), computed between the reference fields and the predicted ones and defined as:

$$\mathcal{L} = \frac{1}{N_{\text{samples}} \times n_x \times n_y} \sum_{i=1}^{N_{\text{samples}}} \sum_{r=1}^{n_y} \sum_{c=1}^{n_x} \left(F^i_{r,c} - \tilde{F}^i_{r,c}\right)^2 \tag{17}$$

where $N_{\text{samples}}$ is the number of samples for every optimization step, $n_x$ and $n_y$ are respectively the number of nodes in the $x$ and $y$ direction for the output encoding mesh, *i.e.* $n_x = 176$ and $n_y = 656$,

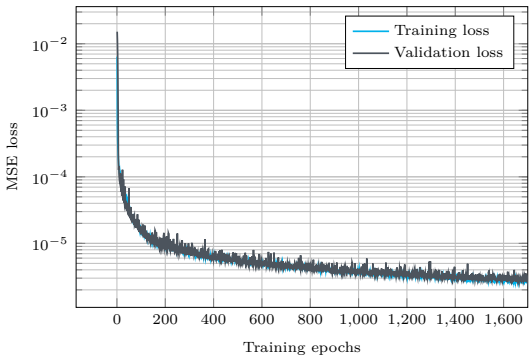



Figure 7: **Training curve** showing the evolution of the MSE loss, for both the training and validation dataset, over the number of epochs.

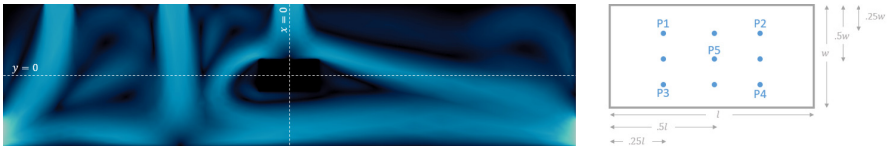



Figure 8: **Location of sampling probes**. On left, probes that capture the variation of the field along a specific line. On right, probes inside working object to capture variation over time.

$F^i_{r,c}$ is the true scalar value at a specific location for a sample $i$ and $\tilde{F}^i_{r,c}$ is the value inferred by the model. The training is performed using the TensorFlow API [65], on an Nvidia Tesla V100 GPU. The optimization is performed using mini-batches of size 32 for a total of 1651 epochs. The training is terminated using the early stopping criterion after a satisfying accuracy is attained and before the performance on the validation dataset starts to deteriorate, as can be seen in figure 7.

## 7 Results and discussion

Although the temperature fields are inferred on the whole computational domain, the main interest of any industrial practitioner will be biased toward the temperature fields within the object being cooled. To focus more on this region, temperature probes are installed at various positions within the object as is demonstrated in figure 8. These probes allow plotting the evolution of the temperature field over the simulation time. Plots of the variation of the temperature field over predefined lines in the computational domain will also be obtained. Moreover, the plot of the temperature field in the region of interest, and the computation of a normalized error, will also allow the evaluation of the model on various data sets with different flow characteristics.

First, the input fields from a simulation with symmetrical inlets, placed at $d_{in1} = d_{in2} = 1.5H$, will be provided for the model to assess its interpolation ability over time. This setup is partially included in the **SymVar** dataset used to train the model where only 12% of the total 1971 time steps are previously seen during training and no snapshot was provided at a step exceeding 2300. This dataset will be referred to as **Sym1.5** in the figures and tables used to assess the model performance. The model extrapolation ability over time was tested for 200 extra time steps. Figure 9 shows the plots of temperature at various probes where almost exact similar fields are inferred by the model over the whole range of provided time steps. If the model is required to predict for further time intervals,

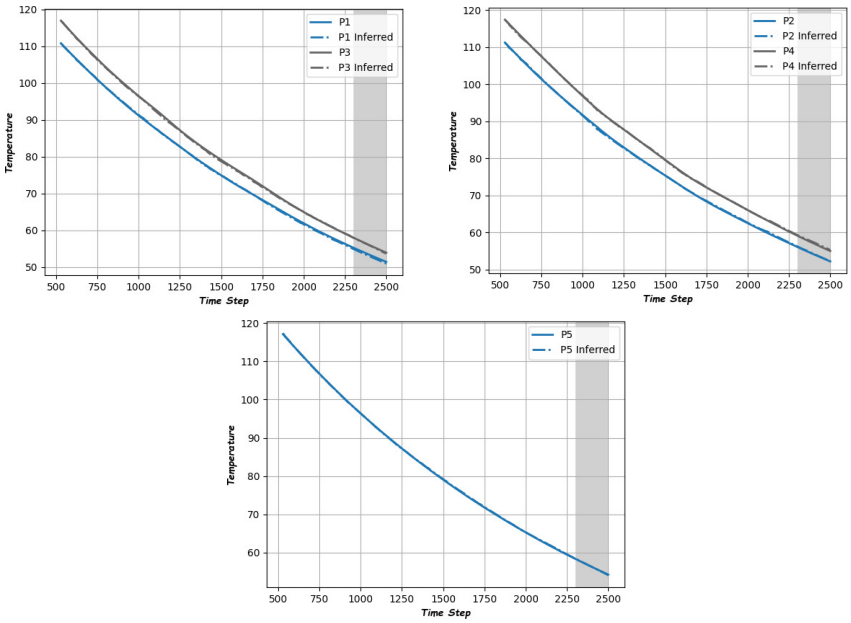



Figure 9: **Plot of computed and inferred field at various probe locations for setup with symmetrical inlets at $d_{in1} = d_{in2} = 1.5$.** The predicted fields are plotted with dashed lines, while the computed ones are plotted with solid lines. The shaded region, shown after time step 2300, represents the region where extrapolation over time occurs.

the time domain of the provided training dataset should be expanded to avoid divergence in model inferrings'.

The generalization capacity of the model to completely unseen inlet setups is also evaluated. Three data sets were created for this purpose: **Sym1.05, UnSym**, and **SameSide**. The first dataset, **Sym1.05**, with symmetrical inlets, placed at $d_{in1} = d_{in2} = 1.05H$, has similar flow characteristics as the dataset used to train the model but with different positions of inlets, not seen before by the model. The remaining two data sets, **UnSym** and **SameSide**, have both their inlets placed at an unsymmetrical location with respect to the origin. However, the **SameSide** dataset, with both inlets placed on the left of the object, shows a very large change in the characteristics of the flow compared to the other sets. Figure 10 shows the flow fields resulting from all three different setups.

Figure 11 shows the line plots of temperature across specific $x$ and $y$ values shown in figure 8. The line plots are obtained at the same single time step for all sets. For the first three data sets, **Sym1.5**, **Sym1.05**, and **UnSym**, the model was able to accurately infer the variation of temperature over the whole computation domain. This can be seen by the overlapping plots of the inferred field and the label field obtained from a CFD simulation. For the **SameSide** dataset, although a larger deviation between the fields can be observed, mainly in the fluid domain where the flow characteristics drastically change, the model is still able to infer the general trend of the variation of the field. To quantify the quality of the inferred fields over the different inlet setups, the $l_2$ normalized error, defined in equation (18), is computed for each sample:

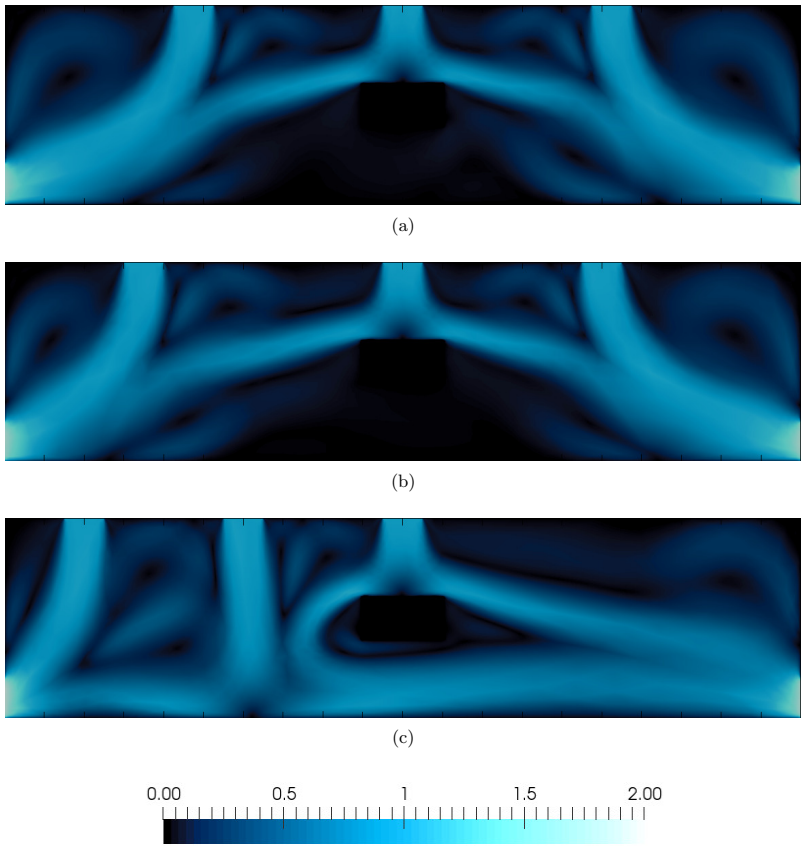



Figure 10: **Velocity magnitude** from the different flow setups used to test the model generalization capacity. Each setup is for a specific dataset: (a) **Sym1.05** dataset, (b) **UnSym** dataset, and (c) **SameSide** dataset.

| Dataset | Sym1.5 | Sym1.05 | UnSym | SameSide |
|---|---|---|---|---|
| Average | $1.27 \times 10^{-2}$ | $1.54 \times 10^{-2}$ | $1.33 \times 10^{-2}$ | $1.09 \times 10^{-1}$ |
| Maximum | $1.92 \times 10^{-2}$ | $2.35 \times 10^{-2}$ | $1.66 \times 10^{-2}$ | $1.35 \times 10^{-1}$ |
| Time Step | 1575 | 567 | 1745 | 637 |
| Focused Average | $2.6 \times 10^{-3}$ | $3.09 \times 10^{-3}$ | $3.58 \times 10^{-3}$ | $3.23 \times 10^{-2}$ |

Table 1: **Summary of obtained error.** The average of the $l_2$ normalized error is computed across all samples. The time step corresponds to the step at which the maximum error is found. The focused average is computed only over the object domain.

$$error = \frac{||T^n - \tilde{F}^n||}{||T^n||}, \tag{18}$$

where $||\cdot||$ is the $l_2$ norm of the physical field. $T^n$ is the true scalar field at time step $n$, and $\tilde{F}^n$ is the model inferred one.

The model performance is approximately similar over the **Sym1.5, Sym1.05**, and **UnSym** sets with an average error ranging from $1.27 \times 10^{-2}$ to $1.54 \times 10^{-2}$. The error increases for the **SameSide** set, as expected due to a large change in flow characteristics, with a maximum not exceeding $1.35 \times 10^{-1}$. All results are summarized in table 1 along with the time step that corresponds to the maximum error in each setup. Moreover, the error in the domain of the object is also computed neglecting the surrounding domain, and shown in the last row of Table 1. This error is an order of magnitude lower than its counterpart due to fewer changes in the object domain with the variation in the positions of the inlets. The error fields maps, computed using the absolute error normalized with the true temperature in Kelvin, are shown in figure 15. Moreover, a visual comparison between the inferred fields and their computed counterpart in the region of interest, *i.e.* the domain of the object, is shown in figure 14. The obtained fields are almost identical except for the **SameSide** set where deviation from the true field occurs but maintains the same range of temperature values.

Finally, the trained network, modeling the scalar transport equation, is incorporated in the solution loop as shown in figure 12. Initially, the CFD solver is utilized to resolve both governing equations, *i.e.* the Navier-Stokes and the transport equations, until all flows are well established. When reaching $t = 50$ sec, the trained deep learning model can start being used to infer the scalar field instead of the CFD solver. The temperature, $T^n$, along with the velocity fields obtained from resolving the Navier-Stokes equation, $u^{n+1}$ and $v^{n+1}$, are interpolated into the required encoding mesh of the model. The model will infer the temperature field $T^{n+1}$ at the next model time step, $t^{n+1} = t^n + 30\Delta t_{solver}$. The inferred temperature field is recycled by encoding it and feeding it back to the model for multiple times $f$. After multiple predictions, the traditional transport solver is utilized to redirect the solution before proceeding again with the model. The model predicted field, $T^{n+f}$, is interpolated back to the unstructured mesh to enable the finite element transport equation solver to operate on. The input required by the traditional solver to resolve the field at the next time step is identical to that of the model. The solver inferred field, $T^{n+f+1}$, can then be fed back to the deep learning model. This sequence is repeated until the whole time domain is spanned. The number of times the model is used consecutively $f$, affects the quality of the solution. Large $f$ values allow advancing with larger time steps throughout the required time interval but may lead to less accurate solutions.

The total time required for the industrial-level CFD solver to resolve both governing equations for $t \in [50, 250]$ sec is 491.92 seconds. Around 22% of this computational time is required for only resolving the scalar temperature field $T$. To explore the potential of our model, the value of $f$ is set to infinity, meaning that the temperature field across the whole time domain will be resolved by the deep learning model without referring back to the scalar finite element solver. Knowing that $\Delta t_{\text{model}} = 30 \times \Delta t_{\text{solver}}$, the model resolves the temperature field across the whole computation domain in 8.71s., *i.e.* around 12 times faster than the finite element scalar equation solver. At last, the $l_2$ normalized error for the predicted fields on the **Sym1.5** dataset, with $f$ set to infinity, maintains an average of $2.49 \times 10^{-2}$ and does not exceed the $3.61 \times 10^{-2}$ level as can be seen in figure 13a. The mean temperature field over

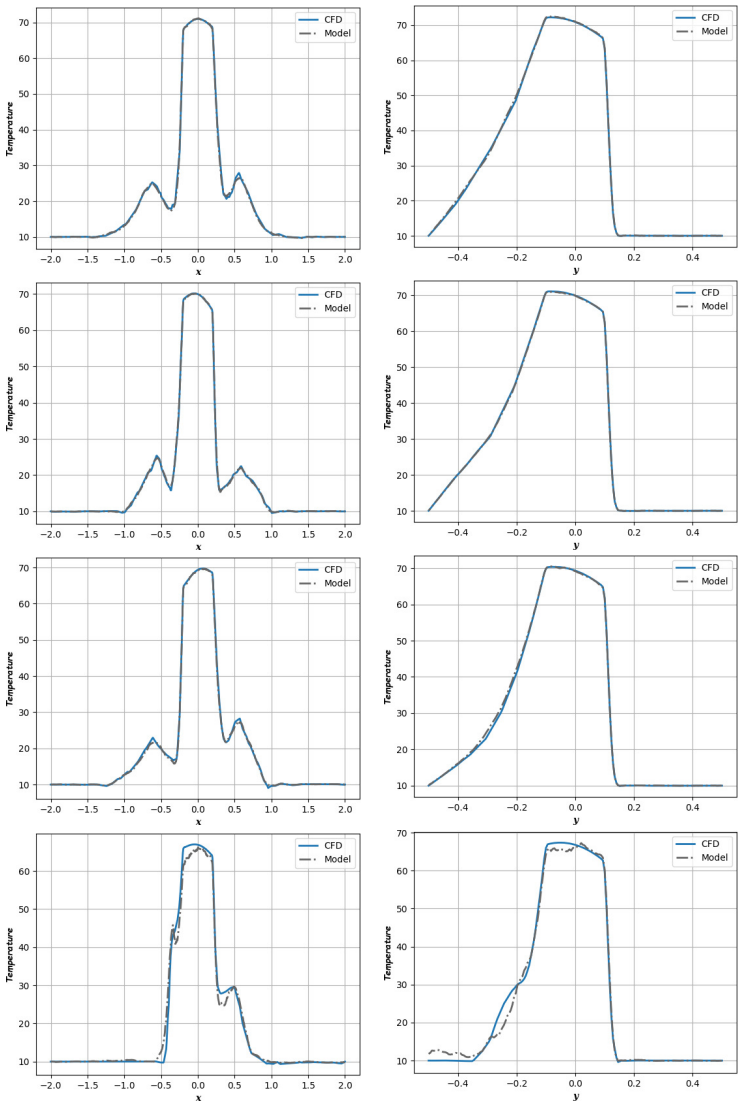



Figure 11: **Line plot of temperature variation over a certain direction**. The temperature field, both computed and inferred, at the same time step, $TS = 1750$, is plotted along both lines, $x = 0$ and $y = 0$, for different inlet setups. Every row of plots corresponds to a separate dataset. The order of sets is: **Sym1.5, Sym1.05, UnSym**, and **SameSide**.

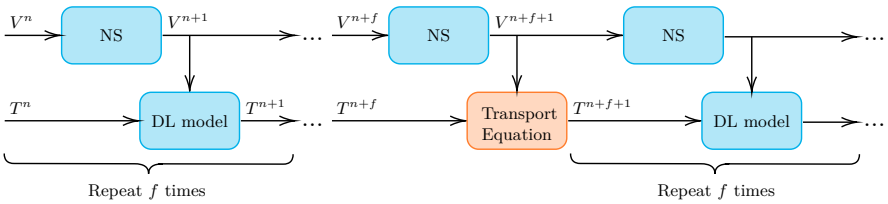
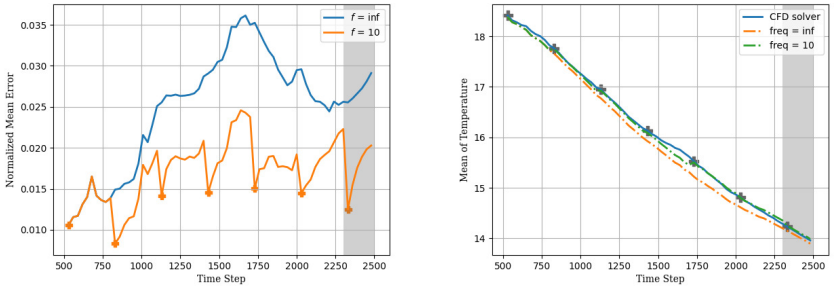



Figure 12: **Implementing the deep learning model in the solution loop**.




(a) (b)

Figure 13: **Implementation results for different values of** $f$ with (a) showing the plot of error evolution and (b) the plot of the average temperature evolution. In both figures, the cross sign is shown for the prediction obtained after the traditional solver is used.

the whole spatial domain is computed and compared to that obtained with the CFD solver as shown in figure 13b. Moreover, the results obtained with $f$ set to 10 are also shown in both figures, 13a and 13b.

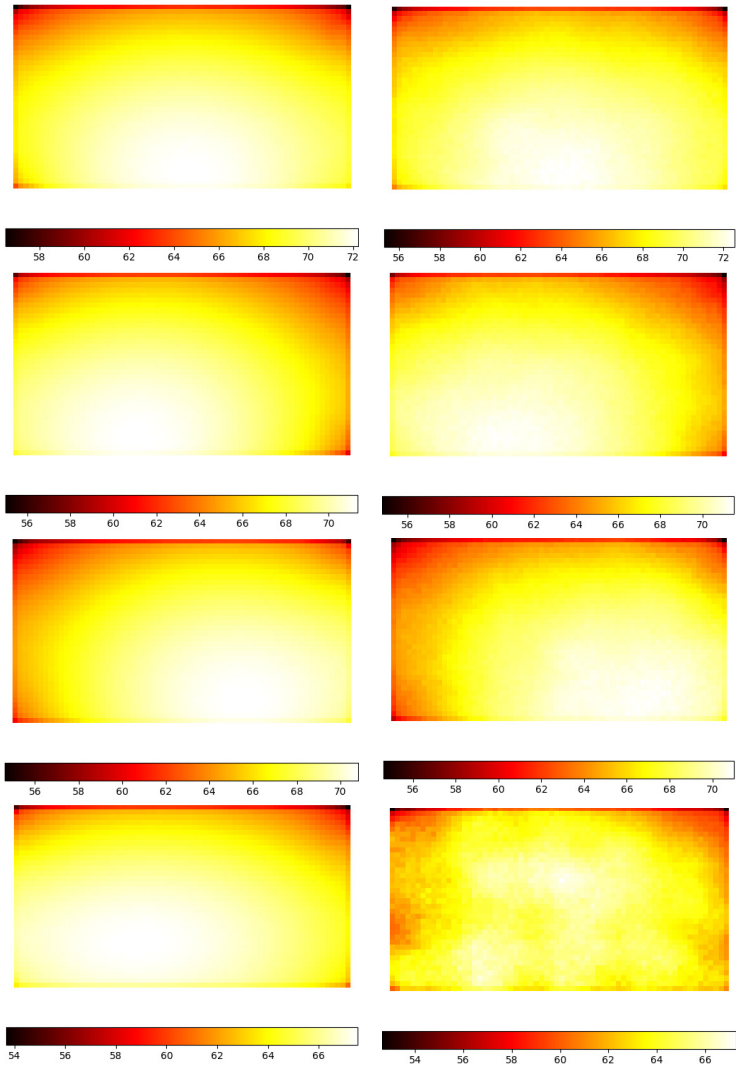



Figure 14: **Temperature field in object domain**. The temperature field, computed on left and inferred on right, at the same time step ($t = 1750\,\Delta t$), is plotted in the region of interest, for different inlet setups. Every row of plots corresponds to a separate dataset. The order of sets is: **Sym1.5, Sym1.05, UnSym**, and **SameSide**.

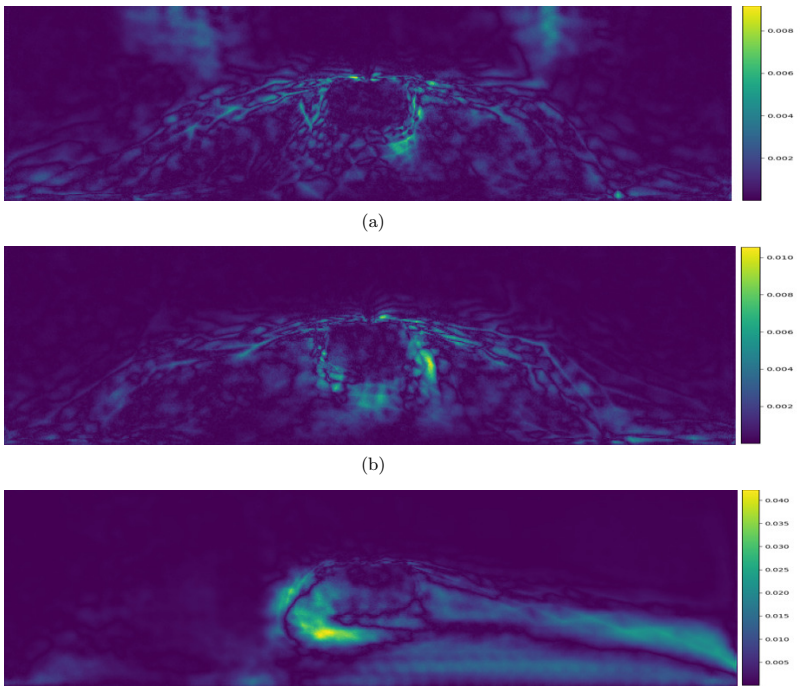



Figure 15: **Error field maps** obtained for the different inlet setups of the (a) **Sym1.05** dataset, (b) **UnSym** dataset and the (c) **SameSide** dataset.

## 8 Conclusion

In this work, a deep learning model, based on a convolutional network with an autoencoder architecture, is employed for inferring a certain scalar field. The model was trained for predicting the temperature field in a conjugate heat transfer problem and thus models the scalar transport equation in the coupled system of equations. Although the model was trained on samples obtained from a few different symmetrical cooling setups, it was able to accurately infer the fields for new inlet positions, whether symmetrical or not, with a maximum error of $1.35 \times 10^{-1}$. Also, the ability of the model to interpolate in the time-domain, permitted training on multiple cooling setups, with a few snapshots for each setup, distributed on this domain. Moreover, the model manages to span the whole time domain independently, *i.e.* without referring back to the scalar transport equation solver, 12 times faster than the industrial level CFD solver, and returns reliable predictions for the evolution of the temperature field with an average $l_2$ normalized error equals to $2.49 \times 10^{-2}$. These results motivate us to exploit the capacity of deep learning models, with similar architecture, for assisting in the simulation of numerous number of physical problems governed by a similar set of equations where the Navier–Stokes equations are coupled with a scalar transport equation responsible for the computation of a certain physical field.

### Data availability

The code of this project is available upon request from the corresponding author.

## References

[1] Jiri Blazek. Computational fluid dynamics: principles and applications. Butterworth-Heinemann, 2015.

[2] Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In 30th aerospace sciences meeting and exhibit, page 439, 1992.

[3] PG Huang, J Bardina, and T Coakley. Turbulence modeling validation, testing, and development. NASA technical memorandum, 110446:147, 1997.

[4] WP Jones and Brian Edward Launder. The prediction of laminarization with a two-equation model of turbulence. International journal of heat and mass transfer, 15(2):301–314, 1972.

[5] David C. Wilcox. Formulation of the k-w turbulence model revisited. AIAA Journal, 46(11):2823–2838, 2008.

[6] E. Hachem, G. Jannoun, J. Veysset, M. Henri, R. Pierrot, I. Poitrault, E. Massoni, and T. Coupez. Modeling of heat transfer and turbulent flows inside industrial furnaces. Simulation Modelling Practice and Theory, 30:35–53, 2013.

[7] E. Hachem, T. Kloczko, H. Digonnet, and T. Coupez. Stabilized finite element solution to handle complex heat and fluid flows in industrial furnaces using the immersed volume method. International Journal for Numerical Methods in Fluids, 68(1):99–121, 2012.

[8] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing, 2018.

[9] Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. A survey of deep learning techniques for neural machine translation, 2020.

[10] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: an overview. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 8599–8603, 2013.

[11] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018, 2018.

[12] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019.

[13] Thomas Daniel, Fabien Casenave, Nissrine Akkari, and David Ryckelynck. Model order reduction assisted by deep neural networks (rom-net). Advanced Modeling and Simulation in Engineering Sciences, 7:1–27, 2020.

[14] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[15] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021.

[16] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[17] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. Proceedings of the National Academy of Sciences, 116(31):15344–15349, 2019.

[18] Chady Ghnatios, George El Haber, Jean-Louis Duval, Mustapha Ziane, and Francisco Chinesta. Artificial intelligence based space reduction of structural models. ESAFORM 2021, 04 2021.

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[20] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215, 2014.

[21] Youssef Mesri, Hugues Digonnet, and Thierry Coupez. Advanced parallel computing in material forming with cimlib. European Journal of Computational Mechanics, 18(7-8):669–694, 2009.

[22] Jungwoo Kim, Dongjoo Kim, and Haecheon Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. Journal of computational physics, 171(1):132–150, 2001.

[23] Cyril Gruau and Thierry Coupez. 3d tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. Computer Methods in Applied Mechanics and Engineering, 194(48-49):4951–4976, 2005.

[24] Thomas JR Hughes, Gonzalo R Feijóo, Luca Mazzei, and Jean-Baptiste Quincy. The variational multiscale method—a paradigm for computational mechanics. Computer methods in applied mechanics and engineering, 166(1-2):3–24, 1998.

[25] Ramon Codina. Stabilization of incompressibility and convection through orthogonal sub-scales in finite element methods. Computer Methods in Applied Mechanics and Engineering, 190(13):1579–1599, 2000.

[26] Y. Bazilevs, V.M. Calo, J.A. Cottrell, T.J.R. Hughes, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. Computer Methods in Applied Mechanics and Engineering, 197(1):173–201, 2007.

[27] Elie Hachem, Hugues Digonnet, Elisabeth Massoni, and Thierry Coupez. Immersed volume method for solving natural convection, conduction and radiation of a hat-shaped disk inside a 3d enclosure. International Journal of numerical methods for heat & fluid flow, 2012.

[28] E. Hachem, S. Feghali, R. Codina, and T. Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. International Journal for Numerical Methods in Engineering, 94(9):805–825, 2013.

[29] Steven R Allmaras and Forrester T Johnson. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. In Seventh international conference on computational fluid dynamics (ICCFD7), volume 1902. Big Island, HI, 2012.

[30] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. Journal of computational physics, 79(1):12–49, 1988.

[31] Stanley Osher and Ronald Fedkiw. Level set methods and dynamic implicit surfaces, volume 153. Springer Science & Business Media, 2006.

[32] James Albert Sethian. Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science, volume 3. Cambridge university press, 1999.

[33] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. Journal of Computational physics, 183(1):83–116, 2002.

[34] Eric W Weisstein. Heaviside step function. https://mathworld. wolfram. com/, 2002.

[35] Suhas V Patankar. Numerical heat transfer and fluid flow. CRC press, 2018.

[36] Suhas V Patankar. A numerical method for conduction in composite materials, flow in irregular geometries and conjugate heat transfer. In International Heat Transfer Conference Digital Library. Begel House Inc., 1978.

[37] Luca Formaggia and Simona Perotto. Anisotropic error estimates for elliptic problems. Numerische Mathematik, 94(1):67–92, 2003.

[38] Johan Hoffman and Claes Johnson. Adaptive finite element methods for incompressible fluid flow. In Error estimation and adaptive discretization methods in computational fluid dynamics, pages 97–157. Springer, 2003.

[39] Cyril Gruau and Thierry Coupez. 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. Computer Methods in Applied Mechanics and Engineering, 194(48-49):Pages 4951–4976, November 2005.

[40] Thierry Coupez. Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing. Journal of computational physics, 230(7):2391–2405, 2011.

[41] Daniele Boffi, Franco Brezzi, Michel Fortin, et al. Mixed finite element methods and applications, volume 44. Springer, 2013.

[42] Leopoldo P Franca and A Nesliturk. On a two-level finite element method for the incompressible navier–stokes equations. International Journal for Numerical Methods in Engineering, 52(4):433–453, 2001.

[43] Ali Ihsan Nesliturk. Approximating the incompressible Navier-Stokes equations using a two-level finite element method. University of Colorado at Denver, 1999.

[44] F. Brezzi, L.P. Franca, T.J.R. Hughes, and A. Russo. $b = \alpha g$. Computer Methods in Applied Mechanics and Engineering, 145(3):329–339, 1997.

[45] Franco Brezzi and Michel Fortin. Mixed and hybrid finite element methods, volume 15. Springer Science & Business Media, 2012.

[46] Thierry Dubois, François Jauberteau, and Roger Temam. Dynamic multilevel methods and the numerical simulation of turbulence. Cambridge University Press, 1999.

[47] Ramon Codina. Stabilized finite element approximation of transient incompressible flows using orthogonal subscales. Computer Methods in Applied Mechanics and Engineering, 191(39):4295–4321, 2002.

[48] E. Hachem, S. Feghali, R. Codina, and T. Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. International Journal for Numerical Methods in Engineering, 94(9):805–825, 2013.

[49] Ramon Codina. Comparison of some finite element methods for solving the diffusion-convection-reaction equation. Computer Methods in Applied Mechanics and Engineering, 156(1):185–210, 1998.

[50] Santiago Badia and Ramon Codina. Analysis of a stabilized finite element approximation of the transient convection-diffusion equation using an ale framework. SIAM Journal on Numerical Analysis, 44(5):2159–2197, 2006.

[51] Alexander N. Brooks and Thomas J.R. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. Computer Methods in Applied Mechanics and Engineering, 32(1):199–259, 1982.

[52] Augusto Cesar Galeão and Eduardo Gomes Dutra do Carmo. A consistent approximate upwind petrov-galerkin method for convection-dominated problems. Computer Methods in Applied Mechanics and Engineering, 68(1):83–95, 1988.

[53] Elie Hachem, Benjamin Rivaux, Thibaud Kloczko, Hugues Digonnet, and Thierry Coupez. Stabilized finite element method for incompressible flows with high reynolds number. Journal of computational physics, 229(23):8643–8665, 2010.

[54] E. Hachem, T. Kloczko, H. Digonnet, and T. Coupez. Stabilized finite element solution to handle complex heat and fluid flows in industrial furnaces using the immersed volume method. International Journal for Numerical Methods in Fluids, 68(1):99–121, 2012.

[55] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pages 665–674, 2017.

[56] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pages 241–246. IEEE, 2016.

[57] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J. Kennedy. Relational autoencoder for feature extraction. 2017 International Joint Conference on Neural Networks (IJCNN), 2017.

[58] Carl Doersch. Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908, 2016.

[59] Kai Fukami, Yusuke Nabae, Ken Kawai, and Koji Fukagata. Synthetic turbulent inflow generator using machine learning. Phys. Rev. Fluids, 4:064603, Jun 2019.

[60] Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence, 2019.

[61] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[62] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 of Proceedings of Machine Learning Research, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[63] Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, Jun Yuan, and Orion Reblitz-Richardson. Mind the pad – {cnn}s can develop blind spots. In International Conference on Learning Representations, 2021.

[64] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[65] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[66] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application, 2020.

## Appendix A Exploring performance for different scaling methods

For deep learning approaches using gradient descent to optimize their parameters, it is suggested to maintain the input features at a similar scale. This practice is motivated by a faster convergence of the optimizer toward the requested minimum [66]. Moreover, scaling the target fields avoids encountering the exploding gradient problem. Multiple methods exist to scale the fields such as the min-max scaler, the standard scaler, the robust scaler, etc. The most common methods used are the min-max scaler, also known as field normalization, and the standard scaler (*i.e.* field standardization). The choice of the suitable scaling method depends on multiple factors and may vary between fields. For example, different resources suggest normalizing in case the field distribution is not Gaussian, and others suggest standardizing if outliers exist. In the present work, the temperature field is normalized, given its non-Gaussian distribution, and the performance of the model is evaluated for both scaling methods available for the velocity field. Moreover, the model is also trained with raw data as a reference for both scaling methods. Equation (A1) is used to normalize a field, while equation (A2) is used to standardize it:

$$X' = \frac{X - min}{max - min}, \tag{A1}$$

$$X' = \frac{X - \mu}{\sigma}, \tag{A2}$$

where $min$, $max$, $\mu$, and $\sigma$ are respectively the minimum, maximum, mean, and standard deviation of the field $X$, obtained based on the training dataset.

The plot of the loss curve over the validation dataset will be used to compare among the different scaling methods. Figure 16 shows the evolution of the validation loss for three similar models trained with fields scaled using different methods for 600 epochs. The inferred temperature field is restored to its original scale before computing the loss to allow the comparison between the different models.

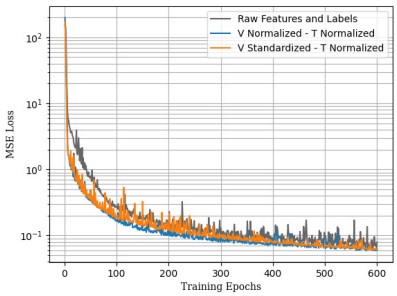



Figure 16: Validation loss for different scaling methods.

By inspecting the above figure, the advantages of scaling your fields before training are obvious since both models with scaled data encounter a faster decrease in the loss than the one with the raw data. Approximately similar performance is observed for both scaling methods, however, we chose to normalize both fields rather than standardizing velocity and normalizing temperature since slightly lower losses are obtained in some training epoch intervals. Using scaled data, the loss at epoch 600 decreases to $5.96 \times 10^{-2}$ for normalizing both fields, and $6.25 \times 10^{-2}$ for standardizing velocity and normalizing temperature, whereas, for raw data, a higher loss, $7.97 \times 10^{-2}$, is obtained.

## Appendix B Exploring performance for different padding methods

To choose the appropriate padding scheme, the model was trained with different padding methods. The investigation was limited to only three schemes and the validation loss, obtained after training is terminated using the early stopping criteria, is used to compare between the different schemes. Figure 17 shows the loss obtained during training over the validation dataset for the three padding schemes: same, reflect, and symmetric padding. Although the performance is slightly affected by the scheme used and all three curves look similar, the model trained with reflect padding scheme returned the lowest mean squared error with a value of $2.6420 \times 10^{-6}$, compared to $2.78 \times 10^{-6}$ and $2.88 \times 10^{-6}$ for same padding and symmetric padding, respectively.

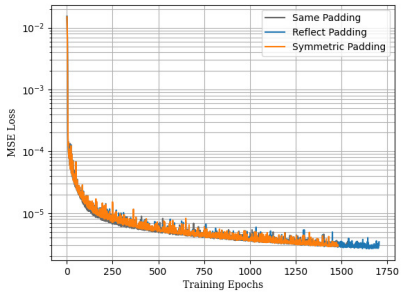
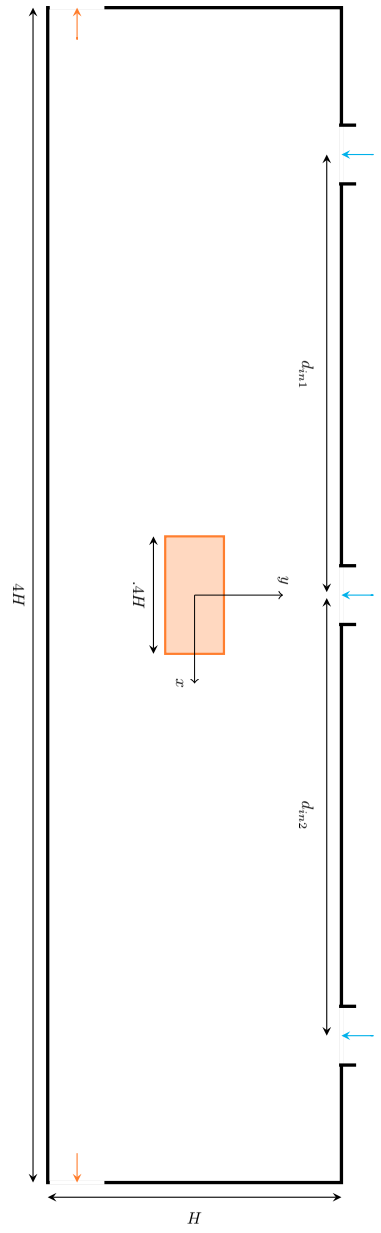



Figure 17: Validation loss for different padding methods.

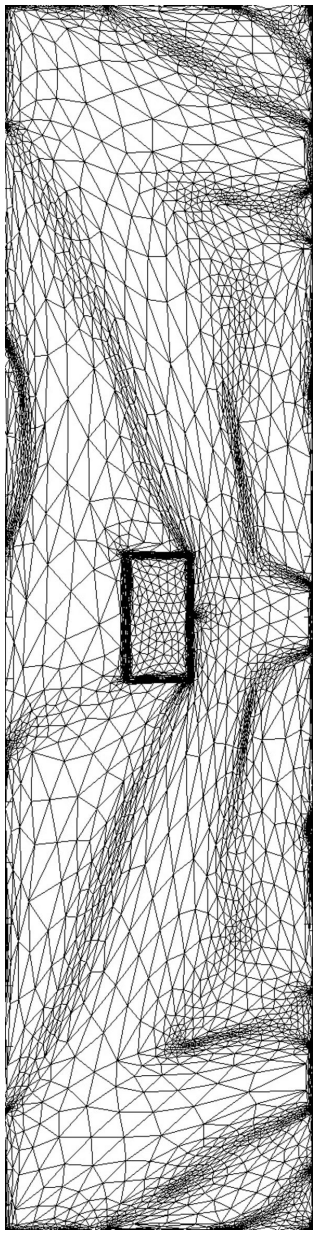
Table 2 summarizes the obtained losses over both the training and the validation dataset. Based on the shown losses, the model trained with reflect padding and obtained at epoch 1650 will be used to predict the temperature field for different setups as detailed in section 7. The slightly larger training loss obtained for the same and reflect padding, compared to that of validation, can be accounted for the stochastic nature of training along with the fact that the training loss is computed half an epoch earlier than the validation loss.

| Padding | Same | Symmetric | Reflect |
|---|---|---|---|
| Training | 3.02e-6 | 2.87e-6 | 2.69e-6 |
| Validation | 2.78e-6 | 2.88e-6 | 2.64e-6 |
| Epoch | 1553 | 1427 | 1651 |

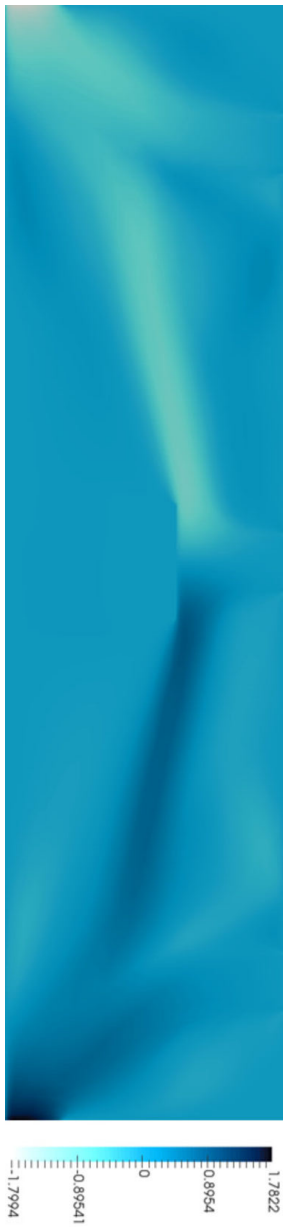
Table 2: Summary of obtained losses for different padding schemes over both the training and the validation dataset



PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

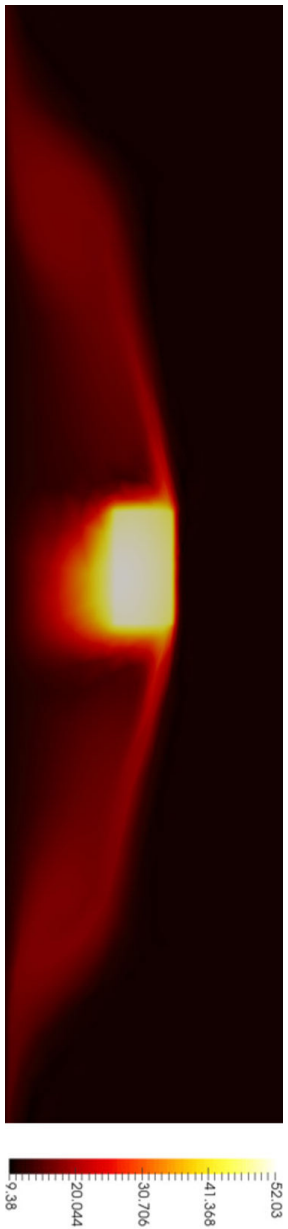


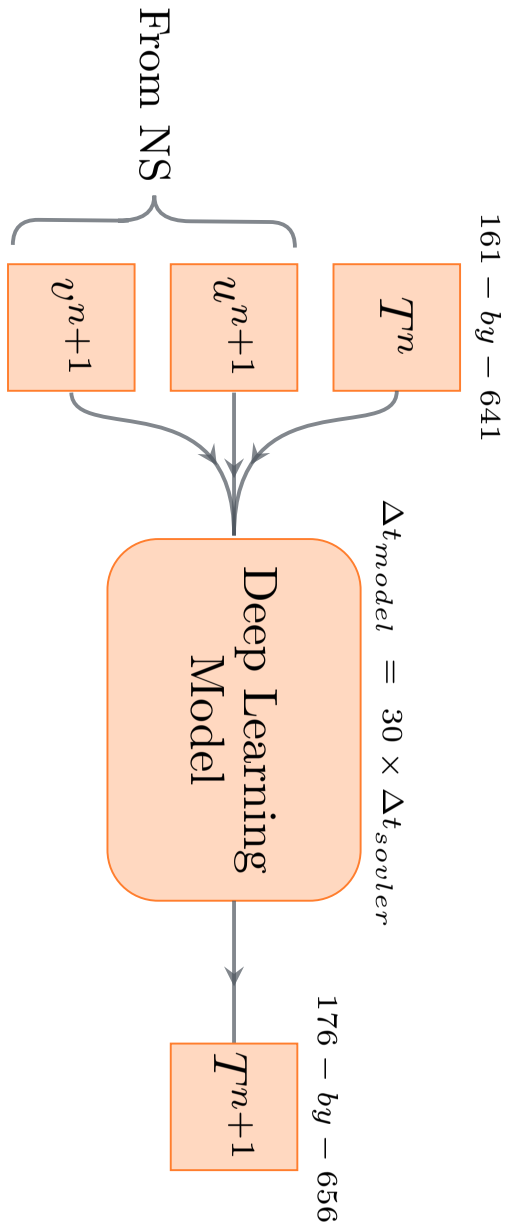
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

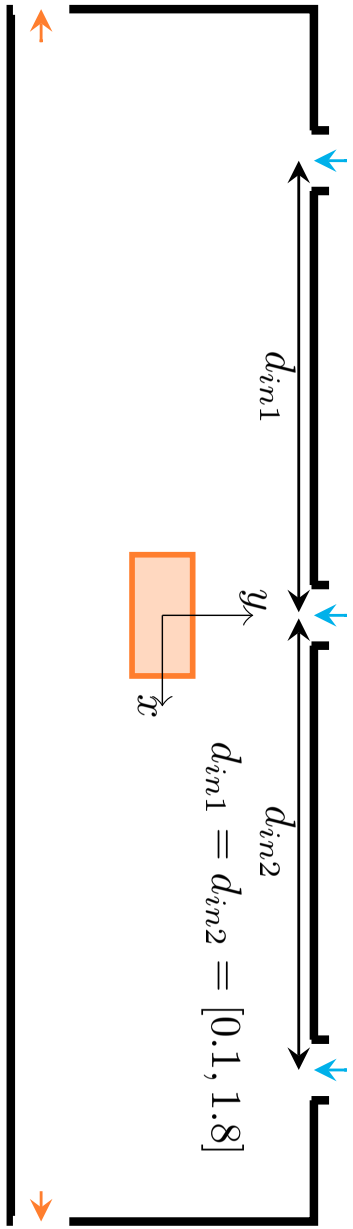


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

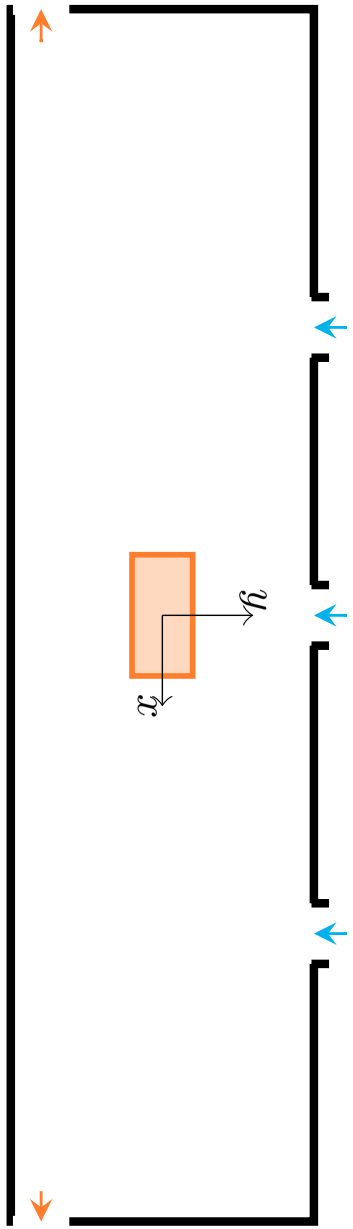
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

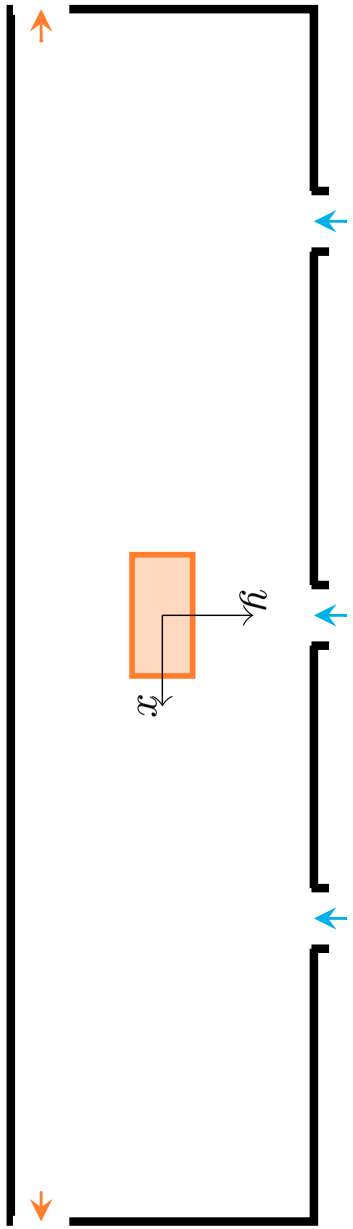


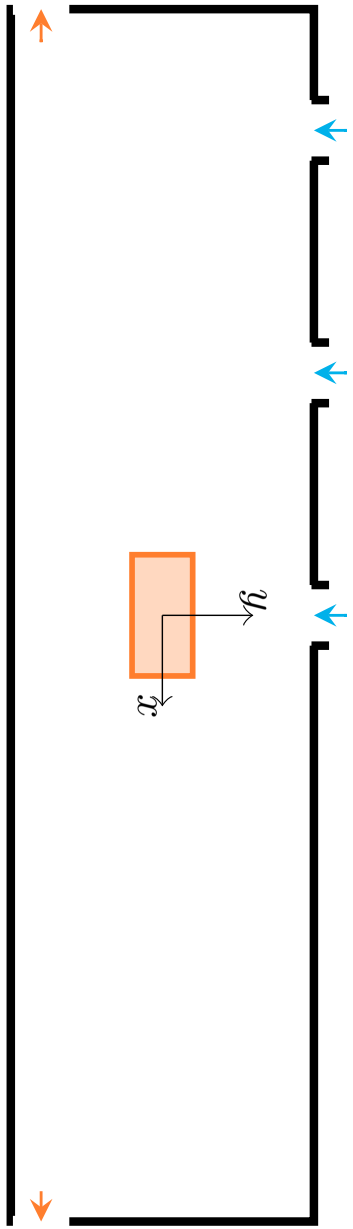


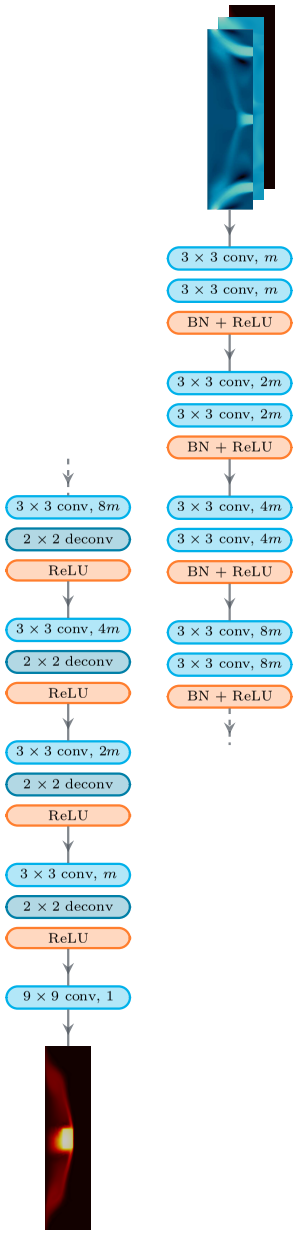


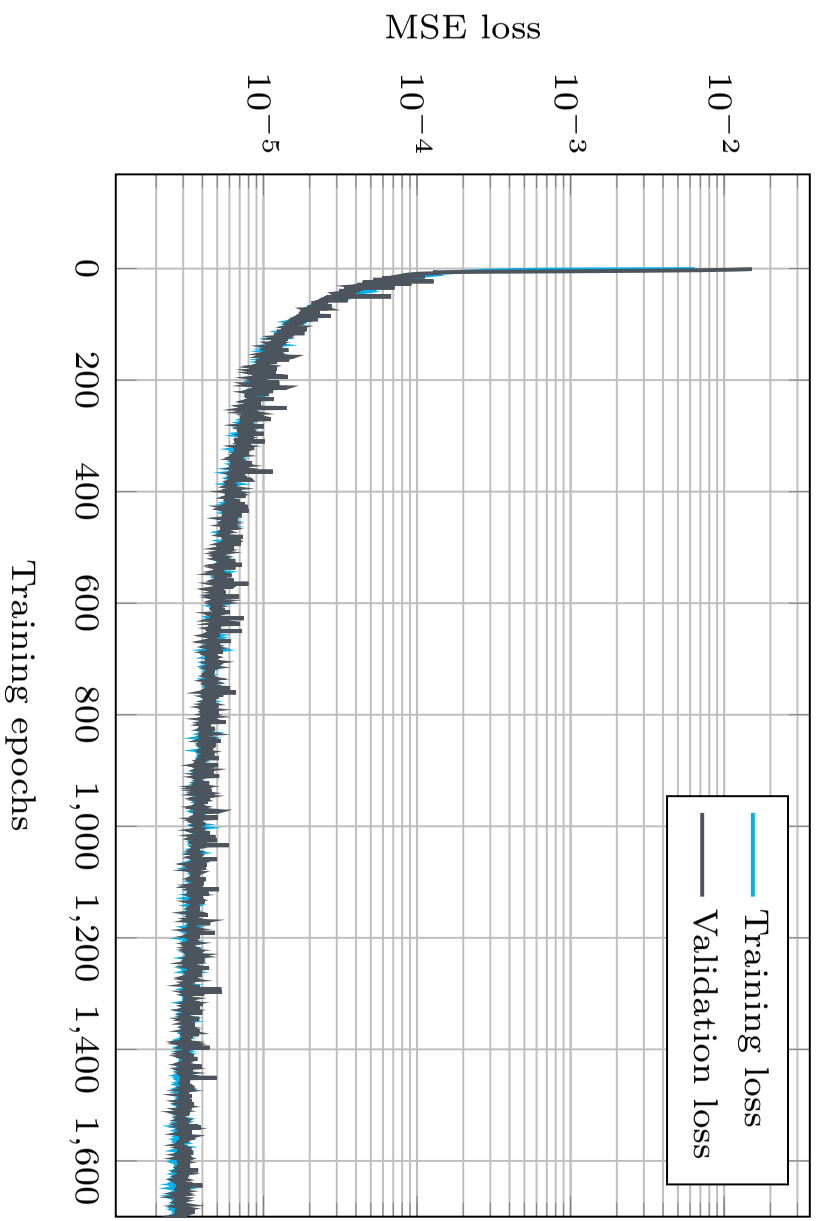
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

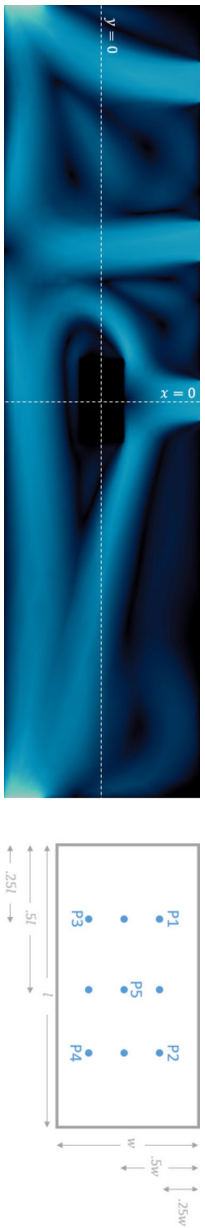


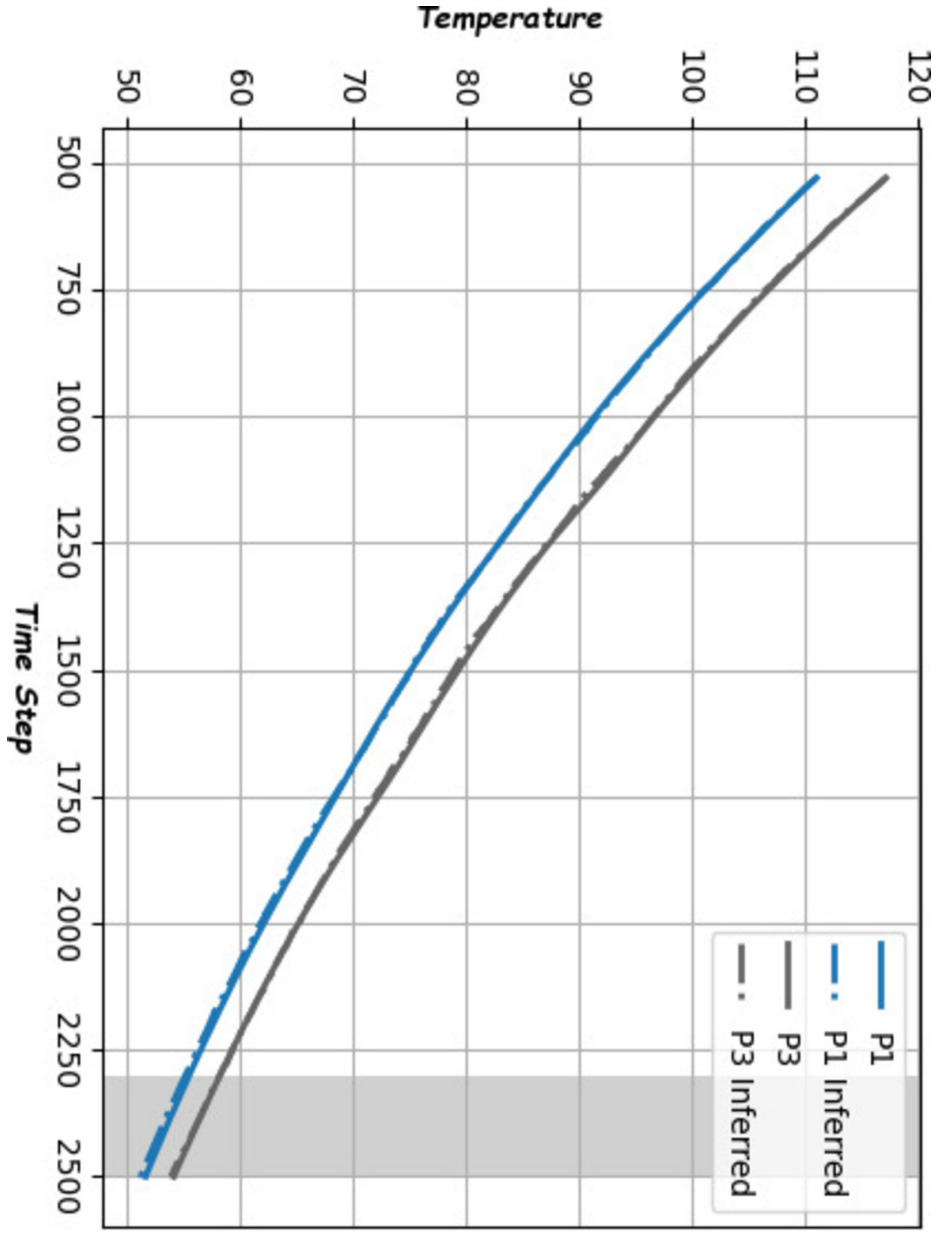


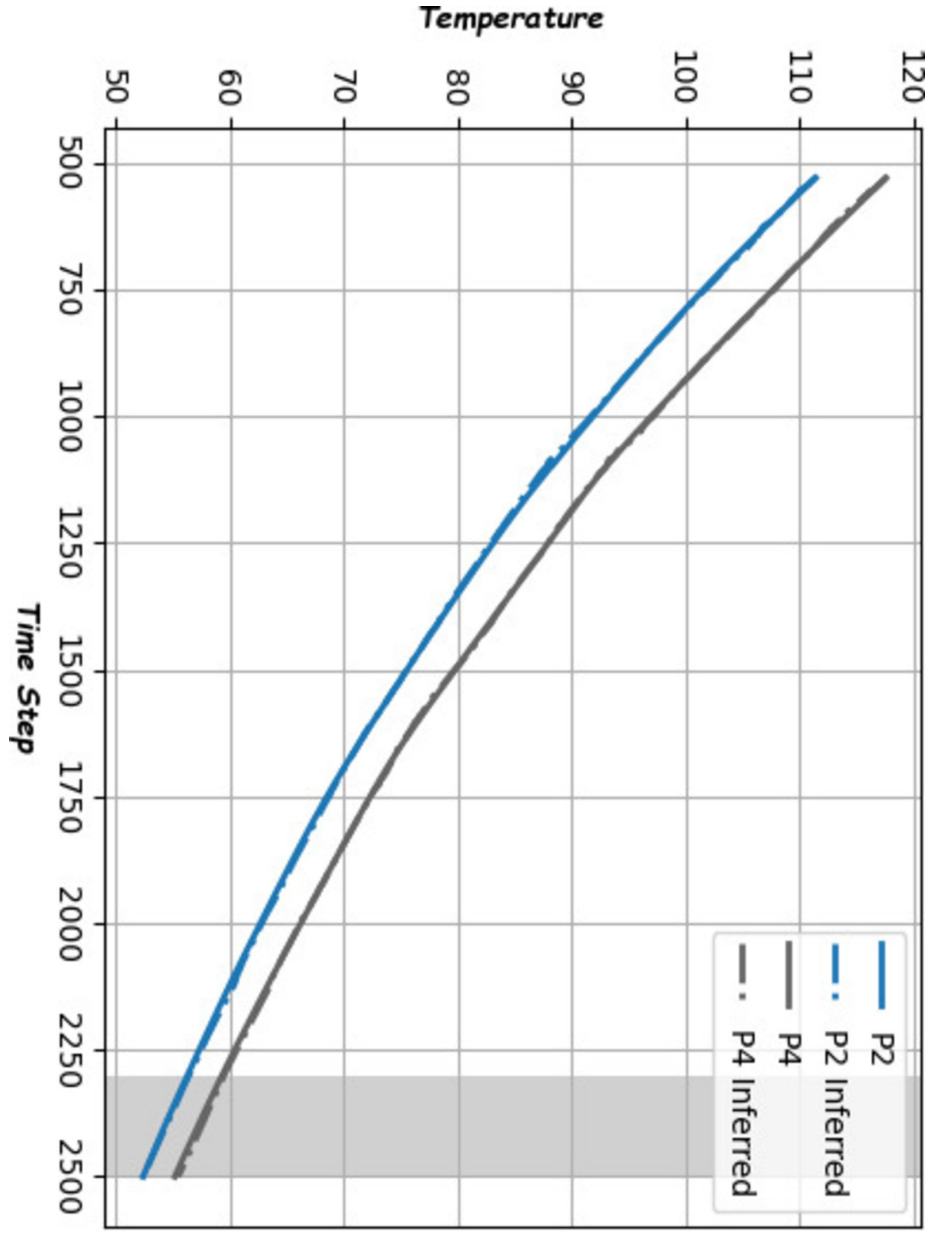




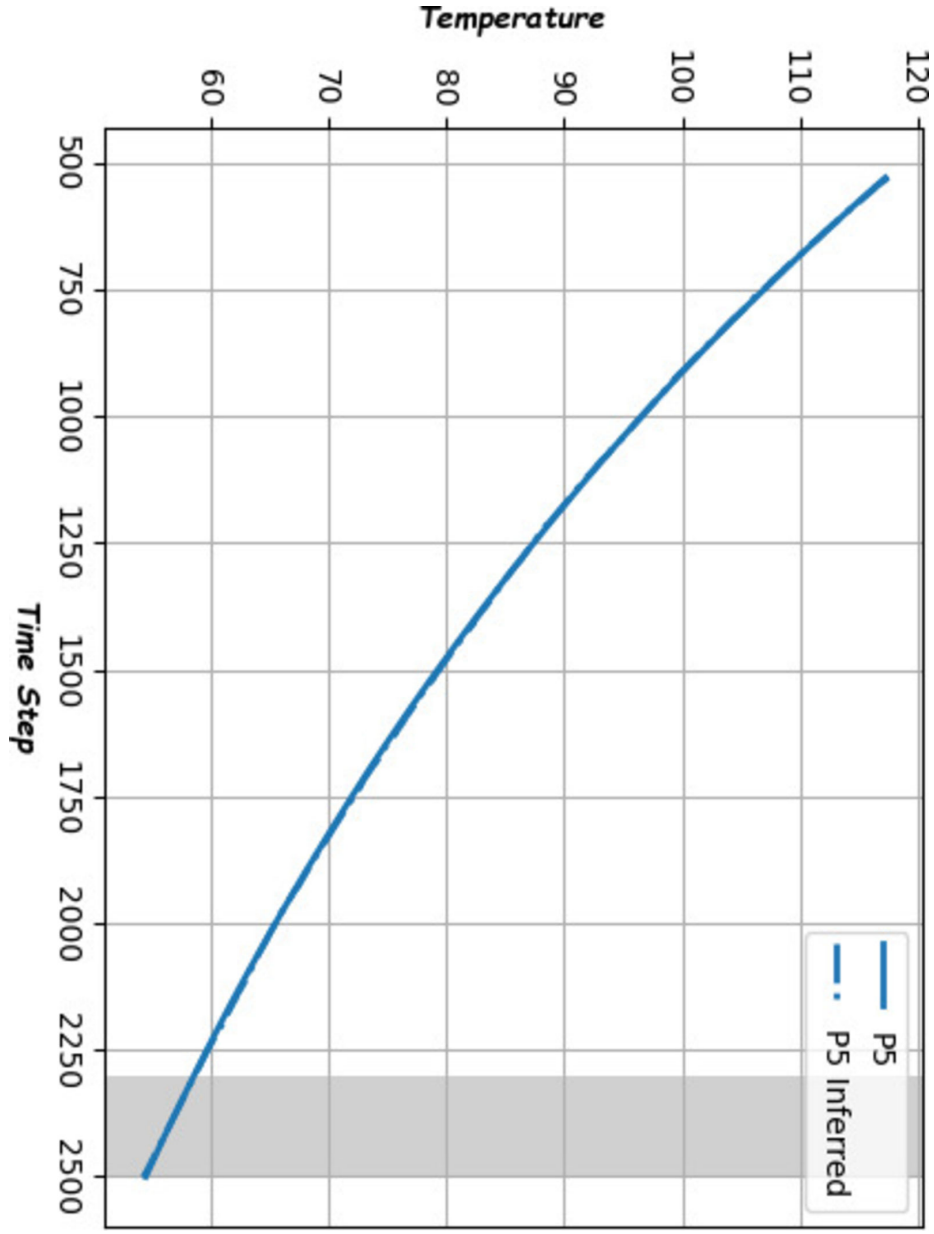


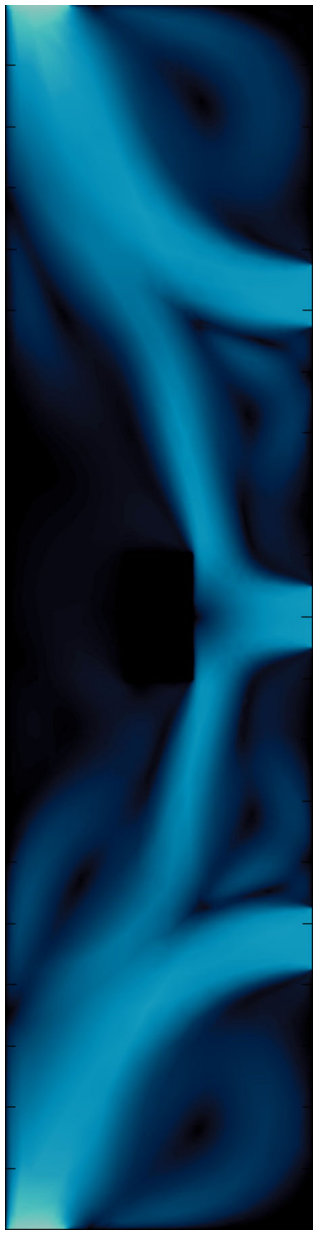




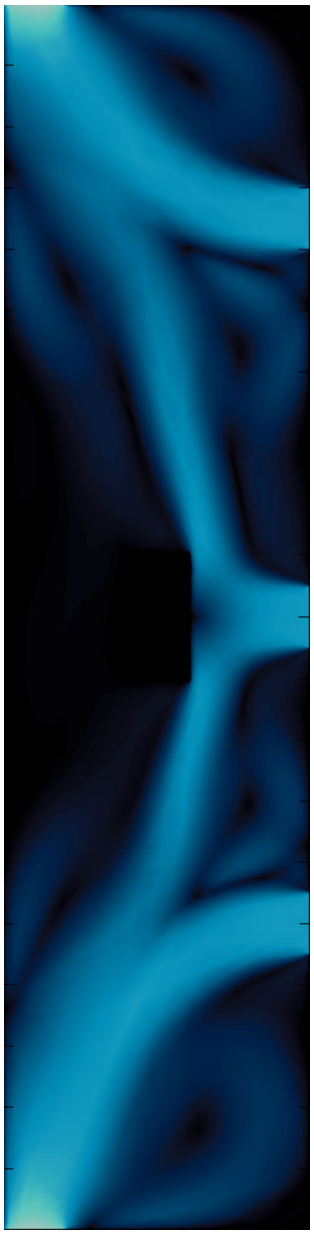


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

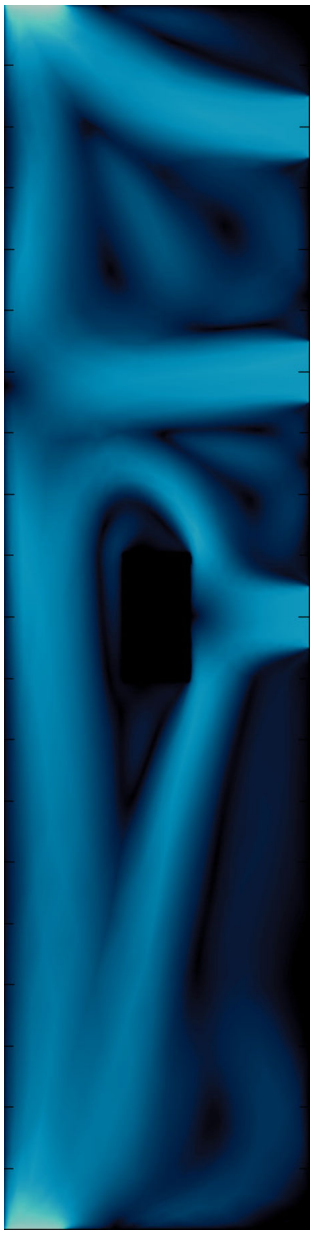


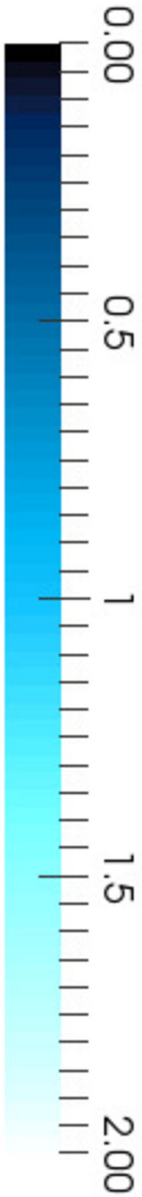


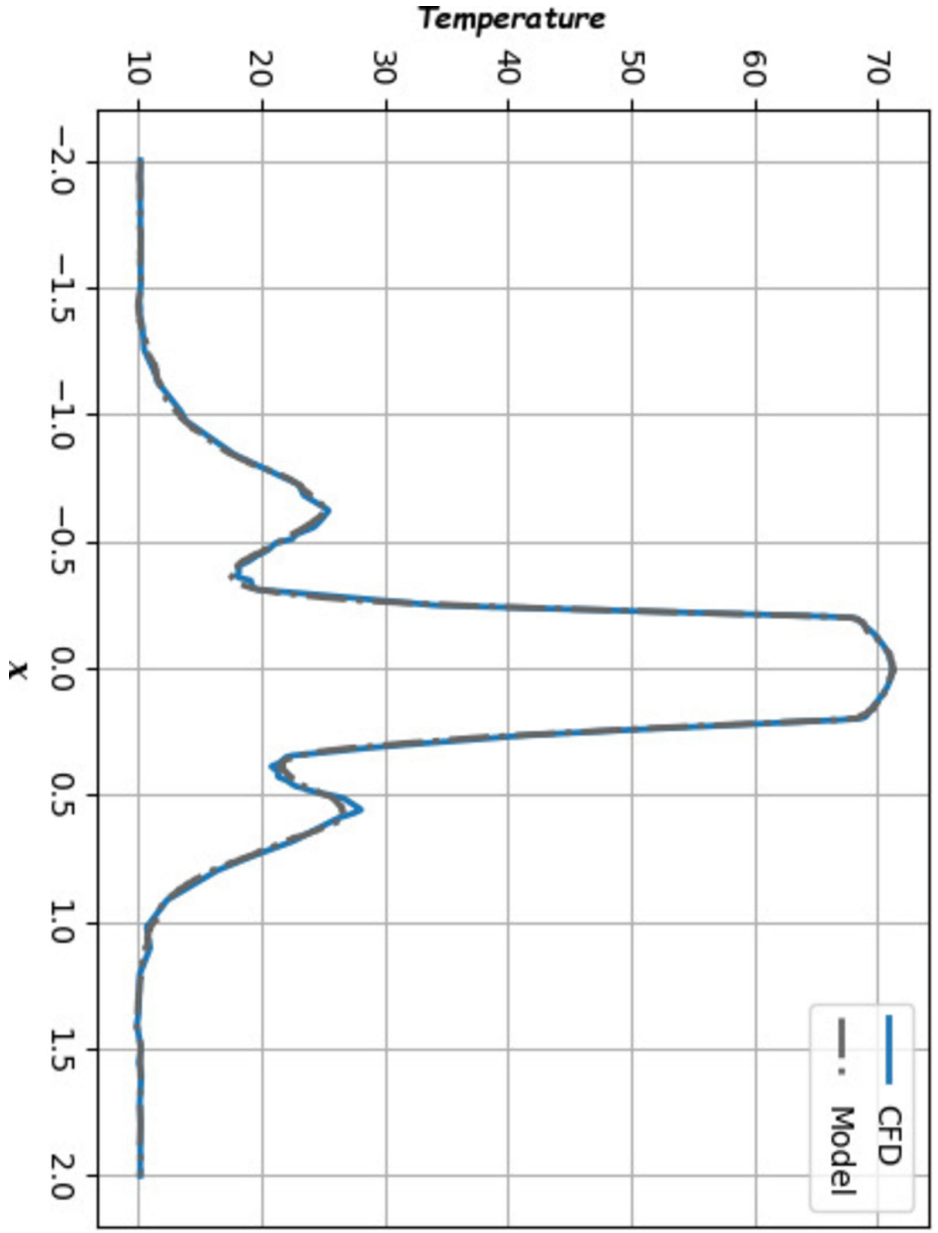
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723



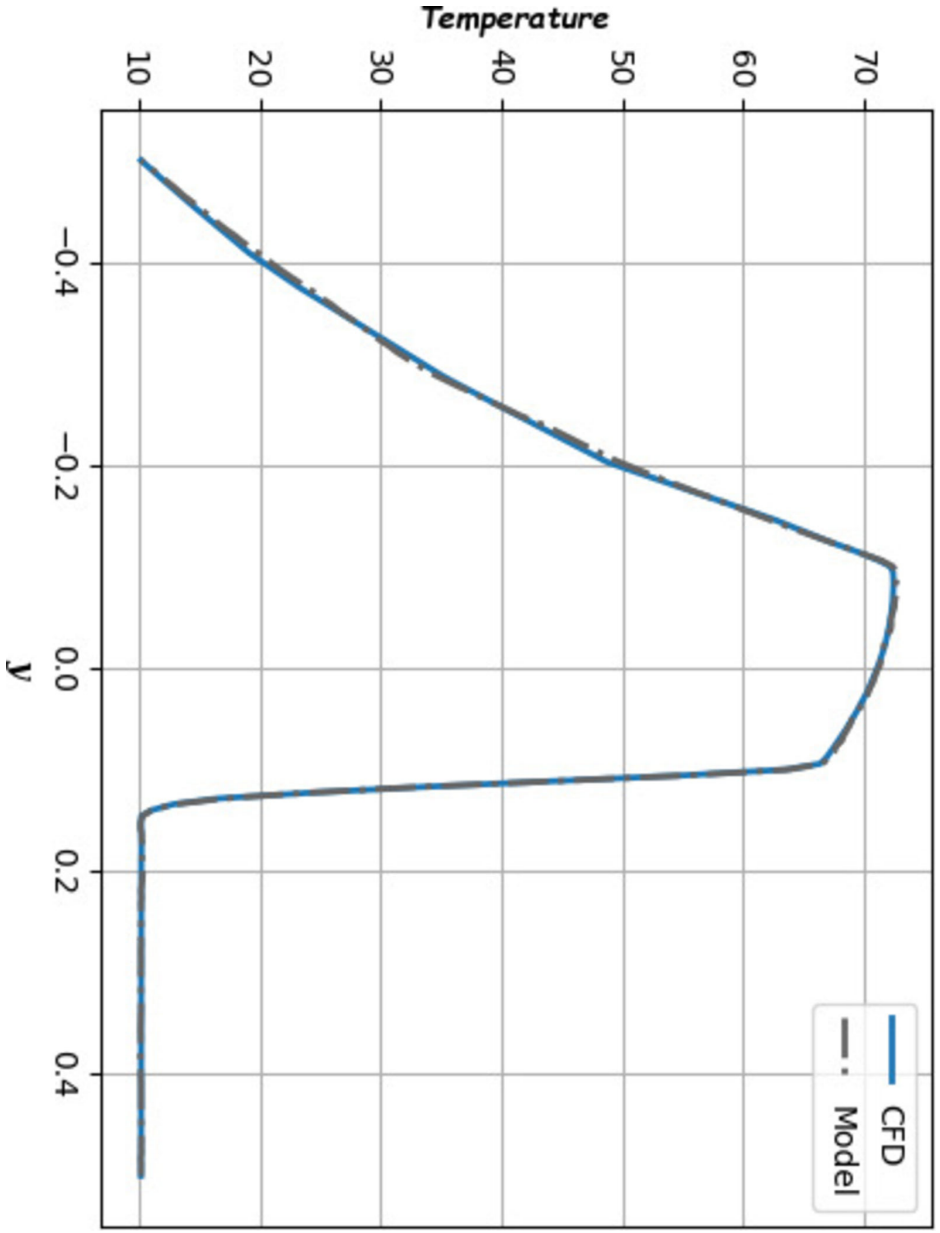
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

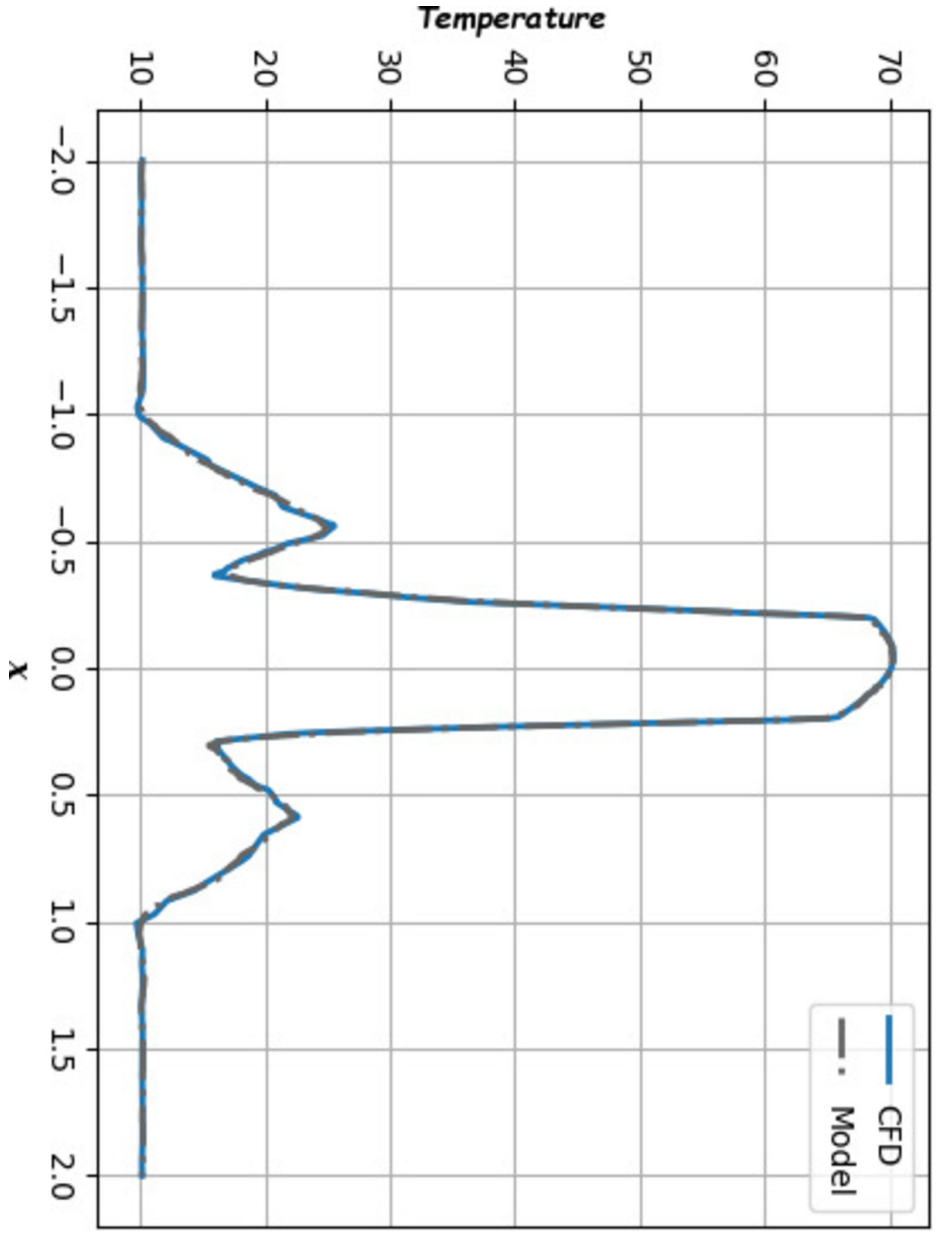




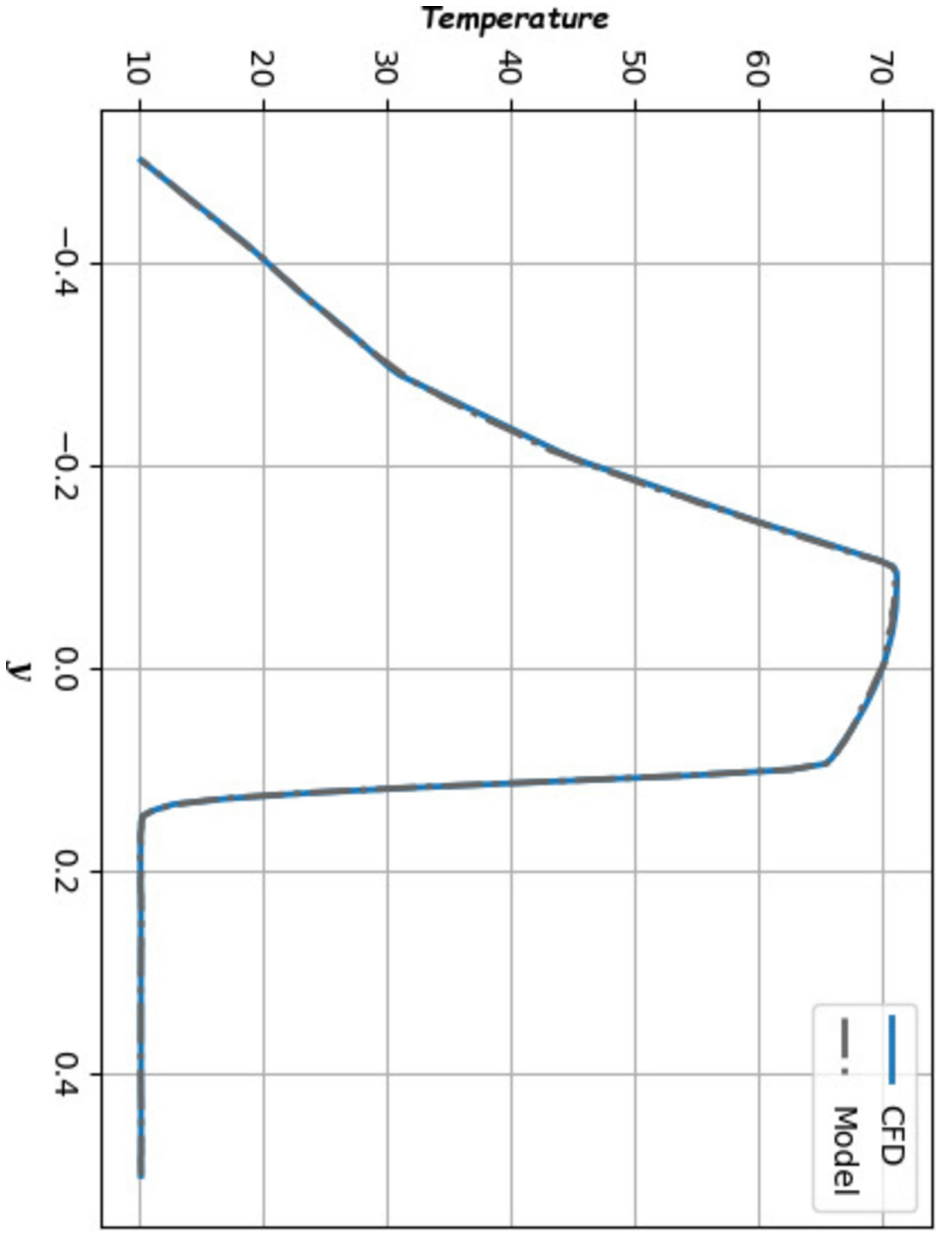


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

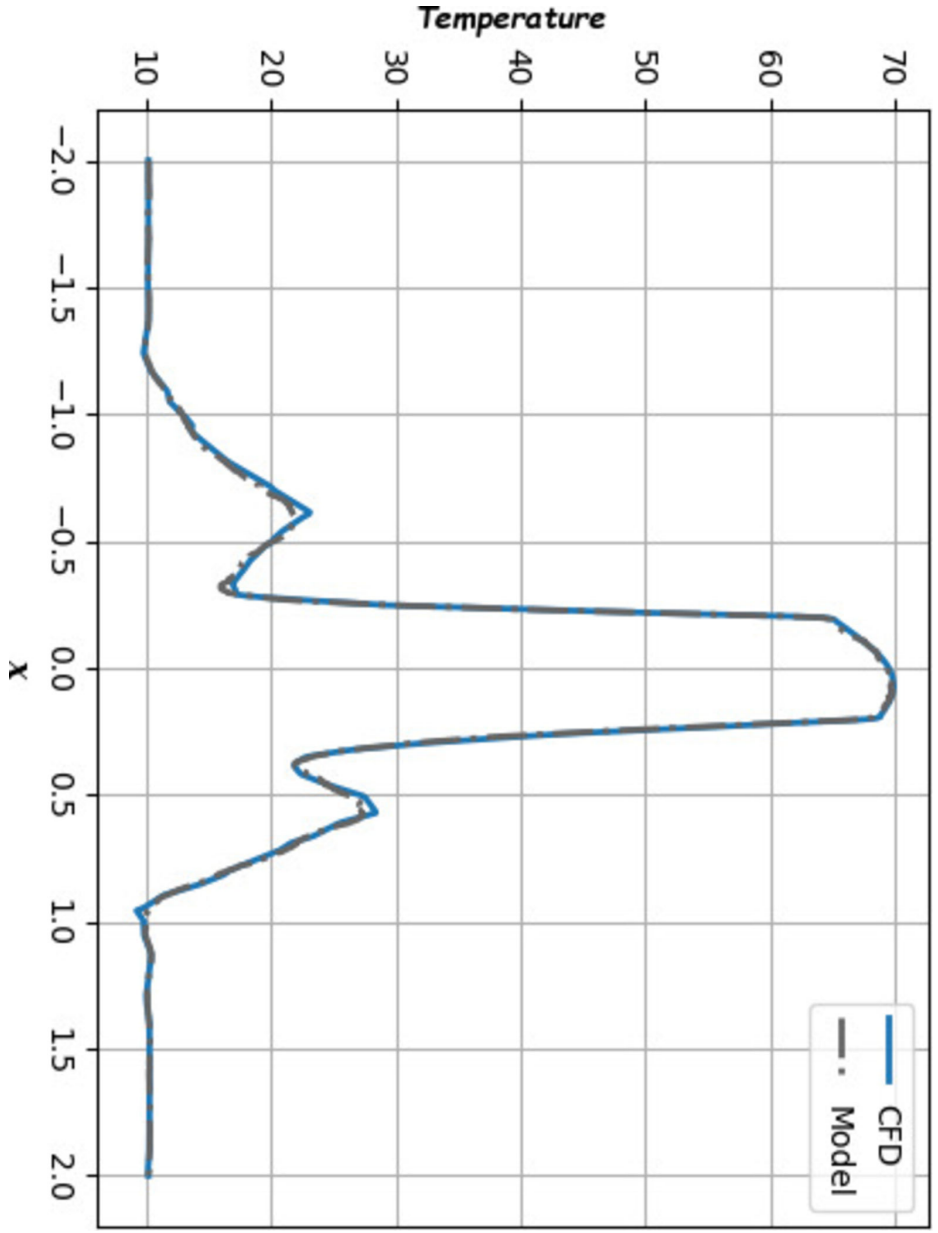




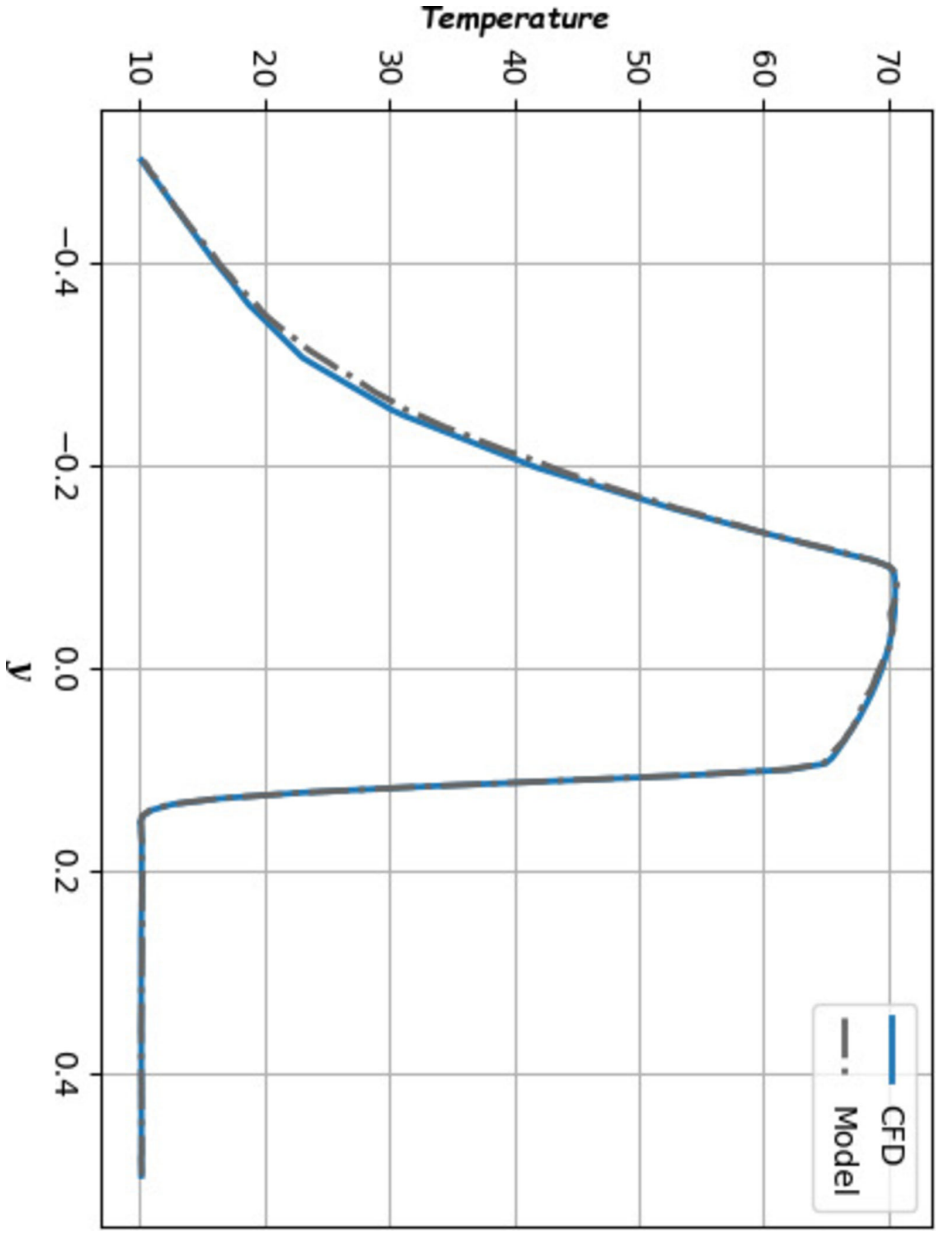
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

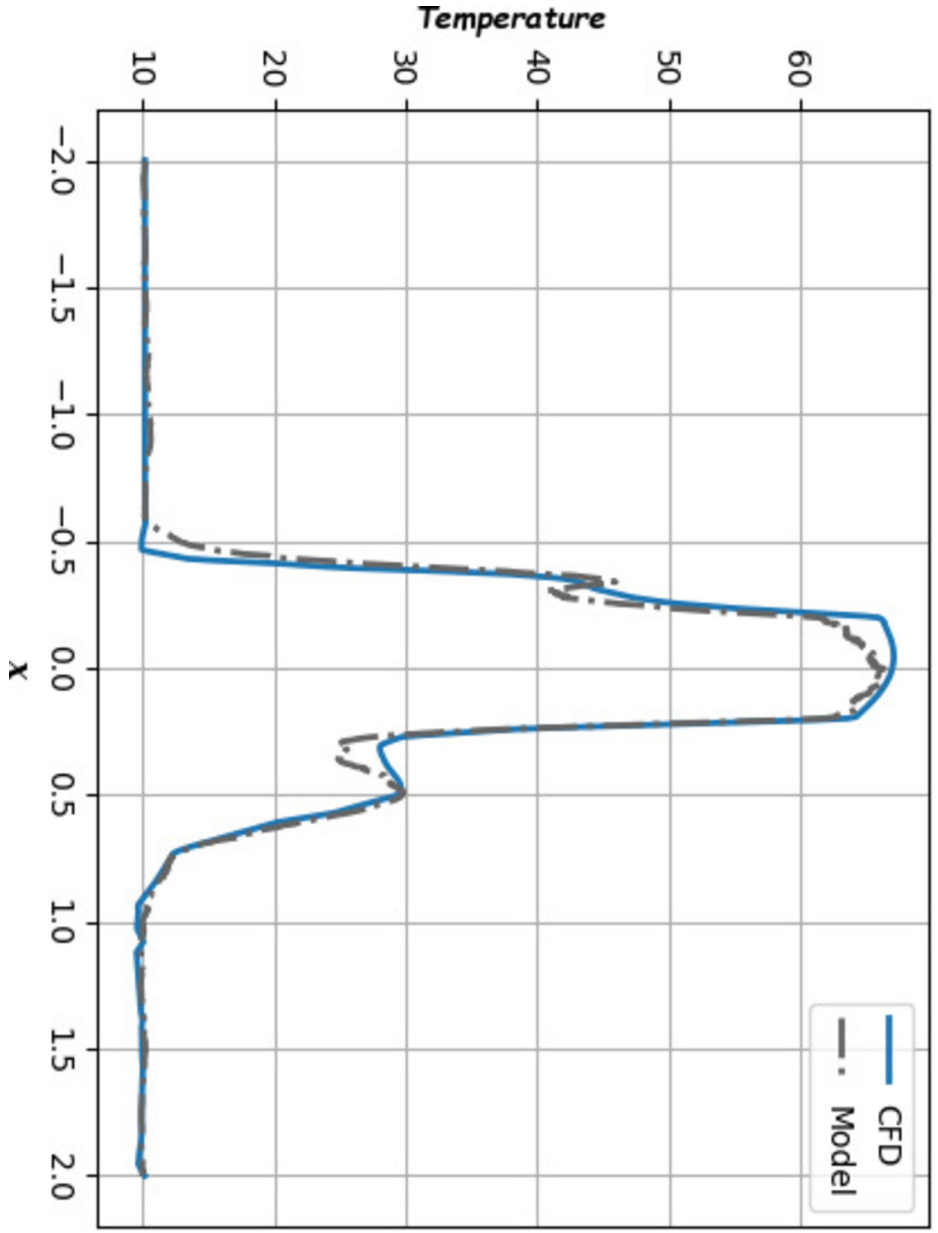


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

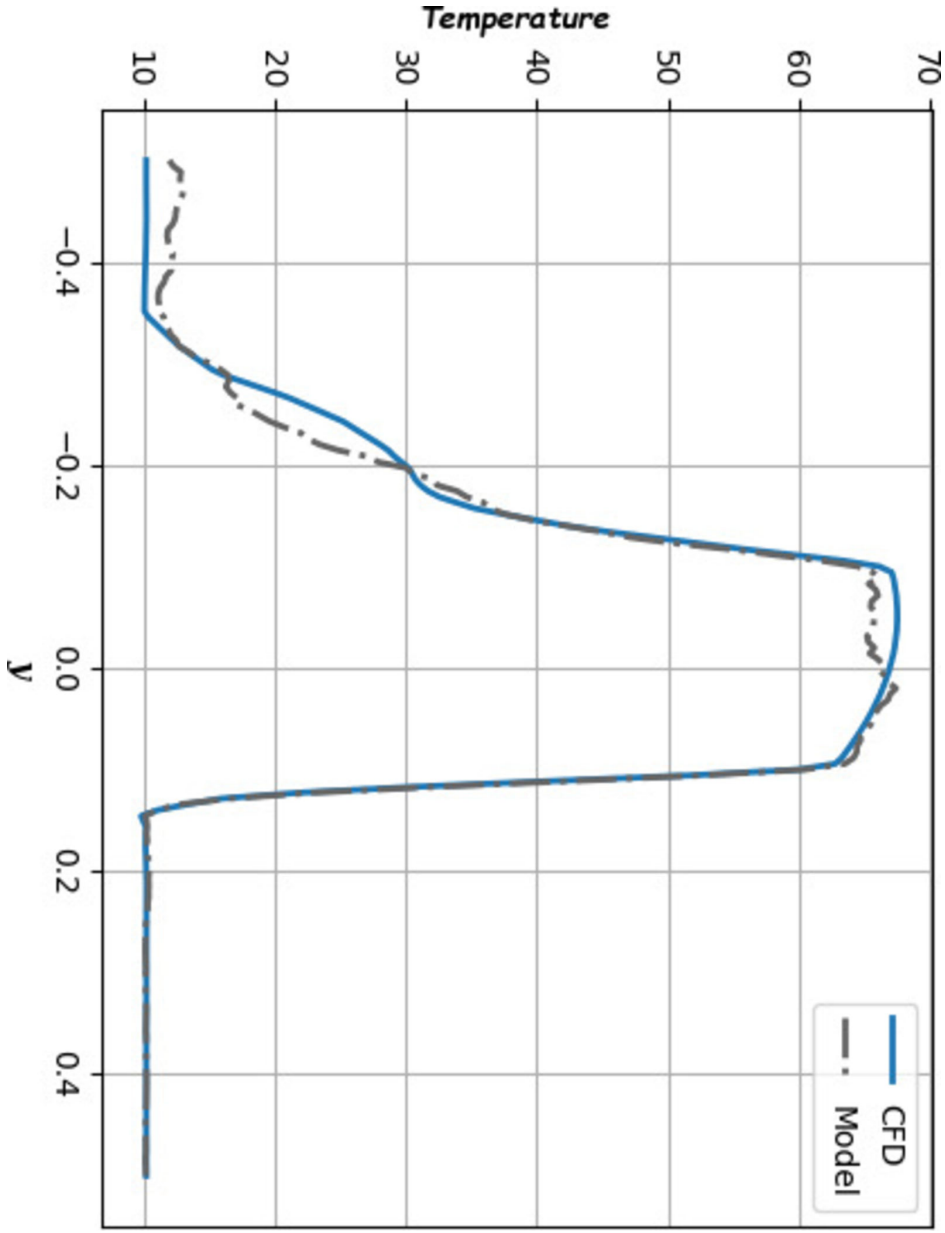


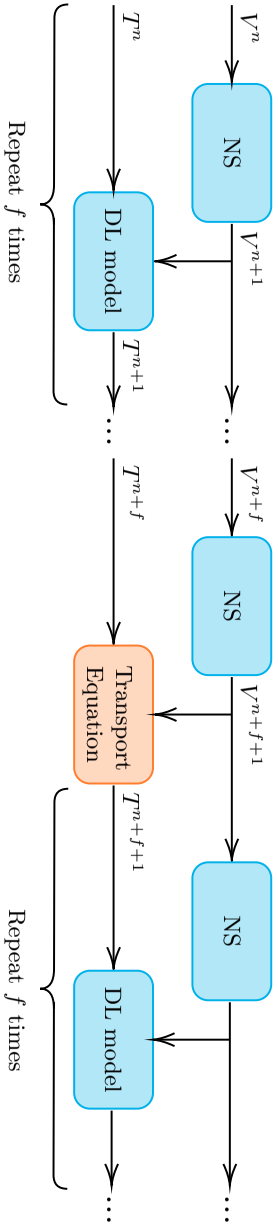
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

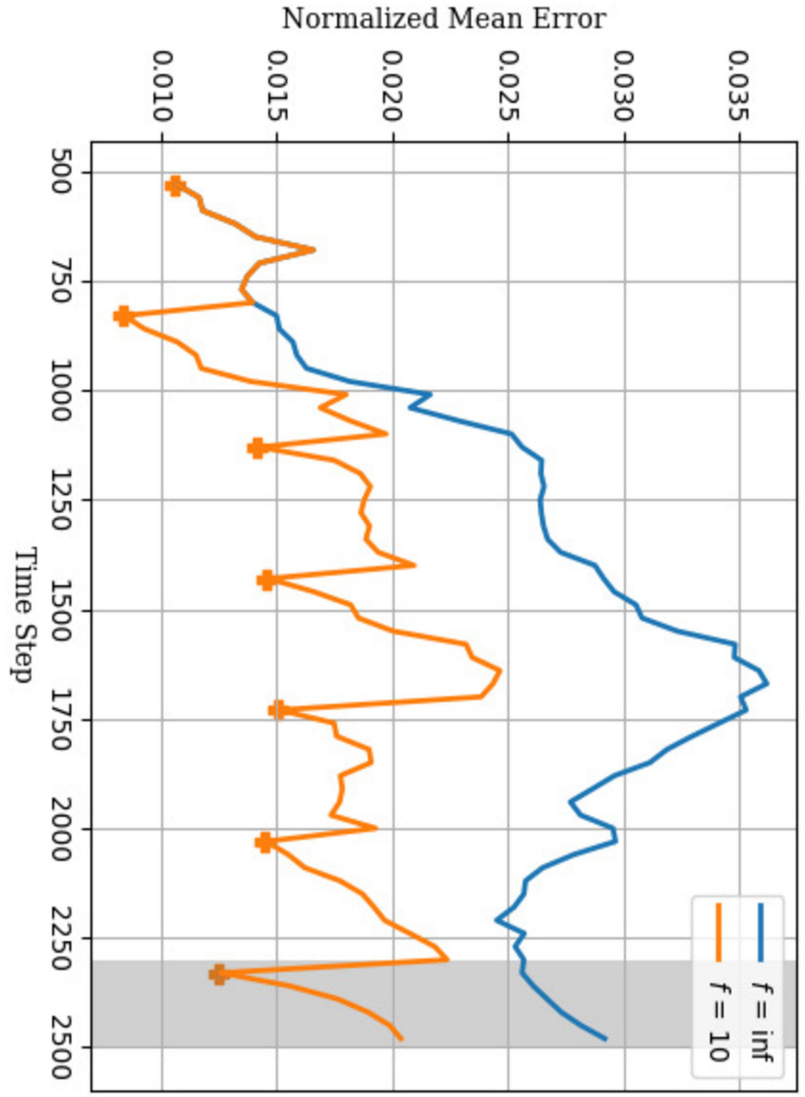


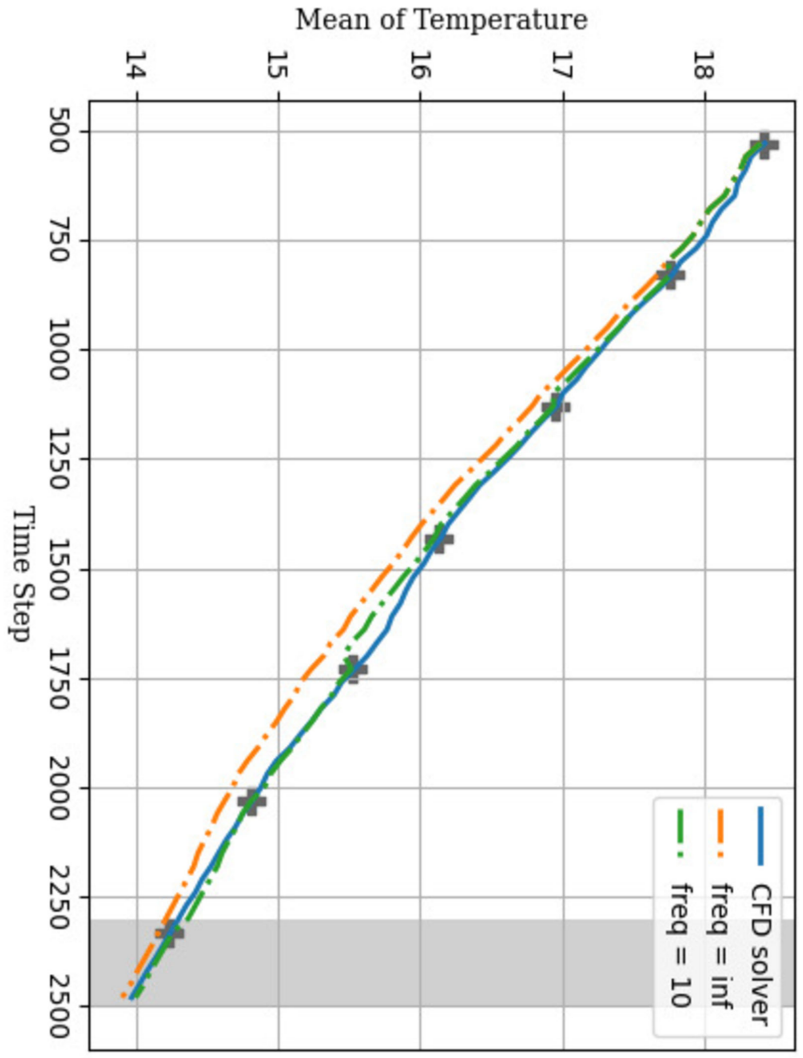


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

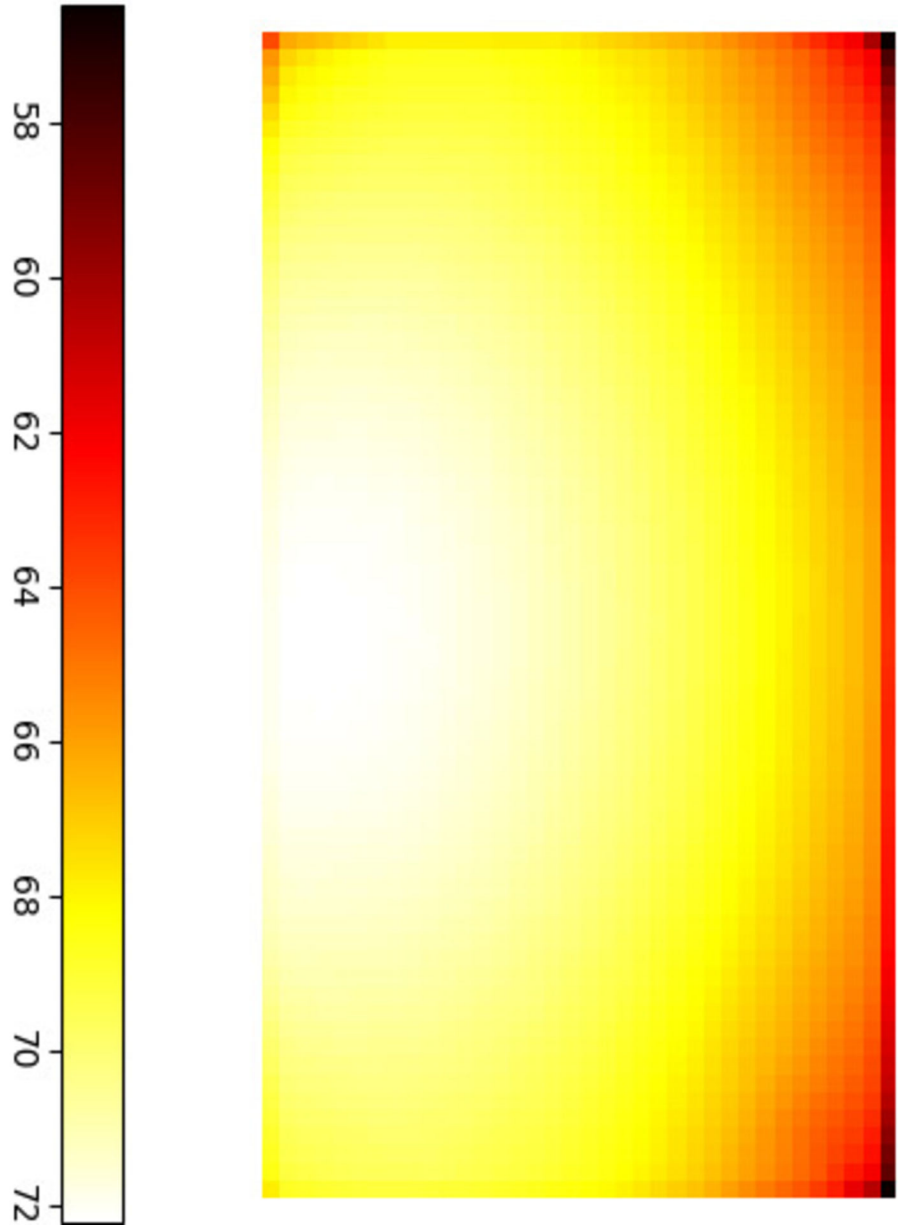


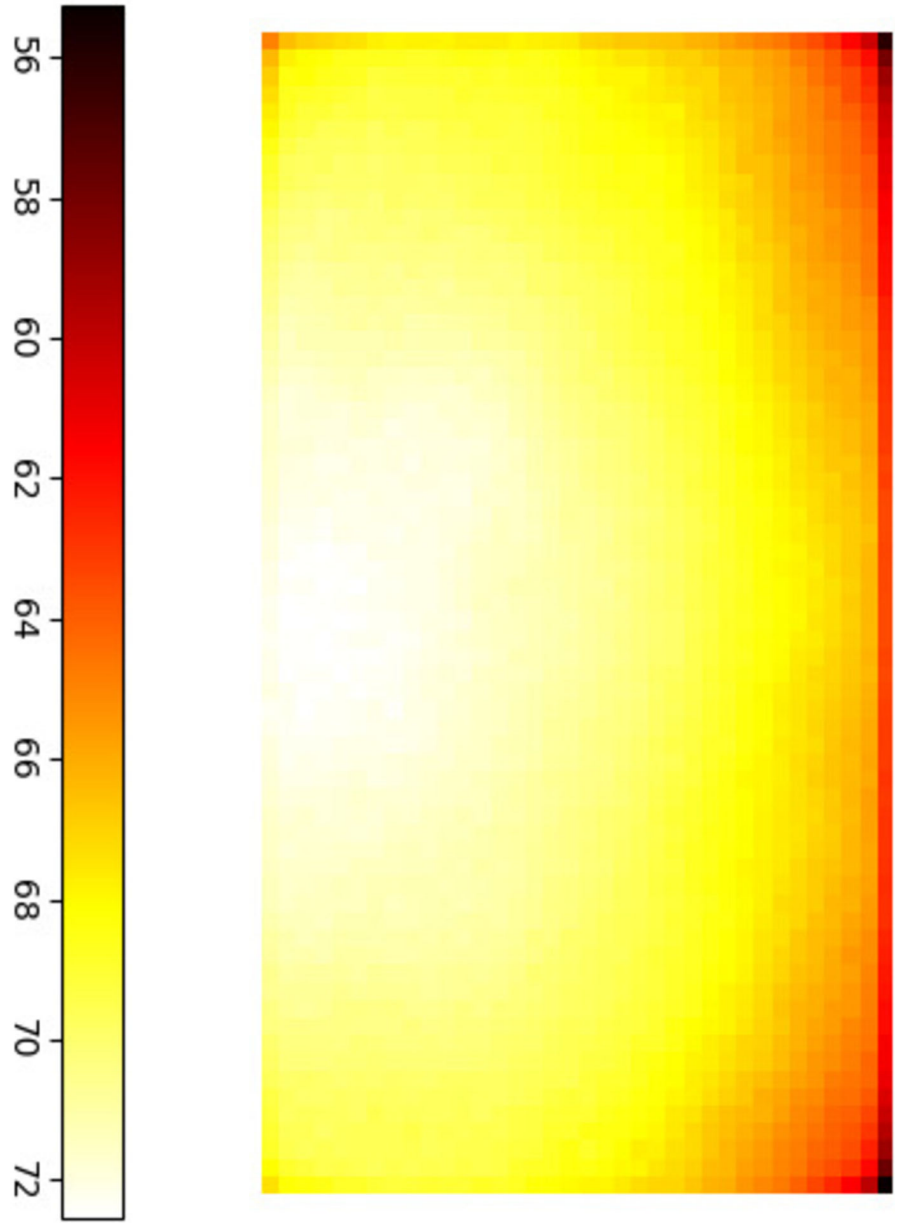


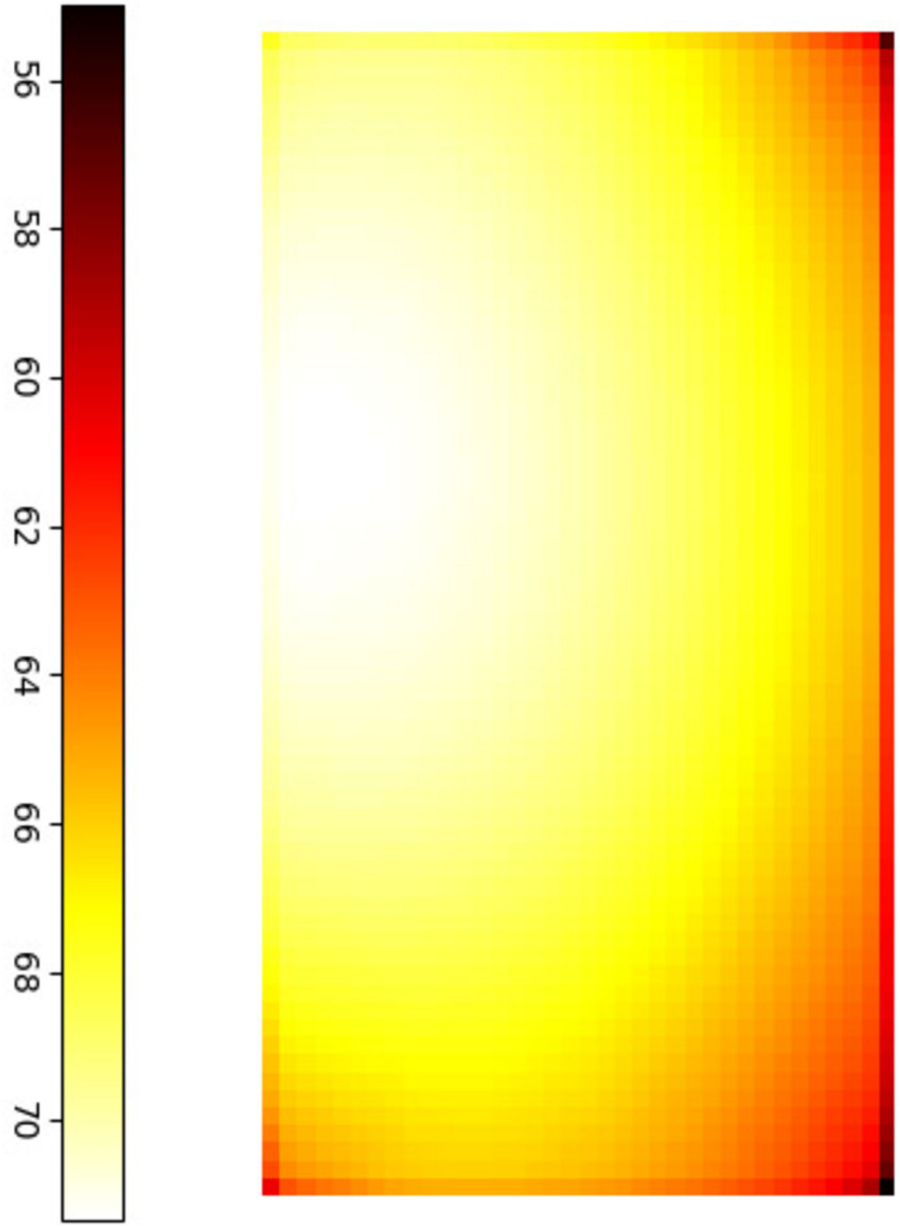


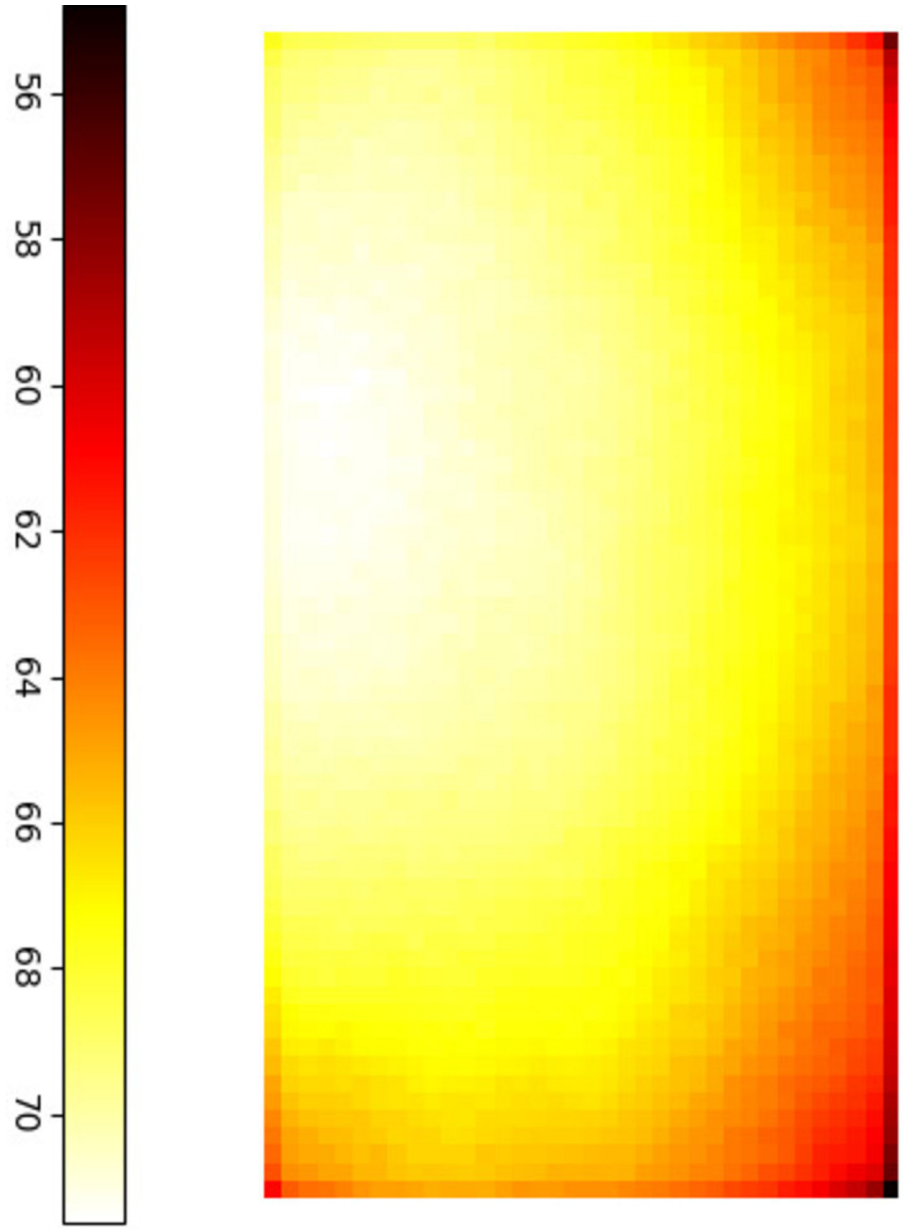


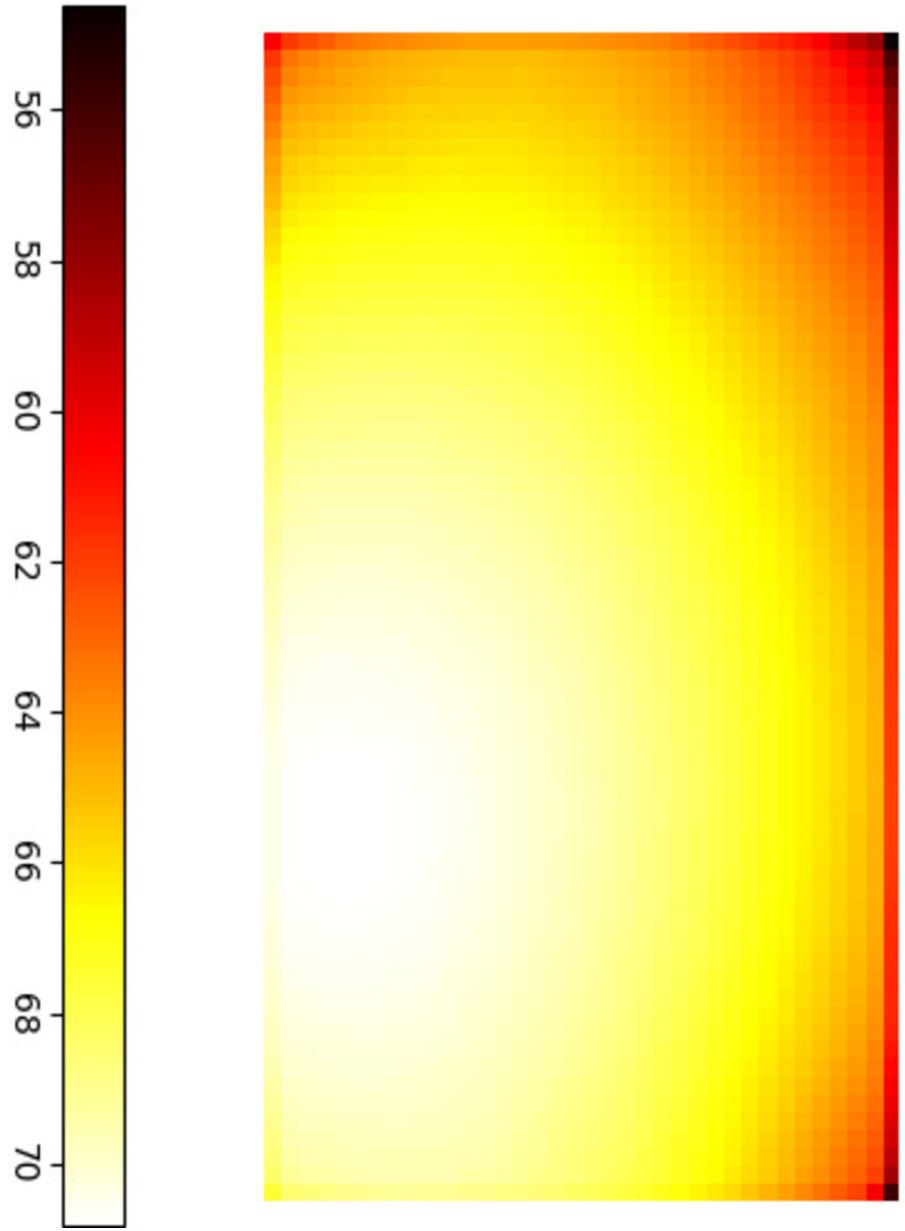
PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

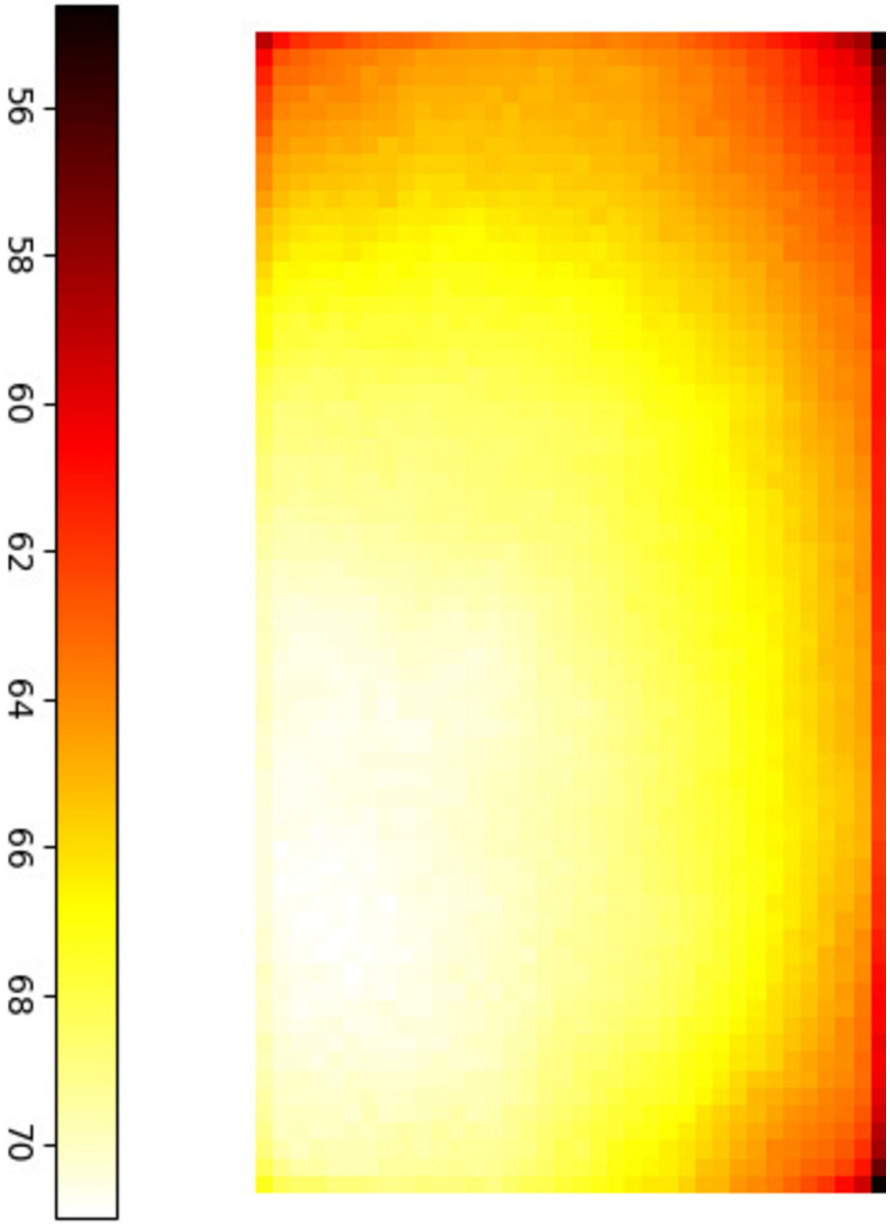




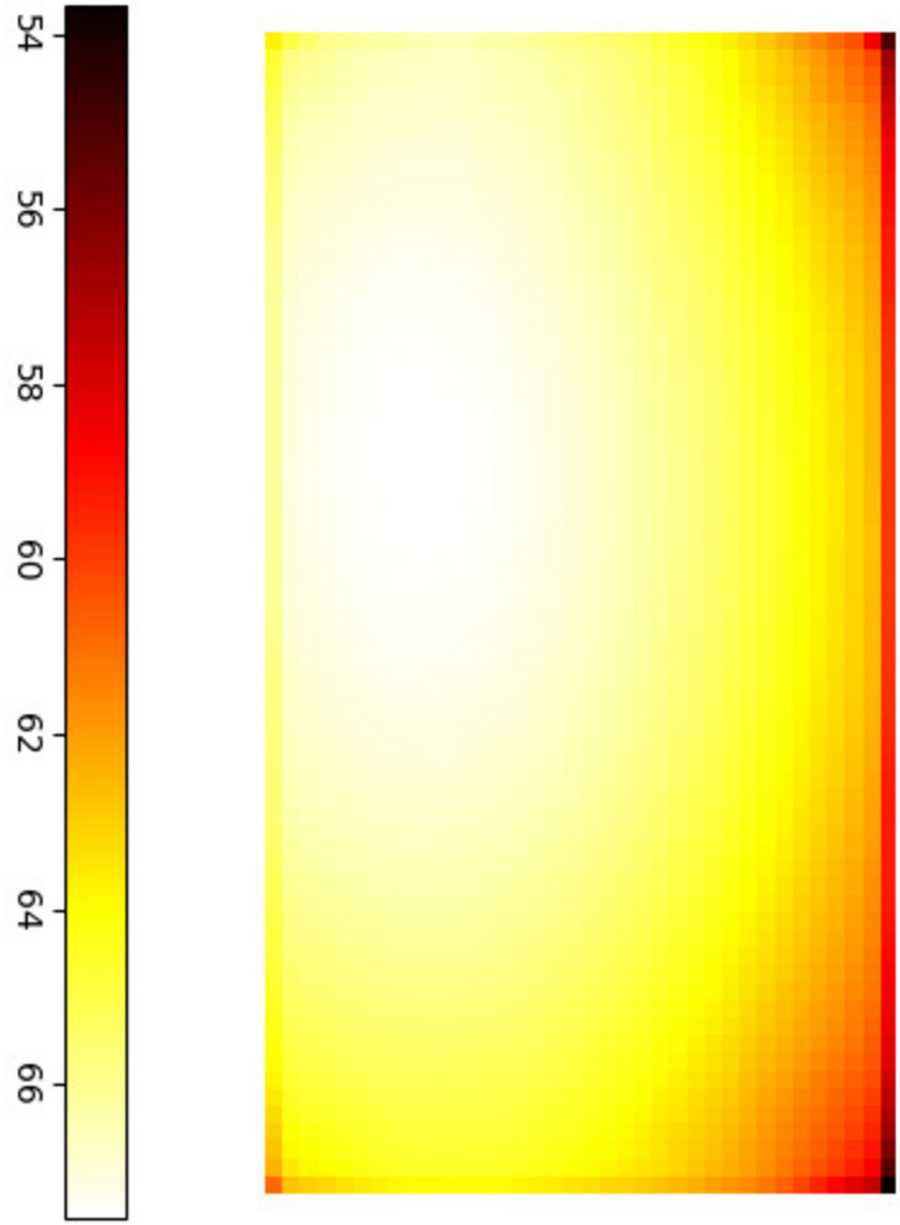


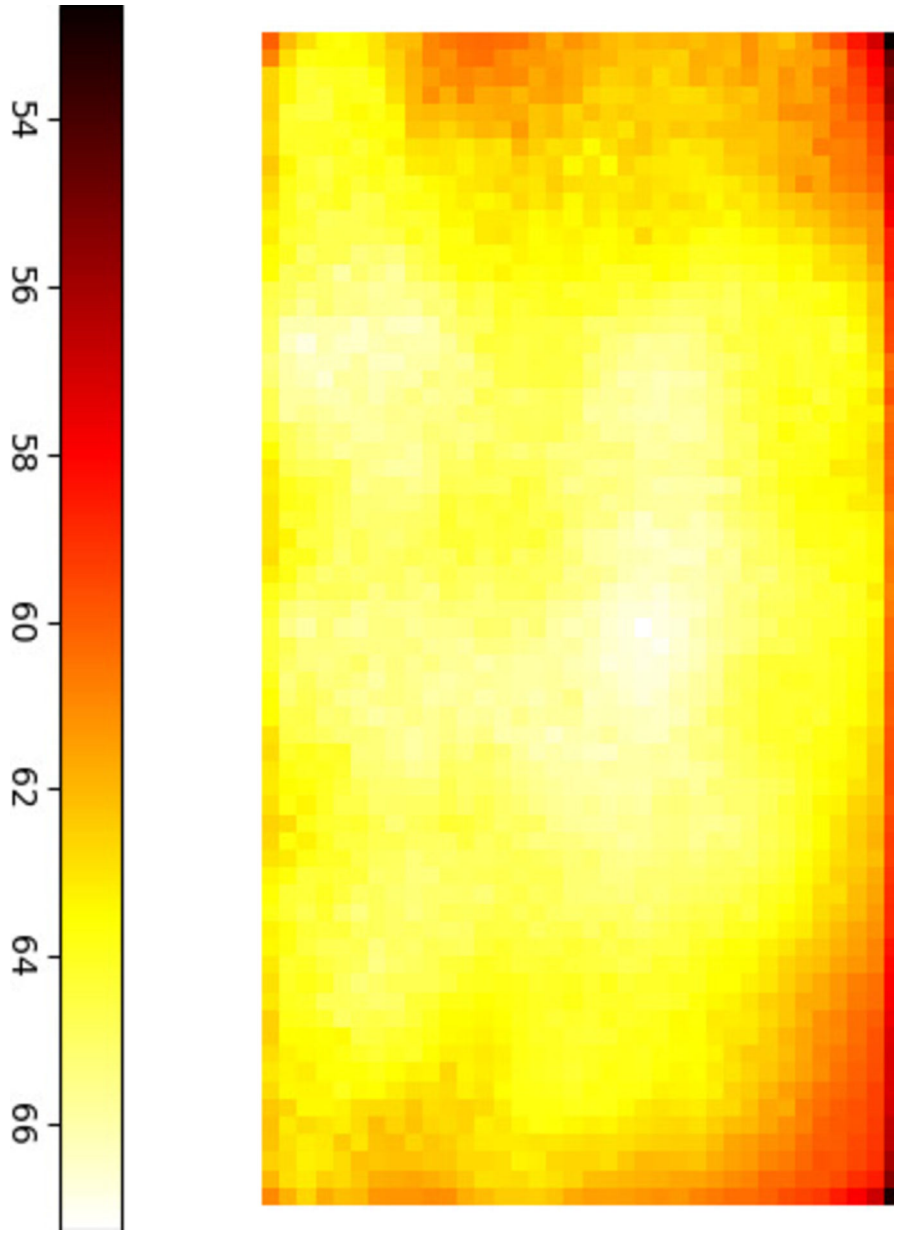




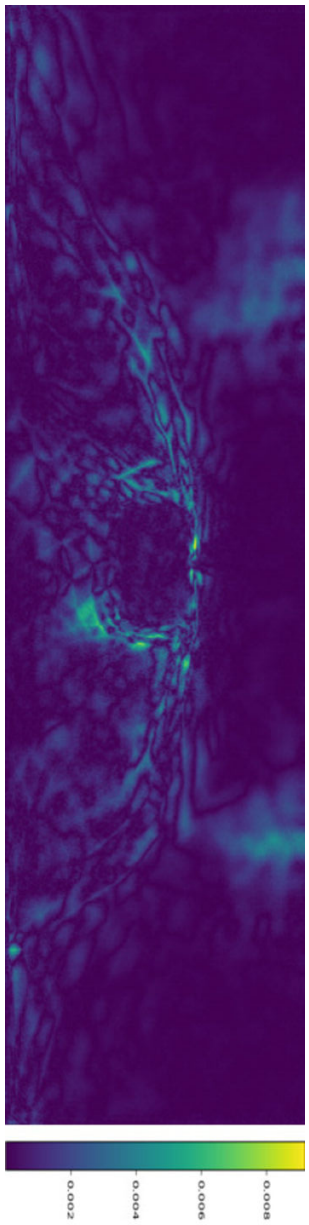


PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723

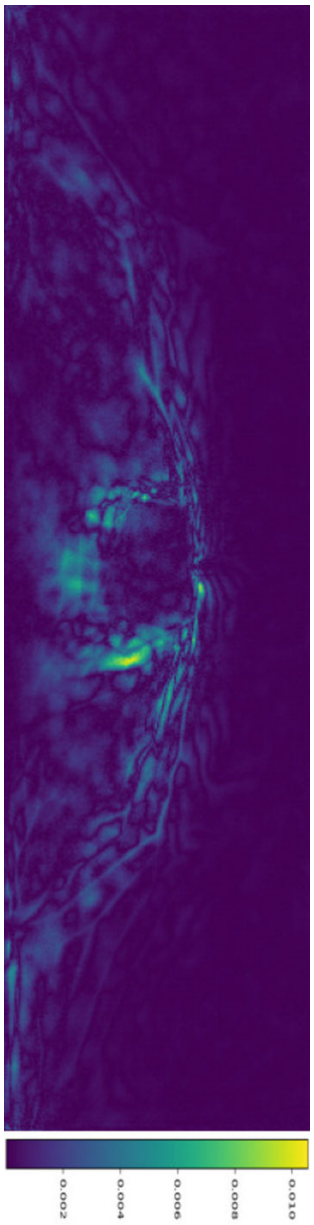




PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723



PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723



PLEASE CITE THIS ARTICLE AS DOI: 10.1063/5.0077723