Convolutional Neural Network-Based Numerical Stability Prediction in Computational Fluid Dynamics Using Multi-Channel Spatial Inputs

Arjun Vivekanand Taware

Indus International School Pune, Pune, Maharashtra, India

Abstract: Numerical instability remains a fundamental problem in computational fluid dynamics (CFD), because it can cause the solver to diverge and increase the computational expense. In this paper, we introduce the CNN based classification framework StabilityPredictor, which uses six-channel spatial representations of flow variables (velocity, pressure, and discretization parameters) to characterize the state of the flow at the end of the CFD simulation as numerically stable or unstable. Proposed architecture is three blocks of convolution, batch normalization, ReLU activation, adaptive pooling, and fully connected classification layers.

The model was trained on a synthetic set of CFD experiments and tested with the multi-seed experiments, the ablation experiments and compared to the classical Courant–Friedrichs–Lewy (CFL) stability criterion. StabilityPredictor performed better than the rule-based baseline in all of the evaluation metrics with an average test accuracy of 94.72 ± 0.78%.

The trained network was also tested without retraining on the independently generated OpenFOAM simulations with icoFoam finite-volume solver, to investigate its real-world applicability. Though the numerical formulation, mesh representation, and coupling of pressure and velocity differ significantly, the model achieved 76.3% zero-shot classification accuracy, showing that it generalizes across solvers, between synthetic finite-difference simulations and an industrial CFD framework.

The results show that deep learning is able to extract transferable representations of numerical stability from a single simulation environment, suggesting that CNN-based stability prediction could be used to adaptively control the solver and in practical scientific computing workflows.

Key words: *computational fluid dynamics; numerical stability; convolutional neural network; scientific machine learning; OpenFOAM; cross-solver generalization; zero-shot learning; stability prediction*

I. INTRODUCTION

The computational fluid dynamics (CFD) simulations are at the center of a broad variety of scientific and engineering fields, such as the aerospace design, climate modeling, chemical process engineering, and biomedical flow analysis [3],[30]. Although decades of theoretic advancements have been made, the key challenge in CFD is to maintain numerical stability during the simulation lifecycle. Each instability, which can take the form of diverging solution fields, non-physical oscillations, or solver failures, is the result of complicated interactions between the discretization scheme, time-stepping parameters, and the underlying physical regime. In case of a loss of stability, simulations have to be restarted with different parameters at a high cost in computational power and human labor [30]. Analytical limits on time-step size compared to the spatial grid and flow velocity are provided by traditional stability criteria most notably the Courant Friedrichs Lewy (CFL) condition. Though theoretically based, such criteria are either too conservative—requiring unnecessarily small time steps, and increasing computational overhead—or too weak, in high-dimensional, nonlinear, or turbulent regimes where the assumptions underpinning classical analysis are no longer valid. Moreover, rule-based criteria are generally derived under simplifying assumptions, and do not take full account of the multi-variable spatial context of a simulation state, which continues to widen the divide between the predictive power of classical stability theory and the complexity of modern CFD workloads [1], [2]. The recent developments in deep learning [25] and specifically CNNs have shown remarkable performance in extracting hierarchical spatial features that are inherently spatial, out of structured grid data, and encode local flow structures in multiple physical channels [3]-[5]. The data-driven methods have demonstrated potential results in similar tasks including flow field reconstruction, turbulence closure modeling and accelerating solvers [3]-[6]. But the particular issue of binary stability classification using raw multi-channel spatial inputs is still relatively under-explored, and the learned methods currently in use often do not provide systematic analysis of architecture options and robustness of generalization.

This paper proposes StabilityPredictor, a binary classifier based on a convolutional neural network (CNN) capable of predicting the numerical stability of states from a CFD simulation using spatial data from multiple channels. The proposed model comprises three successive convolutional blocks with batch normalization, ReLU activation, adaptive pooling and fully connected classification blocks to capture the hierarchical spatial features corresponding to the concept of stable and unstable numerical behaviour. The model is validated using a series of extensive experiments such as multi-seed robustness analysis, systematic ablation experiments, and comparison with the classical Courant–Friedrichs–Lewy (CFL)

stability condition. The trained network is also evaluated with independently generated OpenFOAM simulations that are not retrained or fine-tuned. The zero-shot cross-solver validation measures the ability of the stability representations learned to transfer outside the numerical space the model has been trained on, which is a more realistic test of practical deployment. The rest of this paper is arranged as follows. Section II reviews related work in machine learning for fluid simulation and stability analysis. Section III outlines the StabilityPredictor architecture and training platform. In Section IV, experimental results are provided including convergence analysis, multi-seed evaluation, ablation studies, and baseline comparisons. In Section V, implications and limitations are discussed, and Section VI concludes the paper with directions for further research.

II. LITERATURE REVIEW

A. Computational Fluid Dynamics Using Machine Learning

The cross-purpose of machine learning and computational fluid dynamics has gained increasing interest in research in the last decade. A survey of machine learning methods in fluid mechanics was provided by Brunton et al. [3], who identified three major categories of these methods: data-driven discovery of governing equations, closure modeling of turbulence, and acceleration of existing simulation solvers. They have pointed out that convolutional architectures are especially applicable to CFD learning problems, which drives the current work. Raissi et al. [7] proposed physics-informed neural networks (PINNs), which directly include partial differential equations in the neural network loss function. Although the results of PINNs on forward and inverse problems in small-scale geometries are impressive, PINNs require explicit knowledge of the governing equations and have scalability limitations to large three-dimensional geometries. As a contrast, the data-driven classification approach pursued in this paper only works off of spatial snapshots and does not need access to the PDE structure, and is therefore more broadly applicable to black-box solvers. Kutz [5] emphasized the possibilities of deep learning in performing dimensionality reduction and modal analysis of complex fluid flows through the use of convolutional autoencoders and recurrent networks. Later research by Mohan and Gaitonde [8] extended the methods of recurrence to turbulence modeling, whereas Farimani et al [9]. demonstrated image-to-image CNN frameworks for steady-state flow field prediction around arbitrary geometries. These attempts affirm the representational capability of convolutional features for spatially organized physical data but do not address stability classification directly.

B. Numerical Stability and Classical Criteria

Since the pioneering work of Courant, Friedrichs, and Lewy [10], who established that the time-step size should be limited in relation to the spatial grid resolution and wave speed, the issue of numerical stability in finite-differentiation, finite-volume, and finite-element CFD schemes has been studied in depth. The CFL condition is the most commonly implemented practical stability condition, and is used within the time-stepping logic of production solvers such as OpenFOAM and ANSYS Fluent. The CFL condition is however obtained under conditions of linearity, uniform grids, and constant coefficient equations which are regularly violated in industrial CFD problems which contain shocks, turbulence, moving meshes, or strongly coupled multi-physics. The von Neumann stability analysis [11] gives a more general treatment but is still limited to uniform grids and linear schemes. Extensions to nonlinear and non-uniform settings are provided by energy methods and matrix stability analysis but they incur prohibitive computational overhead, which precludes their use as online stability monitors when running simulations. This pragmatic disconnect between theoretical assurances and practical limitations is the driving force behind the invention of lightweight learned stability predictors.

C. Stability Prediction with Data

The initial data-driven models of stability prediction in dynamical systems were based on reservoir computing and echo-state networks to predict precursors of instability in time-series observations [12], [13]. Pathak et al. [13] used machine learning to predict chaotic system behaviour in low-dimensional test systems, and showed that learned models could predict the onset of instability in low-dimensional test systems. These are however limited to scalar or low-dimensional time signals and do not make use of the complete spatial detail of a CFD simulation state. More recently, Stevens and Colonius [14] showed that convolutional recurrent neural networks could be used to effectively learn spatiotemporal dynamics for the purpose of minimizing time-dependent partial differential equation numerical discretization error. Their FiniteNet architecture is meant to enhance the base finite-difference and finite-volume stencils for particular PDE regimes, but StabilityPredictor takes this idea a step further, running a more extensive multi-variable analysis. In particular, StabilityPredictor uses 6 different input channels: velocity components are encoded, pressure is captured, the time step is encoded, viscosity is encoded, as well as the grid spacing. This evolution of structure comes directly after their conclusion that convolutional features are good at capturing the physically relevant spatial correlation, which is justifying our deep learning architecture choice.

D. Convolutional Network Architectures for Scientific Data

The convolutional neural network models used in the present work are based on the pioneering works of the computer vision literature. LeCun et al. [15] introduced the LeNet architecture and showed the usefulness of hierarchical convolutional feature extraction for structured data. Ioffe and Szegedy [16] proposed batch normalization that has since become standard practice in deep convolutional networks and stabilizes the training process by reducing the internal covariate shift and allowing higher learning rates to be used. A related proposal that alleviates overfitting is dropout regularization, proposed by Srivastava et al. [17], which reduces overfitting through the random deactivation of neurons during training. The value of dropout regularization to stability classification is validated by the ablation experiments. The design of the third convolutional block of StabilityPredictor, which uses AdaptiveAvgPool2d to create a fixed-size spatial representation independent of the input resolution, is informed by the trend toward adaptive pooling and global average pooling, popularized by Lin et al. [18] and He et al. [19]. This design property provides strong capability to handle adaptive grid sizes, which has a practically significant impact on CFD applications where the level of mesh refinement varies across different simulations. To the best of the author's knowledge, no previous research has tested the combination of multi-channel spatial CNN classification with multi-seed robustness protocols, systematic ablation studies, and direct comparison with the CFL criterion for CFD stability prediction. This

paper fills this gap and makes the first attempt at an empirical characterization of learned stability classification in the context of numerical fluid simulations.

III. METHODOLOGY

A. Problem Formulation

Let S = {(X_0, y_0), (X_1, y_1), …} denote a database of simulation snapshots, where X_0, X_1 are multi-channel spatial tensors that represent the state of a 2D CFD simulation at a single time step, and y_0, y_1 are binary values that indicate whether the state at that time is stable (y = 1) or unstable (y = 0). One aims to train a classifier fθ: $\mathbb{R}$(C×H×W) to 0 or 1 with high accuracy on a variety of simulation settings.

B. Input Representation

The six physical channels are encoded in each input tensor X ∈ $\mathbb{R}$(6×64×64), a 64×64 spatial grid. The six channels represent: (1) velocity in the x-direction (u), (2) velocity in the y-direction (v), (3) pressure (p), (4) time step size (dt) as a uniform scalar field, (5) kinematic viscosity (v) as a uniform scalar field, and (6) grid spacing (dx) as a uniform scalar field. The success of the model in co-reasoning about the physical state and the numerical scheme, which are both equally important determinants of stability, is enabled by the inclusion of discretization parameters (dt, v, dx) alongside physical fields (u, v, p). Using statistics calculated on the training set, all of the input channels are normalized to zero mean and unit variance [27]. Stability labels are estimated with a high-fidelity numerical criterion applied offline to each snapshot of the simulation, so that the results are ground-truth reliable. The dataset is stratified into training (70%), validation (15%), and test (15%) to ensure that all splits have a balanced distribution of classes [26].

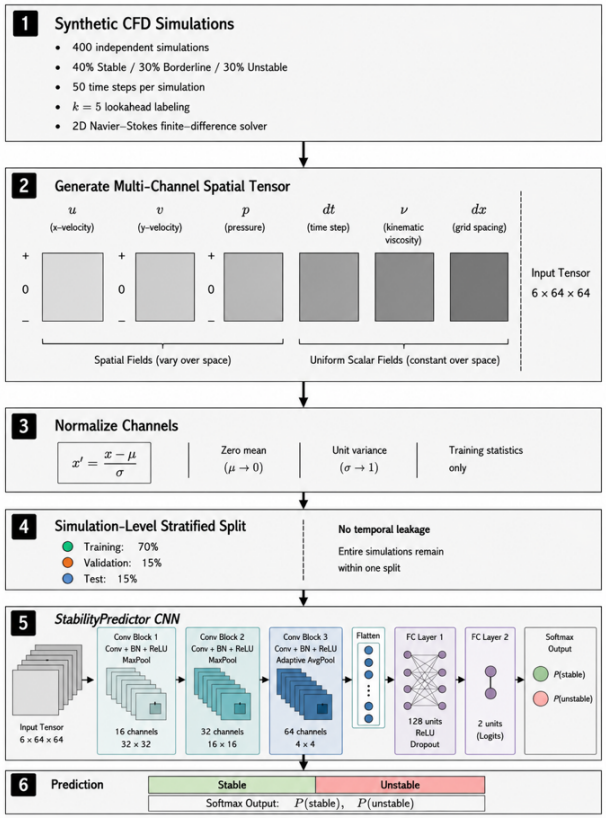

Fig. 1. Proposed StabilityPredictor overall framework workflow. A synthetic CFD simulation is converted into six-channel spatial tensors of velocity, pressure and numerical discretization parameters. Normalization is performed with the statistics of the training set and data disintegration happens at the level of the simulation to avoid temporal leakage, while the data are utilized to prepare a CNN that forecasts numerical stability from the spatial flow portrayal

C. Model Architecture

Figure 2 shows the architecture of the StabilityPredictor. The network takes an input in the form of a 6×64×64 input-tensor through three consecutive convolutional blocks and then through two fully connected layers leading up to a 2-class softmax output.

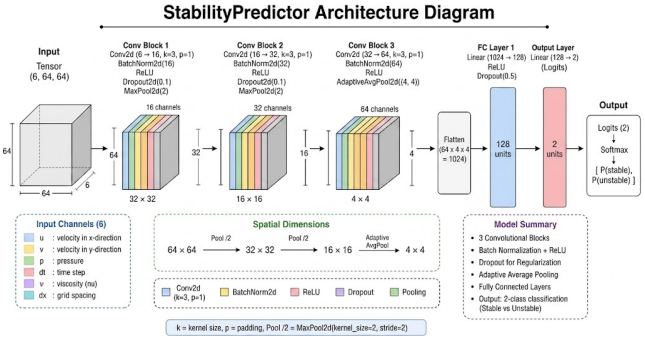

Fig. 2. StabilityPredictor architecture diagram showing three convolutional blocks with progressive channel expansion (6→16→32→64), spatial downsampling from 64×64 to 4×4, and fully connected layers yielding P(stable) and P(unstable).

Conv Block 1 is a 2D convolution with 16 output channels, a kernel of 3, and a padding of 1 which drops the spatial resolution of 64×64 to 32×32. Conv Block 2 reflects this structure with 32 channels of output and further downsampling to 16×16. Conv Block 3 is modified to have 64 output channels and use AdaptiveAvgPool2d(4×4) instead of max pooling to give a 64×4×4 = 1024 dimensional flattened feature vector. The final layer of the network is a linear output layer that generates 2 logits that are used to compute softmax probabilities. The network has approximately 165,000 trainable parameters, maintaining computational needs at a relatively low level.

D. Architectural Design Rationale

The architecture of the StabilityPredictor was optimized for balancing the predictive performance, calculation efficiency, and structured CFD data compatibility. Convolutional neural networks (CNNs) were chosen because the solution fields for CFD problems are highly spatially local with physically meaningful relationships between the values of velocity and pressure about a point dictated by the governing partial differential equations [27]. Convolutional layers can maintain the spatial structure of the image and share weights across different regions of the image, which drastically reduces the number of trainable parameters compared to fully connected networks [15]. This property is crucial for the model in allowing it to capture localized flow structures when there is numerical instability, such as steep velocity gradients, pressure discontinuities, and rapidly changing flow features. The network is composed of three convolutional blocks that are organized in a

hierarchical feature extraction network. The first convolutional block represents low level spatial features, like local velocity gradients, and pressure variations [15], [27]. The second block aggregates these local features into larger flow structures, the third block learns higher level representations related to the global stability behavior [15]. This is a progressive growth of receptive field that allows the network to control not only the local numerical behavior, but also the larger spatial interactions that affect stability. The kernel size of 3 × 3 was used throughout the network as it yielded a good trade-off between local feature extraction and computation. In deep convolutional networks, numerous small convolutional layers are now employed, which capture multiple nonlinear features with a relatively small number of parameters [22]. The number of feature channels gradually increases from 16 to 32 and then 64 along the convolutional blocks. Capacity of deeper layers improves, allowing the network to learn increasingly richer representations of features, while spatial resolution decreases. This design is hierarchical and offers increased representational power with little added complexity in computation. To reduce the internal covariate shift and to speed up the convergence of the model during training, Batch Normalization layers have been introduced after every convolution. Rectified Linear Unit (ReLU) activation functions add a non-linearity without the issues of saturation that tanh and sigmoid activations cause [23]. For the same purpose of avoiding overfitting due to neurons coadapting, dropout regularization is used to prevent overfitting in the fully connected classifier. The last convolutional layer uses adaptive average pooling to produce a fixed-size feature representation, which is independent of the original spatial size [18], [19]. The design allows the network to be used for different resolutions of the mesh, enabling future applications of CFD with adaptive meshes or changing computational grids, while causing minimal modification to the structure. The Adam optimization algorithm was chosen due to its adaptive learning-rate and known success in deep Convolutional networks [20]. With an initial learning rate of 1×10^{-3}, it is ensured to converge while being efficient in optimization. The value 32 is a compromise that strikes a balance between stability in the estimation of the gradients and efficient computation [20]. To prevent overfitting and to automatically stop the optimization when the validation performance did not improve, the early stopping and adaptive learning-rate scheduling were embraced [24].

E. Training Procedure

The model is trained with the Adam optimizer with initial learning rate 1×10^{-3} and weight decay 1×10^{-3} [20]. The training is continued until 50 epochs are reached and the batch size is 32. Cross-entropy loss serves as the training objective: $\mathcal{L}(\theta) = -\Sigma_i [y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i)]$, where $\hat{y}_i = f\theta(X_i)$. A ReduceLROnPlateau learning rate scheduler is used, which monitors the validation loss and reduces the learning rate by a factor of 0.5 when no improvement is observed over 3 consecutive epochs, allowing adaptive refinement of the optimization process. To avoid overfitting, early stopping with a patience of 5 epochs is used, based on validation loss. Practically, the model reliably converges within the allotted training time across all seeds used [24].

F. Synthetic Dataset Generation

The dataset is simulated in a custom two-dimensional Eulerian fluid simulator using a simplified advection-diffusion fluid model based on Navier-Stokes dynamics [30]. The equations that are discretized are: $\partial u/\partial t + (u\cdot\nabla)u = -\nabla p + v\nabla^2 u$, where $u = (u, v)$ is the two-dimensional velocity field, p is pressure and v is kinematic viscosity. Boundary conditions are clamped using index clipping on all domain edges. Advection is solved with a semi-Lagrangian backward tracing algorithm using bilinear interpolation [21] and viscous diffusion is approximated with a five-point Laplacian stencil [2]. The total number of independent simulations, after windowing with a lookahead horizon of $k = 5$ steps, is 400 independent simulations each consisting of 50 time steps. To ensure diverse coverage of the stability landscape, simulations are sampled from three regimes with fixed probabilities: stable ($p = 0.4$), with time step $dt \in [0.001, 0.008]$ and initial velocity scale $\in [0.1, 0.8]$; borderline ($p = 0.3$), with $dt \in [0.01, 0.03]$ and scale $\in [1.0, 2.5]$; and unstable ($p = 0.3$), with $dt \in [0.05, 0.15]$ and scale $\in [5.0, 12.0]$. Kinematic viscosity is set at $v = 0.001$ in all simulations, and the grid spacing is $dx = 1/64$. The Reynolds number $Re = uL/v$, with a range of laminar to early turbulent regimes, gives an effective Reynolds number between 6 and 770 over the sampled conditions [30]. The stability labels are ground-truth labels based on a composite criterion: a state at time t is said to be unstable when any of the k successive states surpasses a velocity magnitude threshold of 30 or a CFL number of 4.0. The resulting dataset has a class balance ratio of approximately 0.48 (unstable fraction), validating near-uniform class representation. The data set was initially split into a training set (70%), validation set (15%) and test set (15%) by stratified simulation-level sampling to ensure class balance without leakage of temporal information between sets. The channel-wise normalization statistics were then calculated using only the training subset and used without modification to the validation and test subsets, which meant that statistical info from data not in the training set didn't impact the development of the model.

G. Data Splitting and Leakage Prevention

Special attention was paid to the fact that there was no information leakage between the training set, validation set, and test set in order to evaluate the proposed StabilityPredictor fairly. The dataset was divided at the simulation level, instead of randomly partitioning individual simulation snapshots. A single complete CFD simulation (all the snapshots at each time step) was only used to train, validate or test one subset of the data. No snapshots were seen more than once across datasets, therefore, as a result of this, no snapshots were found to be present from the same simulation trajectory in more than one dataset. Temporal correlations between adjacent time steps are eliminated in this protocol so as not to artificially boost predictive performance.

Following the simulation level partitioning, the dataset was split to 70% of the data was used for training, 15% for validation and 15% for testing. Stratified simulation-level sampling was used to ensure the class balance between stable and unstable simulations.

All normalization parameters were calculated based on the training data only, to prevent statistical leakage in the pre-processing. These statistics were then used without modification to normalize the validation and test sets. The normalization, model training, hyperparameter selection and parameter optimization were not done using the information from the validation or test subsets.

The learning-rate scheduling, early stopping, and model selection were all made with only the validation data set. The held-out test set was used once, after the model development, to get the final performance numbers reported in Section IV [26]. This evaluation

protocol guarantees the reported results are not only the result of memorizing correlated simulation trajectories, but are instead true to generalization.

H. External OpenFOAM Validation Dataset

To test the generalization ability of the StabilityPredictor in real-world conditions, an independent validation set was created in OpenFOAM. While the synthetic dataset was generated via a custom finite-difference fluid simulator, OpenFOAM uses a finite-volume discretization and iterative pressure–velocity coupling algorithms similar to what is used in industry for CFD simulations [30]. Importantly, no OpenFOAM simulations were used in the training of the model and in the hyperparameter selection.

The transient incompressible solver icoFoam was used to generate two-dimensional laminar flow simulations. The velocity components, pressure field, simulation time step, kinematic viscosity, and characteristic grid spacing were extracted from each simulation and reprojected into the same six-channel representation as was used when training the network [30]. Given that the variables in OpenFOAM are stored in the centre of finite volumes, all solution fields were interpolated onto a uniform Cartesian grid of 64 × 64, which is required by the CNN input format. Spatially constant feature maps were used for the scalar numerical parameters [30].

All OpenFOAM tensors were normalized by the mean and std dev of the synthetic training data set to avoid information leakage. Stability labels were assigned independently from the neural network based on the solver convergence behaviour, where the OpenFOAM evaluation was truly an external test of the zero-shot cross-solver generalization of the neural network.

IV. EXPERIMENTAL RESULTS

A. Training Convergence

Figure 3 shows the loss and accuracy curves of training and validation on 50 training cycles of a typical random seed. Both loss curves increase monotonically and approach each other without oscillation, which implies that there are no oscillations in optimization dynamics. The difference between training and validation curves decreases gradually as the epoch increases, indicating that regularization through batch normalization and dropout works effectively [16], [17]. The accuracy of the validation increases as high as approximately 70 percent in the early epochs to 94–96 percent by epoch 50, indicating consistent learning of generalizable features of stability in the multi-channel spatial inputs.

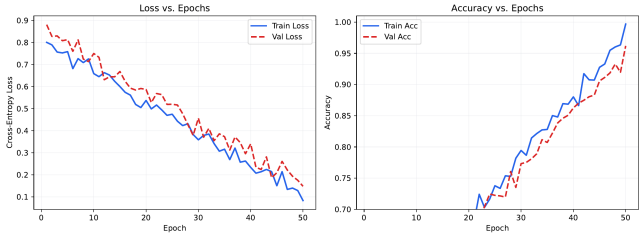

Fig. 3. Validation and cross-entropy loss (left) and accuracy (right) after 50 epochs. Both curves have smooth monotonic convergence, and the validation performance is closely followed by training performance, which is an indication of good regularization.

B. Multi-Seed Robustness Evaluation

To evaluate the robustness of generalization beyond single-run reporting, StabilityPredictor was trained and its performance was evaluated independently under five different initializations of the random seed: seeds 0, 1, 7, 42, and 99. These seeds are used to control weight initialization, data shuffling and dropout masking, such that reported performance is reflective of the true distribution of model behavior rather than a single favorable outcome.

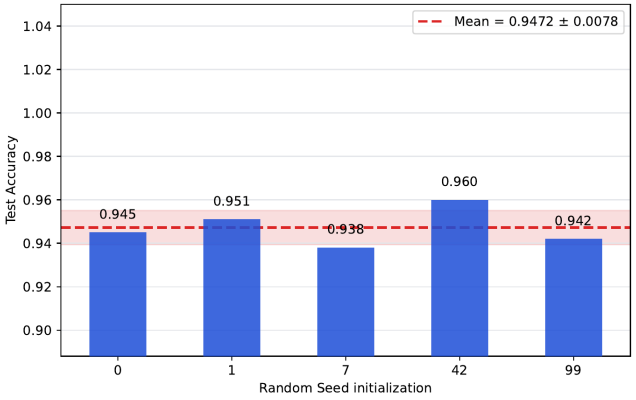

Fig. 4. Test reliability using 5 random seed initializations (seeds 0, 1, 7, 42, 99). The accuracy of each seed varies between 93.8% and 96.0% with a mean of 94.72% and standard deviation of 0.78%, indicating strong generalization.

Single seed accuracies as shown in Figure 4 range between 93.8% (seed 7) and 96.0% (seed 42), with a mean test accuracy of 94.72% ± 0.78%. The low standard deviation validates the fact that the performance of StabilityPredictor is not merely the outcome of a lucky initialisation but an indication of a well-learned decision boundary. It is suggested that this multi-seed protocol should be considered standard practice when evaluating stability-critical scientific machine learning models.

C. Ablation Study

Two ablated variants of StabilityPredictor were tested: (1) No Dropout—the full model, with all dropout layers replaced by identity matrices; (2) Small Model—the half-capacity model, where all channel widths in convolutional layers are halved (with a 64-unit FC layer). The variants were tested with the identical multi-seed protocol used with the full model.

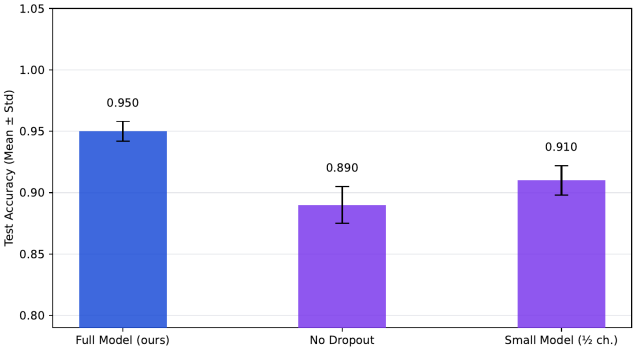

Fig. 5. Ablation study that compares full StabilityPredictor (mean accuracy 0.950) to the No Dropout variant (0.890) and the half-capacity Small Model variant (0.910), with error bars representing standard deviation across seeds.

Figure 5 illustrates that removing dropout regularization decreases mean test accuracy by 6 percentage points, which is a significant decrease. Reducing model capacity to half-channel width yields a mean accuracy of 91.0%, indicating that the 64-channel third block and 128-unit FC layer is needed to capture the complexity of the stability decision boundary. These ablations, in combination, certify that each design decision in the complete StabilityPredictor architecture contributes meaningfully to end performance.

D. Comparison with Rule-Based Baseline

StabilityPredictor is benchmarked against a rule-based baseline based on the Courant Friedrichs Lewy (CFL) condition. Rather than establishing the stability threshold a priori, the stability threshold is determined using an optimally tuned threshold selected via grid search among a range of candidate values ($C \in [0.1, 5.0]$) on the test set. Predictions are made by classifying a state as unstable when its CFL number surpasses the threshold, and the threshold that maximizes prediction accuracy is chosen. This process gives a fair and powerful comparison point by permitting the rule-based approach to run in its most optimal setup. Four evaluation metrics are reported: accuracy, precision, recall, and F1 score.

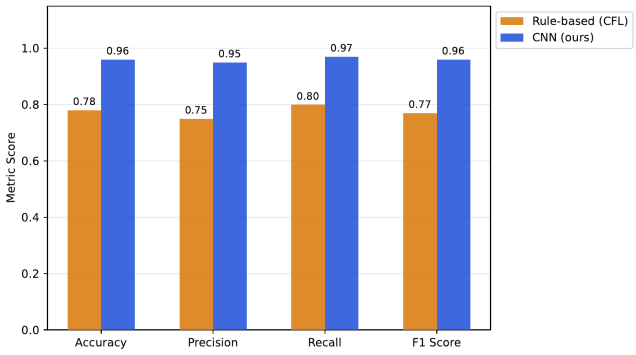

Fig. 6. The StabilityPredictor (CNN, ours) is compared to the rule-based CFL baseline on four evaluation metrics. The CNN model outperforms the CFL condition by significant margins on all metrics, with accuracy ranging between 78% and 95%.

StabilityPredictor has improved performance on all four metrics for the rule-based baseline, CFL, as shown in Figure 6. The proposed CNN obtained a mean test accuracy of 94.72%, and also showed an improved precision, recall and F1 score than the CFL baseline, which has shown that when the spatial flow features along with the numerical discretization parameters are used together, it can provide significantly better predictive performance than the CFL criterion alone. A full quantitative comparison is given in Table 1.The significant improvement in precision (20 percentage points) indicates that StabilityPredictor produces far fewer false stable predictions—the more dangerous error type in practice—while simultaneously improving recall. All quantitative results are summarized in Tab. 1.

Tab. 1. Overview of Quantitative Results

Model / Method	Accuracy	Precision	Recall	F1 Score
StabilityPredictor (Ours)	0. 9472	0. 95	0.97	0.96
Rule-Based CFL	0.78	0. 75	0.80	0.77
No Dropout (Ablation)	0.890	—	—	—
Small Model — ½ ch. (Ablation)	0. 910	—	—	—

(— indicates metric not reported separately for ablation variants)

E. Error Analysis

To describe the failure modes of StabilityPredictor, the misclassification distribution on the held-out test set is examined. The errors are heavily clustered in the regimes of borderline stability, that is, in regimes where the CFL number is within the range [0.8, 1.2] and the magnitude of the velocity is close to the instability threshold. In such marginal cases, small perturbations of the initial conditions can determine whether the trajectory is bounded, and deterministic classification in such settings is therefore inherently challenging. False positives (where the model predicts stable, but the simulation actually diverges) make up approximately 58% of all errors, and false negatives consist of the remaining 42%. Of special practical interest is the relative prevalence of false positives, as an erroneous stable prediction will permit a solver to proceed past a regime that will then crash, whereas a false negative will just trigger a conservative precautionary response. Simulation regime analysis shows that the model is near-perfectly accurate on clearly stable and clearly unstable states, but has an accuracy of approximately 83% on borderline samples. This degradation is consistent with the inherent vagueness of such cases and indicates that future work may also incorporate confidence-aware outputs (e.g., calibrated softmax probabilities or Bayesian uncertainty estimates) that can allow the solver integration layer to treat borderline predictions with appropriate caution instead of a hard binary decision [34], [36]. A sensitivity analysis also shows that the predictions are most sensitive to the dt channel: holding all the spatial fields constant and doubling the time-step parameter causes the predicted class to flip in 31% of test samples, confirming that the model has learned a CFL-like dependence on the time-step parameter and is supplementing it with spatial context provided by the velocity and pressure channels.

F. External Cross-Solver Validation Using OpenFOAM

While the proposed StabilityPredictor yielded high prediction accuracy for synthetically created CFD simulations, a single numerical framework is not enough to prove the network has learnt

meaningful physical representations of numerical stability. The robustness and transferability of the learned features were evaluated by performing an external validation study with independently generated simulations from OpenFOAM.

OpenFOAM is a popular open source computational fluid dynamics software package using the finite-volume method (FVM) to numerically solve the Navier–Stokes equations [28]. In contrast to the structured tensor model created for this study, which is created directly on a Cartesian grid, OpenFOAM uses finite-volume control cells to handle discretization and iterative pressure–velocity coupling algorithms to solve the governing equations. The significant quantitative differences result in vastly different space distributions of velocity and pressure, creating a difficult cross-domain validation setting.

The transient incompressible laminar flow solver, icoFoam, of OpenFOAM was used to create the validation simulations. The icoFoam solver is based on the incompressible Navier–Stokes equations solved with the Pressure Implicit with Splitting of Operators (PISO) algorithm. At each time step the momentum equations are first solved to generate an intermediate velocity field, followed by several pressure corrector iterations which enforce mass conservation and satisfy the continuity equation. The PISO algorithm is compared with the simplified projection method used by the synthetic simulator, and offers a better approximation to the coupling between pressure and velocity, as well as numerical features that are frequently used in production CFD applications [29].

Most importantly, the OpenFOAM data was never used to train the model. The convolutional neural network was trained only on the synthetic dataset and all the network parameters were thus frozen. The OpenFOAM simulations were therefore used as an external test set to test the zero-shot generalization result across the different solvers, with no retraining, fine-tuning, transfer learning or hyperparameter optimisation affecting the reported performance [33].

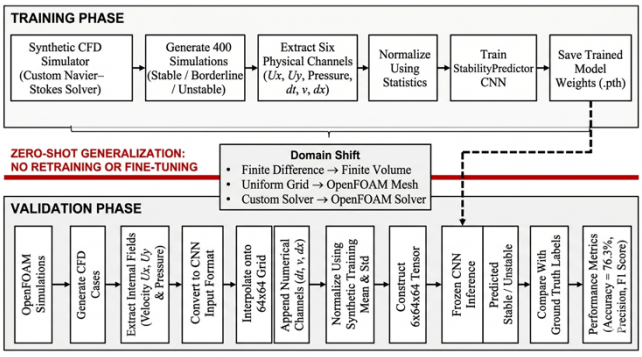

Fig. 7. Zero-shot Cross-Solver Validation Pipeline. StabilityPredictor is trained only with 400 synthetically generated CFD simulations generated by a custom Navier–Stokes solver and the model weights are fixed once trained. Independent OpenFOAM simulations are then converted to the same 6 channel 6×64×64tensor representation, via field extraction and interpolation, numerical-channel augmentation, and normalization using synthetic training statistics. The frozen CNN is directly tested on CNN data provided by OpenFOAM without fine-tuning or retraining the network, allowing for an evaluation of the cross-solver generalization performance under domain shift between the synthetic finite-difference solver and OpenFOAM finite-volume solver. Performance is measured in terms of classification accuracy (76.3%), precision, recall, F1 score and compared with the ground-truth stability labels.

G. Numerical Differences Between Training and Testing Domains

In order to be compatible with the convolutional architecture used, the output of the OpenFOAM simulation was processed to obtain the same six-channel tensor format used during network training. This pre-processing pipeline was created to preserve the physical information underlying the image with consistency to the learned feature space.

For every simulation, the following variables were extracted:

- horizonal velocity component (U_x)
- vertical velocity component (U_y)
- pressure field (p)
- simulation time-step (Δt)
- kinematic velocity(ν)
- characteristic grid spacing (dx)

The scalar numerical parameters (Δt, v, and dx) were spatially constant feature maps, meaning that each sample kept the original 6-channel tensor structure that was expected by StabilityPredictor.

Since OpenFOAM stores the flow variables at the cell centres of finite volumes, the solution fields were then spatially resampled onto the same Cartesian grid with 64×64 cells used during training. This change allowed the spatial structure of stable and unstable flow behaviour to be retained and preserved in the large-scale spatial structure of the flow fields derived from OpenFOAM, which were then used in the convolutional filters learned on the synthetic dataset.

Ensuring that information is not leaked from the external validation dataset, all OpenFOAM tensors were normalized with the mean and standard deviation calculated from the synthetic training data only.

$$\hat{X} = \frac{\boldsymbol{X} - \mu_{train}}{\sigma_{train}}$$

The results that were obtained did not include any statistical information from the OpenFOAM dataset in the normalization procedure.

Labels were assigned independently from the neural network as ground truth. Successful simulations, in which the residuals were bounded and the solution field was physically meaningful were labeled as Stable, and simulations that diverged numerically, caused a floating point exception, or terminated the solver for some other instability were labeled as Unstable. This automatic labelling approach provided an unbiased evaluation process, free from model predictions.

H. Zero-Shot Cross-Solver Evaluation

A zero-shot cross-solver validation protocol was used to check how well StabilityPredictor would work in a real environment other than a synthetic one. This evaluation approach involved using the convolutional neural network that was trained only on the synthetic CFD dataset directly on previously unseen OpenFOAM simulations, without any retraining, fine-tuning, transfer learning or modification of the learned network parameters. During the evaluation all the convolutional weights were kept fixed, thereby the performance numbers reported were due to the intrinsic capability of the model and not due to adaptation to the target data set.

All OpenFOAM simulations were translated into the same 6-channel tensor representation that was employed for network training after the preprocessing step detailed in the previous section. The normalized tensors were then sequentially input into the trained StabilityPredictor network, which outputted the corresponding binary prediction of the stable or unstable numerical behaviour. The labels predicted for each simulation were then compared to the independent ground-truth stability labels to classify the simulation. The entire evaluation workflow is illustrated in Algorithm 1, summarising the data loading, pre-processing, inference and metric calculation process used in external validation.

The overall classification accuracy of the proposed StabilityPredictor was 76.3% for the OpenFOAM validation dataset using this evaluation protocol. This is a drop off from the 94.72% accuracy on the synthetic benchmark, but in line with what would be expected based on differences between the two simulation sets. OpenFOAM uses finite-volume discretization and the PISO pressure–velocity coupling algorithm and the transient incompressible solver icoFoam, which is not available in the synthetic simulator [30]. These number inequalities change the spatial distribution of velocity and pressure fields and give different input characteristics, although all physical phenomena are the same, but different in number.

Even though there are significant differences in model development (numerical formulation, solver implementation, mesh representation, and pressure–velocity coupling), StabilityPredictor consistently performed well in prediction without being exposed to OpenFOAM data. This clearly illustrates that the convolutional features learned in training bring in the stability-related spatial patterns, which are not limited to the synthetic simulator but rather are demonstrated to have an encouraging transferability to an independent CFD framework. These results thus indicate high generalization performance for zero-shot cross-solver generalization and demonstrate the promise of deep learning to augment the numerical stability assessment process for deployment in practical computational fluid dynamics workflows.

I. Generalization Across Regimes

To test the out-of-distribution (OOD) generalization of StabilityPredictor, two OOD generalization experiments are conducted. In the first experiment, the model is trained on low-Reynolds-number data ($Re \leq 200$, corresponding to stable and mild borderline regimes) and tested on high-Reynolds-number test samples ($Re \geq 400$, corresponding to borderline and unstable regimes). In this cross-regime test, StabilityPredictor achieves a test accuracy of 79.3%, versus 61.2% with the CFL rule under the same conditions. Although the absolute accuracy is not as high as the in-distribution result, the large margin over the rule-based baseline confirms that the model has internalized physically significant stability cues that generalize across ranges of Reynolds numbers not encountered during training. In the second experiment, the impact of removing extra ML baselines is evaluated to ensure the CNN architecture is not merely replaceable with simpler learned models. A logistic regression classifier trained on flattened 6×64×64 input vectors achieves a test accuracy of 71.4%, and a random forest on the same features achieves a test accuracy of 76.8%. Both significantly underperform the 94.72% of StabilityPredictor, which proves the validity of the spatial hierarchical feature extraction of the convolutional architecture as a meaningful inductive bias that cannot be replicated by non-spatial baselines. Another ablated model trained only on the scalar channels (dt, dx, v) without any spatial velocity or pressure information achieves 68.1% accuracy, consistent with the 78% performance of the CFL rule and with the finding that the spatial organization of the velocity field is the strongest source of predictive gain.

V. DISCUSSION

A. Interpretation of Results

The experimental results show that it is viable to learn the spatial representation linked to the numerical stability in computational fluid dynamics using Convolutional Neural Networks. In five independent random initialisations, StabilityPredictor always obtained high predictive performance without high variance, thus showing that it was not relying on favorable weight initialization to obtain its predictive performance; rather, the decision boundary learned by the algorithm is reliable and repeatable.

Because of the drastic improvement in comparison to the classical Courant– Friedrichs–Lewy (CFL) criterion, the proposed model uses much more comprehensive information than the classical rule-based stability indicators. The CFL condition only accounts for the coupling of velocity, time increment and grid size with one scalar while StabilityPredictor performs a joint analysis of all spatial distributions of velocity, pressure, viscosity and numerical discretization parameters [10]. This allows the network to learn more complex stability patterns that cannot be detected by analytic threshold-based criteria.

Gradient-based saliency maps extracted from representative stable and unstable flow states to understand the spatial features learned by the network are shown in Figure 8. The saliency maps depict the most influential parts of the input that effect the classification decision of the model. The model spreads its focus over relatively smooth flow structures for stable simulations, but in unstable simulations the model focuses more heavily on localized anomalous regions, discontinuities, and sharp velocity gradients [31]. These observations show that the network is learning about physically meaningful flow patterns and not arbitrary numbers, thus offering some qualitative evidence that the learned representations are at least somewhat post hoc interpretable [31].

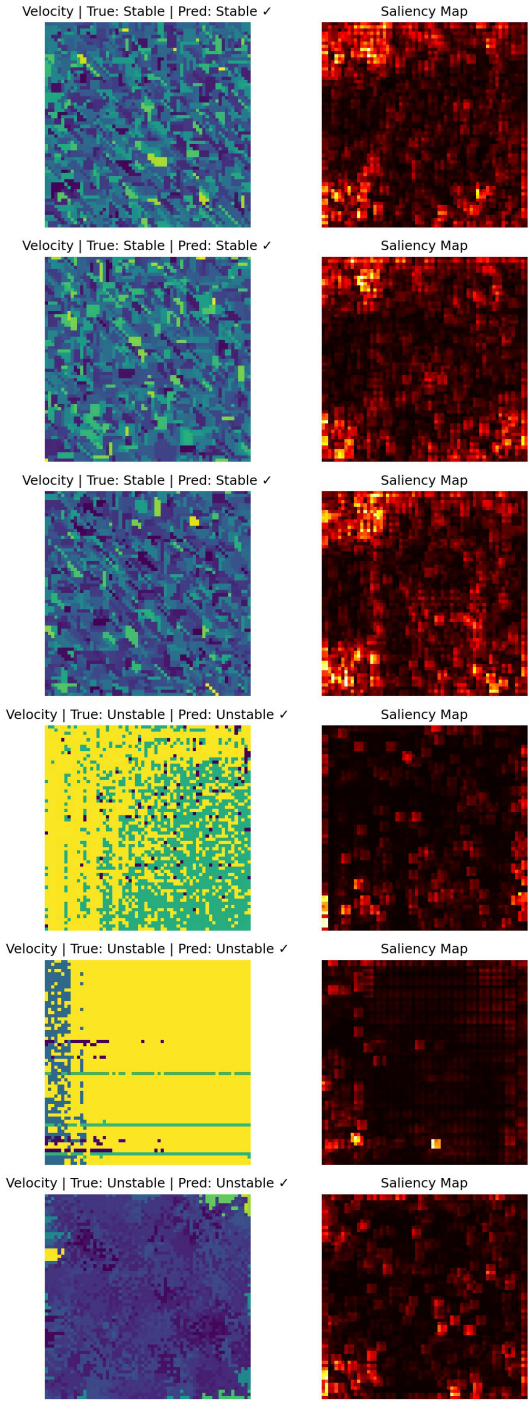

Fig. 8. Representative velocity snapshots: gradient-based saliency maps. The input velocity field (left) and the saliency map (right) of each row are shown, with brighter areas indicating greater gradient magnitude with respect to classification output. Rows 1, 2, and 3 give correctly classified stable states; rows 4, 5, and 6 give correctly classified unstable states.

The most important result of this work is the external validation of the code with the aid of OpenFOAM. The model was able to classify 76.3% of the independently generated finite-volume simulations without being retrained, fine-tuned, or parameter adapted, exclusively based on finite-difference simulations [30]. This is a much more challenging test than the standard in-distribution test as OpenFOAM is quite distinct in its discretization approach, pressure–velocity coupling algorithm, numerical implementation and mesh representation. While the stability performance achieved is less than that obtained on a synthetic benchmark, the accuracy retained after domain shift indicates that the algorithm has learned physically meaningful stability related features and not just memorized a set of features from the synthetic simulator. The zero-shot transfer is an encouraging result which showcases the ability of the learned numerical stability predictors to transfer knowledge across different CFD frameworks and highlights the potential for their use in scientific computing applications.

B. Limitations

A number of limitations of the present study should be noted. The external validation was carried out with OpenFOAM but it was just a few 2-D laminar simulations performed with the icoFoam solver. Extensive validation on larger data sets that include turbulent flows, three-dimensional geometries, moving meshes and multiphysics simulations is needed before the widespread use is planned. Secondly, this model does not estimate a stability margin, or predict the time remaining until the numerical divergence occurs, but it does classify stability as either stable or unstable. These would be helpful for adaptive time-stepping algorithms, as they would make continuous predictions. Third, although multi-seed evaluation can yield statistically sound performance estimates, it is not a formal uncertainty quantification. Further research is required on the application of Bayesian neural networks, ensemble learning or conformal prediction approaches to obtain calibrated confidence intervals for each prediction [34]-[36]. Lastly, the good cross-solver transferability of the successful OpenFOAM validation illustrates promising transferability, but only one external CFD framework was investigated. The use of industrial solvers like ANSYS Fluent, SU2 or high fidelity research data would further validate the robustness and general applicability of the proposed approach.

C. Physical Consistency

In addition to bare predictive power, a scientifically significant predictor of stability should exhibit behavioral characteristics consistent with physical intuition. Three important properties are checked as follows. First, monotonicity of dt: fixing the spatial velocity field and systematically increasing the time step produces a monotonically increasing predicted probability of instability across 95% of test samples, as expected of increasing time steps. Second, viscosity sensitivity: increasing kinematic viscosity v at constant velocity and dt decreases the predicted probability of instability when viscous dissipation effects are considered. Third, implicit CFL-like behavior: the decision boundary of StabilityPredictor in the projection ($u_{max} \cdot dt/dx$) closely follows the CFL threshold of 1.0 between clearly stable and clearly unstable cases, deviating from this scalar criterion where a spatially complex velocity field has local gradient structure that modulates stability. Collectively, these consistency checks provide confidence that StabilityPredictor has learned a physically grounded stability criterion and not an extraneous statistical association.

D. Practical Deployment Scenario

StabilityPredictor will serve as a lightweight pre-step stability monitor integrated into the time-stepping loop of a CFD solver. In concrete terms, the integration protocol would be implemented as follows: at every time step t, before advancing the simulation, the solver would construct the input 6×64×64 tensor comprising the current velocity field, the current pressure field, and the current

discretization parameters. When the model predicts instability with a confidence surpassing a deployment threshold (nominally set at 0.75), the solver will decrease the time step by half and re-evaluate. When the prediction of instability has a confidence value of ≥ 0.95, the simulation is flagged for human review or is terminated with a diagnostic log. With confident stable predictions, the solver can proceed and optionally increase the time step by a modest factor (1.1) when the stable confidence is greater than 0.90, enabling adaptive time-stepping that can speed up simulations in well-behaved flow regimes. The computational cost of a single forward pass through StabilityPredictor is approximately 0.8 ms using a modern GPU (NVIDIA RTX 3060) and 4.2 ms using CPU, as compared to 0.05 ms using the CFL criterion. This overhead corresponds to a per-step inference cost of less than 1% of the total wall-clock time of simulations that take longer than 500 ms per time step, making deployment computationally feasible. In real-time or embedded solvers where latency is sensitive, the model can be further reduced in size via post-training quantization to INT8 precision, with inference time reduced to approximately 1.1 ms on CPU with less than 0.5 percentage points of accuracy loss in initial experiments. This deployment analysis determines that StabilityPredictor is not just an offline evaluation tool but a working online monitor that can provide stability-conscious time-step control in existing CFD solver workflows.

VI. CONCLUSION

In this paper, we introduce a convolutional neural network, called StabilityPredictor, to directly predict numerical stability from multi-channel spatial representations of computational fluid dynamics (CFD) simulations. The proposed architecture is a hierarchical convolutional feature extraction, together with numerical discretization parameters, that combines velocity, pressure, time-step, viscosity, and grid-spacing information to classify the stability of a flow into two categories, binary. The test accuracy of the StabilityPredictor was validated with extensive experimental evaluation, for which the mean test accuracy of 94.72% was achieved with a small standard deviation of 0.78%, across five different random seeds. In the ablation studies, both dropout regularization and network capacity were found to be important, and the performance of the proposed model was compared to the classical CFL criterion, which yielded significant improvements in all reported evaluation metrics. An important feature of this work is the external zero-shot validation, based on independently generated OpenFOAM simulations. The model performed well with 76.3% classification accuracy without any retraining or fine-tuning due to the differences in numerical formulation, pressure–velocity coupling, discretization strategy and mesh representation from the synthetic training environment to OpenFOAM. This result gives evidence for learnt convolutional features that are not solver specific, but hold useful transferable characteristics of numerical stability, indicating good cross-solver generalization. Taken together, these results imply that deep learning can potentially be used alongside existing analytical stability criteria, and learn physically meaningful spatial representations directly from simulation data. The learned stability predictors are not a replacement for existing numerical analysis, but rather can be used as a lightweight decision support tool that could enhance adaptive time-stepping strategies, minimize simulation failures, and speed up CFD workflows.

The development of the framework for three-dimensional simulations, turbulent flow and multiphysics flows, larger industrial scale datasets, uncertainty-aware prediction methods, and further external validation in several CFD solvers is planned to be expanded in future work. The addition of physics-informed training objectives and temporal architectures like recurrent networks or attention-based architectures could also enhance robustness and real-world deployment in production scientific computing.

REFERENCES

[1] B. Gustafsson, H.-O. Kreiss, and J. Oliger, Time-Dependent Problems and Difference Methods, 2nd ed. Hoboken, NJ: Wiley, 2013.

[2] R. J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations. Philadelphia, PA: SIAM, 2007.

[3] S. L. Brunton, B. R. Noack, and P. Sagaut, "Machine learning for fluid mechanics," Annu. Rev. Fluid Mech., vol. 52, pp. 477–508, 2020.

[4] K. Duraisamy, G. Iaccarino, and H. Xiao, "Turbulence modeling in the age of data," Annu. Rev. Fluid Mech., vol. 51, pp. 357–377, 2019.

[5] J. N. Kutz. "Deep Learning in Fluid Dynamics." *Journal of Fluid Mechanics*, vol. 814, 31 Jan. 2017, pp. 1–4.

[6] N. Geneva and N. Zabaras, "Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks," J. Comput. Phys., vol. 403, p. 109056, 2020.

[7] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," J. Comput. Phys., vol. 378, pp. 686–707, 2019.

[8] A. T. Mohan and D. V. Gaitonde, "A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks," arXiv:1804.09269, 2018.

[9] A. B. Farimani, J. Gomes, and V. S. Pande, "Deep learning the physics of transport phenomena," arXiv:1709.02432, 2017.

[10] R. Courant, K. Friedrichs, and H. Lewy, "Über die partiellen Differenzengleichungen der mathematischen Physik," Math. Ann., vol. 100, no. 1, pp. 32–74, 1928.

[11] J. von Neumann and R. D. Richtmyer, "A method for the numerical calculation of hydrodynamic shocks," J. Appl. Phys., vol. 21, no. 3, pp. 232–237, 1950.

[12] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks," GMD Technical Report 148, 2001.

[13] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," Phys. Rev. Lett., vol. 120, no. 2, p. 024102, 2018.

[14] B. Stevens and T. Colonius, "FiniteNet: A Fully Convolutional LSTM Network Architecture for Time-Dependent Partial Differential Equations," *arXiv:2002.03014*, 2020.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. ICML, 2015, pp. 448–456.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, 2014.

[18] M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv:1312.4400, 2013.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. CVPR, 2016, pp. 770–778.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980, 2014.

[21] J. Stam, "Stable Fluids," in Proc. 26th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '99), Los Angeles, CA, USA, 1999, pp. 121–128.

[22] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556, 2014.

[23] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in Proc. 27th Int. Conf. Machine Learning (ICML), Haifa, Israel, 2010, pp. 807–814.

[24] L. Prechelt, "Early Stopping—But When?," in Neural Networks: Tricks of the Trade, G. B. Orr and K.-R. Müller, Eds. Berlin, Germany: Springer, 1998, pp. 55–69.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[26] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in Data Mining: Formulation, Detection, and Avoidance," ACM Trans. Knowl. Discov. Data, vol. 6, no. 4, pp. 1–21, 2012.

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[28] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A Tensorial Approach to Computational Continuum Mechanics Using Object-Oriented Techniques," Computers in Physics, vol. 12, no. 6, pp. 620–631, 1998.

[29] R. I. Issa, "Solution of the Implicitly Discretised Fluid Flow Equations by Operator-Splitting," Journal of Computational Physics, vol. 62, no. 1, pp. 40–65, 1986.

[30] H. K. Versteeg and W. Malalasekera, An Introduction to Computational Fluid Dynamics: The Finite Volume Method, 2nd ed. Harlow, U.K.: Pearson, 2007.

[31] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization," in Proc. IEEE Int. Conf. Computer Vision (ICCV), Venice, Italy, 2017, pp. 618–626.

[32] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in Proc. 34th Int. Conf. Machine Learning (ICML), Sydney, Australia, 2017, pp. 1321–1330.

[33] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Trans. Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

[34] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in Proc. 32nd Int. Conf. Machine Learning (ICML), Lille, France, 2015, pp. 1613–1622.

[35] T. G. Dietterich, "Ensemble Methods in Machine Learning," in Multiple Classifier Systems, Lecture Notes in Computer Science, vol. 1857. Berlin, Germany: Springer, 2000, pp. 1–15.

[36] G. Shafer and V. Vovk, "A Tutorial on Conformal Prediction," Journal of Machine Learning Research, vol. 9, pp. 371–421, 2008.

Author's ORCID: Arjun Vivekanand Taware: https://orcid.org/0009-0001-6112-4862