A Two-Phase Deep Learning Framework for Adaptive Time-Stepping in High-Speed Flow Modeling

Jacob Helwig[1] **Sai Sreeharsha Adavi**[2] **Xuan Zhang**[1] **Yuchao Lin**[1]
Felix S. Chim[2] **Luke Takeshi Vizzini**[2] **Haiyang Yu**[1] **Muhammad Hasnain**[3]
Saykat Kumar Biswas[3] **John J. Holloway**[1] **Narendra Singh**[2] **N. K. Anand**[3]
Swagnik Guhathakurta[2] **Shuiwang Ji**[1]
[1]Department of Computer Science and Engineering, Texas A&M University
[2]Department of Aerospace Engineering, Texas A&M University
[3]J. Mike Walker '66 Department of Mechanical Engineering, Texas A&M University
{jacob.a.helwig,sji}@tamu.edu

Abstract

We consider the problem of modeling high-speed flows using machine learning methods. While most prior studies focus on low-speed fluid flows in which uniform time-stepping is practical, flows approaching and exceeding the speed of sound exhibit sudden changes such as shock waves. In such cases, it is essential to use adaptive time-stepping methods to allow a temporal resolution sufficient to resolve these phenomena while simultaneously balancing computational costs. Here, we propose a two-phase machine learning method, known as ShockCast, to model high-speed flows with adaptive time-stepping. In the first phase, we propose to employ a machine learning model to predict the timestep size. In the second phase, the predicted timestep is used as an input along with the current fluid fields to advance the system state by the predicted timestep. We explore several physically-motivated components for timestep prediction and introduce timestep conditioning strategies inspired by neural ODE and Mixture of Experts. As ShockCast is the first framework for learning high-speed flows, we evaluate our methods by generating two supersonic flow datasets, available at `https://huggingface.co/divelab`. Our code is publicly available as part of the AIRS library (`https://github.com/divelab/AIRS`).

1 Introduction

Learning fluid dynamics aims to accelerate fluid modeling using machine learning models [1, 2]. Because this is an emerging area of research, most current works focus on low-speed scenarios in which flows are assumed to be incompressible. In such cases, the time scale of dynamics is relatively stable, enabling the use of time-stepping schemes with uniform step sizes without substantially affecting solution quality or the required computational effort. In contrast, time scales vary greatly for high-speed flows such that uniform time-stepping is no longer a tractable strategy. For example, supersonic flow occurs when a fluid moves faster than the local speed of sound [3, 4]. The speed of such flows is typically characterized by the Mach number M, defined as the ratio of the flow velocity v to the local speed of sound a as $M = v/a$. Flows in the supersonic regime (commonly $1 < M < 5$) exhibit distinct phenomena with small time scales, including shock waves, expansion fans, and significant compressibility effects [4]. Hypersonic flow refers to extremely high-speed fluid flows, conventionally defined by Mach numbers greater than 5. In the hypersonic regime, encountered

Preprint. Under review.

in the design of spacecraft, missiles, and atmospheric reentry vehicles, flows exhibit unique and complex behaviors such as heating, strong shock wave interactions, and chemical reactions.

For both supersonic and hypersonic flows, the time scale required to accurately resolve these phenomena is much smaller than other parts of the dynamics. Therefore, uniform time-stepping is no longer practical, as it would require the use of the smallest time scale for all steps, inflating the required computation prohibitively. Instead, high-speed flow solvers employ adaptive time-stepping schemes which dynamically adjust the timestep size such that smaller steps are taken in the presence of sharp gradients. Adaptive time-stepping can also benefit neural solvers through more balanced objectives. When using a uniform step size, the amount of evolution the model is required to learn can vary greatly between states with sharp gradients and smoother states, which is an especially pertinent consideration for high-speed flows. By instead inversely scaling the step size according to the rate of change, the difficulties of different training pairs are more evenly distributed.

However, because neural solvers achieve speedup through use of coarsened space-time meshes, classical approaches for determining the timestep size are not applicable. We therefore develop ShockCast, the first machine learning framework (to the best of our knowledge) for temporally-adaptive modeling of high-speed flows. ShockCast consists of two phases: a neural CFL phase, where the timestep is predicted, and a neural solver phase, where the flow field is evolved forward in time by the predicted timestep size. We investigate the effect of physically-motivated components in our neural CFL model, and develop several novel timestep conditioning strategies for neural solvers inspired by neural ODE [5] and Mixture of Experts [6]. To evaluate our framework, we generate two new high-speed flow datasets modeling a circular blast (maximum Mach numbers vary from 0.49 to 2.97 across cases) and a multiphase coal dust explosion (Mach numbers of the initial shock vary from 1.2 to 2.1). Our work represents the first steps towards developing machine learning models for high-speed flows, where there is great potential for neural acceleration due to the immense computational requirements of classical methods.

2 Background

In Section 2.1, we briefly introduce how PDEs are solved numerically before describing the role that the CFL condition plays in this task in Section 2.2. We then overview several prominent machine learning approaches for solving PDEs in Section 2.3.

2.1 Solving Time-Dependent Partial Differential Equations

Time-dependent PDEs are common in engineering, with some of the most prominent applications arising in fluid dynamics. In two spatial dimensions, they typically equate a first derivative in time to some operator $\mathcal{H}$ of spatial derivatives for an unknown D-dimensional solution function $\boldsymbol{u}(x, y, t) \in \mathbb{R}^D$ as

$$\partial_t \boldsymbol{u} = \mathcal{H}(\boldsymbol{u}, \partial_x \boldsymbol{u}, \partial_{xx} \boldsymbol{u}, \partial_y \boldsymbol{u}, \partial_{yy} \boldsymbol{u}, \partial_{xy} \boldsymbol{u}, \ldots), \tag{1}$$

with boundary conditions and initial conditions imposing additional constraints on $\boldsymbol{u}$. To solve a PDE, we need to identify a function $\boldsymbol{u}$ satisfying these constraints. In most real-world applications of PDE modeling, including fluid dynamics, an analytical form of this solution is intractable to obtain, and so we instead rely on producing $\boldsymbol{u}$ in numerical form, that is, obtaining point-wise evaluations on a discrete set of collocation points in space-time. This is done by evolving the PDE forward in time by first approximating the spatial derivatives on the right-hand side of Equation (1) using finite difference methods, finite volume methods, finite element methods [7], or spectral methods [8, 9, 10, 11]. Plugging these quantities into $\mathcal{H}$ gives $\partial_t \boldsymbol{u}(t)$, which is then time-integrated to advance the solution in time by a step size of Δt.

2.2 The Courant–Friedrichs–Lewy Condition

Numerical time integrators are very sensitive to the timestep size Δt. When $\boldsymbol{u}(t)$ is changing rapidly, or more formally, when $\|\partial_t \boldsymbol{u}(t)\|$ grows large, too large Δt can lead to divergence of the numerical solution [3]. In low-speed flows, the time scale does not vary drastically, which is to say that the magnitudes of temporal derivatives do not fluctuate substantially. In such cases, Δt can be chosen as a fixed value to match the smallest time scale, simplifying the numerical solution process. On the other hand, time scales vary greatly in high-speed flows. For example, shock wave interactions in

supersonic and hypersonic flows produce extremely sharp spatial gradients that can only be resolved with small timestep sizes. Following the dissipation of these phenomena, the solution can become smoother such that the time scale is substantially larger. Uniform time-stepping schemes, which must maintain a timestep small enough to resolve sharp gradients even in smooth regions, therefore impose greater computational burden compared to *adaptive time-stepping methods*, where timestep sizes are dynamically adjusted according to the sharpness of the solution's spatial gradients.

Adaptive time-stepping methods employ the Courant–Friedrichs–Lewy (CFL) Condition [12] to determine the timestep size. The CFL condition is a necessary condition on the timestep size to attain convergence of the numerical solution [13]. For a single-phase flow in two spatial dimensions and a target Courant number $C \in (0, 1)$, the CFL condition requires that

$$\Delta t \leq \frac{C}{\lambda_{\max}} \min_{x,y}(\Delta x, \Delta y), \tag{2}$$

where $\min_{x,y}(\Delta x, \Delta y)$ is the minimum cell height and width in the spatial discretization and the maximum wave speed $\lambda_{\max}$ is defined as

$$\lambda_{\max} \coloneqq \max_{x,y} \lambda(x, y) \qquad \lambda(x, y) \coloneqq \max\left(|u(x, y)| + a(x, y), |v(x, y)| + a(x, y)\right), \tag{3}$$

where u and v denote the x and y components of the velocity, and $a(x, y) \coloneqq \sqrt{\gamma RT(x, y)}$ is the local sound speed defined by the ratio of specific heats γ, the specific gas constant R and the temperature T. Intuitively, the CFL condition restricts information flow for stability such that, roughly speaking, information propagates no more than one cell in any direction per timestep, which can be seen from the scaling by the minimum cell size and inverse scaling by the wave speed in Equation (2).

2.3 Neural Solvers

As previously discussed, numerically solving PDEs is a computationally intensive process. Deep surrogate models which can accelerate the solution of PDEs are therefore of great interest. Various approaches have emerged over the last decade, including Physics-Informed Neural Networks [14, 15, 16, 17, 18] and operator learning [19, 20, 21, 22, 23, 24, 25, 26]. Neural solvers have been tailored to a variety of applications of PDE modeling, including subsurface modeling [27, 28], climate and weather modeling [29, 30, 31, 32, 33], and airfoil design [34, 35]. Neural solvers span a diverse array of architectures, including convolutional models [1, 36, 37, 38, 39, 40, 41, 42], transformers [43, 44, 45, 46, 47], and graph neural networks [48, 49, 50, 51].

Speedups over classical methods are primarily achieved by the ability of neural solvers to learn solution mappings on coarsened grids in space-time [52, 53]. To maintain stability, classical methods require computational grids to be sufficiently fine in time, as specified by the CFL condition, as well as in space. On the other hand, neural methods can learn to map between solutions spaced hundreds of classical solver steps apart on much lower-dimensional spatial discretizations, thereby realizing substantial speedups. Additionally, classical methods require that all D flow variables comprising the PDE be evolved, whereas neural solvers can learn to explicitly model only a subset of variables of interest while implicitly learning to incorporate the effect of the omitted variables.

3 Methods

3.1 Learning High-Speed Flows

High-speed flows are one of the most resource-intensive applications of PDE modeling and therefore stand to benefit greatly from the speedup offered by neural solvers. In such settings, time-adaptive meshes present a more balanced one-step objective for neural solvers. When using a uniform step size Δt, the difference between inputs and targets $\|\boldsymbol{u}(t + \Delta t) - \boldsymbol{u}(t)\|$ tends to grow with the rate of change of inputs with respect to time $\|\partial_t \boldsymbol{u}(t)\|$, which follows from the definition of a derivative $\partial_t u(t) \coloneqq \lim_{h\to 0}(u(t + h) - u(t))/h$. This implies that inputs with sharp gradients are less correlated with targets such that the mapping for sharp-gradient inputs is more difficult to learn compared to smoother ones. By instead following an adaptive scheme to inversely scale Δt according to the rate of change of $\boldsymbol{u}(t)$, the difference $\|\boldsymbol{u}(t) - \boldsymbol{u}(t + \Delta t)\|$ is more uniformly distributed across inputs with varying degrees of sharpness in gradients, thereby reducing variance in the training objective. This is a particularly relevant consideration for high-speed flows, where the sharpness of

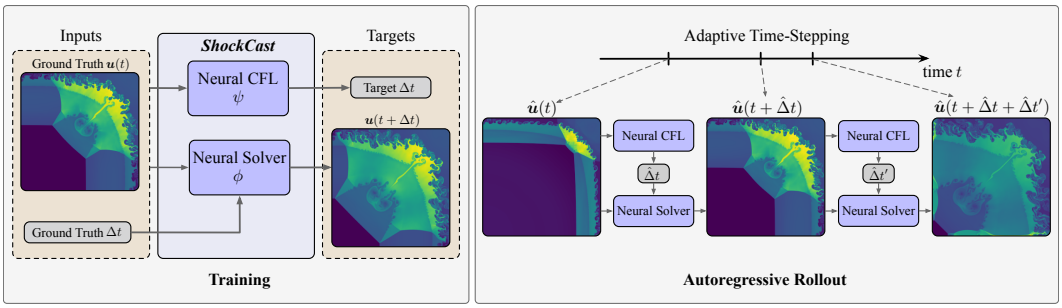

Figure 1: Overview of the ShockCast framework for time-adaptive modeling of high-speed flows. *Left*: Training pipeline. The neural CFL model and time-conditioned neural solver are conditioned on the current flow state and predict the corresponding timestep size Δt and flow state Δt ahead, respectively. *Right*: Inference pipeline. ShockCast autoregressively alternates between predicting the timestep size given the current flow state using the neural CFL model and evolving the flow state forward in time by the predicted timestep size using the neural solver model. Note that the example data are from the circular blast dataset we generated in this work.

gradients varies greatly throughout time due to phenomena such as shock waves. Furthermore, the ability to train neural solvers on time-adaptive meshes allows direct use of solutions produced from high-speed flow solvers without introducing error by interpolating to a uniform grid or modifying solver codes to save solutions at uniform timesteps.

Although well-motivated, the use of adaptive temporal meshes presents several challenges for neural solvers. Because the step size is determined by the solution, it is not known ahead of time and must instead be computed on-the-fly during autoregressive rollout. Importantly, the Δt at inference time must be aligned with those from training to avoid a test-time distribution shift. While the CFL condition is used to determine the step size for classical solvers during data generation, it cannot produce Δt matching those in the training data due to the use of coarsened computational meshes and only modeling a subset of the variables comprising the PDE. Specifically, while neural solvers are trained to advance time by hundreds of classical solver steps with a single forward pass, the CFL condition is used to compute the size of a single solver step, and will therefore suggest a step size orders of magnitude smaller than the neural solver encountered during training. Furthermore, because neural solvers learn on coarsened spatial meshes, the cell sizes Δx and Δy appearing in Equation (2) will not match those used to compute Δt in the training data. Finally, Equation (2) is the condition for a single phase flow, whereas a multiphase setting that features, for example, a solid phase interacting with a liquid phase has a much more complicated form involving a large number of field variables. Direct use of this form would require the neural solver to learn to evolve all of these variables, thereby reducing the model's capacity to capture fields of interest accurately.

These challenges motivate us to develop ShockCast, a two-phase framework consisting of a timestep-conditioned neural solver and a neural CFL model which can emulate the timestep sizes in the training data on a coarsened space-time mesh using only a subset of the field variables. At inference time, each unrolling step utilizes each of the phases in turn. In the first phase, the neural CFL model predicts the timestep size which is used by the neural solver in the second phase to evolve the current flow field forward in time by the predicted timestep. We investigate approaches for better aligning the neural CFL model with the CFL condition, and introduce several novel timestep conditioning strategies for the neural solver.

3.2 A Two-Phase Framework

Our datasets $\mathcal{D} \coloneqq \{\boldsymbol{U}_i\}_i^N$ consist of N numerical solutions to the compressible Navier-Stokes equations produced by a classical high-speed flow solver. Each solution $\boldsymbol{U} \coloneqq \{\boldsymbol{u}_j\}_j^n$ consists of a series of n snapshots on a temporal grid $\mathcal{T} \coloneqq \{t_j\}_j^n$, where $\boldsymbol{u}_j \coloneqq \boldsymbol{u}(t_j) \in \mathbb{R}^{D \times M}$ denotes the solution at time point t_j sampled on a spatial discretization with M mesh points and D fields. Notably, $\mathcal{T}$ is coarsened relative to the grid used by the classical solver by selecting every J-th solution from the solver for a coarsening factor $J \geq 100$.

For the first phase of our framework, we train a neural CFL model ψ to minimize $\mathbb{E}_{j\sim\mathcal{T},\boldsymbol{U}\sim\mathcal{D}}\left[\mathcal{L}_c\left(\psi(\boldsymbol{u}_j), \Delta_j\right)\right]$ for the loss $\mathcal{L}_c$, where we take $\mathcal{L}_c$ to be the MAE. In the second phase, we train a neural solver ϕ to map the solution at the current timestep $\boldsymbol{u}_j$ and the timestep size $\Delta_j \coloneqq t_{j+1} - t_j$ to the subsequent solution t_{j+1} by optimizing the one-step objective given by $\mathbb{E}_{j\sim\mathcal{T},\boldsymbol{U}\sim\mathcal{D}}\left[\mathcal{L}_s\left(\phi(\boldsymbol{u}_j, \Delta_j), \boldsymbol{u}_{j+1}\right)\right]$, where we take $\mathcal{L}_s$ to be the relative error averaged over fields. While the neural solver is trained to emulate the behavior of the classical solver used to generate the data, the neural CFL model is trained to emulate the process by which the timestep sizes are chosen while generating data. It is important to again emphasize that due to the use of a coarsened computational mesh and a reduced number of states being modeled, it is not possible to directly use the CFL condition to deterministically predict the timestep size. At inference time, ShockCast predicts the full solution given the initial condition $\boldsymbol{u}_0$ by alternating between the two phases autoregressively as

$$\hat{\Delta}t \coloneqq \psi(\hat{\boldsymbol{u}}) \qquad\qquad \hat{\boldsymbol{u}}(t + \hat{\Delta}t) = \phi\left(\hat{\boldsymbol{u}}(t), \hat{\Delta}t\right),$$

where $\hat{\boldsymbol{u}}(t)$ denotes the predicted state up until time t. This process, shown in Figure 1, is repeated until a pre-specified stopping time is reached.

3.3 Neural CFL

Inspired by the classical CFL condition, we experiment with several modifications to the input features and internal structure of our neural CFL model. As adaptive time-stepping schemes adjust Δt according to the sharpness of the gradients of $\boldsymbol{u}(t)$, we include the spatial gradients $\nabla\boldsymbol{u}$ computed using finite differences for all fields in $\boldsymbol{u}$ as inputs. From Equation (2), we furthermore observe the dependence of the CFL condition on the max wave speed. The max wave speed is computed by taking the maximum over the local wave speed $\lambda(x, y)$ in all computational cells. This operation can be viewed as a functional mapping λ to the scalar value $\lambda_{\max}$ via max pooling. This motivates us to employ max pooling as our spatial downsampling function. Finally, as previously discussed, the classical CFL condition is not directly applicable due to the use of mesh coarsening and modeling only a subset of the field variables. However, it is possible that the functions comprising the condition can be used to learn a surrogate condition. We therefore add *CFL features* to inputs as the local wave speed $\lambda(x, y)$, the velocity magnitudes $|u(x, y)|$ and $|v(x, y)|$, and the local sound speed $a(x, y)$.

3.4 Timestep Conditioning for Neural Solvers

We now discuss our approaches for timestep conditioning for the neural solver phase.

Time-Conditioned Layer Norm. Several prior works have considered training models to advance time by multiples of a uniform step size [54, 55]. These models have utilized *time-conditioned layer norm*, a technique originally introduced for conditioning diffusion models on the diffusion time [56, 57]. Prior to each layer, the timestep size Δt is embedded into two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with sizes matching the hidden dimension d_{model} of the feature map $\boldsymbol{z}$. These are then applied as a scale and shift following each normalization layer as $\mathrm{LN}(\boldsymbol{z})(1 + \boldsymbol{a}) + \boldsymbol{b}$.

Spatial-Spectral Conditioning. Many neural solvers perform convolutions in Fourier space and may not use normalization layers by default [1, 36]. For these models, the Spatial-Spectral conditioning strategy introduced by [54] can be used to perform timestep conditioning in the frequency domain. Under this scheme, the Fourier transform of the feature map is point-wise multiplied with a complex-valued embedding $\boldsymbol{\xi}$ of Δt as $\mathcal{F}(\boldsymbol{z})\boldsymbol{\xi}$. $\boldsymbol{\xi}$ has different entries for each frequency of $\mathcal{F}(\boldsymbol{z})$, and so to maintain parameter-efficiency, [54] share $\boldsymbol{\xi}$ across all channels of $\boldsymbol{z}$.

Euler Residuals. Neural solvers often employ residual connections [58] as $\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + F_l(\boldsymbol{z}_l)$, where the l-th solver layer F_l includes spatial integration operations such as convolution or attention. Many works have studied the relationship between residual connections and Euler integration [59, 60, 61], and even extended it to more general classes of integrators [5, 62]. For a function of time $\boldsymbol{v}$, Euler integrators approximate time integration as

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \int_t^{t+\Delta t} \partial_t \boldsymbol{v}(\tau)\mathrm{d}\tau \approx \boldsymbol{v}(t) + \Delta t \partial_t \boldsymbol{v}(t).$$

When viewing the evolution of the latent features $\boldsymbol{z}_l \mapsto \boldsymbol{z}_{l+1}$ from layer-to-layer as the evolution of some latent map $\boldsymbol{z}(t) \in \mathbb{R}^{d_{\text{model}}}$ in time, the time integration of $\boldsymbol{z}$ carried out numerically by the Euler integrator corresponds exactly to residual connections. In our scenario, we interpret the l-th layer representation $\boldsymbol{z}_l$ corresponding to the input field $\boldsymbol{u}(t)$ and timestep size Δt as the latent form of some intermediate state $\boldsymbol{u}(t + \alpha\Delta t)$, where $\alpha \in [0, 1]$ increases monotonically with depth in the network. This interpretation leads to *Euler Residuals*, in which the period of time integration executed by F_l is related to the timestep size Δt as $\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \boldsymbol{a}F_l(\boldsymbol{z}_l)$, where $\boldsymbol{a}$ is an affine transformation of Δt as $\boldsymbol{a} = \boldsymbol{W}\Delta t + \boldsymbol{c}$ for $\boldsymbol{W}, \boldsymbol{c} \in \mathbb{R}^{d_{\text{model}}}$.

Mixture of Experts. Mixture of Experts [6, 63, 64] has emerged as an effective approach to scaling up models and has been applied in PDE modeling tasks [65]. The Mixture of Experts (MoE) layer consists of a gating network G_l and K experts $F_{l,1}, \ldots, F_{l,k}$. The gating network is often a simple MLP, while the experts can have more complex architectures. Based on layer inputs $\boldsymbol{z}_l$, the gating network weighs the outputs of the experts according to the gate $G_l(\boldsymbol{z}_l) \in \mathbb{R}^K$ as $\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \sum_k^K G_l(\boldsymbol{z}_l)_k F_{l,k}(\boldsymbol{z}_l)$. The gating network can then learn to partition the latent space such that a subset of experts is relegated to specializing in particular areas. As our task involves learning to evolve dynamics by variable time lengths, we instead condition our gating network only on the timestep Δt as

$$\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \sum_k^K G_l(\Delta t)_k \left(\boldsymbol{a}_k F_{l,k}(\boldsymbol{z}_l)\right),$$

where $\boldsymbol{a}_k$ is an affine transformation of Δt for the k-th expert, and plays the same role as for the Euler residuals. This enables each layer to have experts specializing in short time integration periods where $\boldsymbol{u}(t)$ contains sharp gradients, as well as experts for handling longer timesteps where the dynamics behave more smoothly.

4 Related Work

Learnable spatial re-meshing for PDEs has been an active area of research, with advances made in supervised [66, 67, 68] and reinforcement learning [69, 70, 71] frameworks. However, to the best of our knowledge, we are the first to consider learning to temporally re-mesh and utilize data with adaptable temporal resolution, which are both vital for developing models for high-speed flows. Works in this direction have instead developed various schemes using temporally uniform data. [72] recently introduced a pipeline wherein a timestep prediction model is trained using an unsupervised loss designed to avoid timesteps that are too small. A second module then predicts temporal derivatives of various orders such that the solution can be queried using a Taylor expansion at any point up until the predicted timestep, enabling supervision using the temporally-uniform ground truth data. Following a similar continuous-time strategy, [73] propose to learn an interpolator such that arbitrary time points between temporally uniform training data can be queried. [74] employ a conditional neural field to map from initial conditions to arbitrary query times. Other works learn to map forward in time by various multiples of a uniform step size, with [75, 76] training different models for each step size and [54, 55] training one model to be shared across step sizes using time-conditioned layer norm. Importantly, [54, 55] consider the timestep size to be known *a priori*, with [54] treating this as a benchmark for probing the ability of neural solvers to respond to timestep conditioning and [55] using it as a pre-training task. As detailed in Section 3.1, this assumption is not realistic in the setting we consider here.

5 Experiments

5.1 Datasets

We consider two settings of high-speed flows in the supersonic regime. We discuss the generation of these cases in Appendix A and visualize solutions in Appendix E.

Coal Dust Explosion. The first setting we consider is a multiphase problem containing both gaseous air and granular coal particles. The simulation begins with a thin, uniform layer of coal dust settled on the bottom of a channel. Near the left boundary, we initialize a normal shock. We vary the initial

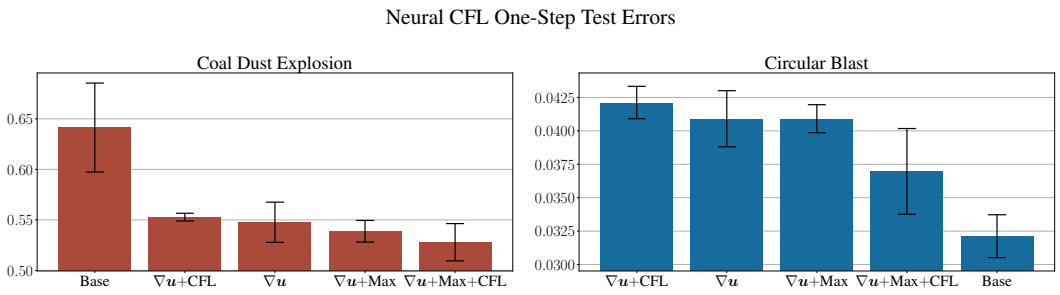

Figure 2: One-step MAE of Neural CFL models on Δt averaged over 3 training runs, where Δt is normalized to have standard deviation 1. Error bars are $\pm$ 2 standard errors.

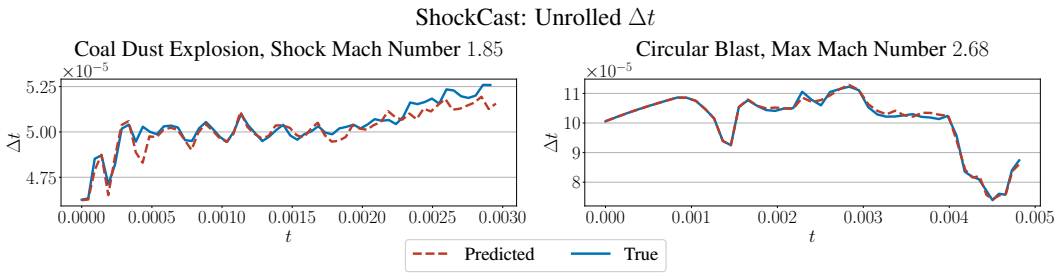

Figure 3: Δt predicted by autoregressive unrolling of ShockCast with F-FNO+Euler conditioning neural solver backbone for a selected solution.

strength of the shock between Mach 1.2 and 2.1 along with the particle diameter from case to case for a total of 100 cases, with 90 for training and 10 for evaluation. Once the simulation starts, the normal shock travels to the right as shown in Figure 7, where it interacts with the dust layer, first compressing it and later generating instabilities at the gas-dust layer interface. These instabilities further grow with time into turbulent vortical structures, which raise the dust in the channel and mix them with the air. The amount of mixing depends both on the initial shock strength and the particle diameter. We train our models to predict the velocity and temperature of the gas and the volume fraction describing the percentage of coal dust at each point.

Circular Blast. The second setting we consider is a two-dimensional circular blast case, which represents a two-dimensional version of the Sod's shock tube problem [77]. We initialize a circular region of high pressure such that the pressure inside the circle is substantially higher than its

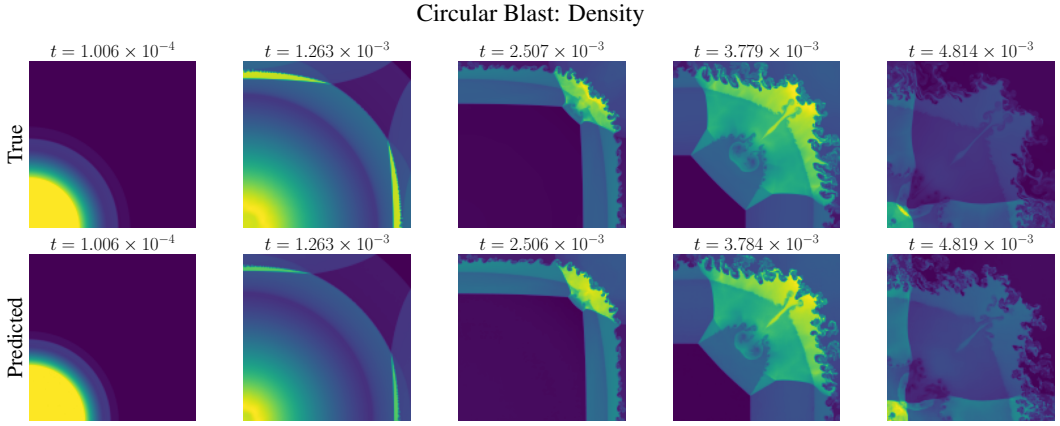

Figure 4: Comparison of the ground truth (top) and predicted (bottom) density fields for the circular blast data. We obtain predictions with autoregressive unrolling of ShockCast.

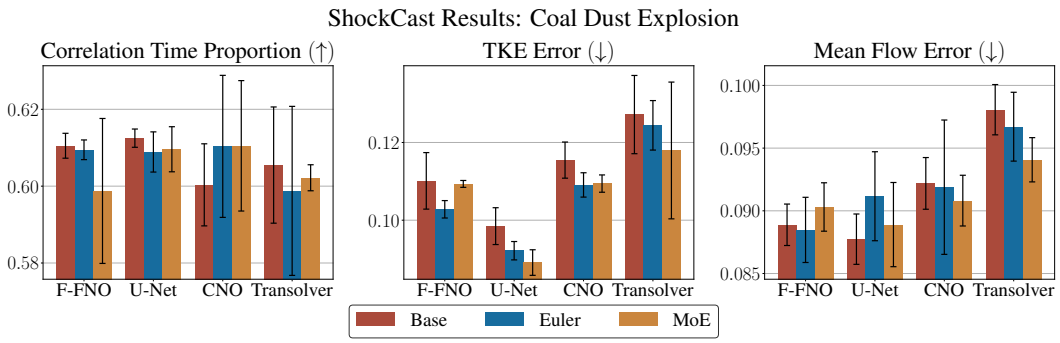

Figure 5: Coal dust explosion results averaged over three neural solver training runs with best performing neural CFL model. Error bars are $\pm$ 2 standard errors.

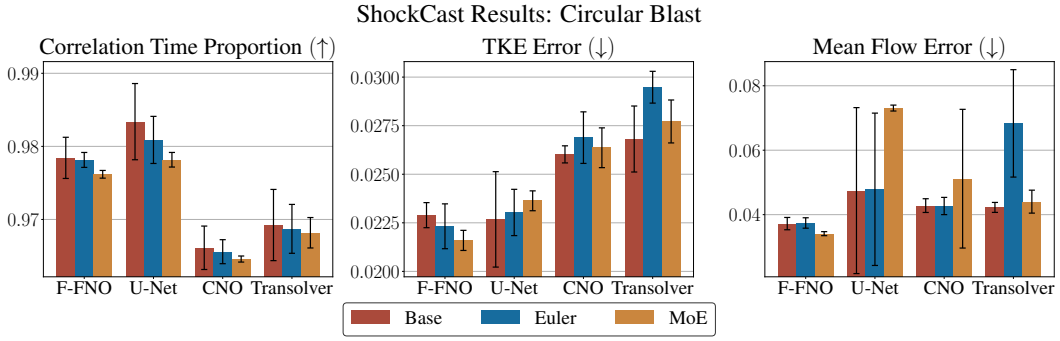

Figure 6: Circular blast results averaged over three neural solver training runs with best performing neural CFL model. Error bars are $\pm$ 2 standard errors.

surroundings as shown in Figure 8. We vary the ratio of these initial pressures from 1.99 to 50 to produce a set of 99 cases split into 90 training cases and 9 evaluation cases. Once the simulation starts, a circular shock travels radially outward, while an expansion wave travels in the opposite direction. This continues until the outward moving shock reflects from the boundaries and travels inwards toward the origin. The interaction of the reflected shocks with the post-shock gas generates instabilities which grow into turbulent structures. Once these reflected shocks reach the origin, they reflect again, thus propagating radially outward. This continues repeatedly, while with each reflection, the shocks lose strength. The maximum Mach number, which correlates with the initial pressure ratio, varied from 0.49 to 2.97 across cases. We train our models to predict the velocity, temperature, and density fields.

5.2 ShockCast Backbones

We use the ConvNeXt architecture [78] as the backbone architecture for our Neural CFL model trained with the noise injection strategy from [79] with noise level 0.01. We experiment with a variety of neural solver architectures. Multiscale processing has been highlighted as a vital component of neural solvers [54], and thus, we explore the interaction between our timestep conditioning strategies and various multiscale processing mechanisms: two hierarchical mechanisms and two global mechanisms. The first hierarchical model we consider is the U-Net [80]. We use the modern U-Net variant from [54], which closely resembles architectures used by diffusion models [81]. The Convolutional Neural Operator [42] extends the U-Net into the neural operator framework using anti-aliasing techniques from [82]. Both architectures use a hierarchical approach to processing dynamics on different scales, whereas the Fourier Neural Operator [1] employs global Fourier convolutions in the frequency domain. The variant we employ here is the Factorized FNO [36] (F-FNO), which enhances the scalability of FNO by applying convolutions one spatial dimension at a time. As an alternative to parallel multi-scale processing with Fourier convolutions, attention enables mesh points both distant and local to share information with one another. Transolver [83] reduces the quadratic complexity of attention by coarsening using a learnable soft pooling operation. For each architecture,

we experiment with each of the timestep conditioning strategies introduced in Section 3.4. We refer to the Base version of models as the one using either spatial-spectral conditioning in the case of F-FNO or time-conditioned layer norm in the case of the remaining architectures. We discuss these architectures and training procedures in greater detail in Appendices B and C, respectively.

5.3 Metrics

To evaluate predicted solutions, we compute the Pearson's correlation coefficient for each field and at each timestep with the ground truth data. This requires the predicted fields to be on the same temporal grid as the ground truth data, for which we use linear interpolation. The correlation time [53, 84, 47] for a given field is defined as the last time t before the correlation sinks below a threshold, which we take to be 0.9 here. We then average this time across all fields and report it as a percentage of the full simulation time. We additionally use the predicted fields to compute several of the primary physical quantities of interest to practitioners. The mean flow is computed by averaging each flow field over time, while the Turbulence Kinetic Energy (TKE) is calculated as the sum of the variances of the fluctuating part of the velocity field. These quantities are given by

$$\bar{\boldsymbol{u}} \coloneqq \frac{1}{T}\int_0^T \boldsymbol{u}(t)\mathrm{d}t \qquad \mathrm{TKE} \coloneqq \frac{1}{2T}\int_0^T (u(t)-\bar{u})^2 + (v(t)-\bar{v})^2 \mathrm{d}t, \tag{4}$$

respectively. We report the relative error averaged over each variable for the mean flow, and relative error of the TKE field, where the integrals in Equation (4) are approximated using the trapezoidal rule. We discuss these metrics in greater detail in Appendix D.

5.4 Results

We examine the one-step MAE of several variants of the neural CFL model in Figure 2. For the circular blast, a single-phase problem where the CFL condition is determined entirely by the velocity and temperature fields, the base model yields the best performance, as the modeled variables by themselves are sufficient to accurately predict the timestep size. In contrast, the coal dust explosion has a more complicated form of the CFL condition to account for both the solid phase describing the coal dust and the gas phase. The modeled variables primarily focus on the gas phase, with the exception of the volume fraction. In this more challenging setting, we observe substantial benefits from our physically-motivated enhancements to the neural CFL model. Our best results are achieved when using max-pooling, with the spatial gradient of the flow state $\nabla \boldsymbol{u}$ and CFL features added to inputs. In Figure 3, we examine the predicted Δt obtained through autoregressive unrolling of the ShockCast framework in each setting and observe a close match with the ground truth.

We visualize unrolled predictions on the circular blast density field in Figure 4. In Figures 5 and 6, we examine the performance of ShockCast realized with each of the neural solver architectures and timestep conditioning strategies. For both the coal dust explosion and circular blast settings, ShockCast achieves the strongest performance in terms of correlation time when leveraging a U-Net backbone with time-conditioned layer norm. On the coal dust explosion cases, MoE conditioning and Euler conditioning with a U-Net backbone achieve the first and second best performance in terms of TKE error, respectively. Similarly, the TKE error for the circular blast is lowest for Euler and MoE conditioning strategies with a F-FNO backbone. Finally, while U-Net with time-conditioned layer norm has the best mean flow error for the coal dust explosion setting, the F-FNO with MoE variant of ShockCast has the lowest circular blast mean flow error. We present extended results in Appendix F.

6 Conclusion

In this work, we develop machine learning methods for modeling high-speed flows with adaptive time-stepping. To this end, we propose ShockCast, a two-phase framework that learns timestep sizes in the first phase and evolves fluid fields by the predicted step size in the second phase. To evaluate ShockCast, we generate two new supersonic flow datasets. Results show that ShockCast is effective at learning to temporally re-mesh and evolve fluid fields. Our work represents the first steps towards developing machine learning models for high-speed flows, where there is great potential for neural acceleration due to immense computational requirements of classical methods.

Acknowledgments

This work was supported in part by the National Science Foundation under grant IIS-2243850. The authors express their gratitude to Professor Ryan Houim of the University of Florida for providing access to HyBurn, the computational fluid dynamics (CFD) code utilized for the simulations presented in this study. The CFD calculations presented in this work were partly performed on the Texas A&M high-performance computing cluster Grace.

References

[1] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[2] Xuan Zhang, Limei Wang, Jacob Helwig, Youzhi Luo, Cong Fu, Yaochen Xie, ..., and Shuiwang Ji. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023.

[3] John D. Anderson. *Fundamentals of Aerodynamics*. McGraw Hill, New York, 7th edition, 2023.

[4] John D. Anderson. *Modern Compressible Flow: With Historical Perspective*. McGraw-Hill Education, 4th edition, 2020.

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[6] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[7] Junuthula Narasimha Reddy, NK Anand, and Pratanu Roy. *Finite element and finite volume methods for heat transfer and fluid dynamics*. Cambridge University Press, 2022.

[8] David Gottlieb and Steven A Orszag. *Numerical analysis of spectral methods: theory and applications*. SIAM, 1977.

[9] David Gottlieb and Jan S Hesthaven. Spectral methods for hyperbolic problems. *Journal of Computational and Applied Mathematics*, 128(1-2):83–131, 2001.

[10] Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and Thomas A Zang. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007.

[11] David A Kopriva. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers*. Springer Science & Business Media, 2009.

[12] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

[13] Sören Bartels. *Numerical approximation of partial differential equations*, volume 64. Springer, 2016.

[14] Maziar Raissi, Paris Perdikaris, and George 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.

[15] Saykat Kumar Biswas and NK Anand. Three-dimensional laminar flow using physics informed deep neural networks. *Physics of Fluids*, 35(12), 2023.

[16] Saykat Kumar Biswas and NK Anand. Interfacial conditioning in physics informed neural networks. *Physics of Fluids*, 36(7), 2024.

[17] Woojin Cho, Kookjin Lee, Donsub Rim, and Noseong Park. Hypernetwork-based meta-learning for low-rank physics-informed neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Smruti Shah and NK Anand. Physics-informed neural networks for periodic flows. *Physics of Fluids*, 36(7), 2024.

[19] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[20] Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[21] Zongyi Li, Miguel Liu-Schiaffini, Nikola Borislavov Kovachki, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Learning dissipative dynamics in chaotic systems. In *Advances in Neural Information Processing Systems*, 2022.

[22] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

[23] Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3d pdes. *arXiv preprint arXiv:2309.00583*, 2023.

[24] Michael Poli, Stefano Massaroli, Federico Berto, Jinkyoo Park, Tri Dao, Christopher Re, and Stefano Ermon. Transform once: Efficient operator learning in frequency domain. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[25] Jacob Seidman, Georgios Kissas, Paris Perdikaris, and George J Pappas. Nomad: Nonlinear manifold decoders for operator learning. *Advances in Neural Information Processing Systems*, 35:5601–5613, 2022.

[26] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[27] Chengyuan Deng, Shihang Feng, Hanchen Wang, Xitong Zhang, Peng Jin, Yinan Feng, Qili Zeng, Yinpeng Chen, and Youzuo Lin. Openfwi: Large-scale multi-structural benchmark datasets for full waveform inversion. *Advances in Neural Information Processing Systems*, 35:6007–6020, 2022.

[28] Tailin Wu, Qinchen Wang, Yinan Zhang, Rex Ying, Kaidi Cao, Rok Sosic, Ridwan Jalali, Hassan Hamam, Marko Maucec, and Jure Leskovec. Learning large-scale subsurface simulations with a hybrid graph network simulator. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4184–4194, 2022.

[29] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *arXiv preprint arXiv:2211.02556*, 2022.

[30] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, et al. Graphcast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

[31] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[32] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Timo Ewalds, Andrew El-Kadi, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, Remi Lam, and Matthew Willson. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint arXiv:2312.15796*, 2023.

[33] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. Climax: A foundation model for weather and climate. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[34] Florent Bonnet, Jocelyn Ahmed Mazari, Paola Cinella, and Patrick Gallinari. Airfrans: High fidelity computational fluid dynamics dataset for approximating reynolds-averaged navier-stokes solutions. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022.

[35] Jacob Helwig, Xuan Zhang, Haiyang Yu, and Shuiwang Ji. A geometry-aware message passing neural network for modeling aerodynamics over airfoils. *arXiv preprint arXiv:2412.09399*, 2024.

[36] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023.

[37] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[38] Jacob Helwig, Xuan Zhang, Cong Fu, Jerry Kurtin, Stephan Wojtowytsch, and Shuiwang Ji. Group equivariant Fourier neural operators for partial differential equations. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[39] Gege Wen, Zongyi Li, Qirui Long, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. Real-time high-resolution co 2 geological storage prediction using nested fourier neural operators. *Energy & Environmental Science*, 16(4):1732–1741, 2023.

[40] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[41] Xuan Zhang, Jacob Helwig, Yuchao Lin, Yaochen Xie, Cong Fu, Stephan Wojtowytsch, and Shuiwang Ji. Sinenet: Learning temporal dynamics in time-dependent partial differential equations. In *The Twelfth International Conference on Learning Representations*, 2024.

[42] Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of pdes. *Advances in Neural Information Processing Systems*, 36, 2024.

[43] Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

[44] Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023.

[45] Steeven Janny, Aurélien Bénéteau, Madiha Nadri, Julie Digne, Nicolas Thome, and Christian Wolf. EAGLE: Large-scale learning of turbulent fluid dynamics with mesh transformers. In *International Conference on Learning Representations*, 2023.

[46] Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: Auto-regressive denoising operator transformer for large-scale PDE pre-training. In *Forty-first International Conference on Machine Learning*, 2024.

[47] Benedikt Alkin, Andreas Fürst, Simon Lucas Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[48] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[49] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[50] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[51] Masanobu Horie and Naoto Mitsume. Physics-embedded neural networks: Graph neural pde solvers with mixed boundary conditions. *Advances in Neural Information Processing Systems*, 35:23218–23229, 2022.

[52] Kimberly Stachenfeld, Drummond B Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. In *International Conference on Learning Representations*, 2021.

[53] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[54] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized PDE modeling. *Transactions on Machine Learning Research*, 2023.

[55] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *arXiv preprint arXiv:2405.19101*, 2024.

[56] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

[57] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[59] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International conference on machine learning*, pages 3276–3285. PMLR, 2018.

[60] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017.

[61] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020.

[62] Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[63] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[64] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[65] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

[66] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[67] Wenbin Song, Mingrui Zhang, Joseph G Wallwork, Junpeng Gao, Zheng Tian, Fanglei Sun, Matthew Piggott, Junqing Chen, Zuoqiang Shi, Xiang Chen, et al. M2n: Mesh movement networks for pde solvers. *Advances in Neural Information Processing Systems*, 35:7199–7210, 2022.

[68] Mingrui Zhang, Chunyang Wang, Stephan C Kramer, Joseph G Wallwork, Siyi Li, Jiancheng Liu, Xiang Chen, and Matthew Piggott. Towards universal mesh movement networks. *Advances in Neural Information Processing Systems*, 37:14934–14961, 2024.

[69] Tailin Wu, Takashi Maruyama, Qingqing Zhao, Gordon Wetzstein, and Jure Leskovec. Learning controllable adaptive simulation for multi-scale physics. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

[70] Niklas Freymuth, Philipp Dahlinger, Tobias Daniel Würth, Simon Reisch, Luise Kärger, and Gerhard Neumann. Swarm reinforcement learning for adaptive mesh refinement. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[71] Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, et al. Reinforcement learning for adaptive mesh refinement. In *International conference on artificial intelligence and statistics*, pages 5997–6014. PMLR, 2023.

[72] Zhikai Wu, Shiyang Zhang, Sizhuang He, Sifan Wang, Min Zhu, Anran Jiao, Lu Lu, and David van Dijk. Tante: Time-adaptive operator learning via neural taylor expansion. *arXiv preprint arXiv:2502.08574*, 2025.

[73] Steeven Janny, Madiha Nadri, Julie Digne, and Christian Wolf. Space and time continuous physics simulation from partial observations. In *The Twelfth International Conference on Learning Representations*, 2024.

[74] Jan Hagnberger, Marimuthu Kalimuthu, Daniel Musekamp, and Mathias Niepert. Vectorized conditional neural fields: A framework for solving time-dependent parametric partial differential equations. In *Forty-first International Conference on Machine Learning*, 2024.

[75] Yuying Liu, J Nathan Kutz, and Steven L Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A*, 380(2229):20210200, 2022.

[76] Asif Hamid, Danish Rafiq, Shahkar Ahmad Nahvi, and Mohammad Abid Bazaz. Hierarchical deep learning-based adaptive time stepping scheme for multiscale simulations. *Engineering Applications of Artificial Intelligence*, 133:108430, 2024.

[77] Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of computational physics*, 27(1):1–31, 1978.

[78] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[79] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[81] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[82] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in neural information processing systems*, 34:852–863, 2021.

[83] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 53681–53705. PMLR, 21–27 Jul 2024.

[84] Phillip Lippe, Bastiaan S Veeling, Paris Perdikaris, Richard E Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. *arXiv preprint arXiv:2308.05732*, 2023.

[85] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. AMReX: A Framework for Block-Structured Adaptive Mesh Refinement. *Journal of Open Source Software*, 4(37):1370, 2019.

[86] Ryan W Houim and Kenneth K Kuo. A low-dissipation and time-accurate method for compressible multi-component flow with variable specific heat ratios. *Journal of Computational Physics*, 230(23):8527–8553, 2011.

[87] Ryan W Houim and Elaine S Oran. A multiphase model for compressible granular–gaseous flows: formulation and initial tests. *Journal of fluid mechanics*, 789:166–220, 2016.

[88] Dinshaw S Balsara and Chi-Wang Shu. Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy. *Journal of Computational Physics*, 160(2):405–452, 2000.

[89] M Pino Martín, Ellen M Taylor, Minwei Wu, and V Gregory Weirs. A bandwidth-optimized weno scheme for the effective direct numerical simulation of compressible turbulence. *Journal of Computational Physics*, 220(1):270–289, 2006.

[90] Zhijun Shen, Wei Yan, and Guangwei Yuan. A robust hllc-type riemann solver for strong shock. *Journal of Computational Physics*, 309:185–206, 2016.

[91] Amiram Harten, Peter D Lax, and Bram van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. *SIAM review*, 25(1):35–61, 1983.

[92] Fernando F Grinstein, Len G Margolin, and William J Rider. *Implicit large eddy simulation*, volume 10. Cambridge university press Cambridge, 2007.

[93] Elaine S Oran, Jay P Boris, and Jay P Boris. *Numerical simulation of reactive flow*, volume 2. Citeseer, 2001.

[94] Ben Thornber, Andrew Mosedale, Dimitris Drikakis, David Youngs, and Robin JR Williams. An improved reconstruction method for compressible flows with low mach number features. *Journal of computational Physics*, 227(10):4873–4894, 2008.

[95] Dimitris Drikakis, Marco Hahn, Andrew Mosedale, and Ben Thornber. Large eddy simulation using high-resolution and high-order methods. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1899):2985–2997, 2009.

[96] Ben Thornber, Andrew Mosedale, and Dimitris Drikakis. On the implicit large eddy simulations of homogeneous decaying turbulence. *Journal of Computational Physics*, 226(2):1902–1929, 2007.

[97] Swagnik Guhathakurta and Ryan W Houim. Impact of particle diameter and thermal radiation on the explosion of dust layers. *Proceedings of the Combustion Institute*, 39(3):2905–2914, 2023.

[98] Swagnik Guhathakurta and Ryan W Houim. Influence of thermal radiation on layered dust explosions. *Journal of Loss Prevention in the Process Industries*, 72:104509, 2021.

[99] Swagnik Guhathakurta. *Effect of Radiative Heat Transfer on the Structure and Propagation of Layered Coal-Dust Explosions*. PhD thesis, University of Florida, 2021.

[100] Swagnik Guhathakurta and Ryan W Houim. Propagation and severity of coal-dust explosions and the effect of radiation in different channel lengths. In *Proceedings of the 29th international colloquium on the dynamics of explosions and reactive systems*, 2023.

[101] Jacob W Posey, Brayden Roque, Swagnik Guhathakurta, and Ryan W Houim. Mechanisms of prompt and delayed ignition and combustion of explosively dispersed aluminum powder. *Physics of Fluids*, 33(11), 2021.

[102] Hsiao-Chi Li and Ryan W Houim. Pore-scale resolved simulation of quenching, acceleration, and transition to detonation of hydrogen explosions by metal foams. *Combustion and Flame*, 259:113118, 2024.

[103] Hsiao-Chi Li. *Flame Quenching or Accelerating by Metal Foam in a Square Channel Using Immersed Boundary Method*. University of Florida, 2022.

[104] Joshua W Hargis, Anthony Egeln, Ryan Houim, and Daniel R Guildenbecher. Visualization of post-detonation fireball flowfields and comparison to cfd modeling. *Proceedings of the Combustion Institute*, 40(1-4):105230, 2024.

[105] Anthony A Egeln, John C Hewson, Daniel R Guildenbecher, Ryan T Marinis, Marc C Welliver, and Ryan W Houim. Post-detonation fireball modeling: Validation of freeze out approximations. *Physics of Fluids*, 35(6), 2023.

[106] T Farrukh, R Houim, D Guildenbecher, M Welliver, and S Balachandar. Particle and fluid time scales in a spherical multiphase blast flow. *Shock Waves*, pages 1–19, 2025.

[107] Eleuterio F Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media, 2013.

[108] Bram Van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of computational Physics*, 32(1):101–136, 1979.

[109] Nico Fleischmann, Stefan Adami, and Nikolaus A Adams. A shock-stable modification of the hllc riemann solver with reduced numerical dissipation. *Journal of computational physics*, 423:109762, 2020.

[110] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[111] William A Falcon. Pytorch lightning. *GitHub*, 3, 2019.

[112] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[113] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Appendix

A CFD Problem Description and Numerical Models

We conduct the numerical simulations in this work using the HyBurn code, which implements high-order Godunov schemes to solve the governing equations. HyBurn is a finite volume solver that employs parallelization and adaptive mesh refinement (AMR) via the AMReX library [85]. It uses a low-dissipation, WENO-based high-order Godunov method [86, 87, 88, 89, 90, 91] and models turbulence using Implicit Large Eddy Simulation [92, 93, 94, 95, 96]. HyBurn solves compressible Navier-Stokes equations for Eulerian-Eulerian granular multiphase reactive flows, employing advanced numerical techniques developed by [87]. It uses an explicit third-order three-stage Runge-Kutta time-stepping algorithm along with two independent CFL numbers – one for the hyperbolic and the other for the parabolic terms of the compressible Navier-Stokes equations. The actual timestep size is determined by whichever is the smaller of the two. HyBurn has been extensively validated and verified against various compressible multiphase and reactive flow problems [86, 87]. More details about the algorithms and the applications of HyBurn can be found in previous work [97, 98, 99, 100, 101, 102, 103, 104, 105, 106].

A.1 Coal Dust Explosion

The first setting we consider is loosely based on the coal dust explosion simulations in [97, 98, 99, 100]. The major differences are that in this study we use a normal shock instead of a detonation to mimic the primary explosion in a coal mine, and there are no chemical reactions involved. The computational geometry we use is a $25\,\mathrm{cm}$ by $5\,\mathrm{cm}$ two-dimensional rectangular channel containing air, with a thin uniform layer of coal dust settled on the bottom of the channel. We keep the left and right boundaries open, while the top and bottom boundaries are symmetry. This is a multiphase problem containing both gaseous air and granular coal particles. Near the left boundary, we initialize a normal shock using post-shock conditions to the left of the shock and quiescent air ($300\,\mathrm{K}$ temperature, $1\ \mathrm{atm}$ pressure) ahead of it. We vary the initial strength of the shock between Mach 1.2 and 2.1 along with the particle diameter between $1\,\mu\mathrm{m}$ and $150\,\mu\mathrm{m}$ from case to case for a total of 100 cases. We set the initial volume fraction of the dust layer to 47% and the particles to be monodisperse. We model the coal particles as being composed of inert ash only, with a solid-phase density of $1330\,\mathrm{kg\,m^{-3}}$.

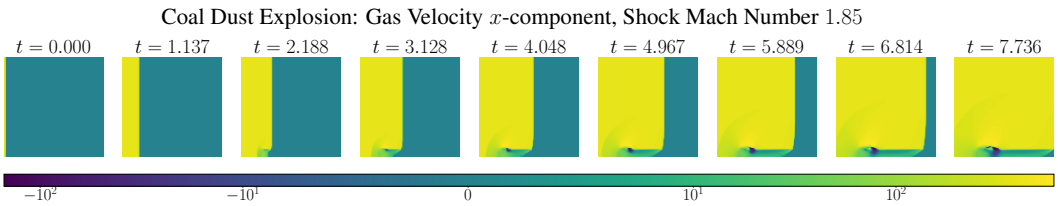

Figure 7: Initial gas velocity x-component for a selected coal dust explosion case. Times are in units of 10^{-5} seconds and the downsampling factor relative to the classical solver solution is $100\times$ compared to $500\times$ used for training ShockCast. The initial shock can be seen to be moving from left to right.

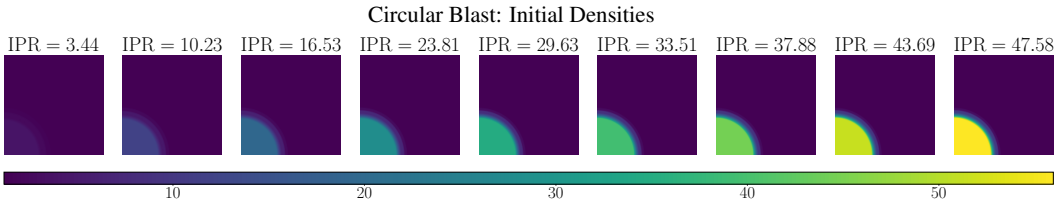

Figure 8: Initial density field for the circular blast evaluation cases with varying Initial Pressure Ratios (IPR). Due to the ideal gas law, increasing the pressure also increases the density.

We use two AMR levels to give an effective resolution of $\sim 0.12\,\mathrm{mm}$ at the finest level. We use a fifth-order WENO interpolation scheme and the HLL [107] flux reconstruction method. We set the CFL numbers for both the hyperbolic and parabolic terms to 0.6.

Once the simulation starts, the normal shock travels to the right as shown in Figure 7, where it interacts with the dust layer, first compressing it and later generating instabilities at the gas-dust layer interface. These instabilities further grow with time into turbulent vortical structures, which raise the dust in the channel and mix them with the air. The amount of mixing depends both on the initial shock strength and the particle diameter. We cap the total simulation time at $3\,\mathrm{ms}$ for all cases.

A.2 Circular Blast

The second setting we consider is a two-dimensional circular blast case, which represents a two-dimensional version of the Sod's shock tube problem [77]. We initialize a circular region of high pressure such that the pressure inside the circle is substantially higher than its surroundings as shown in Figure 8. We vary the ratio of these initial pressures from 1.99 to 50 to produce a set of 99 cases. To reduce the computational cost, we only simulate a quarter of the circle. We use symmetry boundary conditions for all boundaries to allow for the generated shocks and expansion waves to reflect when incident on them. We model a single gaseous phase throughout the computational domain. We set the initial temperatures to $300\,\mathrm{K}$ everywhere, with the gas initially at rest. We use two Adaptive Mesh Refinement (AMR) levels – triggered by predefined pressure and density ratio thresholds between any two adjacent computational cells – to give an effective resolution of $\sim 0.98\,\mathrm{mm}$ at the finest level, which is sufficient to resolve the shocks. We use a fifth-order MUSCL [108] interpolation scheme and the HLLC-LM [109] flux reconstruction method. We set the CFL numbers for both the hyperbolic and parabolic terms to 0.8.

Once the simulation starts, a circular shock travels radially outward, while an expansion wave travels in the opposite direction. This continues until the outward moving shock reflects from the boundaries and travels inwards toward the origin. The interaction of the reflected shocks with the post-shock gas generates instabilities which grow into turbulent structures. Once these reflected shocks reach the origin, they reflect again, thus propagating radially outward. This continues repeatedly, while with each reflection, the shocks lose strength. We cap the total simulation time at $5\,\mathrm{ms}$ for all cases.

Model	Initial Learning Rate	Latent Dimension	Key Hyperparameters
U-Net	2×10^{-4}	64	3 downsampling/upsampling levels
CNO	2×10^{-4}	27	4 downsampling/upsampling levels, 6 residual blocks per level
F-FNO	1×10^{-3}	96	32 modes, 12 layers
Transolver	6×10^{-4}	192	8 layers, 8 attention heads, 8 slices
ConvNeXT	2×10^{-4}	96	ConvNeXt-T (See Section 3 of [78])

Table 1: Hyperparameters for neural solver and neural CFL models used in both the coal dust explosion setting and the circular blast setting.

Model	Parameters (M)	GigaFLOPs	Peak Train Memory (GiB)
U-Net			
Base	140.861	94.328	13.465
Euler	140.861	94.328	14.152
MoE	131.129	85.839	21.307
CNO			
Base	45.710	16.028	8.951
Euler	45.704	16.028	9.629
MoE	174.339	55.917	27.783
F-FNO			
Base	15.372	20.901	18.953
Euler	15.374	20.901	20.164
MoE	15.697	20.762	37.242
Transolver			
Base	11.139	191.112	41.895
Euler	10.336	191.110	41.881
MoE	9.911	182.924	62.432
ConvNeXT	27.822	1.736	11.117

Table 2: Model parameter counts, GigaFLOPs per forward pass, and peak GPU memory usage during training. FLOPs were computed with a batch size of 1 on the coal dust explosion dataset, while GPU memory was computed for a batch size of 32 on a single A100 GPU.

B Neural Solvers

Previous works have highlighted the importance of multiscale processing mechanisms in designing effective neural solvers [54]. This is at least in part due to the temporal coarsening approach taken by neural solvers. While differential operators defining PDEs are primarily local in nature, their effects become increasingly global as the timestep size is increased beyond that of the classical solver. It is therefore vital for neural solvers to incorporate spatial processing mechanisms that enable modeling of phenomena on both local and global scales. Thus, we explore a variety of neural solver backbones in ShockCast spanning both hierarchical and parallel multi-scale processing mechanisms. Within the parallel framework, we consider both convolution-based and attention-based mechanisms.

U-Net. The U-Net [80] is one of the most prominent examples of a hierarchical mechanism. The U-Net is composed of a downsampling path and an upsampling path. To achieve a global receptive field, the input features sampled on the original solution mesh are first sequentially downsampled using pooling operations or strided convolutions. At each resolution, the downsampled features are convolved and point-wise activations are applied. Following the downsampling path, an inverse upsampling path is applied, consisting of upsampling operations which are either transposed con-

volutions or interpolation, with convolution and non-linearities again applied at each resolution. To restore high-frequency details that were lost along the downsampling path, skip connections from respective resolutions along the downsampling path concatenate feature maps at each stage of the upsampling path. The U-Net architecture we employ in our framework is the "modern U-Net" architecture from [54], which closely resembles architectures used by diffusion models [81].

CNO. The Convolutional Neural Operator [42] adapts the U-Net into the neural operator framework. Neural operators [26, 25] are a class of neural solver which aim to maintain the continuous nature of the underlying PDE solution despite the discretization of training data. In doing so, they enable trained neural solvers to be evaluated on discretizations differing from the training data. CNO extends these properties to the U-Net architecture using anti-aliasing techniques from [82].

F-FNO. While the previous architectures take a hierarchical approach to multi-scale processing, Fourier Neural Operators [1] process information on multiple scales in parallel using global Fourier convolutions. Convolution kernels are parameterized in the frequency domain such that the complex-valued weights to be learned represent the coefficients of the kernel function in the Fourier basis. Due to the convolution theorem, convolutions in the frequency domain are carried out via point-wise multiplication of frequency modes. Furthermore, because Fourier basis functions have global support, with high frequency modes describing fine details and low frequencies describing the "background" of the function, information on multiple spatial scales is processed in parallel. To execute these convolutions more efficiently, FNOs truncate the number of non-zero modes in each kernel to a threshold such that only the lowest frequencies are present in the Fourier expansion of the kernel function. Building on this efficiency, Factorized Fourier Neural Operators [36] perform convolutions one spatial dimension at a time such that kernels are a function of only one spatial dimension. This enables deeper F-FNOs, enhancing the expressive capacity of the architecture.

Transolver. As an alternative to parallel multi-scale processing with Fourier convolutions, attention enables mesh points both distant and local to share information with one another. However, due to its quadratic complexity, adapting Transformers to PDE modeling tasks, where the number of mesh points can be on the order of thousands and above, presents computational challenges [44, 43, 65]. Transolver [83] reduces this complexity by performing attention on a coarsened mesh. The coarsening is achieved by a learnable soft pooling operation, where the soft assignments are not entirely based on clustering local points together.

C Training Details

C.1 Datasets

We coarsen the coal dust explosion cases in time by saving every 100 steps, and applied further coarsening to the saved steps by a factor of 5 for an overall coarsening factor of $500\times$ relative to the CFD solver. The coarsest AMR level gave a spatial resolution of 104×520. We only use the left-most fifth of the domain along the horizontal axis such that the training resolution was 104×104. We train models on the velocity and temperature fields of the gas, as well as the volume fraction describing the percentage of coal comprising each computational cell.

For the circular blast cases, we save solutions every 100 CFD steps. The coarsest AMR level for the circular blast setting gives a spatial resolution of 256×256, which we coarsened further to the training resolution of 128×128 using averaging. We train models on the velocity, temperature and density fields.

C.2 Training Pipeline

We implement our training pipeline in PyTorch [110] using PyTorch Lightning [111]. Depending on model training memory requirements, we train models on between 1-2 80 GiB A100 GPUs or between 1-8 11 GiB RTX 2080 GPUs. We optimize all models using the Adam optimizer [112] with a cosine learning rate scheduler [113]. We use a batch size of 32 for all neural solver models, and a batch size of 320 for neural CFL models. We train neural solver models for 400 epochs, resulting in over 75K training updates for the coal dust explosion dataset and over 50K training updates for the circular blast dataset. We train neural CFL models for 800 epochs using training noise with a level of

0.01 [79]. On the coal dust explosion dataset, this results in over 12K training updates, while for the circular blast dataset, this results in over 6K training updates. We present initial learning rates and other key hyperparameters used for all models in Table 1.

C.3 Parameter Counts, FLOPs and Peak GPU Memory

In Table 2, we present parameter counts, forward FLOPs, and peak training memory for the coal dust explosion cases. We compute FLOPs with a batch size of one using the `FlopCounterMode` module from `torch.utils.flop_counter` module, and report training memory observed on a single A100 GPU using a batch size of 32.

In our experiments, we offset increased computation when using the MoE timestep conditioning strategy by reducing the latent dimension of all models except CNO according to the square root of the number of experts used. We use four experts for all models except for Transolver, where we use two experts due to high memory consumption. For CNO, we ran into stability issues when attempting to scale the number of parameters beyond those reported in [55]. As can be seen in Table 2, this led to a reduced amount of computation for the CNO variants of ShockCast relative to the other neural solver backbones. However, when using MoE timestep conditioning, we found that we could stably train CNO with 4 experts with no reduction in the embedding dimension.

D Metrics

Correlation time proportion. We compute Pearson's correlation coefficient for each field and at each timestep with the ground truth data. This requires the predicted fields to be on the same temporal grid as the ground truth data, for which we use linear interpolation. The correlation time [53, 84, 47] for a given field is defined as the last time t before the correlation sinks below a threshold, which we take to be 0.9 here. We then average this time across all fields and report it as a proportion of the full simulation time such that a perfect prediction would have correlation time proportion of 1.

Mean flow and Turbulence Kinetic Energy. The mean flow is defined as the flow states averaged over time, while the turbulence kinetic energy is the sum of the variances of the fluctuating part of the velocity field. These quantities are given by

$$\bar{\boldsymbol{u}} \coloneqq \frac{1}{T}\int_0^T \boldsymbol{u}(t)\mathrm{d}t \qquad \mathrm{TKE} \coloneqq \frac{1}{2T}\int_0^T (u(t)-\bar{u})^2 + (v(t)-\bar{v})^2 \mathrm{d}t. \tag{5}$$

We approximate the integrals in Equation (5) using the trapezoidal rule as

$$\int_0^T f(t)\mathrm{d}t \approx \frac{1}{2}\sum_j^{n-1} \Delta_j \left(f(t_{j+1}) + f(t_j)\right),$$

where $\Delta_j \coloneqq t_{j+1} - t_j$.

E Solution Visualizations

Here, we visualize true and predicted fields for ShockCast using the F-FNO with Euler conditioning neural solver backbone on a selected solution from the evaluation datasets. For the coal dust explosion setting, this solution has shock Mach number of 1.85, while for the circular blast setting, it has a max Mach number of 2.68. We visualize TKE fields in Figures 9 and 10 and mean flow fields in Figures 11 and 12. When visualizing the instantaneous fields, we subsample the true solution in time to reduce the number of snapshots. For each of the snapshots $\boldsymbol{u}(t)$ in the subsampled solution, we find the snapshot from the solution $\hat{\boldsymbol{u}}(\hat{t})$ autoregressively unrolled by ShockCast with the closest predicted time $\hat{t}$ to t. We then visualize the true snapshots $\boldsymbol{u}(t)$ alongside the corresponding closest-in-time predicted snapshots $\hat{\boldsymbol{u}}(\hat{t})$, as well as the residual between each pair $|\boldsymbol{u}(t) - \hat{\boldsymbol{u}}(\hat{t})|$. We present these visualizations in Figures 13 to 16 for each of the fields in the coal dust explosion case and in Figures 17 to 20 for each of the fields in the circular blast case.

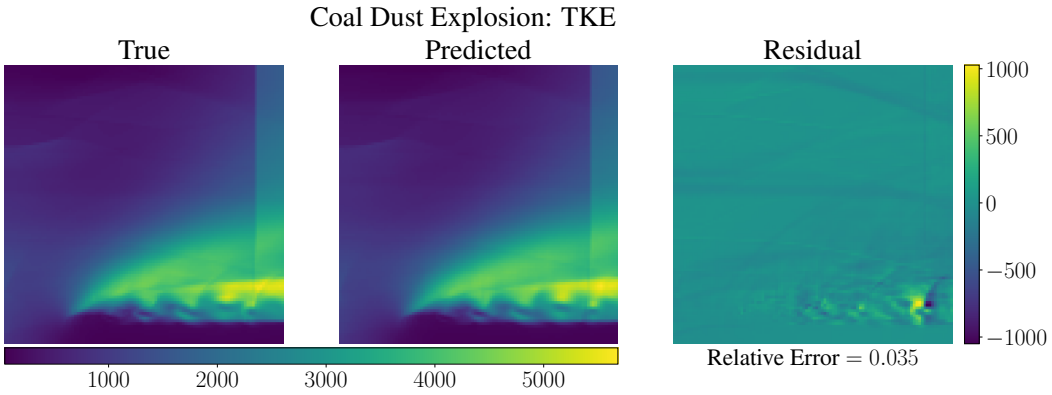

Figure 9: TKE for coal dust explosion.

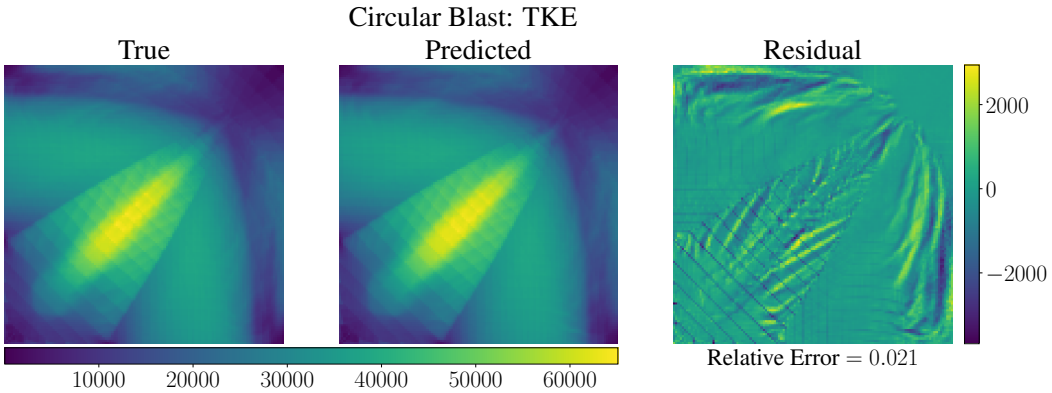

Figure 10: TKE for circular blast.

F Extended Results

In this section, we present the numerical values of average evaluation errors and their corresponding standard errors as *mean (standard error)*. In Tables 3 and 4, we present one-step errors for ShockCast. We note that the timestep predicted by the neural CFL model will not perfectly match the ground truth timestep such that the prediction from the neural solver model will be for a time which differs from the ground truth. To compute the unrolled errors in Tables 5 and 6 and correlation time proportions shown in Tables 7 and 8, we linearly interpolate ShockCast predictions in time to be sampled on the same temporal grid as the ground truth data. As can be seen in Figure 15, the volume fraction field at later timesteps can be sparse, and so we clamp the norm of the ground truth field in the denominator of the relative error to have a minimum value of 1. For the mean flow results, which we show in Tables 9 and 10, and TKE results that we present in Table 11, the target quantities involve integrating the instantaneous fields with respect to time, and thus, no interpolation is required.

G Limitations and Future Directions

As previously discussed, neural solvers can benefit from time-adaptive schemes, as varying the timestep size according to the rate of change can lead to more balanced one-step objectives across flow states with varying gradient sharpness. Here, we have supervised our neural CFL model using timesteps resulting from coarsening a temporal mesh computed using the CFL condition. However, approaches that learn to adapt timestep sizes based on a policy that balances solution accuracy with computational cost, as is done by [69] for spatial remeshing, may lead to further improvements.

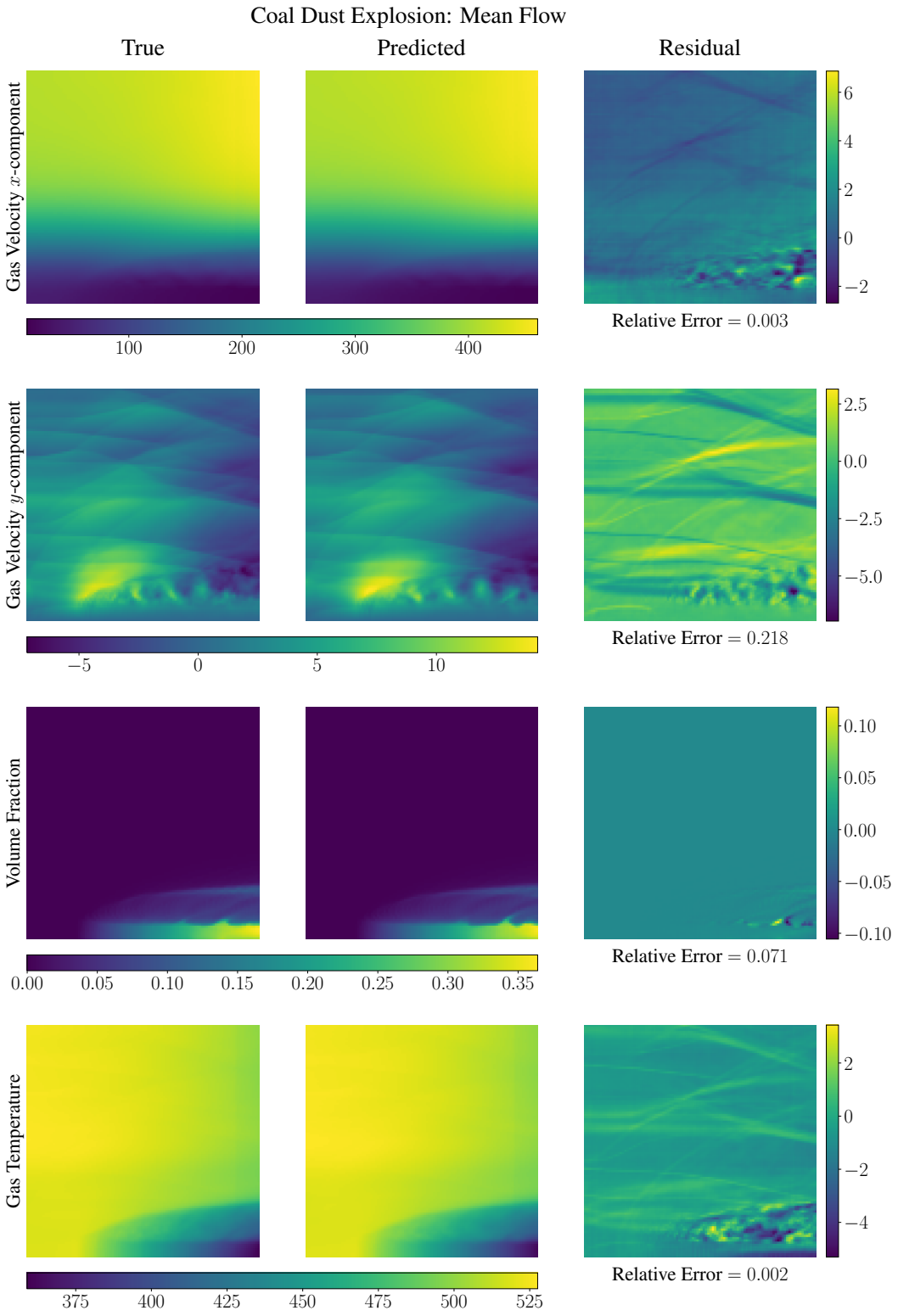

Figure 11: Mean flow for coal dust explosion.

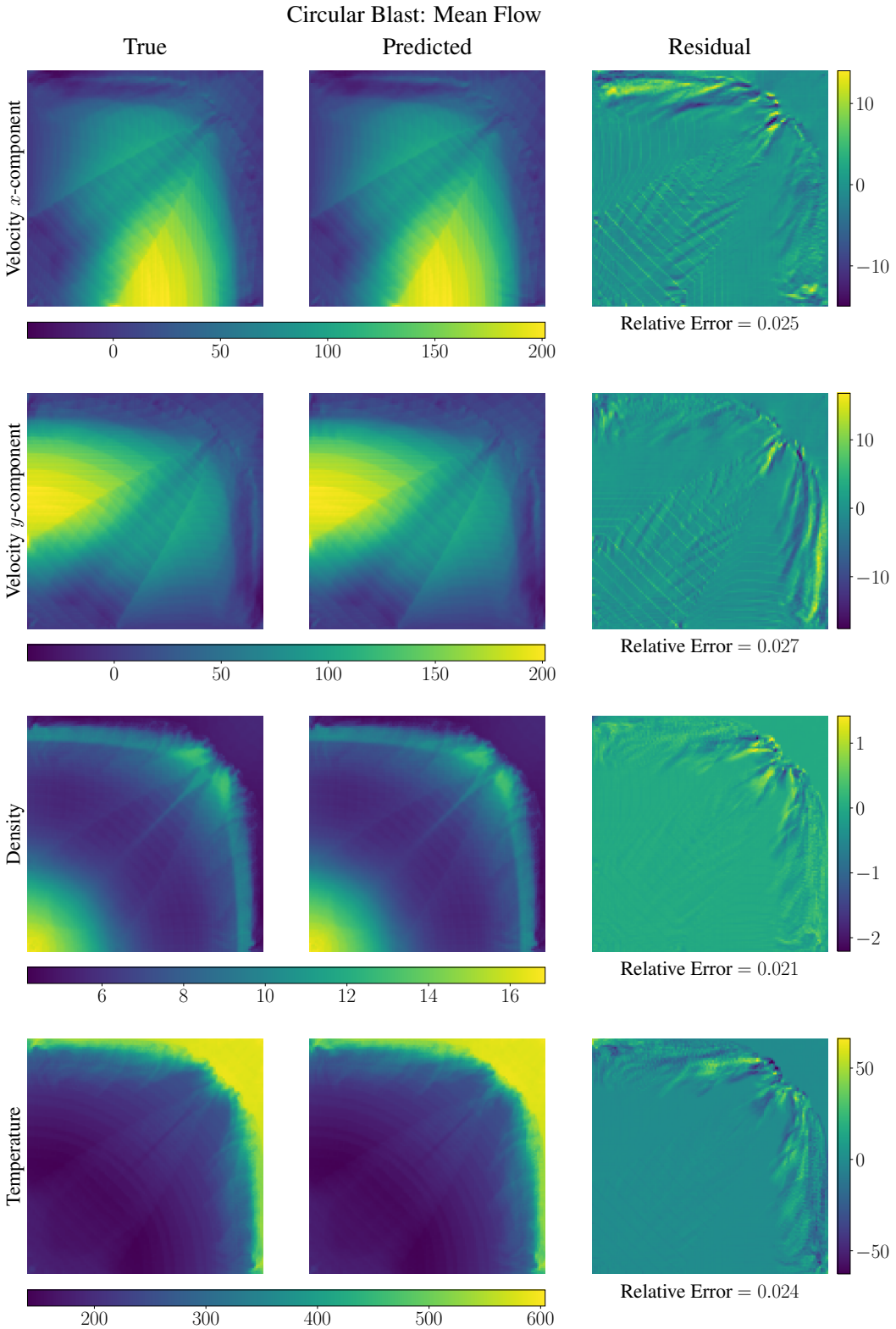

Figure 12: Mean flow for circular blast.

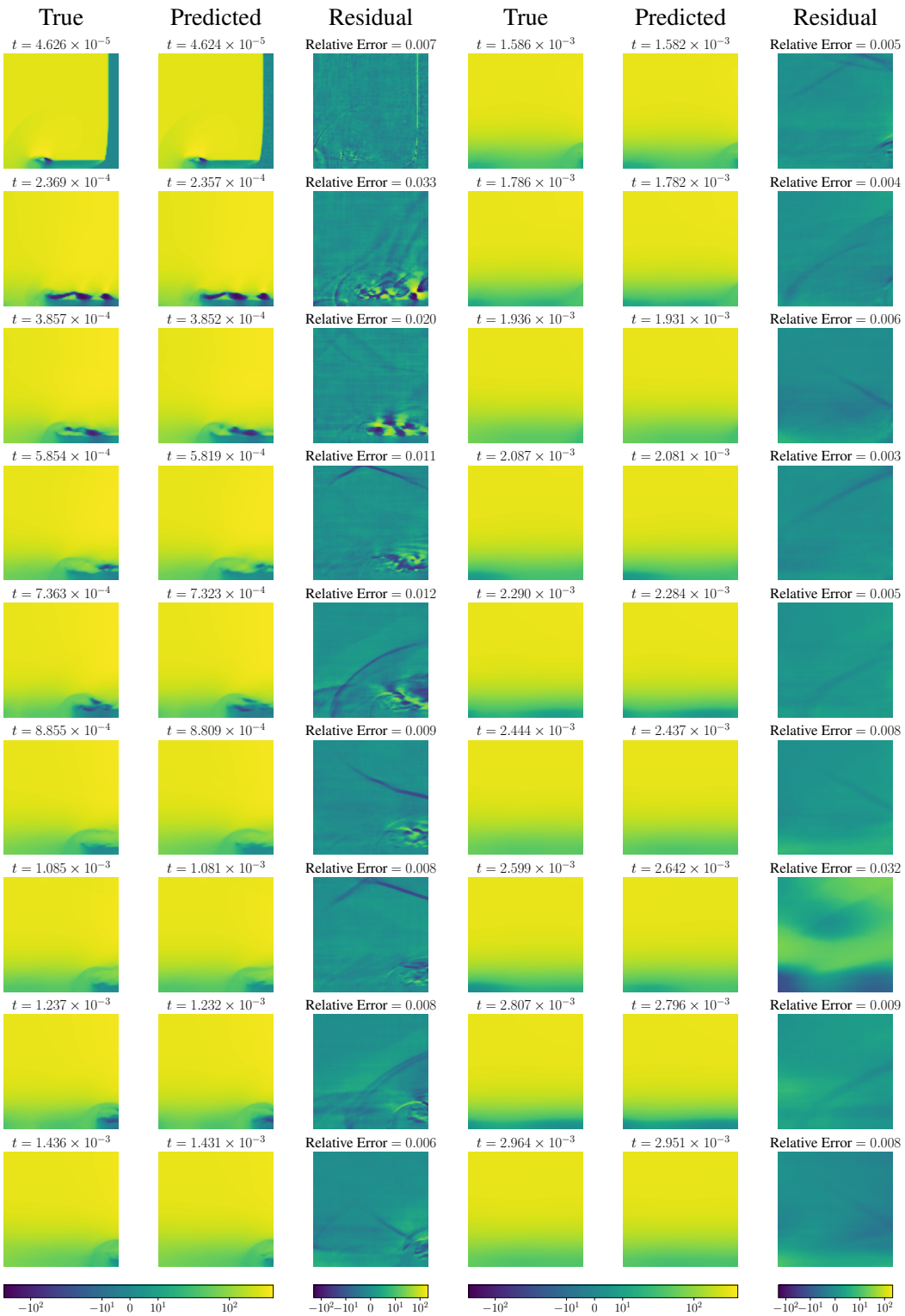

Figure 13: Gas velocity x-component for coal dust explosion.

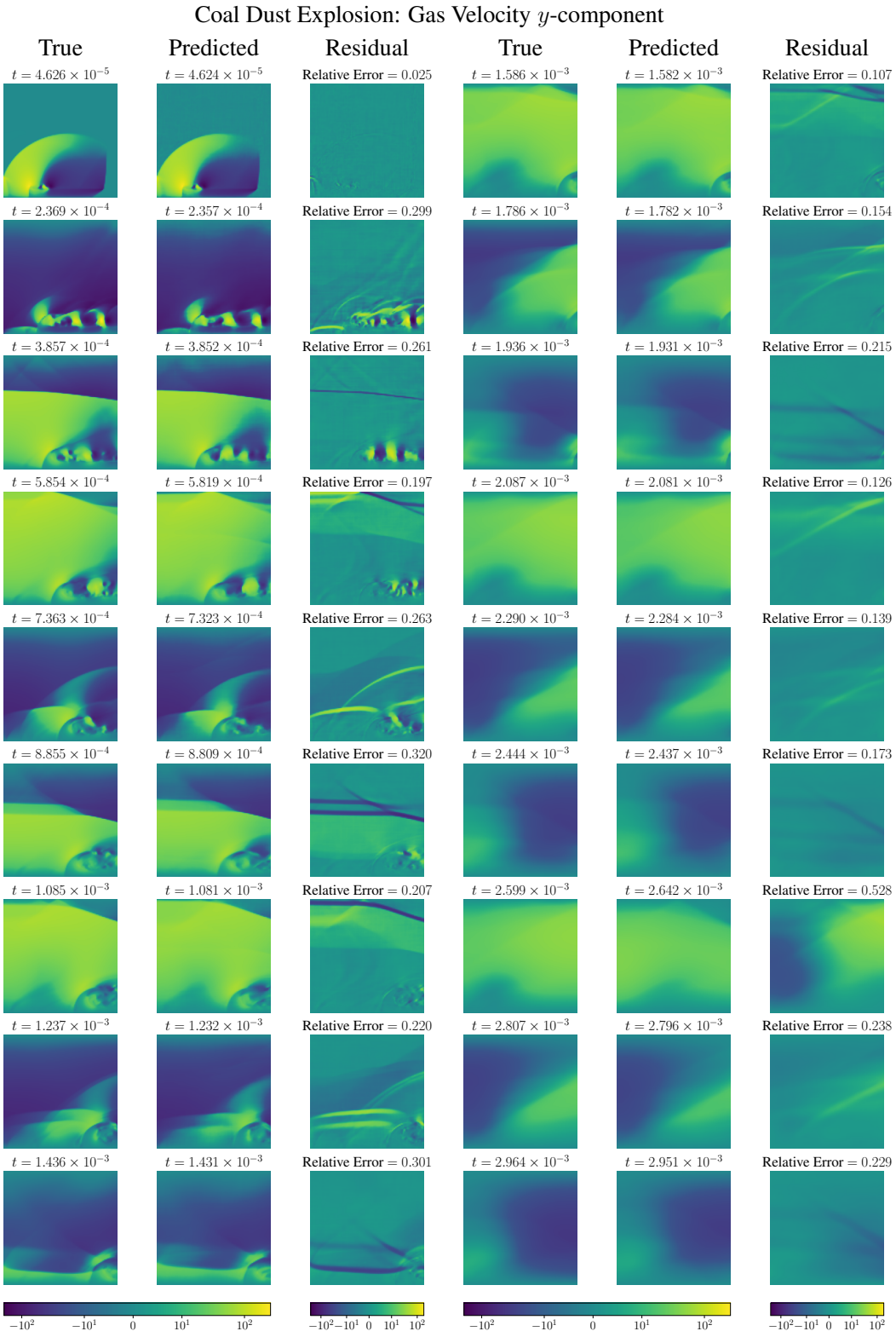

Figure 14: Gas velocity y-component for coal dust explosion.

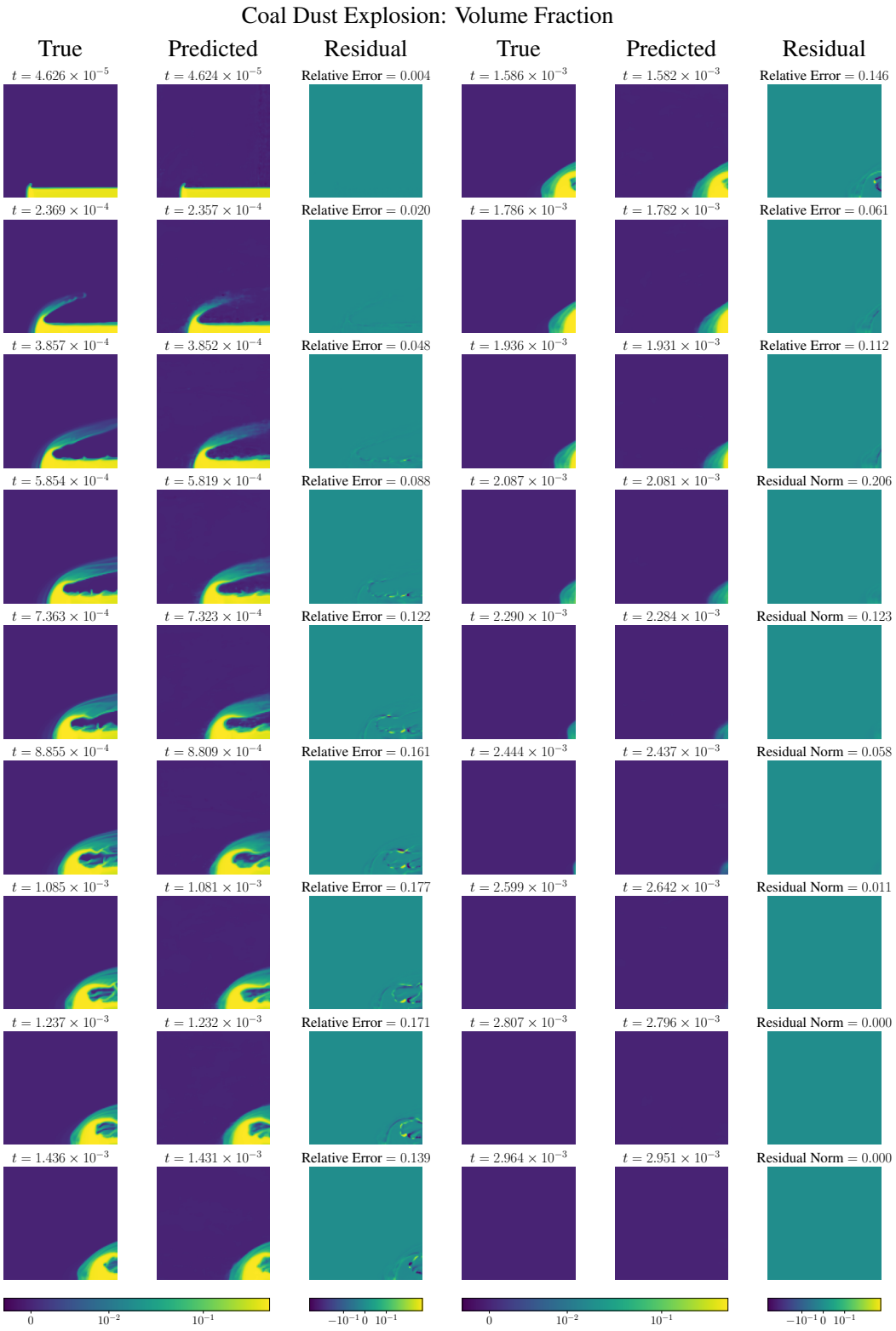

Figure 15: Volume fraction for coal dust explosion.

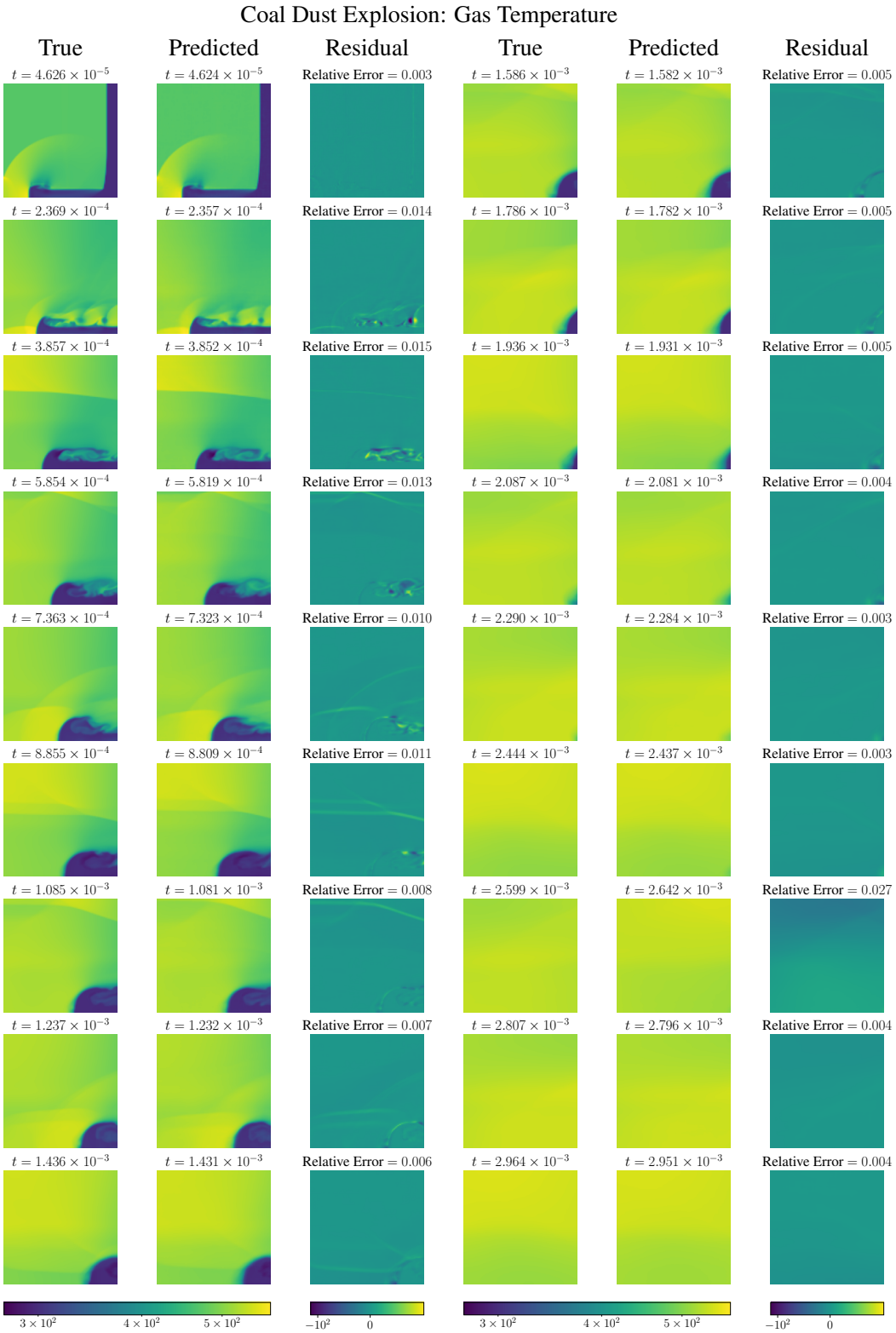

Figure 16: Gas temperature for coal dust explosion.

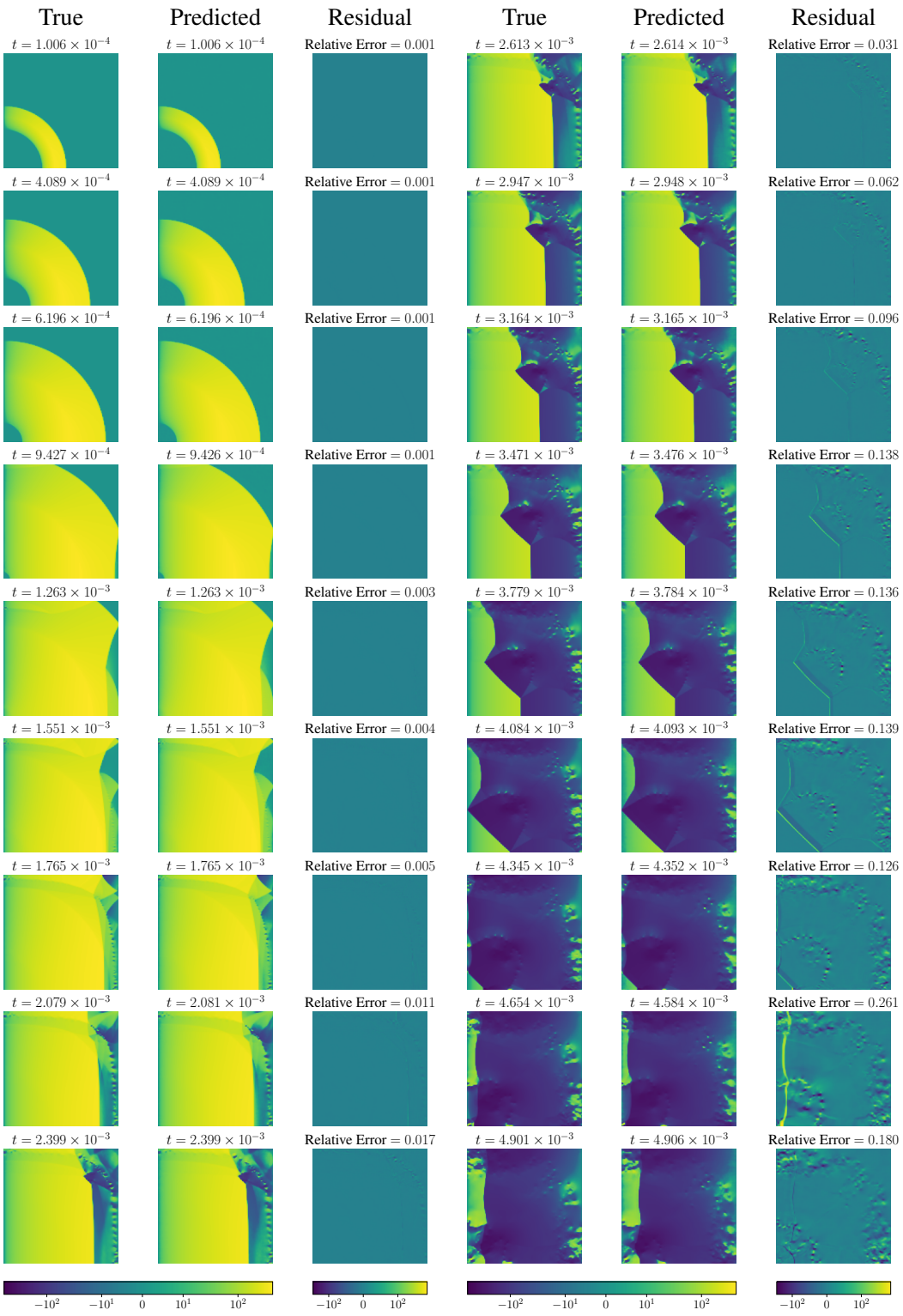

Figure 17: Velocity x-component for circular blast.

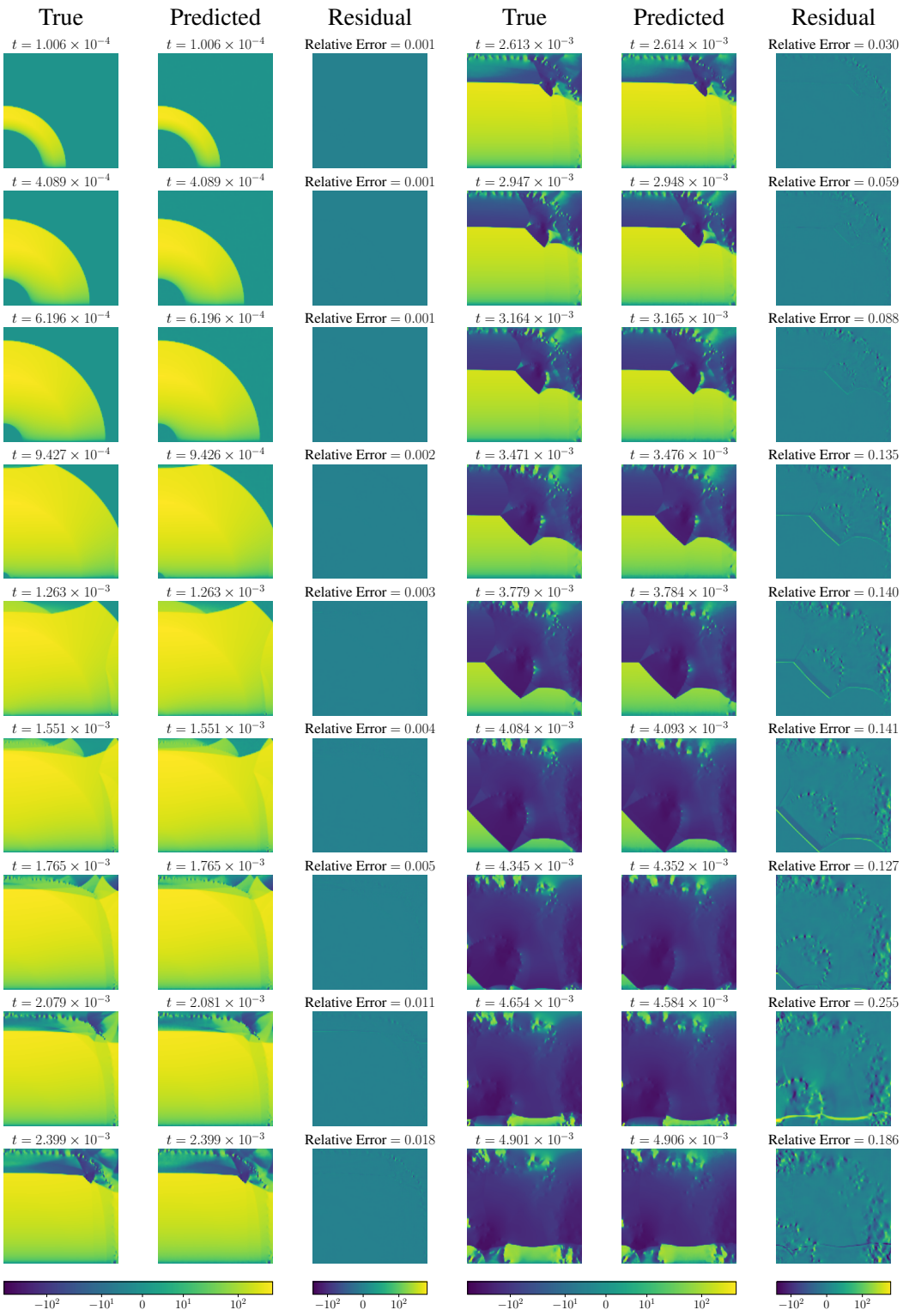

Figure 18: Velocity y-component for circular blast.

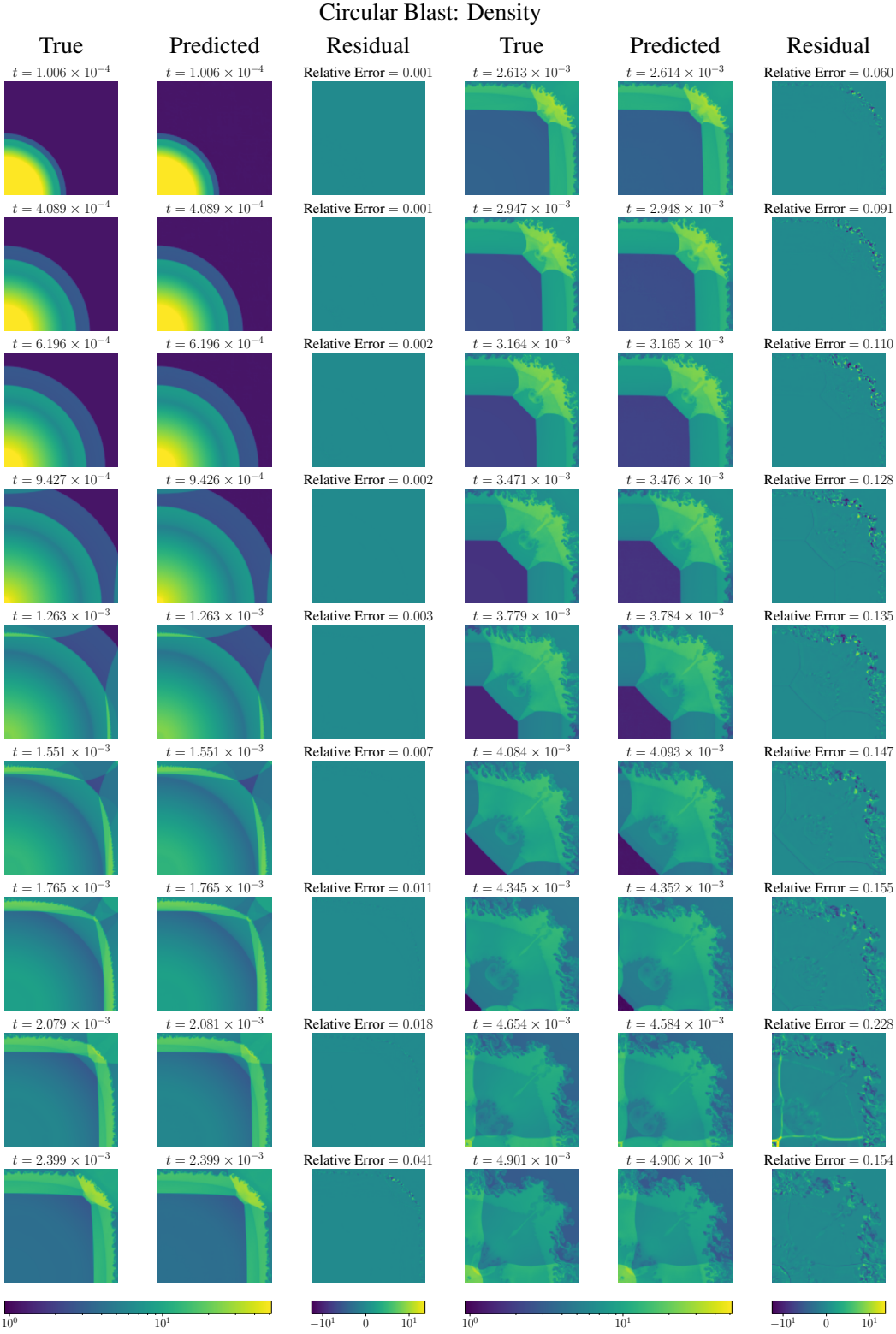

Figure 19: Density for circular blast.

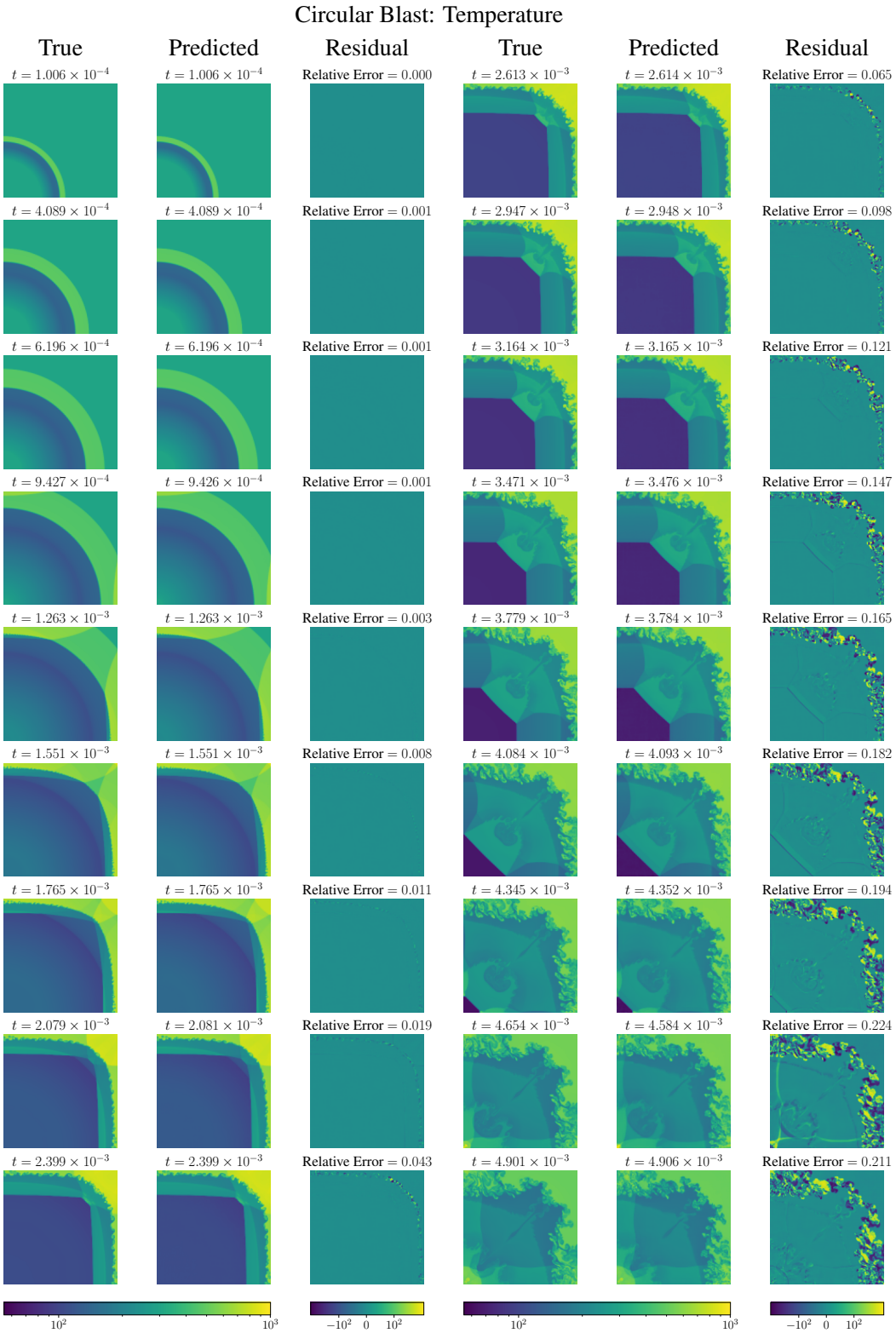

Figure 20: Temperature for circular blast.

Coal Dust Explosion: One-Step Prediction Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Gas Velocity x-component	Gas Velocity y-component	Volume Fraction	Gas Temperature	Mean
CNO					
Base	0.94 (0.01)	9.94 (0.02)	3.03 (0.01)	0.38 (0.00)	3.57 (0.00)
Euler	0.97 (0.01)	10.27 (0.08)	3.15 (0.04)	0.38 (0.00)	3.69 (0.02)
MoE	0.92 (0.00)	9.93 (0.04)	2.86 (0.02)	0.36 (0.00)	3.52 (0.01)
F-FNO					
Base	0.93 (0.00)	10.27 (0.06)	3.04 (0.01)	0.36 (0.00)	3.65 (0.02)
Euler	0.93 (0.00)	10.20 (0.01)	3.01 (0.03)	0.36 (0.00)	3.62 (0.01)
MoE	0.93 (0.00)	10.33 (0.03)	3.05 (0.01)	0.36 (0.00)	3.67 (0.01)
Transolver					
Base	1.22 (0.02)	12.98 (0.19)	2.62 (0.01)	0.44 (0.01)	4.32 (0.05)
Euler	1.18 (0.02)	12.83 (0.22)	2.60 (0.04)	0.44 (0.01)	4.26 (0.07)
MoE	1.20 (0.01)	12.83 (0.07)	2.53 (0.02)	0.43 (0.00)	4.25 (0.02)
U-Net					
Base	0.92 (0.01)	10.32 (0.06)	2.80 (0.02)	0.35 (0.00)	3.59 (0.02)
Euler	0.91 (0.01)	10.27 (0.05)	2.82 (0.02)	0.35 (0.00)	3.59 (0.02)
MoE	0.93 (0.01)	10.36 (0.07)	2.88 (0.01)	0.35 (0.00)	3.63 (0.02)

Table 3: Relative error for one-step predictions on evaluation split of coal dust explosion cases.

Circular Blast: One-Step Prediction Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Velocity x-component	Velocity y-component	Density	Temperature	Mean
CNO					
Base	2.08 (0.01)	2.08 (0.01)	1.82 (0.01)	1.73 (0.01)	1.93 (0.01)
Euler	2.05 (0.01)	2.08 (0.02)	1.79 (0.01)	1.69 (0.01)	1.90 (0.01)
MoE	1.75 (0.03)	1.75 (0.04)	1.49 (0.01)	1.42 (0.01)	1.60 (0.01)
F-FNO					
Base	1.87 (0.00)	1.87 (0.00)	1.93 (0.00)	2.05 (0.00)	1.93 (0.00)
Euler	1.85 (0.01)	1.84 (0.01)	1.93 (0.00)	2.04 (0.00)	1.92 (0.01)
MoE	1.87 (0.00)	1.86 (0.00)	2.00 (0.01)	2.14 (0.00)	1.97 (0.00)
Transolver					
Base	1.21 (0.01)	1.21 (0.01)	0.95 (0.01)	0.93 (0.01)	1.07 (0.01)
Euler	1.27 (0.02)	1.27 (0.02)	1.01 (0.01)	0.99 (0.01)	1.14 (0.01)
MoE	1.28 (0.01)	1.29 (0.01)	1.02 (0.01)	0.99 (0.01)	1.15 (0.01)
U-Net					
Base	1.52 (0.00)	1.51 (0.01)	1.34 (0.00)	1.37 (0.00)	1.44 (0.00)
Euler	1.52 (0.00)	1.52 (0.00)	1.34 (0.00)	1.38 (0.01)	1.44 (0.00)
MoE	1.71 (0.01)	1.70 (0.01)	1.49 (0.01)	1.52 (0.01)	1.61 (0.01)

Table 4: Relative error for one-step predictions on evaluation split of circular blast cases.

Coal Dust Explosion: Unrolled Prediction Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Gas Velocity x-component	Gas Velocity y-component	Volume Fraction	Gas Temperature	Mean
CNO					
Base	3.09 (0.08)	45.30 (1.25)	18.02 (1.00)	1.22 (0.05)	16.91 (0.59)
Euler	3.07 (0.09)	45.71 (1.15)	17.53 (0.34)	1.19 (0.03)	16.88 (0.35)
MoE	3.04 (0.02)	45.53 (0.17)	17.58 (0.70)	1.18 (0.01)	16.83 (0.15)
F-FNO					
Base	2.87 (0.08)	42.70 (0.38)	16.72 (0.09)	1.10 (0.01)	15.85 (0.13)
Euler	2.80 (0.00)	42.51 (0.66)	16.98 (0.27)	1.09 (0.01)	15.84 (0.20)
MoE	2.89 (0.06)	43.27 (0.73)	17.16 (0.32)	1.13 (0.01)	16.11 (0.14)
Transolver					
Base	3.33 (0.07)	42.59 (1.20)	19.59 (0.21)	1.20 (0.02)	16.68 (0.26)
Euler	3.33 (0.02)	43.95 (0.65)	19.26 (0.70)	1.22 (0.03)	16.94 (0.30)
MoE	3.21 (0.11)	42.61 (0.84)	19.56 (0.60)	1.22 (0.03)	16.65 (0.38)
U-Net					
Base	3.03 (0.05)	44.66 (0.49)	16.26 (0.23)	1.12 (0.01)	16.27 (0.16)
Euler	2.93 (0.03)	44.82 (0.40)	16.86 (0.25)	1.12 (0.00)	16.43 (0.08)
MoE	3.00 (0.04)	45.02 (0.47)	17.12 (0.10)	1.14 (0.01)	16.57 (0.15)

Table 5: Relative error for unrolled predictions on evaluation split of coal dust explosion cases.

Circular Blast: Unrolled Prediction Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Velocity x-component	Velocity y-component	Density	Temperature	Mean
CNO					
Base	6.93 (0.08)	6.99 (0.12)	7.63 (0.02)	7.96 (0.03)	7.37 (0.05)
Euler	6.98 (0.13)	7.06 (0.12)	7.58 (0.06)	7.92 (0.06)	7.38 (0.08)
MoE	6.74 (0.06)	6.77 (0.05)	7.48 (0.07)	7.85 (0.05)	7.21 (0.01)
F-FNO					
Base	5.89 (0.03)	5.86 (0.01)	5.59 (0.01)	5.89 (0.01)	5.81 (0.01)
Euler	5.70 (0.04)	5.73 (0.06)	5.56 (0.03)	5.87 (0.03)	5.71 (0.04)
MoE	5.91 (0.08)	5.95 (0.10)	5.68 (0.01)	6.00 (0.03)	5.89 (0.04)
Transolver					
Base	7.35 (0.31)	7.35 (0.32)	7.59 (0.05)	8.07 (0.02)	7.59 (0.17)
Euler	7.54 (0.16)	7.31 (0.11)	7.63 (0.09)	8.11 (0.04)	7.65 (0.09)
MoE	7.27 (0.17)	7.23 (0.23)	7.62 (0.10)	8.08 (0.06)	7.55 (0.14)
U-Net					
Base	5.45 (0.07)	5.47 (0.10)	5.16 (0.03)	5.44 (0.04)	5.38 (0.05)
Euler	5.50 (0.01)	5.44 (0.02)	5.24 (0.06)	5.55 (0.06)	5.43 (0.03)
MoE	5.46 (0.02)	5.50 (0.02)	5.30 (0.04)	5.56 (0.03)	5.45 (0.01)

Table 6: Relative error for unrolled predictions on evaluation split of circular blast cases.

Coal Dust Explosion: Correlation Time Proportion $\times 10^{-2}$ (↑)					
Model	Gas Velocity x-component	Gas Velocity y-component	Volume Fraction	Gas Temperature	Mean
CNO					
Base	80.00 (0.00)	19.05 (0.17)	80.04 (1.10)	61.05 (1.06)	60.03 (0.53)
Euler	80.00 (0.00)	21.40 (2.96)	78.53 (1.59)	64.21 (0.92)	61.04 (0.92)
MoE	80.00 (0.00)	19.48 (0.32)	79.27 (1.07)	65.45 (2.44)	61.05 (0.85)
F-FNO					
Base	80.00 (0.00)	19.76 (0.28)	78.74 (0.32)	65.71 (0.30)	61.05 (0.16)
Euler	80.00 (0.00)	20.72 (1.46)	78.21 (0.23)	64.86 (0.94)	60.95 (0.13)
MoE	80.00 (0.00)	16.17 (2.99)	77.64 (0.68)	65.69 (0.25)	59.88 (0.94)
Transolver					
Base	80.00 (0.00)	21.76 (2.71)	75.25 (0.28)	65.19 (0.48)	60.55 (0.76)
Euler	80.00 (0.00)	19.23 (4.55)	75.80 (0.69)	64.48 (0.48)	59.88 (1.10)
MoE	80.00 (0.00)	19.36 (0.34)	76.46 (0.74)	65.06 (0.05)	60.22 (0.17)
U-Net					
Base	80.00 (0.00)	21.90 (1.68)	79.92 (1.13)	63.17 (0.10)	61.25 (0.12)
Euler	80.00 (0.00)	20.02 (0.15)	79.27 (1.01)	64.27 (0.56)	60.89 (0.26)
MoE	80.00 (0.00)	22.08 (1.50)	78.65 (0.96)	63.12 (0.11)	60.96 (0.29)

Table 7: Correlation time proportion for unrolled predictions on evaluation split of coal dust explosion cases.

Circular Blast: Correlation Time Proportion $\times 10^{-2}$ (↑)					
Model	Velocity x-component	Velocity y-component	Density	Temperature	Mean
CNO					
Base	100.00 (0.00)	100.00 (0.00)	93.38 (0.39)	93.07 (0.23)	96.61 (0.15)
Euler	100.00 (0.00)	100.00 (0.00)	93.35 (0.12)	92.89 (0.22)	96.56 (0.08)
MoE	100.00 (0.00)	100.00 (0.00)	93.04 (0.09)	92.79 (0.06)	96.46 (0.02)
F-FNO					
Base	100.00 (0.00)	100.00 (0.00)	96.07 (0.42)	95.30 (0.15)	97.84 (0.14)
Euler	100.00 (0.00)	100.00 (0.00)	95.77 (0.06)	95.49 (0.14)	97.81 (0.05)
MoE	100.00 (0.00)	100.00 (0.00)	95.52 (0.06)	94.96 (0.05)	97.62 (0.03)
Transolver					
Base	100.00 (0.00)	100.00 (0.00)	94.11 (0.50)	93.58 (0.47)	96.92 (0.24)
Euler	100.00 (0.00)	100.00 (0.00)	94.11 (0.39)	93.37 (0.39)	96.87 (0.17)
MoE	100.00 (0.00)	100.00 (0.00)	94.18 (0.42)	93.09 (0.01)	96.82 (0.10)
U-Net					
Base	100.00 (0.00)	100.00 (0.00)	97.22 (0.84)	96.13 (0.21)	98.34 (0.26)
Euler	100.00 (0.00)	100.00 (0.00)	96.25 (0.42)	96.10 (0.23)	98.09 (0.16)
MoE	100.00 (0.00)	100.00 (0.00)	95.70 (0.07)	95.57 (0.13)	97.82 (0.05)

Table 8: Correlation time proportion for unrolled predictions on evaluation split of circular blast cases.

Coal Dust Explosion: Mean Flow Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Gas Velocity x-component	Gas Velocity y-component	Volume Fraction	Gas Temperature	Mean
CNO					
Base	1.00 (0.02)	23.03 (0.39)	12.42 (0.17)	0.42 (0.01)	9.22 (0.10)
Euler	0.97 (0.05)	22.68 (0.86)	12.71 (0.30)	0.39 (0.01)	9.19 (0.27)
MoE	0.91 (0.04)	22.79 (0.25)	12.25 (0.28)	0.37 (0.01)	9.08 (0.10)
F-FNO					
Base	0.82 (0.10)	22.59 (0.12)	11.80 (0.12)	0.35 (0.01)	8.89 (0.08)
Euler	0.79 (0.04)	22.29 (0.25)	11.99 (0.31)	0.32 (0.01)	8.85 (0.13)
MoE	0.80 (0.02)	22.44 (0.13)	12.55 (0.24)	0.34 (0.00)	9.03 (0.10)
Transolver					
Base	1.44 (0.09)	23.07 (0.60)	14.32 (0.17)	0.39 (0.03)	9.81 (0.10)
Euler	1.37 (0.10)	23.47 (0.70)	13.46 (0.13)	0.39 (0.01)	9.67 (0.14)
MoE	1.25 (0.16)	22.78 (0.11)	13.20 (0.15)	0.40 (0.02)	9.41 (0.09)
U-Net					
Base	1.06 (0.07)	22.50 (0.24)	11.17 (0.30)	0.36 (0.02)	8.77 (0.10)
Euler	1.03 (0.05)	23.33 (0.30)	11.74 (0.41)	0.37 (0.02)	9.12 (0.18)
MoE	1.06 (0.02)	22.53 (0.48)	11.62 (0.25)	0.35 (0.01)	8.89 (0.17)

Table 9: Relative error for mean flow on evaluation split of coal dust explosion cases.

Circular Blast: Mean Flow Relative Error $\times 10^{-2}$ ($\downarrow$)					
Model	Velocity x-component	Velocity y-component	Density	Temperature	Mean
CNO					
Base	6.25 (0.12)	6.20 (0.24)	2.35 (0.06)	2.33 (0.07)	4.28 (0.11)
Euler	6.24 (0.23)	6.25 (0.21)	2.30 (0.05)	2.28 (0.06)	4.27 (0.13)
MoE	7.94 (2.10)	7.98 (2.14)	2.30 (0.06)	2.24 (0.05)	5.12 (1.08)
F-FNO					
Base	5.77 (0.22)	5.70 (0.15)	1.73 (0.01)	1.68 (0.02)	3.72 (0.10)
Euler	5.77 (0.19)	5.81 (0.12)	1.72 (0.02)	1.66 (0.03)	3.74 (0.08)
MoE	5.12 (0.13)	5.12 (0.07)	1.71 (0.01)	1.69 (0.02)	3.41 (0.03)
Transolver					
Base	6.21 (0.13)	6.04 (0.23)	2.27 (0.01)	2.37 (0.01)	4.22 (0.08)
Euler	11.59 (1.40)	10.82 (1.95)	2.43 (0.01)	2.49 (0.01)	6.83 (0.83)
MoE	6.52 (0.28)	6.30 (0.42)	2.34 (0.05)	2.44 (0.02)	4.40 (0.18)
U-Net					
Base	7.94 (2.63)	7.90 (2.45)	1.62 (0.05)	1.53 (0.03)	4.75 (1.29)
Euler	7.97 (2.34)	7.92 (2.32)	1.67 (0.05)	1.58 (0.02)	4.79 (1.18)
MoE	12.89 (0.02)	12.95 (0.12)	1.76 (0.03)	1.63 (0.02)	7.31 (0.05)

Table 10: Relative error for mean flow on evaluation split of circular blast cases.

Coal Dust Explosion: TKE Relative Error $\times 10^{-2}$ ($\downarrow$)	
Model	TKE
CNO	
Base	$11.55\,(0.23)$
Euler	$10.91\,(0.16)$
MoE	$10.94\,(0.11)$
F-FNO	
Base	$11.01\,(0.36)$
Euler	$10.28\,(0.11)$
MoE	$10.93\,(0.04)$
Transolver	
Base	$12.72\,(0.50)$
Euler	$12.44\,(0.32)$
MoE	$11.79\,(0.88)$
U-Net	
Base	$9.85\,(0.24)$
Euler	$9.21\,(0.12)$
MoE	$8.91\,(0.16)$

Circular Blast: TKE Relative Error $\times 10^{-2}$ ($\downarrow$)	
Model	TKE
CNO	
Base	$2.60\,(0.02)$
Euler	$2.69\,(0.07)$
MoE	$2.64\,(0.05)$
F-FNO	
Base	$2.29\,(0.03)$
Euler	$2.23\,(0.06)$
MoE	$2.16\,(0.03)$
Transolver	
Base	$2.68\,(0.08)$
Euler	$2.95\,(0.04)$
MoE	$2.77\,(0.06)$
U-Net	
Base	$2.27\,(0.12)$
Euler	$2.30\,(0.06)$
MoE	$2.36\,(0.03)$

Table 11: Relative error for TKE on evaluation splits.

H Broader Impacts

Neural PDE solvers have been applied in accelerating dynamics simulations across a variety of real-world applications of PDE modeling, including weather and climate forecasting, aerodynamics modeling, and subsurface modeling. As neural solvers often do not include guarantees on generalization or stability over long time-integration periods, it is vital to perform rigorous validation before relying on predictions in applications. Here, we have explored the potential of neural solvers to accelerate modeling of high-speed flows. High-speed flows play an important role in the design of a variety of applications with potential for societal impact, including spacecraft, missiles, and atmospheric reentry vehicles. It is therefore important to closely monitor the development of works along this direction.