16th World Congress in Computational Mechanics (WCCM)
21-26 July 2024, Vancouver, Canada
A. Korobenko, M. Laforest, S. Prudhomme, R. Vaziri (Eds)

DEEP CONVOLUTIONAL ARCHITECTURES FOR FORECASTS IN FLOW PROBLEMS

AZZEDDINE SOULAÏMANI[1], PRATYUSH BHATT[2], YASH KUMAR[1], AND MOHAMED MOOSA[1,3]

[1] École de technologie supérieure
1100 Rue Notre-Dame W., Montreal, H3C 1K3, Quebec, Canada
azzeddine.soulaimani@etsmtl.ca

[2] Delhi Technological University
P4X9+Q8X, Bawana Rd, Shahbad Daulatpur Village, Rohini, New Delhi, 110042, Delhi, India

[3] Univeristé de Montréal
2900 Boul Edouard-Montpetit, Montreal QC H3T 1J4, Canada

Key words: Non-intrusive reduced-order modeling; Deep autoencoders; LSTM; TCN; CNN; Attention mechanism, Time-dependent flow problems

Abstract. This article provides a summary of our latest research, where we investigate the application of data-driven deep learning methods to simulate the dynamics of physical systems that are governed by partial differential equations (PDEs). The main challenge is the long-term temporal extrapolation for fluid dynamics problems that exhibit steep gradients and discontinuities. We make use of deep learning techniques, specifically designed for time-series predictions like LSTM, TCN, and Attention mechanism, as well as CNN. These methods are employed to model the dynamics of systems primarily influenced by advection. We propose a combination of a Convolutional Autoencoder (CAE) model for data compression and a novel CNN-based for forecasts. These models take a series of high-fidelity vector solutions and predict the solutions for the following time steps using auto-regression. To reduce complexity and computational demands during both online and offline stages, we implement deep auto-encoder networks. These techniques are used to compress the high-fidelity snapshots before feeding them into the forecasting models. Our models are evaluated on numerical benchmarks, such as the 1D Burgers' equation and Stoker's dam-break problem, to assess their long-term predictive accuracy, even in scenarios that extrapolate beyond the training domain. The model that demonstrates the highest accuracy is subsequently used to simulate a hypothetical dam break in a river with real 2D bathymetry. Due to space constraints, only a selection of results is showcased, with additional findings available in our work [1] and the newer ones will also be presented in the talk. Our findings indicate that the proposed CNN future-step predictor offers significantly accurate forecasts in the considered spatiotemporal problems.

1 Introduction

The process of obtaining numerical solutions of PDEs through traditional high-fidelity computational solvers can be highly costly. Consequently, this approach becomes inefficient for multi-queries applications like optimization and uncertainty quantification, where a large number of simulations are necessary for analysis. Reduced-order models (ROMs) serve as an effective alternative to these computationally demanding numerical solvers.

Deep learning models typically excel in interpolation scenarios, where they can make predictions for unseen inputs that fall within the distribution of the training data. However, during the inference stage of real-world applications, the input data may lie outside the distribution seen during training. This situation is known as extrapolation. In such cases, deep learning models may encounter large errors and even fail due to their inability to handle out-of-distribution inputs effectively. Some recent research has focused on predicting solution instances outside the training domain for various fluid problems (Liu et al [2], Heaney et al.[3], Jacquier et al. [4]).

The remaining challenge lies in long-term temporal extrapolation for fluid problems characterized by sharp gradients and discontinuities. Our study investigates convolutional architectures (specifically LSTM, TCN, Attention mechanism, and CNN) to obtain accurate solutions for time steps far beyond the training domain, focusing on advection-dominated test cases.

The subsequent sections are organized as follows. Section 2 provides details about the dataset structure used for training and testing, followed by a presentation of the autoencoders for space compression and the forecasting convolutional architectures. In Section 3, the models are tested on the one-dimensional Burgers' problem. For more results see [1]. Finally, section 4 presents some concluding remarks.

2 Methodology

2.1 Dataset

The dataset is composed of T solution vectors/snapshots: v^i with n_s nodes ($v^i \in \mathbb{R}^{n_s}$) at time-steps $i \in \{1, 2, \ldots, T\}$ obtained using a high-fidelity PDEs solver. For the autoencoder models, the output is the reconstruction of the input, therefore the training and validation input and output data are snapshot vectors v^i. For the forecasting models (Figure 1), N samples are used for training; in each sample, the input is a sequence of n_t snapshots (lookback window = n_t): $V = [v^{i-n_t+1}, \ldots, v^{i-1}, v^i]$, with $V \in \mathbb{R}^{n_s \times n_t}$, and the corresponding output is the vector at the time-step immediately after the sequence end - $v^{i+1} \in \mathbb{R}^{n_s}$.

For extrapolative testing (Figure 2), a sequence of n_t vectors from the start of the dataset, $V = [v^1, \ldots, v^{n_t-1}, v^{n_t}] \in \mathbb{R}^{n_s \times n_t}$ is fed to the model to produce the vectors at all the subsequent time-steps: $[v^{n_t+1}, v^{n_t+2}, \ldots, v^T] \in \mathbb{R}^{n_s \times (T-n_t)}$ in an auto-regressive manner, i.e, first only a single subsequent snapshot v^{n_t+1} is predicted, which is then concatenated with previous $n_t - 1$ vectors and passed to the forecasting model to produce vector v^{n_t+2}. In summary, we are using an auto-regressive approach to predict future vectors beyond the observed data. The process involves concatenating previously predicted vectors to make subsequent predictions.

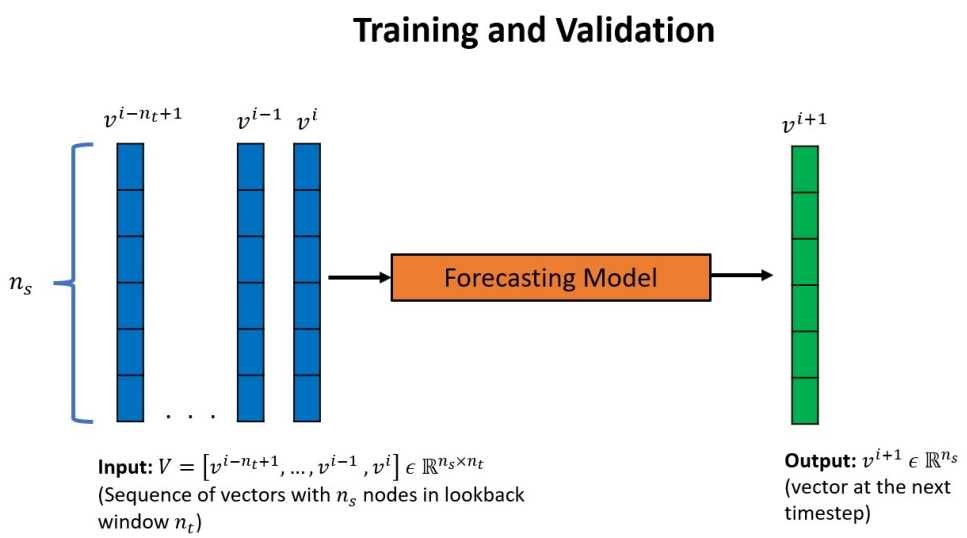


Figure 1: Training and validation method.

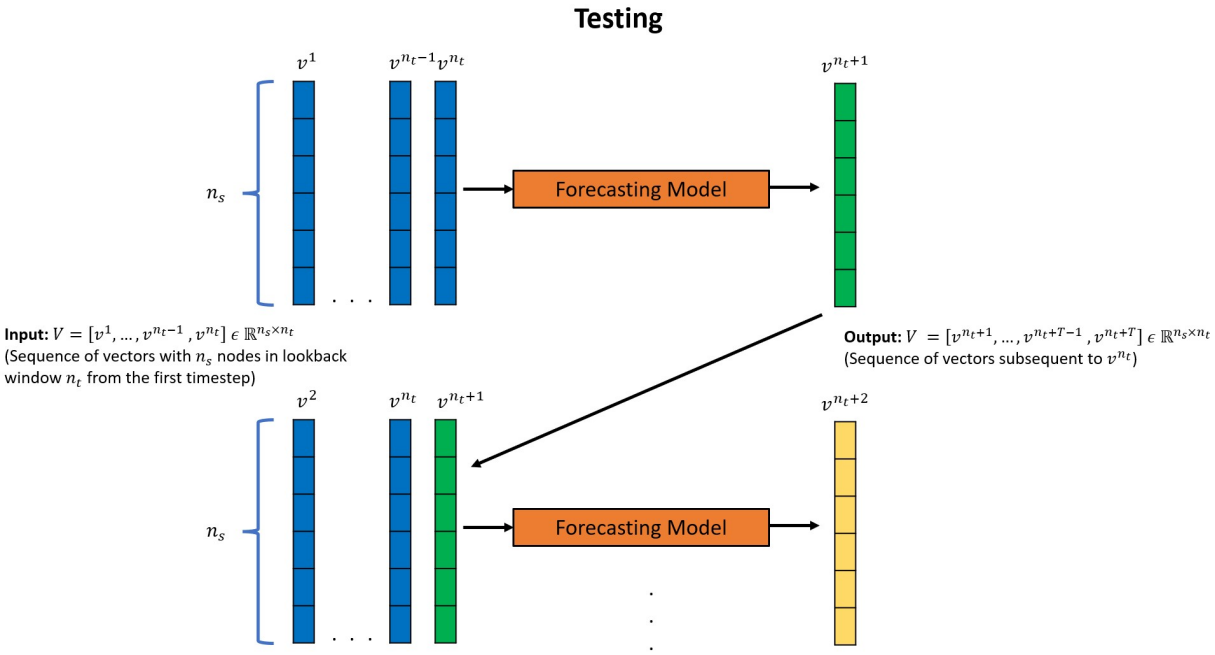


Figure 2: Autoregressive testing method for forecasting models.

2.2 Non-intrusive reduced-order modeling(NIROMs)

NIROMs leverage autoencoders [5] to create data-driven reduced-order models, allowing efficient prediction while avoiding the complexities of directly working with high-fidelity snapshots. An autoencoder learns the approximation of the identity mapping, χ: $v^i \rightarrow v^i_{ae}$ such that

$v^i \approx v^i_{ae}$ and χ: $\mathbb{R}^{n_s} \rightarrow \mathbb{R}^{n_s}$, where n_s is the number of nodes in the solution vector v^i. This process is accomplished using a two-part architecture. The first part of the autoencoder network is the encoder χ_e, which maps a high-dimensional input vector v^i to a low-dimensional latent vector z^i: $z^i = \chi_e(v^i; \theta_e)$ and $z^i \in \mathbb{R}^m$ $(m \ll n_s)$. The second part is called a decoder, χ_d, which maps the latent vector z^i to an approximation v^i_{ae} of the high-dimensional input vector v^i: v^i_{ae} = $\chi_d(z^i; \theta_d)$. The combination of these two parts yields an autoencoder network of the form χ : $v^i \rightarrow \chi_d \circ \chi_e(v^i)$. The restriction (dim($z^i$) = m) $\ll$ (n = dim(v^i)) forces the autoencoder model to learn the salient features of the input data via compression into a low-dimensional space. Problems involving data of high spatial dimension use, generally, convolutional auoencoders CAEs.

2.3 Forecasting Techniques

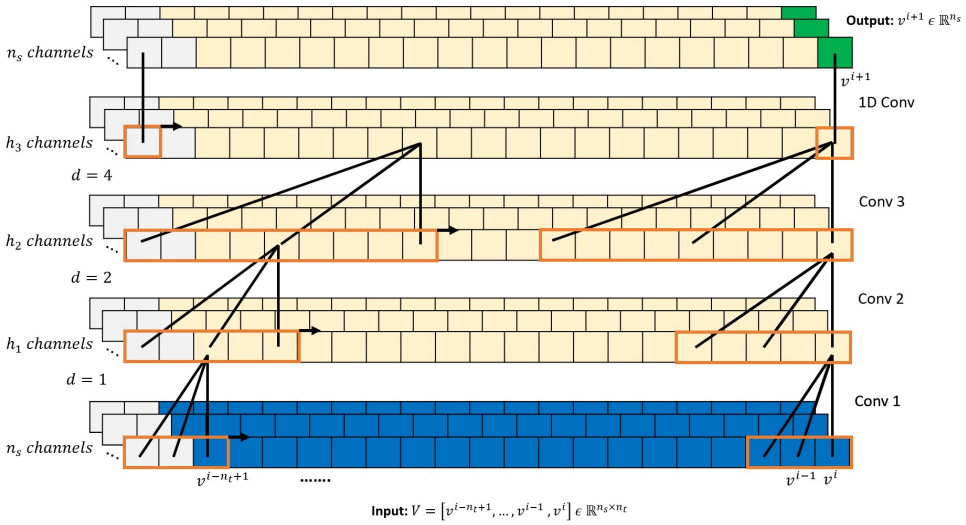
The dataset post compression by the encoder (χ_e) produces N samples of the form: $Z = [z^{i-n_t+1}, \ldots, z^{i-1}, z^i] \in \mathbb{R}^{m \times n_t}$, $z^{i+1} \in \mathbb{R}^m$ *which are used to train the following forecasting models*.

2.3.1 Long Short-Term Memory (LSTM)

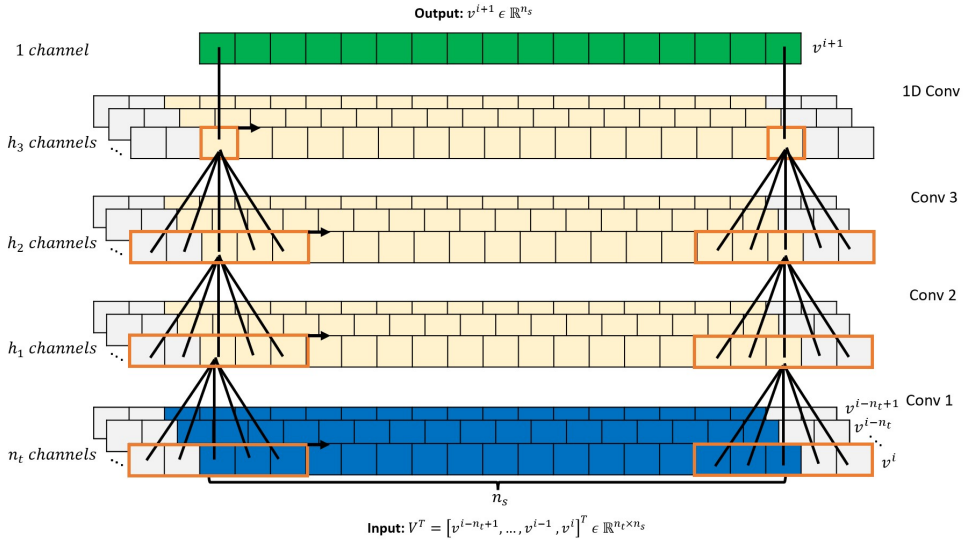
LSTM [6] is a special type of recurrent neural network (RNN) that is well-suited for performing regression tasks based on time series data. The main difference between the traditional RNN and the LSTM architecture is the capability of an LSTM memory cell to retain information over time and an internal gating mechanism that regulates the flow of information in and out of the memory cell [7]. The LSTM cell consists of three parts, also known as gates, that have specific functions. The first part called the forget gate, chooses whether the information from the previous step in the sequence is to be remembered or can be forgotten. The second part called the input gate, tries to learn new information from the current input to this cell. The third and final part, called the output gate, passes the updated information from the current step to the next step in the sequence.

2.3.2 Temporal Convolution Network (TCN)

The Temporal Convolution Network (TCN) operates based on two key principles [8]: The network generates an output that matches the length of the input, and it prevents any future information from leaking into the past. To ensure adherence to the first principle, the TCN employs a 1D fully convolutional network (FCN). In this network, each hidden layer is the same length as the input layer, and zero padding is introduced to maintain the length of subsequent layers consistent with their predecessors. To uphold the second principle, the TCN utilizes causal convolutions, which are achieved by padding only at the beginning of input sequences, where the output at time i is convolved only with elements from time i and earlier in the previous layer (Figure 3). A TCN also makes use of dilated convolutions that enable an exponentially large receptive field.



(a)



(b)

Figure 3: (a) 1D dilated filters convolving on the temporal dimension of vectors (b) 1D CNN filters convolving on the spatial dimension of vectors.

2.3.3 A proposed Convolution Neural Network (CNN) for time forecasting

When convolving along the temporal axis, the (*standard*) TCN model uses information available from all the prior time steps (due to the large receptive field) to evaluate the next time step, as sketched in Figure 3. The model takes in a sequence of n_t vectors corresponding to a look-back window of size n_t : $V = [v^{i-n_t+1}, \dots, v^{i-1}, v^i]$, with $V \in \mathbb{R}^{n_s \times n_t}$. The filters convolve

along the temporal axis for all the n_s vector nodes since the nodes are passed in as channels. However, the results produced from this model do not propagate beyond the training domain. Therefore, another model is proposed here, where the dilated convolutions of the TCN model convolve along the spatial axis, and thus *uses the information available from (or correlation between) the neighboring nodes to determine the future time-step value of the node*. This model takes in a sequence of n_t vectors corresponding to a look-back window of size n_t in a transposed manner, such that the n_t solution vectors are on separate channels: $V^T \in \mathbb{R}^{n_t \times n_s}$, where $V = [v^{i-n_t+1}, \ldots, v^{i-1}, v^i]$. This model produces significantly better results than the TCN on a temporal axis, but the causal padding and dilations employed are of no significance when the convolution filter operates along the spatial axis. Another architecture for modeling the system dynamics, with 1D convolutions and without any dilations or causal paddings is therefore proposed.

The proposed forecasting model of CNN takes in a sequence of vectors with n_t time-steps in a transposed manner as its input: $V^T \in \mathbb{R}^{n_t \times n_s}$, where $V = [v^{i-n_t+1}, \ldots, v^{i-1}, v^i]$, so that the filter convolves on *the spatial dimension* of size n_s, and the n_t vectors lie on separate channels, as shown in Figure 3. The CNN architecture (Figure 3) consists of X residual blocks (X is a hyperparameter), in which the input to each block, after transformation (to make the channels equal) from a 1D Convolution layer (kernel = 1 and channels = 1) is added to the output from the block. A residual block consists of two convolution layers, each followed by a weight normalization and a leaky ReLU activation layer.

2.3.4 Attention Mechanism

The attention mechanism is a sophisticated concept used in neural network architectures, particularly for tasks like natural language processing and time series analysis. Unlike traditional RNNs and LSTMs, which struggle to capture long-range dependencies, attention allows the model to dynamically focus on relevant parts of the input sequence. It operates through three main components: queries, keys, and values. The model calculates a score for each key for the current query, determining the relevance of each input element. These scores are normalized using a softmax function to produce attention weights, which indicate the importance of each input element. The final output is a weighted sum of the value vectors, aggregating the most relevant information from the input sequence [9] .

The primary benefits of the attention mechanism include its ability to capture long-range dependencies and improve interpretability by highlighting which parts of the input the model focuses on. This mechanism enhances the model's performance by ensuring that the most pertinent information is prioritized during each step of the sequence processing. Additionally, attention provides insights into the decision-making process of the model, making its operations more transparent and understandable. By allowing the model to selectively prioritize different parts of the input sequence, the attention mechanism has revolutionized sequence modeling, leading to significant improvements in tasks such as machine translation, text generation, and time series prediction [10].

3 Numerical tests: 1D Burgers' problem

The test case involves the one-dimensional Burgers' equation along with the initial and Dirichlet boundary conditions given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial t^2} \tag{1}$$

$$x \in [0, L], u(0, t) = 0 \tag{2}$$

$$u(x, 0) \equiv u_0 = \frac{x}{1 + \sqrt{\frac{1}{t_0}}\exp(Re\frac{x^2}{4})} \tag{3}$$

where the length $L = 1$ and the maximum time $T_{max} = 2$. The solutions obtained from the above equations produce sharp gradients even with smooth initial conditions if the viscosity ν is sufficiently small. The analytical solution to the problem is given by:

$$u(x, t) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{t_0}}\exp(Re\frac{x^2}{4t+4})} \tag{4}$$

where $t_0 = \exp(\frac{Re}{8})$ and $Re = 1/\nu$. High-fidelity solution vectors are produced by directly assessing the analytical solution across a uniformly discretized spatial domain with 200 grid points ($n_s = 200$) at 250 uniform time-steps ($T = 250$) for two distinct Revalues: 300 and 600. These solution vectors are subsequently utilized to train the autoencoder and forecasting models.

For the autoencoder training, 200 solution vectors are randomly selected at different time steps, while the remaining 50 are used for validation. The training set for the forecasting model consists of the first 150 compressed samples, with each sample containing n_t consecutive solution vectors (i.e., the lookback window equals n_t), where n_t is a hyperparameter. The validation set includes the next 10 samples. For testing, n_t latent vectors from the beginning of the dataset are input into the forecasting model to predict the subsequent time steps using auto-regression (see Table 1).

To evaluate the performance of the previous architectures, the following metrics are used: Mean Squared Error (L_2 norm) MSE: The average of the square of the difference between the actual and predicted values over N samples. Mean Absolute Error ((L_2 norm) MAE, and {Relative L_2 Norm Error: The relative L_2 norm error (referred to as error).

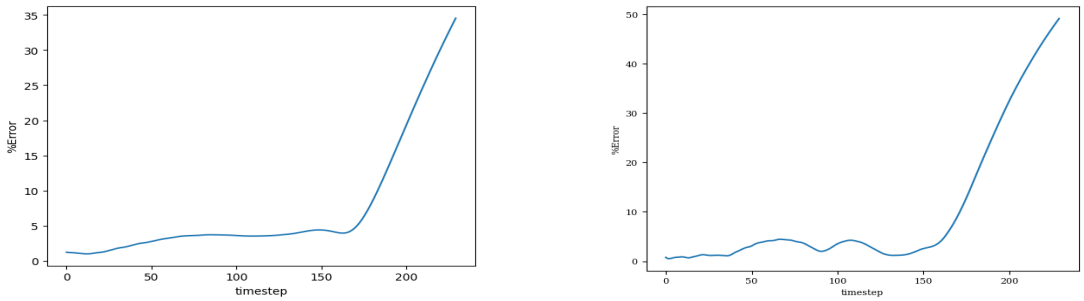
In the following, the encoder comprises two layers with 8 and 32 channels respectively, and a latent dimension of 50. The 1D convolution has a kernel size of 3, and each layer includes padding of size 1. All the hidden layers employ swish activation.

3.1 LSTM model

Various sets of hyperparameters considered for the LSTM network, for both Re= 300 and 600 are summarized in Table 2.

Dataset	Samples	Input	Output
Training	1	$[z^1, \ldots, z^{n_t-1}, z^{n_t}]$	z^{n_t+1}
	2	$[z^2, \ldots, z^{n_t}, z^{n_t+1}]$	z^{n_t+2}
	...	...	...
	150	$[z^{150}, \ldots, z^{n_t+148}, z^{n_t+149}]$	z^{n_t+150} (training end)
Validation	151	$[z^{151}, \ldots, z^{n_t+149}, z^{n_t+150}]$	z^{n_t+151}
	...	...	...
	160	$[z^{160}, \ldots, z^{n_t+158}, z^{n_t+159}]$	z^{n_t+160}
Testing	1	$[z^1, \ldots, z^{n_t-1}, z^{n_t}]$	$[z^{n_t+1}, \ldots, z^{249}, z^{250}]$

Table 1: Burgers' problem: Training, Validation and Testing dataset.



(a) Re = 300 (b) Re = 600

Figure 4: Burgers' problem: L_2 relative error of the autoregressive predictions with increasing time for Re = 300 (a) and Re = 600 (b) for the LSTM model.

The models are trained in batches of size 15. The model with the least validation loss has a lookback window of size 10 and a single LSTM layer with hidden dimension 50 for both Re= 300 and 600. The extrapolation error plots (Figure 4) obtained from these models show that the LSTM model accurately predicts the solution vectors for time steps within the training domain ($i \leq 150$), but the solution does not change for time steps outside the training domain, and so the relative error increases drastically.

3.2 TCN model

The TCN model consists of multiple TCN blocks, each with the same kernel size and number of channels, but with dilations increasing by a factor of 2 in subsequent blocks. The hyperparameters for the TCN network for both Re= 300 and 600 are summarized in Table 3.

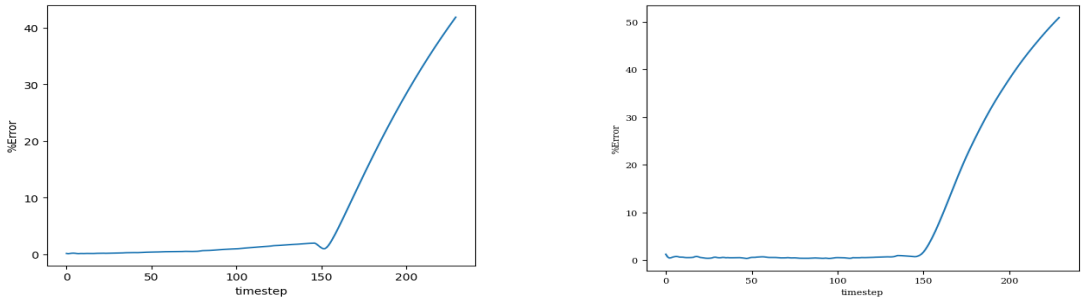
For Re= 300, the best model has 3 temporal blocks, each having 64 channels, whereas, for Re= 600, it has 2 TCN blocks with kernel size 3 and 64 channels each. The error plots (Figure 6) derived from these models suggest that the TCN model precisely predicts the solution vectors for time steps within the training domain, but its accuracy diminishes after training concludes.

Hyperparameters	Values
Sequence length (n_t)	5, 10, 20
LSTM layers	1, 2, 3
hidden/latent dimension (m)	12,25,50
Loss Function	MSE

Table 2: Burgers' problem: Hyperparameters for the LSTM network.

Hyperparameters	Values
Sequence length (n_t)	5, 10, 20
TCN block channels	[32, 32], [64, 64], [32, 32, 32], [64,64,64]
latent dimension (m)	12, 25, 50
Kernel Size(k)	3, 5, 7, 9
Loss Function	MSE

Table 3: Burgers' problem: Hyperparameters for the TCN network.



(a) Re = 300 (b) Re = 600

Figure 5: Burgers' problem, L_2 relative error of the auto-regressive predictions with increasing time for Re = 300 (a) and Re = 600 (b) for the TCN model (over time).

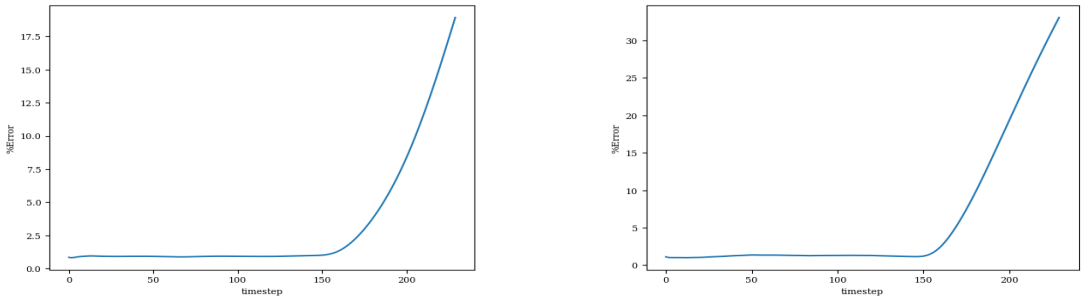
Interestingly, when the same model architecture is applied to the input sequence, such that the dilated 1D convolutions spread along the spatial axis with each solution vector on a distinct channel, it yields accurate forecasts, even beyond the training domain. This finding promotes the creation of a more straightforward predictive/forecasting model, one that is free of dilations and causal padding, as the exponentially expanding receptive field becomes redundant.

3.3 The Attention Mechanism

Our experiments, detailed in Table 4, tested configurations to ensure proper attention mechanism functioning. We chose the number of attention heads as a divisor of the latent dimension, varying both to find optimal performance.

Hyperparameters	Values
Sequence length (n_t)	5, 10, 20
Number of Attention Heads	1,2,5,10,25
latent dimension (m)	12,25,50
Loss Function	MSE

Table 4: Burgers' problem: Hyperparameters for the Attention mechanism.



(a) Re = 300 (b) Re = 600

Figure 6: Burgers' problem, L_2 relative error of the auto-regressive predictions with increasing time for Re = 300 (a) and Re = 600 (b) for the Attention model (over time).

The best setup was 10 heads and a latent dimension of 50, significantly outperforming LSTM and TCN networks in capturing temporal dependencies. The attention mechanism helped focus on relevant input parts, improving learning and prediction. However, the model struggled to extrapolate time steps beyond the training domain, limiting its generalization to unseen future data.

3.4 The proposed CNN model

The CNN model consists of two residual blocks, thus compressing the spatial dimension to 50 in the latent vector. Each block possesses the same kernel size and number of channels. The hyperparameters for the CNN network for both Re=300 and 600 are summarized in Table 5.

The model with the least validation loss has a lookback window of size 10, and each of its blocks has a kernel size of 3 and 50 channels for both Re= 300 and 600. It is clear from the extrapolation and error plots (Figure 7 and 8) that the CNN model accurately models the latent dynamics, and predicts solution vectors accurately for time steps beyond the training domain.

4 Conclusion

This study proposes a Convolutional Autoencoder (CAE) model for compression and a novel CNN future-step predictor for forecasting vector solutions for subsequent time steps. The approximation accuracy and time extrapolation capabilities of the model are evaluated using the

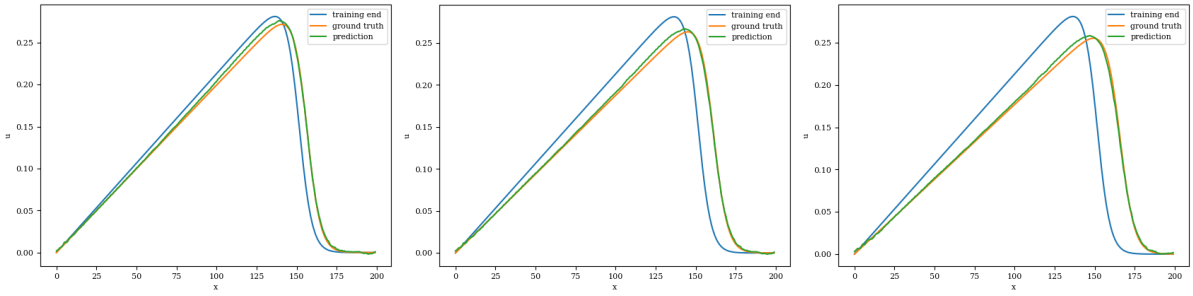
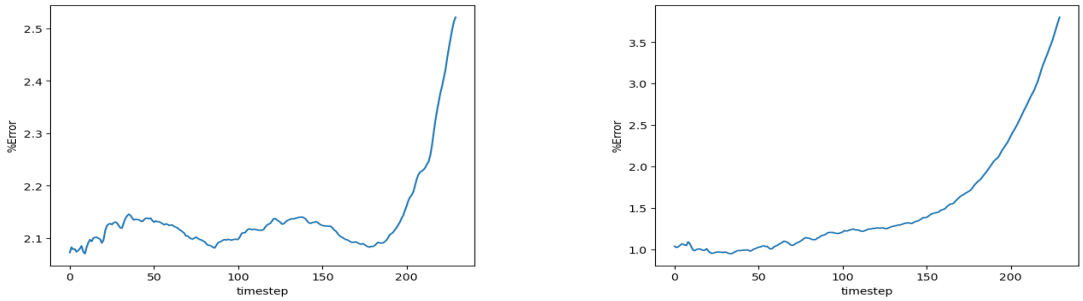


Figure 7: Burgers' problem: Extrapolative auto-regressive predictions using the proposed CNN model, for Re= 300 and for time-steps = 180, 200 and 220; the training end time-step = 160.



(a) Re = 300 (b) Re = 600

Figure 8: Burgers' problem: L_2 relative error of the auto-regressive predictions using the CNN model with increasing time for Re = 300 (a) and Re = 600 (b).

Burgers' problem. More results can be found in [1]. The well-known models built especially for time-series forecasts (LSTM and TCN), produce acceptable results within the training domain, but the solution stops changing during extrapolation. However, when the dilated convolutions propagate on the spatial axis, the models produce good predictions for extrapolation as well. Additionally, since the proposed CNN model uses convolutions, it allows parallel training and evaluation for long input sequences, unlike LSTM. Future work will enhance the model by integrating physical mechanisms during the forecasting phase.

5 Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada and MITACS. The computing resources were provided by the Digital Research Alliance of Canada. The source codes developed are available at https://github.com/Yash-11/lstm_ROM.

REFERENCES

[1] P. Bhatt, Y. Kumar, A. Soulaimani, Deep convolutional architectures for extrapolative forecasts in time-dependent flow problems, Advanced Modeling and Simulation in Engi-

Hyperparameters	Values
Sequence length (n_t)	5, 10, 20
CNN block channels	[50, 50], [100, 100], [200, 200]
latent dimension (m)	12,25,50
Kernel Size(k)	3, 5, 7, 9
Loss Function	MSE

Table 5: Burgers' problem: Hyperparameters for the CNN network.

neering Sciences 10 (11 2023). doi:10.1186/s40323-023-00254-y.

[2] C. Liu, R. Fu, D. Xiao, R. Stefanescu, P. Sharma, C. Zhu, S. Sun, C. Wang, Enkf data-driven reduced order assimilation system, Engineering Analysis with Boundary Elements 139 (2022) 46–55. doi:10.1016/j.enganabound.2022.02.016.

[3] C. E. Heaney, Z. Wolffs, J. A. Tómasson, L. Kahouadji, P. Salinas, A. Nicolle, I. M. Navon, O. K. Matar, N. Srinil, C. C. Pain, et al., An ai-based non-intrusive reduced-order model for extended domains applied to multiphase flow in pipes, Physics of Fluids 34 (5) (2022) 055111. doi:10.1063/5.0088070.

[4] P. Jacquier, A. Abdedou, V. Delmas, A. Soulaïmani, Non-intrusive reduced-order modeling using uncertainty-aware deep neural networks and proper orthogonal decomposition: Application to flood modeling, Journal of Computational Physics 424 (2021) 109854. doi:10.1016/j.jcp.2020.109854.

[5] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, `http://www.deeplearningbook.org`.

[6] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–80. doi:10.1162/neco.1997.9.8.1735.

[7] S. Theodoridis, Machine learning a Bayesian and Optimization Perspective, Elsevier, Academic Press, 2020.

[8] S. Bai, J. Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv:1803.01271 (2018).

[9] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate (2016). arXiv:1409.0473.

[10] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention (2016). arXiv:1502.03044.