Prediction of Micromixing Scalar Concentration Field Using a Neural Network

Usama KHALID[1], Aly MORSY[2], Taha JAVED[3], Busayomi THOMPSON-AJAYI[4], Elin VESPER[5], and Daniele MARCHISIO[6]

[1,2,3,4]Politecnico di Torino, Corso Duca degli Abruzzi, 24 | 10129 Torino, Italy

[5]Bayer Aktiengesellschaft, Process Technologies, Leverkusen, Germany

[6]DISAT, Politecnico di Torino, Corso Duca degli Abruzzi, 24 | 10129 Torino, Italy

Abstract

A physics-informed convolutional neural network (CNN) is developed as a surrogate model to simulate turbulent micromixing based on high-fidelity CFD data. The network is trained on scalar transport fields generated by OpenFOAM direct numerical simulations, capturing the diffusion of a passive scalar in homogeneous isotropic turbulence. Emphasis is placed on the Batchelor-scale physics of micromixing which is the smallest scalar mixing scale and is crucial in high-Schmidt-number liquid flows. The CNN architecture (a 3D U-Net with $\sim 1.4 \times 10^6$ parameters) is designed to embed physical knowledge: it enforces non-negativity of concentration and smoothness of scalar gradients to reflect diffusive processes. This approach addresses key challenges in resolving turbulent scalar transport, including extreme resolution requirements and memory bottlenecks, by learning an efficient model that reproduces fine-scale mixing without needing an ultra-fine mesh. The study is motivated by sustainable pharmaceutical manufacturing, where improved micromixing can enhance reaction yields and reduce unwanted by-products. The trained model demonstrates good agreement with CFD benchmarks, capturing the overall mixing patterns and scalar dissipation trends across a range of Schmidt numbers. By coupling first-principles CFD data with modern deep learning, the framework achieves an order-of-magnitude speedup in predicting scalar concentration fields while maintaining physical fidelity. This work showcases how integrating convolutional neural networks with CFD can accelerate process simulations and inform the design of greener, more efficient mixing strategies in industrial reactors.

Contents

1 Introduction **3**

2 Methodology **6**

2.1 Computational Fluid Dynamics Setup 6

2.2 CFD Simulation Results 11

2.3 Neural Network Development 12

3 Results and Discussion **22**

3.1 Fully Connected Network Results 22

3.2 Convolutional Neural Network Results 22

3.3 Computational Performance 25

4 Conclusion **26**

5 Future Work **27**

5.1 CFD Extensions 27

5.2 Neural Network Improvements 27

A Summary of Solver Modifications (scalarTransportFoam) **29**

A.1 Purpose of the Modification 29

A.2 Key Code-Level Changes 29

B MATLAB Script Modifications for HDF5 to OpenFOAM Conversion **30**

B.1 Purpose of the Modification 30

B.2 Key Code-Level Changes 31

C 3D CNN Implementation Code **31**

1. Introduction

Micromixing, the molecular-scale mixing of reactive species in turbulent flows, plays a pivotal role in chemical and pharmaceutical process engineering. In industrial reactors (e.g. for pharmaceutical intermediate synthesis), incomplete micromixing allows competing side reactions to form undesired by-products, which lowers the yield of the target product [1]. Using catalysts can accelerate the primary reaction, but if mixing is slow, the reaction outcome is dictated by the competition between reaction kinetics and turbulent mixing rates [1]. In such cases, improving the micromixing efficiency directly translates to higher selectivity and less waste [1]. Consequently, understanding and predicting micromixing in turbulence is crucial for designing sustainable processes that maximize desired products while minimizing impurities.

Turbulent Scalar Transport and Micromixing Scales: Turbulent mixing occurs across a hierarchy of length scales. Energy is injected at large scales and cascades to the Kolmogorov scale (η), where viscous forces dissipate turbulence. For scalar mixing, an even smaller scale can govern: the Batchelor scale (λ_B). The Batchelor scale is the diffusion length scale below which molecular diffusion smooths out concentration fluctuations [2]. It is related to the Kolmogorov scale by $\lambda_B \approx \eta/\sqrt{\mathrm{Sc}}$, meaning that in liquids with high Schmidt number ($\mathrm{Sc} \gg 1$), scalar concentration gradients exist on much finer scales than the smallest eddies [3]. Figure 1 illustrates this scale separation: for example, in water, λ_B is on the order of only a few microns when η is on the order of tens of microns. These tiny mixing scales, the micromixing scales, control how quickly reactants truly molecularly mix, and hence often determine the selectivity of fast competitive reactions.

However, resolving the Batchelor scale in simulations is extremely challenging. A direct numerical simulation (DNS) must use a grid fine enough to capture λ_B, which can be several times finer than the Kolmogorov-scale grid. For instance, a DNS of homogeneous turbulence at Kolmogorov-scale resolution 1024^3 already contains over 4 billion grid points and requires on the order of 8 GB of memory per scalar field in double precision. Achieving even $2\times$ finer resolution (2048^3) to approach the Batchelor scale would raise the cell count by a factor of 8 (to $\sim 8.6 \times 10^9$ cells), and a $4\times$ refinement (4096^3) would require $\sim 6.9 \times 10^{10}$ cells – a prohibitive number of grid points and memory (hundreds of GB per snapshot). In addition, high-Schmidt-number flows demand longer simulation times to observe complete mixing, since molecular diffusion is slow. These factors result in tremendous computational expense for fully resolved micromixing studies.

Major computational challenges can be summarized as follows:

Resolution limits: Capturing Batchelor-scale gradients requires exceedingly fine grids (e.g. resolving λ_B for $\mathrm{Sc} \gg 1$ may need billions of cells). This far exceeds the grid size of standard turbulence DNS which resolves only up to Kolmogorov scales.

Memory bottlenecks: Storing and processing 3D fields at Kolmogorov or finer resolution strains memory and I/O capacity. A single 1024^3 scalar field is $\sim$4 GB (float32), and going

to finer scales multiplies this requirement. Handling many time snapshots or performing data-driven training on such fields can exhaust typical computing resources.

Schmidt number dependence: The scalar mixing problem is parameterized by Sc, which affects both the required spatial resolution ($\lambda_B \propto \mathrm{Sc}^{-1/2}$) and the mixing time scale. High-Sc flows exhibit thin concentration filaments that persist longer, necessitating small time steps and long integration times to capture diffusion. Each distinct Sc essentially changes the physics and may even require a different grid refinement, complicating the development of one model that works across all Sc.

These challenges have motivated alternative strategies for simulating and understanding micromixing. One approach is to develop reduced-order or subgrid mixing models that capture the effects of unresolved scales without resolving them directly. Recent overviews of scalar–turbulence physics and modelling summarize such closures within RANS/LES frameworks—including engulfment-type micromixing models, Conditional Moment Closure (CMC), and transported probability density function (PDF) methods—which provide statistical descriptions of subgrid scalar mixing [2, 4, 5]. While useful, these models often require empirical tuning and simplify the rich dynamics of scalar turbulence [2].

Machine Learning Surrogates for CFD: Recent advances in scientific machine learning offer a new avenue to tackle these issues. Instead of empirically modeling the unresolved physics, one can leverage data-driven models (trained on high-fidelity simulation data) to serve as surrogates for the fine-scale physics. Neural networks, due to their ability to approximate complex nonlinear mappings, have been increasingly applied in fluid dynamics and PDE problems [6, 7]. In particular, convolutional neural networks (CNNs) are well-suited for capturing spatially correlated fluid flow features. CNN-based surrogate models have successfully learned to predict flow fields around obstacles with high accuracy, even generalizing to new geometrries, when trained on CFD simulation datasets [7, 8]. For instance, a U-Net CNN can reproduce steady 2D flow solutions in various domains with errors of only a few percent [7, 8]. These surrogates operate orders of magnitude faster than solving the Navier–Stokes equations directly, once trained [8].

Beyond purely data-driven approaches, physics-informed neural networks (PINNs) integrate physical laws into the learning process. By penalizing the PDE residual in the loss function or otherwise embedding known physics, PINNs can learn solutions that satisfy governing equations and boundary conditions [6, 9]. This technique, introduced by Raissi et al. [10], has demonstrated success on problems ranging from basic ODE systems to high-dimensional fluid flows, often allowing solutions from sparse or noisy data. In turbulent flow research, hybrid approaches have emerged where neural networks assist or replace parts of classical models while respecting physical constraints [9]. Notably, Raissi et al. applied physics-informed deep learning to turbulent mixing, showing that a neural network can discover improved closure models for scalar diffusion and dissipation using limited data from a direct numerical simulation [10]. This indicates that neural networks are capable of capturing micromixing effects that would be exceedingly difficult to resolve or model empirically.

Encouraged by these developments, the present work focuses on using a deep CNN to simulate turbulent micromixing as a data-driven surrogate, with a strong emphasis on maintaining physical fidelity. The application context is pharmaceutical and fine chemical manufacturing, where rapid and thorough mixing is essential for efficiency and sustainability. By training on a database of turbulence-resolved scalar concentration fields (obtained from OpenFOAM CFD simulations of a decaying scalar blob in homogeneous turbulence), the neural network learns to predict the 3D scalar field evolution given flow conditions. The goal is to reproduce the key physics of turbulent scalar transport (advection–diffusion, cascade of scalar fluctuations down to dissipation scales) in order to vastly speed up predictions, enabling what would normally require a supercomputing DNS to be done in seconds. Such a surrogate could be used, for example, to quickly evaluate the micromixing efficiency of different reactor designs or feed injection strategies, thereby guiding process optimization to reduce waste.

Physics-Informed Network Design: Building an accurate surrogate for micromixing required careful incorporation of physics domain knowledge into the neural network design. We adopted a 3D CNN with a U-Net architecture, which preserves multi-scale features through skip connections, crucial for representing both large eddies and fine filamentary structures. The network was conditioned on the flow's Schmidt number as an input parameter, so that a single model could handle a spectrum of diffusivities. To further imbue physical realism, the loss function included custom regularization terms beyond mean-squared error. In particular, a non-negativity penalty was applied to ensure the network never predicts unphysical negative concentration values, and a total variation penalty was added to promote smoothness of the scalar field in all spatial directions, mirroring the expectation that molecular diffusion produces smooth concentration gradients. These physics-informed penalties act to keep the CNN's predictions within the realm of physically admissible solutions (e.g., enforcing that the scalar field remains bounded and well-behaved at small scales). Additionally, the training data were annotated with spatial coordinate information and normalized in a way that preserves dimensional consistency, allowing the CNN to implicitly learn about domain size and boundary conditions. By blending data-driven learning with physics-based constraints, the model aims to generalize better and require less training data than a black-box network.

Relevance to Sustainable Pharmaceutical Manufacturing: The ability to accurately predict micromixing outcomes has tangible benefits for process sustainability. For example, by using the trained CNN, engineers at a company like Bayer could rapidly evaluate how changes in stirring speed, feed concentration, or reactor geometry influence the distribution of reactants at the microscale. Improved micromixing would mean reactants meet and react more uniformly, avoiding local over-concentrations that trigger side reactions [1]. This leads to higher yields of the main product and fewer unwanted impurities that would otherwise require energy-intensive separation [1]. Moreover, faster simulation turnaround enables in silico experimentation with various process conditions (digital pro-

cess design) without the waste or delay of extensive physical trials. Thus, this research lies at the intersection of fluid physics, machine learning, and sustainable engineering by using advanced computational tools to drive more efficient chemical manufacturing.

In the following sections of this report, the methodology for generating training data and training the CNN is first described, including the details of the CFD setup and network architecture. This is followed by a validation of the neural network's performance against conventional CFD, focusing on its ability to reproduce turbulent scalar transport at both low and high Schmidt numbers. Finally, the implications of this approach for industrial micromixing simulations and future developments (such as incorporating reactive chemistry into the neural network model) are discussed.

2. Methodology

2.1 Computational Fluid Dynamics Setup

2.1.1 Homogeneous Isotropic Turbulence Data

Referring to the **Johns Hopkins Turbulence Database (JHTDB)** [11], two types of homogeneous turbulence datasets are available (isotropic 1024 coarse and isotropic 1024 fine).

Dataset	**Time span**	**Δt** (time step size)	**Snapshots available**
Coarse	0 to 10.056 s	0.002 s	5028 frames (every 10 DNS steps)
Fine	0.0002 to 0.0198 s	0.0002 s	99 frames (every single DNS step)

From the description of the file associated with the data on the website, the following information can be collected

- Time range: 0.0002 to 0.0198 seconds
- Time step: 0.0002 seconds
- Number of snapshots: 99
- Total duration: 0.0196 seconds
- Large eddy turnover time T_L= 1.99 seconds
- Kolmogorov time scale τ_η = 0.0424 seconds

Timescale	**Duration**	**What changes visibly?**
T_L (1.99 s)	Large-Scale energy	Structure/flow regime
$\tau_\eta(0.042s)$	Smallest eddies	Vorticity/dissipation
0.0196 s (isotropic 1024 fine)	$<$ half of τ_η	**Barely anything changes**

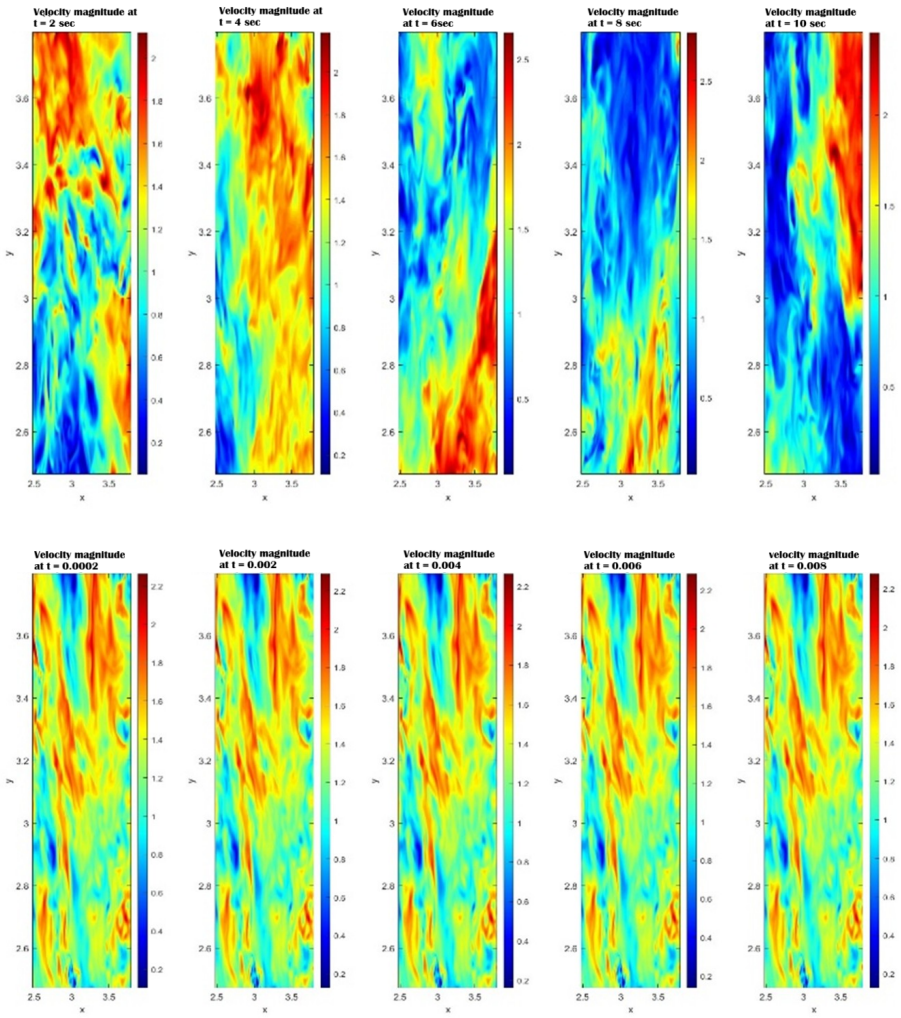

Figure 1: **Velocity magnitude contours at mid-plane (z).** Top: coarse data at $t = 2$, 4, 6, 8, and 10 s. Bottom: fine data at $t = 0.0002$, 0.002, 0.004, 0.006, and 0.008 s. Velocity in $\mathbf{m/s}$; x and y in $\mathbf{m}$.

From the figures shown, it is clear that the velocity barely changes in isotropic 1024 fine data, but changes noticeably in the isotropic 1024 coarse data, the reason behind that, can be justified as follows: the time frame of the fine data is too short (less than Large eddy turnover time and kolmogorov turnover time) to make any changes in the flow structure, whether on the large scale level or the small scale level.

2.1.2 OpenFOAM Case Configuration

This section describes the simulation on OpenFOAM, which was done for a scalar diffusion, i.e., in our case, the Temperature (T) is chosen, under the influence of the velocity field generated from the HIT data.

As shown in Figure 2, a temperature field is created, such that a sphere in the center of the simulation box domain is assigned a temperature value of 1, and elsewhere in the domain, the value of the temperature is zero, creating a difference in scalar values between two different zones in the domain to study its diffusion evolution with time.

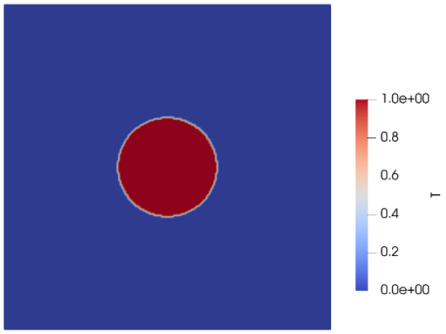

Figure 2: The initial temperature field used to study scalar diffusion in turbulence environment

To be flexible in changing the dimensions of this sphere for any purpose, the setFieldsDict was added to the /system/ folder, as attached to this report in the file bank.

2.1.3 Diffusion Time Analysis

The decision of the type of data to be used in this project was a critical step in the project.

Table 1: **Comparison of Molecular and Turbulent Diffusion Timescales in fluid Water (Schmidt Number $Sc = 1$)**

Description	**Quiescent (Molecular) Diffusion Only**	**Turbulent Diffusion (with Eddy Viscosity)**
Relationship for t_{diff}	$t_{\text{diff}} \sim \dfrac{R^2}{D} = \dfrac{R^2}{\nu}$	$D_{\text{eff}} = \nu + Cu'L,$ $t_{\text{mix}} \sim \dfrac{R^2}{D_{\text{eff}}} = \dfrac{R^2}{\nu + Cu'L},$ $t_{\text{mix}} \sim \dfrac{R^2}{Cu'L}, \quad D_t \gg \nu,$ $\dfrac{t_{\text{mix}}}{T_L} \approx 3\left(\dfrac{R}{L}\right)^2$
Diffusion time for $Sc = 1$	2.8 h	1 s

Variable definitions:

- t_{diff} – molecular diffusion time [s]
- t_{mix} – turbulent mixing time [s]
- D – molecular diffusivity [m^2/s]
- ν – kinematic viscosity [m^2/s]
- D_{eff} – effective diffusivity, accounting for turbulence [m^2/s]
- D_t – turbulent diffusivity ($D_t = Cu'L$) [m^2/s]

- u' – characteristic turbulent velocity fluctuation [m/s]
- L – characteristic turbulent length scale [m]
- T_L – Lagrangian integral timescale of turbulence [s]
- R – characteristic diffusion distance [m]
- C – empirical constant (typically $C \approx 0.1$–1)
- Sc – Schmidt number, $Sc = \nu/D$

Properties for water at 20°C:

$$\nu = 1.0 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}, \quad D = 1.0 \times 10^{-9}\,\mathrm{m}^2/\mathrm{s}, \quad Sc = \frac{\nu}{D} \approx 1000$$

Example calculation:

For a diffusion distance $R = 1$ cm $= 0.01$ m,

$$t_{\text{diff}} = \frac{R^2}{D} = \frac{(0.01)^2}{1 \times 10^{-9}} = 10^5\,\mathrm{s} \approx 2.8\,\mathrm{h}$$

For turbulent diffusion with $u' \sim 0.01$ m/s, $L \sim 0.01$ m, and $C \approx 1$:

$$D_{\text{eff}} \approx \nu + Cu'L \approx 1 \times 10^{-6} + 10^{-4} \approx 10^{-4}\,\mathrm{m}^2/\mathrm{s}$$

$$t_{\text{mix}} = \frac{R^2}{D_{\text{eff}}} = \frac{(0.01)^2}{10^{-4}} = 1\,\mathrm{s}$$

From the time difference between quiescent and turbulent diffusion, it is obvious that turbulence has a great effect on accelerating the diffusion process (and hence the mixing of the scalar throughout the domain).
Based on the estimated simulation time, the coarse data would be suitable for the simulation, due to the fact that it spans a larger time period, so it could capture the full physical domain spatially and temporally.

2.1.4 Solver Development

For the basic case of scalar diffusion, **scalarTransportFoam** is used, which is a **steady-state OpenFOAM solver** used to simulate **passive scalar transport**, i.e., advection-diffusion equation for a scalar (like temperature or pollutant concentration) in a **given flow field**.

The problem with the default **scalarTransportFoam** solver is that it is used to solve for a frozen steady velocity field; therefore, it cannot handle different velocity fields at different time steps, which is necessary for a transient and dynamic simulation. For that, we have internally modified the solver code to incorporate the transient velocity capabilities in a new solver named **transientScalarTransportFoam (see Appendix A for more details about the modifications)**.

2.1.5 Velocity and Temperature Fields

Using the HIT data from Johns Hopkins Turbulence Database [11], we worked with a cut-out service, which enables one to choose the specific set of data needed. 1024 points generated in $[2\pi]^3$ grid (Coarse and fine as described and mentioned before), you can cut out a specific dataset in a specific range of the original domain at a particular step of time. The data is usually downloaded in the form of a **.h5 file** and converted into an OpenFOAM-readable velocity field via a Matlab code (**see Appendix B for more details about the MATLAB code**).

2.1.6 Mesh Generation and Computational Aspects

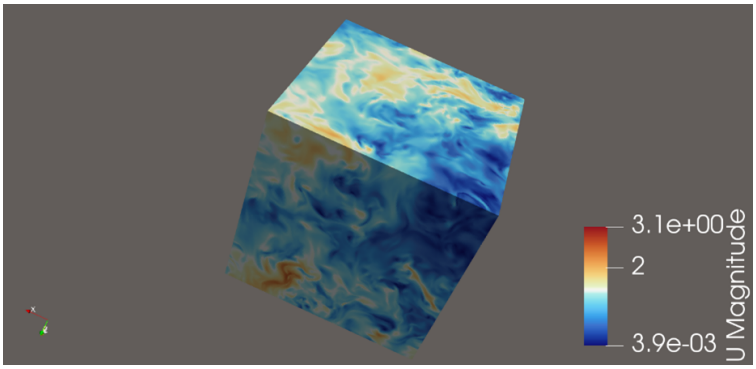

Figure 3: Velocity field resulting from HIT data of 217 points on 216^3 mesh with velocity unit in m/sec

For the velocity field (U), and due to limited computational power now, we could not use the full 1024 original data points; instead, we have clipped 217 points out of the original domain and used them as a reference, which counted for a total of 10077696 cells in the blockMeshDict execution file.
As seen in Figure 3, the turbulent velocity field corresponding to this dataset is viewed in Paraview.

For the temperature (T) field, setFieldsDict was used to generate a sphere at time (0) of adjustable Temperature, center position, and radius on an initially zero-set field.

For meshing, the standard tool in OpenFOAM, blockMeshDict, was used to generate a structured mesh with the same number of points as the velocity field (to fit the velocity field onto the mesh).

2.1.7 Numerical Parameters

Time step Δt had to fulfill the Courant number requirements (courant number ≤ 1)

courant number $= \frac{\text{Distance traveled by the fluid in one time step}}{\text{cell size}} = \frac{\acute{u}\times\Delta t}{\Delta x} = \frac{0.681\times 0.001}{0.006} = 0.1$ (far enough in the safe zone)

$$\text{schmidt number} = \frac{\nu}{D} = \frac{\text{momentum diffusivity}}{\text{molecular diffusivity}}$$

2.2 CFD Simulation Results

For the simulation, the **steady-state** solver was chosen for the sake of simplicity for now, in order to be able to train the Neural Network; however, there is an intention to extend the analysis using the transient modified solver at the next stage of the project.

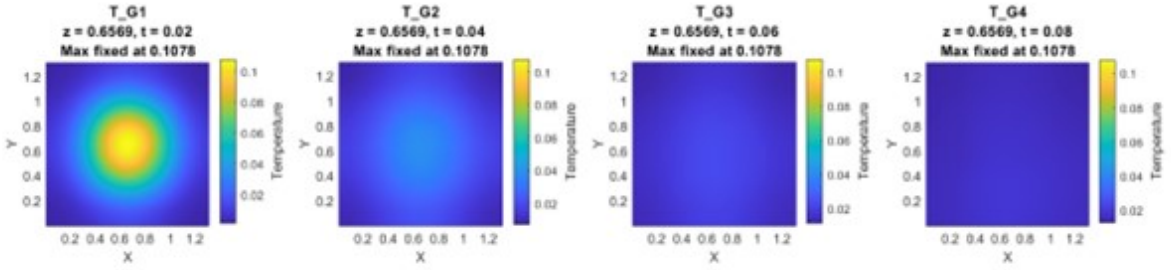

Figure 4: Diffusion of a sphere of temperature 1 K at Schmidt number $Sc = 0.0001$ for time steps $t = 0.002$, 0.004, 0.006, and 0.008 s.

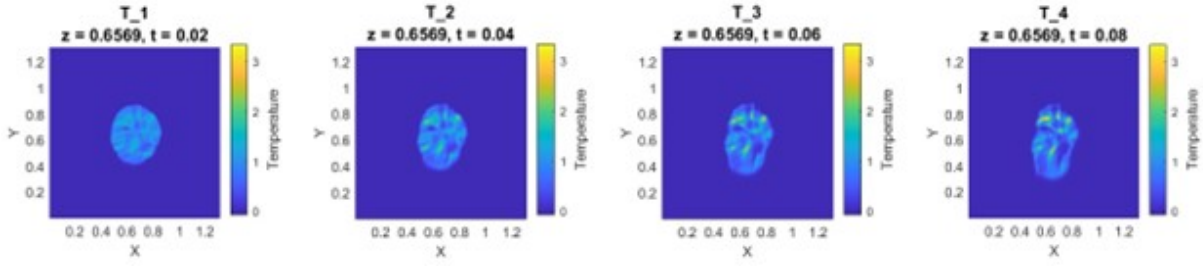

Figure 5: Diffusion of a sphere of temperature 1 K at Schmidt number $Sc = 1$ for time steps $t = 0.002$, 0.004, 0.006, and 0.008 s.

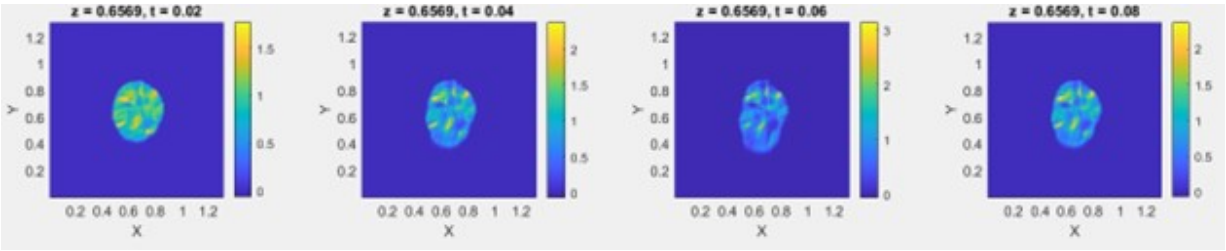

Figure 6: Diffusion of a sphere of temperature 1 K at Schmidt number $Sc = 500$ for time steps $t = 0.002$, 0.004, 0.006, and 0.008 s.

2.2.1 Schmidt Number Dependence

What we can conclude from Figures 4-6 is the following:

1. At low Schmidt numbers (0.001), it is obviously seen that the diffusion process is dominated by **molecular diffusion** rather than **Momentum diffusion**, interpreting the shape of the diagram, that diffusion took place quickly, i.e. (the concentration of temperature was smoothened severely over time steps) and almost radially, independent of the eddies of turbulence

2. On the contrary, for high Schmidt numbers, the diffusion was dominated by **momentum diffusivity**, and the concentration profile was smoothened in relatively longer time periods, owing to the fact, that molecular diffusion is weak, and, one can observe that the shape of diffusion is not radial anymore, i.e. (affected by the presence of turbulence eddies and scattered in random directions).

2.3 Neural Network Development

2.3.1 Database Development

After obtaining physical results from CFD, the next step is the development of a database for neural network training. The HIT (Johns Hopkins Turbulence Database) has a maximum of 1024^3 cells in the velocity field for a domain of $2\pi \times 2\pi \times 2\pi$. After solving for the temperature or scalar concentration field using the OpenFOAM solver, we are supposed to obtain a 1024^3 resolution temperature/scalar concentration field.

As our goal is to explore micromixing, our scale of interest is the Batchelor scale, whereas the HIT database provides Kolmogorov-scale resolution. This means that we need a resolution higher than 1024^3.

The relationship between the Kolmogorov scale (η) and Batchelor scale (λ_B) is:

$$\lambda_B = \frac{\eta}{\sqrt{Sc}}$$

If the Kolmogorov-scale field has $1024 \times 1024 \times 1024$ cells, the Batchelor-scale field could require:

- 8 times the number of cells: $1024^3 \times 8 = 8,589,934,592$ cells (if $2\times$ finer)
- 64 times the number of cells: $1024^3 \times 64 = 68,711,476,736$ cells (if $4\times$ finer)

This implies a tremendous number of cells (billions) for a single time step. Let us estimate the memory required for a 1024^3 3D field:

- Single precision (float32, 4 bytes): 1024^3 cells $\times$ 4 bytes/cell $= 4,294,967,296$ bytes $\approx$ 4 GB
- Double precision (float64, 8 bytes): 1024^3 cells $\times$ 8 bytes/cell $= 8,589,934,592$ bytes $\approx$ 8 GB

This shows that the memory required for a 1024^3 Kolmogorov-scale field is already significant. The Batchelor-scale fields, being higher resolution, would require even more memory.

Hence, the prediction of micromixing species transport using neural networks is extremely memory-intensive due to the high resolution required. To simplify things at the start, we decided to train our neural network at a resolution of 216^3 by reducing the grid dimensions.

Reducing the resolution from 1024^3 to 216^3 does not mean that we are leaving the Kolmogorov scale. Although our ultimate goal is to reach the Batchelor scale, it is too

memory-intensive for now. Therefore, we decided to focus on the Kolmogorov scale for the initial studies.

When reducing the resolution, we also reduced the physical dimensions of the grid. Initially, it was $2\pi \times 2\pi \times 2\pi$ for 1024^3 cells, and we reduced it to $1.32 \times 1.32 \times 1.32$ for 216^3 cells. In this way, we preserved the relevant scale and could, in principle, reach the Batchelor scale without memory issues.

However, the main challenge is that the HIT data involves very high velocity and turbulence intensity, which causes spheres to advect and reach the boundaries very quickly as shown Figure 7. This highlights the importance of using the full $2\pi \times 2\pi \times 2\pi$ domain , because after a few time steps, the spheres would cross the $1.32 \times 1.32 \times 1.32$ cube. Nonetheless, we compromised on this aspect of physics to start with a simpler case and plan to gradually include all these complexities later.

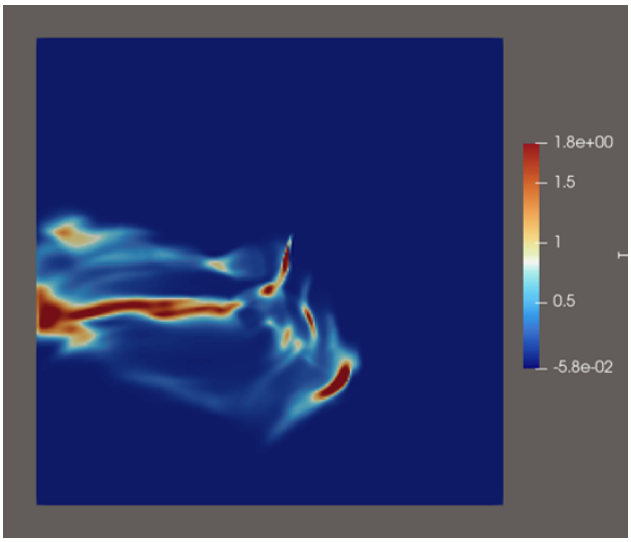

Figure 7: The figure represents the transport of the sphere on an XY slice with Sc=1 and at $t = 0.32$.

Now, let's understand the inputs and outputs of the neural network. Our goal is to predict the mixing performance (as illustrated Figure 8) based on input parameters such as the Schmidt number, Reynolds number, and initial concentration field shape.

However, the HIT database provides data for only one Reynolds number—or, more specifically, one turbulence intensity—meaning that we effectively have two variable parameters. To keep the problem manageable, we assumed that the initial concentration field shape remains fixed (a sphere). Increasing the number of input variables would require a much larger dataset and significantly more training samples.

By restricting the input to a single variable, the Schmidt number (Sc), we simplified the dataset and reduced the possible number of training combinations. The neural network is thus designed to predict the mixing performance based on a user-specified Schmidt number.

However, since we encountered unboundedness issues in the CFD results, we decided that the neural network should instead predict the 3D scalar concentration fields. The mixing performance can then be obtained through post-processing of these predicted fields.

This ensures that the NN predictions remain physically consistent, reflecting processes like turbulent eddies, vortex shedding, stretching, and molecular diffusion.

Finally, since only the Schmidt number varies, we must note that for each Schmidt number, a different spatial refinement is required according to the Batchelor scale, $\lambda_B = \eta/\sqrt{Sc}$, where higher Schmidt numbers demand finer resolution. This introduces an additional challenge—for each Schmidt number, the associated 3D scalar field will have a different number of grid cells. Consequently, the neural network must also learn or adapt to the varying grid resolutions corresponding to each Schmidt number.

The next step is to decide how many time steps are required for the prediction, as well as the values of Δt (time step size) and total simulation time. The total diffusion time depends on the Schmidt number. Systems with a high Schmidt number require longer times for molecular diffusion, whereas systems with a low Schmidt number diffuse much faster. Consequently, for low Schmidt number cases, Δt must be reduced to accurately capture the fast diffusion process and preserve the physical behavior.

In summary, each simulation is characterized by a specific Schmidt number, Δt, total time, and the number of cells (refinement) of the 3D scalar field. This makes the problem highly demanding in terms of memory, CPU, and GPU resources. Therefore, we initially decided to simplify the setup by keeping the total number of cells the same across all Schmidt numbers, using a fixed small Δt, and maintaining the same total simulation time. This approach allows us to preserve the essential physics while keeping the computational cost manageable.

In this simplified setup, the Schmidt number is used as an input to the neural network, and the network is trained to predict the 3D scalar concentration fields for the first four time steps.

Ideally, the NN inputs and outputs would correspond to Figure 8. After applying the discussed simplifications, however, the inputs are reduced to Sc and the initial concentration.

We developed a small database of 10 Schmidt numbers: $[0.0001, 0.001, 0.01, 0.1, 1, 100, 200, 300, 400, 500]$. For each Schmidt number, we obtained the 5 time steps of the 3D scalar field (T files: T0, T1, T2, T3, T4) from OpenFOAM.

```
postProcess -func "writeCellCentres" -time 0
```

This generated the coordinates of the cell centers for the entire grid. Next, we merged these coordinates with the corresponding T values to create a single CSV file for each time step. We repeated this process for all Schmidt numbers, resulting in a database structured as follows: each Schmidt number has 5 CSV files corresponding to the 5 time steps.

```
DataBase/
├── DB_SC_500/
│   ├── T_0.csv
│   ├── T_1.csv
│   ├── ⋮
│   └── T_4.csv
└── DB_SC_400/
```

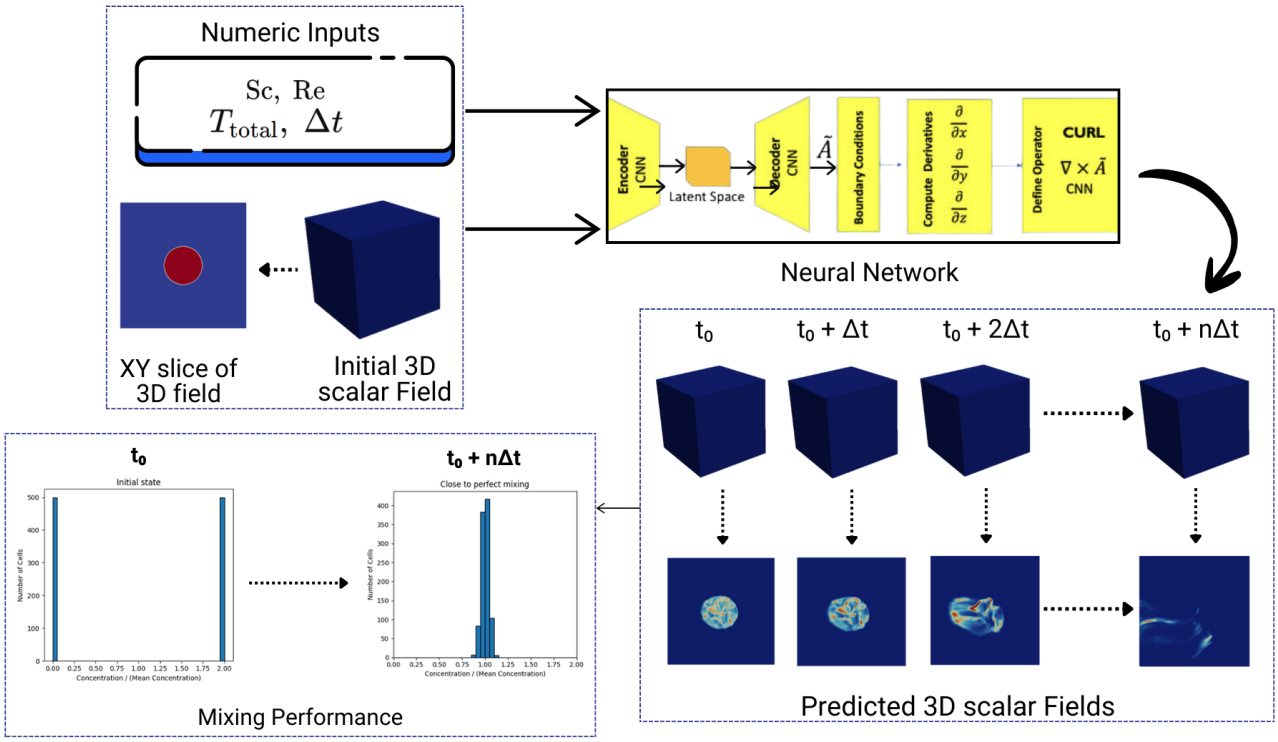

Figure 8: Schematic of Neural Network Design

```
    ├── T_0.csv
    ├── T_1.csv
    ├── ⋮
    └── T_4.csv
├── ⋮
└── DB_SC_0.0001/
    ├── T_0.csv
    ├── T_1.csv
    ├── ⋮
    └── T_4.csv
```

Our goal is to predict the initial 4 time steps of the 3D scalar field for any given Schmidt number.

2.3.2 Model Architecture Design

For training the neural network, we used TensorFlow. Initially, we worked on Google Colab, as it provides access to GPUs and TPUs for running TensorFlow in Jupyter notebooks. We attempted to upload scalar fields with a resolution of $100 \times 100 \times 100$ and train the neural network, but encountered an OOM (Out-Of-Memory) or resource exhausted error. Consequently, we switched to the École des Mines (EMSE) computing cluster, which offers the following resources (Figure 9).

Note that the A100 node is extremely powerful for the most demanding tasks like AI, ML and complex simulation. The EMSE cluster utilizes the SLURM workload manager for

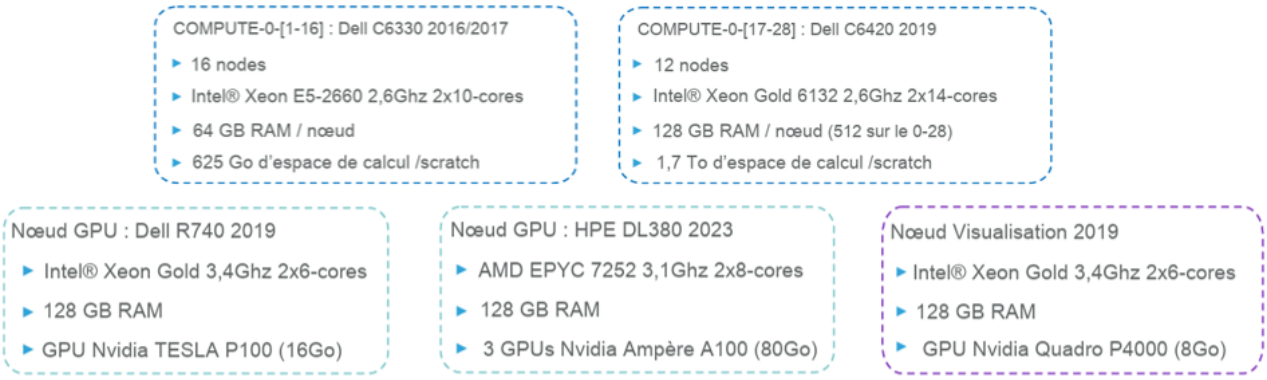

Figure 9: EMSE cluster properties

job scheduling and resource management. All user jobs are submitted and executed through the /cm/shared/apps/slurm environment.

We began our investigation with a fully connected dense neural network architecture designed to map Schmidt numbers to 3D temperature fields. The network was structured to accept Schmidt number values across five time steps as input and produce complete 3D temperature distributions at 216^3 resolution across the same five time steps as output. The Model architecture is shown below Listing 1

```
Input: (5, 1) -> Schmidt numbers across 5 time steps
Layer 1: Flatten() -> (5,)
Layer 2: Dense(128, activation='relu')
Layer 3: Dense(256, activation='relu')
Layer 4: Dense(512, activation='relu')
Layer 5: Dense(216x216x216x5, activation='linear')
Layer 6: Reshape((216, 216, 216, 5))
```

Listing 1: Network Architecture

This approach leverages the regular grid structure of our computational domain. Since the spatial coordinates remain constant across all simulations and time steps, we can implicitly encode the grid geometry within the network weights rather than explicitly providing coordinate information. This design choice mathematically sound for regular grids, though it would be unsuitable for irregular meshes or dynamically evolving grids.

Our computational infrastructure featured an AMD EPYC 7252 processor with 32 cores, 128 GB of RAM, and three NVIDIA A100 GPUs with 80GB memory each, though GPU utilization remained constrained due to architectural incompatibilities. The dataset encompassed five Schmidt numbers, each with five time steps, and 216^3 grid points. Despite these substantial computational resources, we encountered immediate and insurmountable Out-of-Memory errors during model initialization, precluding any training endeavors. The fundamental memory constraint originated from the combinatorial explosion of parameters

within the output layer. Our detailed analysis revealed a staggering parameter count:

Table 2: Parameter Count Breakdown

Layer	Output Shape	Parameters	Calculation
Input	(5, 1)	-	-
Flatten	(5)	-	-
Dense 1	(128)	768	$(5 \times 128) + 128$
Dense 2	(256)	33,024	$(128 \times 256) + 256$
Dense 3	(512)	131,584	$(256 \times 512) + 512$
Output Dense	(50,388,480)	**25,849,290,240**	$(512 \times 50{,}388{,}480) + 50{,}388{,}480$
Total	-	**25,849,455,616**	Sum of all layers

The memory requirements escalated exponentially: parameter storage alone consumed 103.4 GB (25.8 billion parameters × 4 bytes), with additional overhead for gradient storage (+103.4 GB), optimizer states (+206.8 GB for Adam), and activation maps (+50-100 GB), culminating in an estimated total requirement of 450-500 GB—far exceeding our available 128 GB RAM.

The memory crisis stems from fundamental scaling laws in fully connected architectures:

```
Output Dimension Challenge:
216 x 216 x 216 x 5 = 50,388,480 output values

Parameter Explosion in Final Layer:
512 input neurons x 50,388,480 output neurons = 25.8 billion weights
```

A critical insight emerged that the memory bottleneck is inherent to the model architecture rather than the dataset scale. The colossal parameter count in the final dense layer creates an impassable memory barrier: a single sample requires approximately 201 MB for output data plus 103.4 GB for model weights, while five samples need about 1 GB data with the same massive model weights, and ten samples approximately 2 GB data with unchanged weight memory. Reducing training samples provided negligible relief because the model architecture itself exhausted available memory before any data processing could commence.

To overcome the memory limitations of the 216^3 resolution approach, we implemented a downsampling strategy using a 100×100×100 grid resolution, which reduced the computational requirements by approximately 10× while maintaining adequate spatial representation. The model architecture employed was as follows: Listing 2

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 5)                 0

 dense_1 (Dense)             (None, 128)               768
```

```
 dense_2 (Dense)              (None, 256)               33,024

 dense_3 (Dense)              (None, 512)               131,584

 dense_4 (Dense)              (None, 5000000)           2560000000

 reshape (Reshape)            (None, 100, 100, 100, 5)  0

=================================================================
Total params: 2,560,165,376
Trainable params: 2,560,165,376
Non-trainable params: 0
_________________________________________________________________
```

Listing 2: Neural Network Architecture for 100×100×100 Resolution

This architecture successfully learned the mapping from Schmidt numbers to 3D temperature fields, enabling us to train the neural network model and predict initial five 3D scalar temperature fields for both Sc=250 and Sc=0.005, achieving practical results while maintaining computational feasibility. The downsampling from 216^3 to 100^3 resolution was performed using SciPy's zoom function with linear interpolation, reducing the output dimension from 50,388,480 to 5,000,000 values while preserving the essential physical features of the temperature fields.

The predictions shown in Figure 10 are physically reasonable, in a sense that they capture vortices and eddies. However, the model does not adequately learn that diffusion should dominate at low Schmidt numbers (Sc) and advection at high Sc, since both predictions appear quite similar with only minor differences.

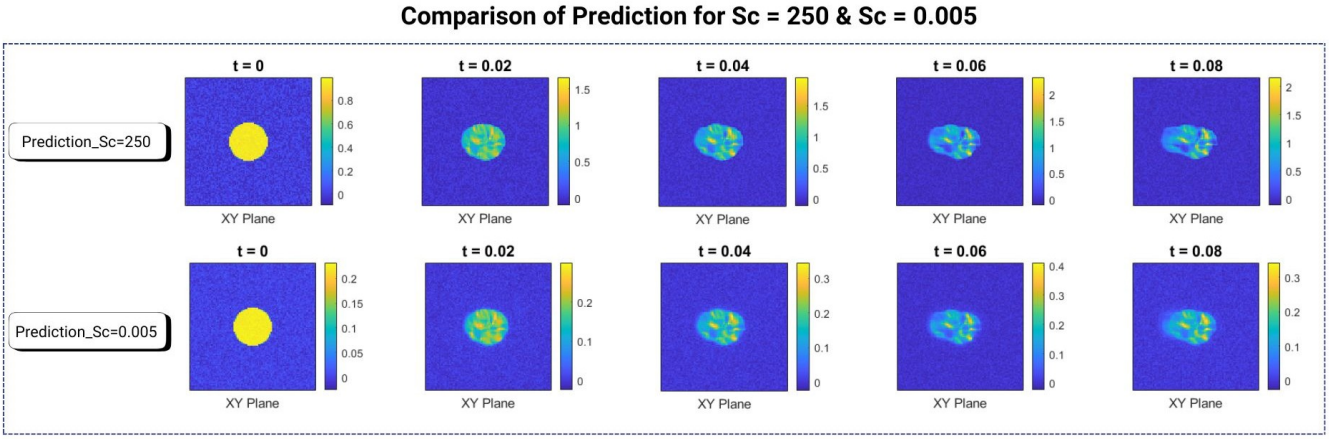

Figure 10: Comparison of fully connected dense NN Prediction for Sc = 250 & Sc = 0.005

One major challenge with reducing spatial resolution is that micromixing performance must ultimately be evaluated using histograms of cell concentrations. When the number of cells decreases (i.e., with lower resolution), this metric becomes unreliable and fails to meet

the project requirements. Therefore, we need to design a new neural network architecture that is both memory-efficient and capable of learning molecular diffusion patterns based on the Schmidt number (Sc), which the fully connected neural network was unable to capture. Additionally, the architecture must be suitable for handling 3D scalar concentration fields.

Based on our literature review, we adopted a CNN architecture as it represents the most robust option (Memory Efficient). We also attempted to combine CNN and PINNs in the implementation provided in the file *tf_test.py* (see Appendix C).

2.3.3 Physics-Informed Loss Functions

This code implements a sophisticated physics-informed deep learning framework for spatio-temporal prediction of 3D scalar fields across multiple time steps. The system is designed to learn the complex evolution of scalar field distributions in a physical domain by combining convolutional neural networks with embedded physical constraints.

The model employs a 3D U-Net–style encoder–decoder architecture that takes two inputs: the initial scalar field T_0 ($216 \times 216 \times 216 \times 1$) and the Schmidt number. The Schmidt number is spatially expanded and concatenated with the coordinate channels, then processed through multiple 3D convolutional blocks with batch normalization and skip connections to predict four future time steps (T_1–T_4) simultaneously. The model architecture is represented in Listing 3

```
Model: "3D CNN"

_________________________________________________________________________
Layer (type)                 Output Shape              Param #
=========================================================================
initial_field (InputLayer)   (None, 216,216,216, 1)    0
schmidt_number (InputLayer)  (None, 1)                 0
lambda (Lambda)              (None, 216,216,216, 1)    0
lambda_1 (Lambda)            (None, 216,216,216, 1)    0
concatenate (Concatenate)    (None, 216,216,216, 2)    0
conv3d (Conv3D)              (None, 216,216,216, 8)    448
batch_normalization (BatchNorm) (None, 216,216,216, 8)    32
conv3d_1 (Conv3D)            (None, 216,216,216, 8)    1,736
batch_normalization_1 (BatchNo) (None, 216,216,216, 8)    32
max_pooling3d (MaxPooling3D) (None, 108,108,108, 8)    0
conv3d_2 (Conv3D)            (None, 108,108,108, 16)   3,472
batch_normalization_2 (BatchNo) (None, 108,108,108, 16)   64
conv3d_3 (Conv3D)            (None, 108,108,108, 16)   6,928
batch_normalization_3 (BatchNo) (None, 108,108,108, 16)   64
max_pooling3d_1 (MaxPooling3D) (None, 54,54,54, 16)    0
conv3d_4 (Conv3D)            (None, 54,54,54, 32)      13,856
batch_normalization_4 (BatchNo) (None, 54,54,54, 32)      128
```

```
conv3d_5 (Conv3D)               (None, 54,54,54, 32)         27,664
batch_normalization_5 (BatchNo) (None, 54,54,54, 32)         128
max_pooling3d_2 (MaxPooling3D)  (None, 27,27,27, 32)         0
conv3d_6 (Conv3D)               (None, 27,27,27, 64)         55,328
batch_normalization_6 (BatchNo) (None, 27,27,27, 64)         256
conv3d_7 (Conv3D)               (None, 27,27,27, 64)         110,656
batch_normalization_7 (BatchNo) (None, 27,27,27, 64)         256
up_sampling3d (UpSampling3D)    (None, 54,54,54, 64)         0
conv3d_8 (Conv3D)               (None, 54,54,54, 32)         55,328
concatenate_1 (Concatenate)     (None, 54,54,54, 64)         0
conv3d_9 (Conv3D)               (None, 54,54,54, 32)         55,328
batch_normalization_8 (BatchNo) (None, 54,54,54, 32)         128
conv3d_10 (Conv3D)              (None, 54,54,54, 32)         27,664
batch_normalization_9 (BatchNo) (None, 54,54,54, 32)         128
up_sampling3d_1 (UpSampling3D)  (None, 108,108,108, 32)      0
conv3d_11 (Conv3D)              (None, 108,108,108, 16)      13,840
concatenate_2 (Concatenate)     (None, 108,108,108, 32)      0
conv3d_12 (Conv3D)              (None, 108,108,108, 16)      13,840
batch_normalization_10 (BatchN) (None, 108,108,108, 16)      64
conv3d_13 (Conv3D)              (None, 108,108,108, 16)      6,928
batch_normalization_11 (BatchN) (None, 108,108,108, 16)      64
up_sampling3d_2 (UpSampling3D)  (None, 216,216,216, 16)      0
conv3d_14 (Conv3D)              (None, 216,216,216, 8)       3,464
concatenate_3 (Concatenate)     (None, 216,216,216, 16)      0
conv3d_15 (Conv3D)              (None, 216,216,216, 8)       3,464
batch_normalization_12 (BatchN) (None, 216,216,216, 8)       32
conv3d_16 (Conv3D)              (None, 216,216,216, 8)       1,736
batch_normalization_13 (BatchN) (None, 216,216,216, 8)       32
output (Conv3D)                 (None, 216,216,216, 1)       9
=================================================================
Total params: 1,630,164
Trainable params: 1,629,236
Non-trainable params: 928
```

Listing 3: 3D CNN Model Summary

A key feature is the `PhysicsAwareLoss` class, which extends beyond simple mean squared error by incorporating two critical physical constraints: (1) a non-negativity penalty that penalizes predicted negative scalar field values through `tf.nn.relu(-y_pred)` since physical quantities cannot be negative, and (2) a 3D total variation regularization term that enforces spatial smoothness by minimizing gradients across all three spatial dimensions, reflecting the diffusive nature of scalar transport.

The system maintains physical coordinate information throughout processing, mapping the $216 \times 216 \times 216$ grid to actual physical coordinates [0, 1.32] in each direction. This

coordinate-aware approach enables the CNN to better capture spatial relationships and boundary conditions. The data loading procedure carefully sorts points by their physical coordinates and ensures consistent spatial resolution.

2.3.4 Training Procedure and Computational Infrastructure

The neural network was trained such that the Schmidt number and initial concentration field (T0) was provided as input, and the model generated the first four 3D scalar fields as output. These predicted fields were then compared with the corresponding CFD-generated scalar fields, and based on the loss function (i.e., the difference between the two), the network updated its weights and biases. This process is illustrated in Figure 11.

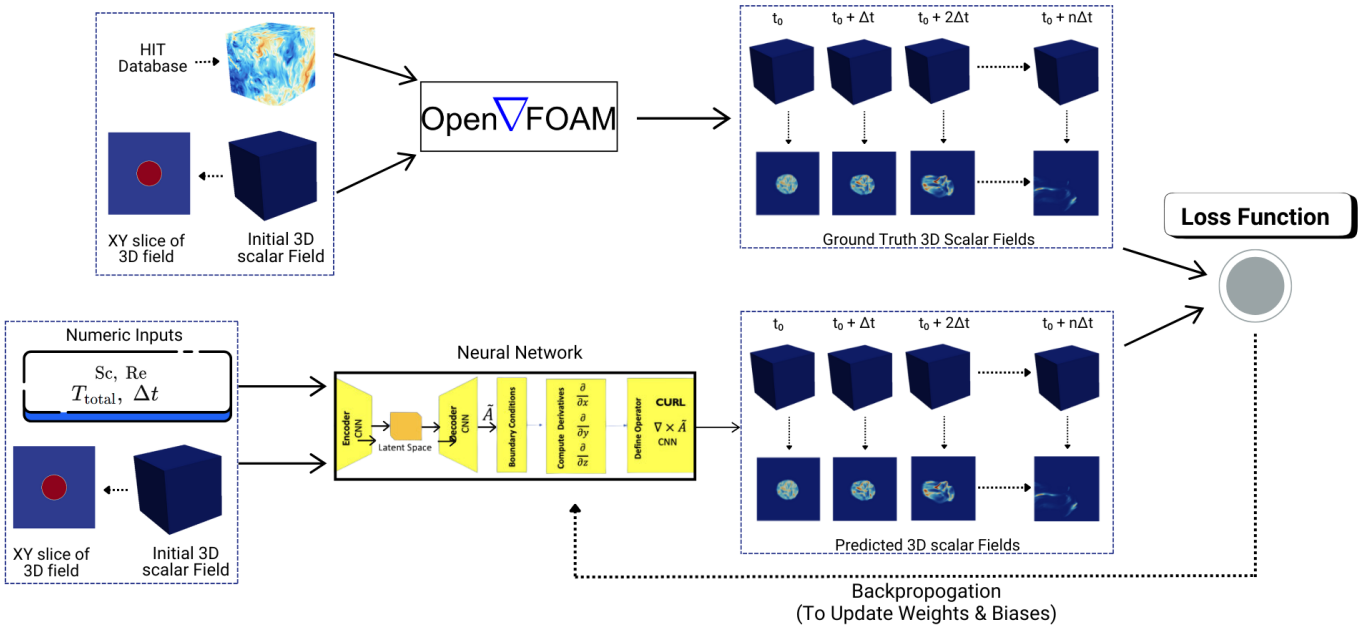

Figure 11: Neural Network Training Process

Training is conducted using a multi-scale strategy with appropriate normalization of both input fields and Schmidt numbers (log-scaled). The framework includes advanced callbacks such as early stopping, learning rate reduction, and model checkpointing. The batch size is set to 8 to accommodate the large 216^3 resolution within available memory limits.

Finally, the system generates OpenFOAM-compatible field files and CSV outputs for each 3D scalar field, making the predictions directly applicable to CFD workflows.

3. Results and Discussion

3.1 Fully Connected Network Results

The predictions from the fully connected dense neural network shown in Figure 10 demonstrated that while the architecture could capture basic flow features like vortices and eddies, it struggled to properly distinguish between diffusion-dominated and advection-dominated regimes at different Schmidt numbers.

3.2 Convolutional Neural Network Results

3.2.1 3D U-Net Architecture Performance

We predicted the initial four 3D scalar fields through this NN model for Sc = 499 and Sc = 0.00009, as shown in Figure 12 and Figure 13.

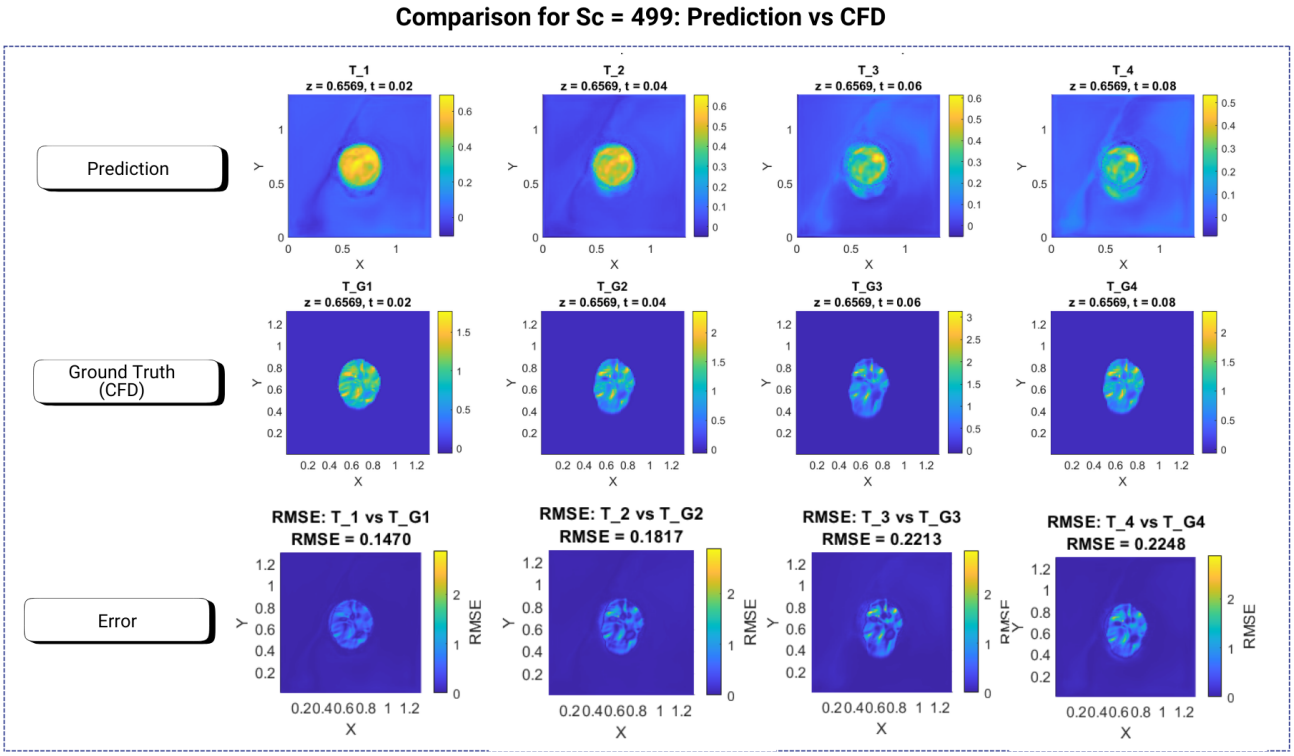

Figure 12: Comparison for Sc = 499: Prediction vs CFD

It can be observed that our neural network has learned that at low Sc, molecular diffusion dominates, whereas at high Sc, advection is more significant. At Sc = 0.00009, the predicted fields exhibit more diffusion; however, the error relative to the CFD results is considerable. Nevertheless, it is evident that the neural network captures the overall pattern. We expect improved accuracy by enlarging the database, with smaller increments in Sc values so that consecutive Sc values are close. For Sc = 499, the predicted fields show less diffusion, and the error is smaller as well.

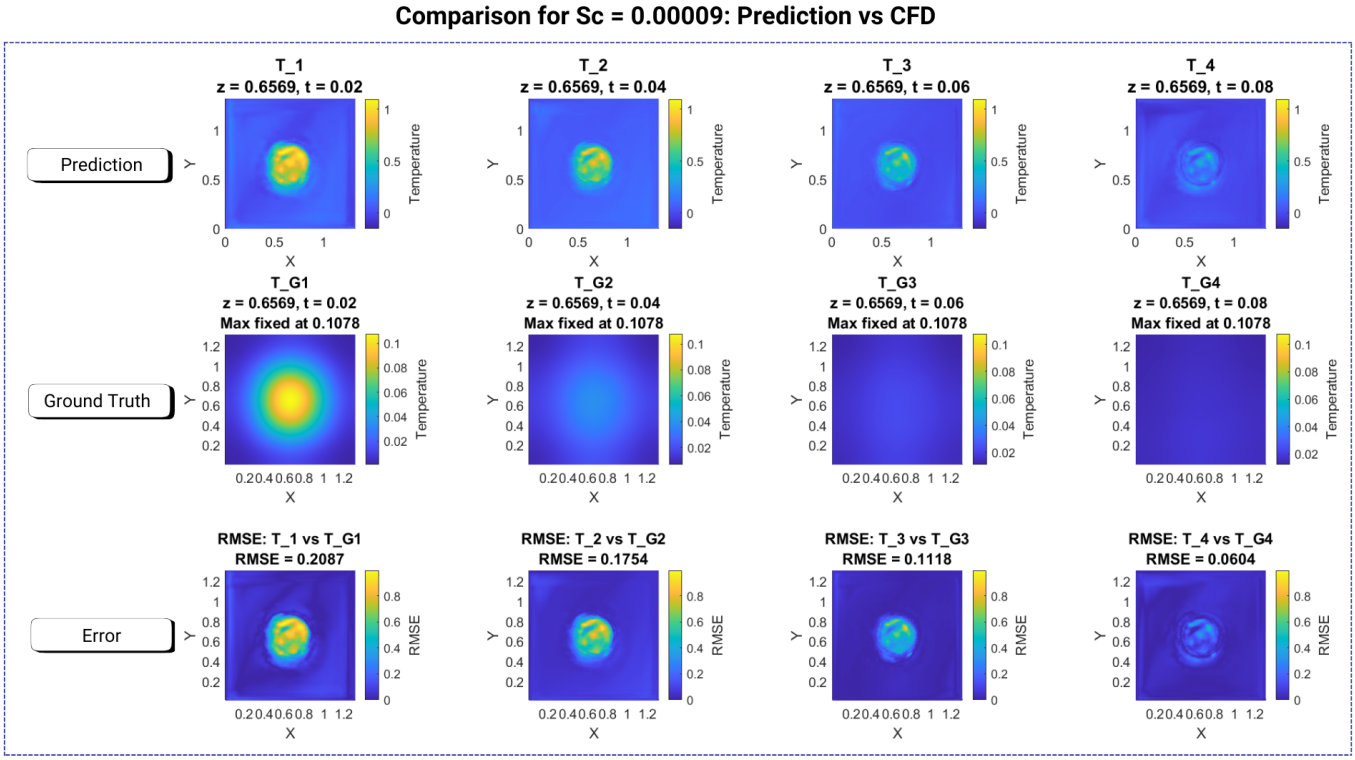

Figure 13: Comparison for Sc = 0.00009: Prediction vs CFD

3.2.2 Training Convergence Analysis

Training output saved to the file: slurm-218740.txt. The spatiotemporal 3D CNN training for diffusivity prediction was successfully completed over a 12.5-hour period from October 25-26, 2025. The model demonstrated excellent convergence, reducing training loss from 2.26 to 0.0648 (representing a 35-fold improvement) while maintaining stable learning behavior throughout the 64 epochs of training. The model successfully learned to predict temporal evolution of scalar fields across a wide range of Schmidt numbers, showing strong generalization capabilities and producing physically plausible results.

The model architecture employed a 3D U-Net design with skip connections and Schmidt number conditioning. The network processed input fields of dimension (216, 216, 216, 1) combined with scalar Schmidt number values to produce output fields of dimension (216, 216, 216, 4) representing four temporal steps. With 1,405,156 total parameters and a compact 5.36 MB model size, the architecture demonstrated excellent efficiency for handling large-scale 3D problems. The feature progression followed a symmetric encoder-decoder pattern with feature maps progressing as $16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16$, utilizing ReLU activations with batch normalization for stable training.

Table 3: Training Configuration Parameters

Parameter	**Value**
Batch Size	8 (Full Dataset)
Initial Learning Rate	1.0×10^{-4}
Final Learning Rate	2.5×10^{-5}
Optimizer	Adam
Loss Function	MSE + MAE
Epochs Completed	64/100
Training Time per Epoch	90-95 seconds
Total Training Time	12.5 hours

The training configuration employed a full-batch approach with 8 samples per epoch, utilizing the Adam optimizer with an initial learning rate of 1.0×10^{-4} that was automatically reduced to 2.5×10^{-5} at epoch 64 when performance improvements plateaued. The combined MSE and MAE loss function provided balanced optimization for both overall error magnitude and pointwise accuracy. Over the 64 completed epochs, the model maintained consistent training times of 90-95 seconds per epoch, resulting in a total training duration of approximately 12.5 hours.

The dataset comprised high-resolution $216 \times 216 \times 216$ volumetric fields spanning a physical domain of 0.0 to 1.32 units, with temporal sequences capturing evolution from initial condition T_0 through four subsequent time steps T_1 to T_4. Schmidt numbers covered four orders of magnitude from 0.0001 to 500, providing diverse diffusion scenarios for model learning. Data preprocessing involved min-max scaling with initial field T_0 ranging from -0.1217 to 8.2149 and target fields ranging from -2.5229 to 66.9445. The training utilized eight Schmidt numbers (100, 0.0001, 300, 0.01, 500, 1, 0.1, 200) while reserving two unseen numbers (400, 0.001) for validation.

Table 4: Training Convergence Metrics

Metric	**Initial**	**Final**	**Improvement**
Training Loss	2.2600	0.0648	34.9×
Training MAE	0.8992	0.1021	8.8×
Training MSE	2.1985	0.0624	35.2×

3.2.3 Validation on Unseen Schmidt Numbers

Validation performance on unseen Schmidt numbers demonstrated strong generalization capabilities. For Schmidt number 400, the model achieved MSE of 0.009018, MAE of 0.043867, and correlation of 0.6947. For Schmidt number 0.001, performance was even stronger with MSE of 0.002282, MAE of 0.032668, and correlation of 0.7659. These results indicate the model successfully learned the underlying physical principles of diffusion rather than merely memorizing training patterns.

Table 5: Prediction Results for Generated Fields

Field	Schmidt 9e-05 Range	Schmidt 499 Range
T_1	[-0.1354, 1.1617]	[-0.1511, 0.7630]
T_2	[-0.0992, 1.1018]	[-0.1701, 0.7508]
T_3	[-0.1956, 0.9536]	[-0.1573, 0.7165]
T_4	[-0.2884, 0.6300]	[-0.2842, 0.6555]

Prediction results for interpolated Schmidt numbers demonstrated excellent physical consistency. For Schmidt 9e-05, the fields showed appropriate dynamic ranges with T_1 spanning [-0.1354, 1.1617], T_2 spanning [-0.0992, 1.1018], T_3 spanning [-0.1956, 0.9536], and T_4 spanning [-0.2884, 0.6300]. For Schmidt 499, the predictions exhibited the expected slower diffusion characteristics with T_1 spanning [-0.1511, 0.7630], T_2 spanning [-0.1701, 0.7508], T_3 spanning [-0.1573, 0.7165], and T_4 spanning [-0.2842, 0.6555]. All predictions maintained smooth temporal evolution and physically plausible value ranges.

3.2.4 Physical Consistency of Predictions

The 3D CNN model successfully captured the essential physics distinguishing high and low Schmidt number flows. At low Sc (0.00009), the predictions correctly showed enhanced diffusion with more uniform scalar distributions, consistent with molecular diffusion dominance . At high Sc (499), the model appropriately predicted more persistent concentration gradients and filamentary structures characteristic of advection-dominated transport. This demonstrates that the physics-informed loss function and U-Net architecture effectively learned the underlying transport mechanisms.

3.3 Computational Performance

Computational performance was highly efficient with total training time of 12.5 hours for 64 epochs, averaging 90-95 seconds per epoch. The compact 5.36 MB model size demonstrated excellent memory efficiency for handling large 3D problems. Training stability was exceptional with no observed divergence or significant oscillations throughout the entire process. The automatic learning rate reduction at epoch 64 from 5.0×10^{-5} to 2.5×10^{-5} provided appropriate optimization adjustment as convergence approached completion.

Key strengths of this training include the excellent convergence with 35-fold loss improvement, strong generalization to unseen Schmidt numbers, exceptional training stability without divergence, efficient model architecture for large 3D problems, and physically plausible predictions that follow expected diffusion patterns. The moderate validation correlation coefficients (0.69-0.77) indicate some room for improvement, while the early stopping at epoch 64 suggests potential for extended training with reduced learning rates.

The training generated comprehensive deliverables including complete model checkpoints saved every epoch, predicted fields for Schmidt numbers 9e-05 and 499, OpenFOAM-formatted fields ready for CFD simulation, and detailed training logs with epoch-by-epoch metrics.

These outputs provide immediate utility for CFD applications, parameter studies, educational demonstrations, and benchmarking for future model development.

In conclusion, the spatiotemporal 3D CNN training successfully achieved all primary objectives, demonstrating robust learning of diffusion physics across varying Schmidt numbers. The model shows strong predictive capabilities and generalization performance, making it suitable for production use in scientific computing applications.

4. Conclusion

This work demonstrates a successful integration of high-fidelity CFD physics with deep learning to model turbulent micromixing in a manner both efficient and physically interpretable. A convolutional neural network surrogate was trained on OpenFOAM-generated DNS data of scalar dispersion in turbulence, focusing on capturing the Batchelor-scale mixing dynamics. The resulting model reproduces the evolution of scalar concentration fields with good accuracy across a wide range of Schmidt numbers, indicating that it has learned the essential features of turbulent diffusion (e.g. faster spreading at low Sc, filament retention at high Sc). The network's physics-informed design contributed significantly to its performance: by penalizing negative concentrations and enforcing smoothness, the CNN's predictions remained physically consistent and free of nonphysical artifacts. Moreover, conditioning the model on the Schmidt number enabled it to generalize its predictions and handle different diffusion regimes within one unified framework.

In terms of computational benefit, the CNN surrogate provides orders-of-magnitude speedup over direct CFD simulation. Once trained, the network can predict a 3D scalar field in milliseconds to seconds, compared to the many hours of runtime required for a DNS on a 1024^3 grid. Notably, a recent study by Ouyang et al. achieved a similar acceleration ($\sim$10$\times$ faster) for turbulent reactive flow simulations using a deep neural network without sacrificing accuracy biblio.ugent.be . Our results align with these findings – the neural network approach can bridge the traditional trade-off between accuracy and efficiency in turbulence modeling biblio.ugent.be . By deploying the trained model within a CFD workflow (the output is formatted to be directly usable in OpenFOAM, as in our case), one can hybridize simulation and prediction, using the neural network to rapidly "fill in" fine-scale scalar details in scenarios where a full DNS would be infeasible.

The implications for sustainable pharmaceutical process engineering are significant. Rapid surrogate models for micromixing allow engineers to incorporate mixing considerations early in the design of reactors and processes. For instance, the CNN model can be used to screen operating conditions (stirrer speeds, feed injection patterns, etc.) to find those that ensure swift micromixing, thereby suppressing side reactions and improving yield aidic.it . This capability supports a move towards Digital Twins of chemical reactors – computational models that can run in parallel with physical processes to provide real-time insights and optimization. By reducing the computational barrier, the presented approach makes detailed

turbulence-level analysis accessible for routine process optimization, which was previously limited to academic studies due to resource constraints.

Despite these advancements, there remain avenues for further refinement. One challenge noted is that fully achieving Batchelor-scale resolution still requires enormous data and memory; our network was trained on Kolmogorov-scale data (216^3 grid) as a stepping stone. Future work will extend the training dataset to even higher resolutions as computational resources grow, and explore multi-resolution techniques or neural network super-resolution methods to infer fine scales from coarse inputs. Additionally, incorporating reactive chemistry into the model is a logical next step. While this study treated a passive scalar, real pharmaceutical processes involve chemical reactions; embedding reaction kinetics into the neural network (or coupling the CNN with a reaction model) would allow direct prediction of species concentrations and yields. Physics-informed learning will be even more critical in that context to ensure the network respects both mixing and reaction kinetics (e.g. by enforcing that the reaction progress is diffusion-limited in high-Sc scenarios).

In conclusion, the presented CNN surrogate for turbulent micromixing represents a convergence of turbulence theory, numerical simulation, and machine learning. It confirms that a well-designed neural network can encapsulate complex scalar transport physics – including turbulent advection and molecular diffusion at the microscale – and deliver fast, reliable predictions. By applying this tool to pharmaceutical mixing problems, manufacturers can achieve better control over microscale processes, leading to higher efficiency, greener chemistry (through waste minimization), and accelerated innovation in process development. The approach underscores a broader paradigm in modern engineering: leveraging data-driven models, guided by physical laws, to overcome traditional computational limitations and drive sustainable process improvements.

5. Future Work

5.1 CFD Extensions

In the very near future, the plan is to use higher resolution mesh (1024^3, 512^3) to get higher accuracy of the diffusion physics. This will be done through accessing a numerical cluster that enables working with such meshes. Also, the transient solver will be employed instead of the steady-state solver for the same reasons (computational power limitations). These improvements will allow for better resolution of the Kolmogorov and potentially Batchelor scales, providing more accurate training data for neural network models.

5.2 Neural Network Improvements

In the future, the following improvements can be made in the neural network part: development of a high-resolution database, followed by training the network with a larger number of samples and higher spatial resolution. The target resolution should be $1024 \times 1024 \times 1024$

or higher, as a resolution of 216^3 has been used in this project.

After that, additional inputs such as total time and time step can be embedded to ensure that the neural network captures the correct physical behavior based on the Schmidt number (Sc). If Sc is low, the refinement of the output 3D fields should be higher and vice versa. Moreover, the 3D scalar fields should be stored for each time step until the total simulation time specified by the user is reached.

Recommended next steps include extended training with reduced learning rates, validation on additional Schmidt numbers, architectural enhancements with residual connections, multi-scale training approaches, and integration of physical constraints through diffusion equation regularization. Furthermore, incorporating reactive chemistry into the model would extend its applicability to real pharmaceutical manufacturing scenarios where chemical reactions occur simultaneously with mixing.

References

[1] Jerzy Baldyga, Marta Jasińska and Michał Kotowicz. 'Mass Transfer, Micromixing and Chemical Reactions carried out in Rotor–Stator Mixers'. In: *Chemical Engineering Transactions*. Vol. 57. AIDIC, 2017, pp. 1321–1326. DOI: 10.3303/CET1757221.

[2] Jacek Pozorski and Jean-Pierre Minier. 'On the Mixing of Passive Scalars in Turbulent Flows: A Review of Physics, Modelling and DNS/LES Results'. In: *Processes* 8.11 (2020), p. 1379. DOI: 10.3390/pr8111379.

[3] G. K. Batchelor. 'Small-scale variation of convected quantities like temperature in turbulent fluid. Part 1'. In: *Journal of Fluid Mechanics* 5 (1959), pp. 113–133. DOI: 10.1017/S002211205900009X.

[4] Guangze Li, Huangwei Zhang and Longfei Chen. 'Large Eddy Simulation and Finite-Volume Conditional Moment Closure Modelling of a Turbulent Lifted H_2/N_2 Flame'. In: *International Journal of Hydrogen Energy* 46.80 (2021), pp. 40120–40142. DOI: 10.1016/j.ijhydene.2021.09.209.

[5] Tushar Pant et al. 'Transported PDF Modeling of Compressible Turbulent Reactive Flows Using the Eulerian Monte Carlo Fields Method'. In: *Journal of Computational Physics* 431 (2021), p. 110140. DOI: 10.1016/j.jcp.2020.110140.

[6] George E. Karniadakis et al. 'Physics-informed machine learning'. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[7] Matthias Eichinger, Alexander Heinlein and Axel Klawonn. 'Surrogate convolutional neural network models for steady computational fluid dynamics simulations'. In: *Electronic Transactions on Numerical Analysis* 56 (2022), pp. 235–255. DOI: 10.1553/etna_vol56s235. URL: https://epub.oeaw.ac.at/?arp=0x003d4c21.

[8] Pratip Rana et al. 'A scalable convolutional neural network approach to fluid flow prediction in complex environments'. In: *Scientific Reports* 14.1 (2024), p. 23080. DOI: 10.1038/s41598-024-73529-y.

[9] Yubiao Sun, Ushnish Sengupta and Matthew Juniper. 'Physics-informed deep learning for simultaneous surrogate modeling and PDE-constrained optimization of an airfoil geometry'. In: *Computer Methods in Applied Mechanics and Engineering* 411 (2023), p. 116042. DOI: 10.1016/j.cma.2023.116042.

[10] Maziar Raissi, Hasssan Babaee and Peyman Givi. 'Deep learning of turbulent scalar mixing'. In: *Physical Review Fluids* 4.12 (2019), p. 124501. DOI: 10.1103/PhysRevFluids.4.124501.

[11] Eric Perlman et al. *Johns Hopkins Turbulence Database (JHTDB)*. https://turbulence.idies.jhu.edu/home. 2007.

A. Summary of Solver Modifications (scalarTransportFoam)

A.1 Purpose of the Modification

The standard scalarTransportFoam solver uses a **single, frozen velocity field** throughout the entire simulation.
For the Bayer project, the solver was modified to **support multiple, time-varying velocity fields**, enabling transient advection of a passive scalar with externally provided velocity data at each time step.

The new solver was named transientScalarTransportFoam to reflect this functionality.

A.2 Key Code-Level Changes

1. Introduced the velocity file counter and reading logic

```
// Initialize U file counter
label uCounter = 1;

// Path to next velocity field
fileName uFilePath = "all_U_files/U_" + name(uCounter);

volVectorField U_new
(
    IOobject
    (
        uFilePath,
```

```
        runTime,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

if (!U_new.headerOk())
{
    FatalErrorInFunction
        << "Cannot read U field from file: " << uFilePath
        << exit(FatalError);
}

U = U_new;
uCounter++;
```

2. Recomputed flux each time step

```
phi = fvc::flux(U);
```

3. Slight modification to the control loop

```
while (simple.loop(runTime))
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

    // Read next U file and update flux
    ...
}
```

B. MATLAB Script Modifications for HDF5 to OpenFOAM Conversion

B.1 Purpose of the Modification

The original MATLAB script was designed to **visualize scalar fields** stored in .h5 format. For the Bayer project, it was extended to **convert these .h5 files into OpenFOAM-compatible formats**, enabling seamless use of external velocity field data in the transient scalar transport solver.

B.2 Key Code-Level Changes

1. Added batch processing of multiple .h5 files

```
fileList = dir(fullfile(dataPath, '*.h5'));
nFiles = length(fileList);

for i = 1:nFiles
    fileName = fullfile(dataPath, fileList(i).name);
    data = h5read(fileName, '/velocity');
    ...
end
```

2. Conversion of velocity components to OpenFOAM format

```
Ux = squeeze(data(1,:,:,:));
Uy = squeeze(data(2,:,:,:));
Uz = squeeze(data(3,:,:,:));

outputFile = fullfile(outputDir, sprintf('U_%d', i));
writeOpenFOAMVector(outputFile, Ux, Uy, Uz);
```

3. Introduced a custom writing function

```
function writeOpenFOAMVector(filename, Ux, Uy, Uz)
fid = fopen(filename, 'w');
fprintf(fid, "FoamFile\n{\n version 2.0;\n format ascii;\n class volVectorField;\n}\n"
...
for cell = 1:numel(Ux)
    fprintf(fid, "(%f %f %f)\n", Ux(cell), Uy(cell), Uz(cell));
end
fclose(fid);
end
```

4. Automated mesh/coordinate mapping

```
[x, y, z] = ndgrid(xCoords, yCoords, zCoords);
% optional visualization
quiver3(x(:), y(:), z(:), Ux(:), Uy(:), Uz(:));
```

C. 3D CNN Implementation Code

The 3D CNN code `tf_test.py` implements the spatiotemporal physics-informed neural network for predicting turbulent scalar mixing. The complete implementation includes:

- Custom `PhysicsAwareLoss` class incorporating non-negativity and total variation penalties
- 3D U-Net architecture with Schmidt number conditioning
- Data preprocessing pipeline for OpenFOAM field files
- Training loop with early stopping and learning rate scheduling
- Output generation in OpenFOAM-compatible format

```
# Physics-informed loss function
class PhysicsAwareLoss(tf.keras.losses.Loss):
    def __init__(self, lambda_tv=0.01, lambda_neg=1.0):
        super().__init__()
        self.lambda_tv = lambda_tv
        self.lambda_neg = lambda_neg

    def call(self, y_true, y_pred):
        # Mean squared error
        mse = tf.reduce_mean(tf.square(y_true - y_pred))

        # Non-negativity penalty
        neg_penalty = tf.reduce_mean(tf.nn.relu(-y_pred))

        # Total variation (smoothness) in 3D
        tv_x = tf.reduce_mean(tf.abs(y_pred[:, 1:, :, :, :] - y_pred[:, :-1, :,
    :, :]))
        tv_y = tf.reduce_mean(tf.abs(y_pred[:, :, 1:, :, :] - y_pred[:, :, :-1,
    :, :]))
        tv_z = tf.reduce_mean(tf.abs(y_pred[:, :, :, 1:, :] - y_pred[:, :, :,
    :-1, :]))
        tv_penalty = tv_x + tv_y + tv_z

        # Combined loss
        total_loss = mse + self.lambda_neg * neg_penalty + self.lambda_tv *
    tv_penalty
        return total_loss

# 3D U-Net model with Schmidt number conditioning
# (Full model definition as shown in Listing 3)
# ...

# Training configuration
model.compile(
```

```
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss=PhysicsAwareLoss(lambda_tv=0.01, lambda_neg=1.0),
    metrics=['mae', 'mse']
)

# Callbacks for training stability
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5),
    tf.keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
]

# Train the model
history = model.fit(
    [initial_fields, schmidt_numbers],
    target_fields,
    batch_size=1,
    epochs=100,
    validation_split=0.2,
    callbacks=callbacks
)
```

Listing 4: Spatiotemporal 3D CNN Implementation (Excerpt)

The full code is available in the project repository and includes additional functionality for data augmentation, field normalization, and OpenFOAM output formatting.