Deep Learning-Based Reduced Order Modeling for the Navier-Stokes Equation

Numerical Analysis for Machine Learning - A.Y. 2023/2024

Luigi Pagani High-Performance Computing Engineering
Politecnico di Milano - Milan, Italy
Email: luigi2.pagani@mail.polimi.it
Student ID: 10677832

CONTENTS

I. Introduction and Project Goal

This report explores the application of deep learning-based reduced order models (DL-ROMs) to efficiently solve the Navier-Stokes equations, which are fundamental in fluid dynamics. We focus on using autoencoders for reduced order modelling to approximate high-dimensional PDE solutions with low-dimensional representations, significantly reducing computational costs.[4] The Navier-Stokes equations present a challenging case study for reduced-order modelling due to their nonlinear nature and the wide range of scales involved in fluid flow. By applying the DL-ROM to these equations, we aim to demonstrate the potential of this approach in handling complex, nonlinear PDEs efficiently. Our study will investigate the accuracy and computational efficiency of DL-ROM compared to high-fidelity numerical solutions of the Navier-Stokes equations.

II. Deep Learning-Based Reduced Order Models (DL-ROMs)

A. Overview

Deep learning-based reduced order models (DL-ROMs) are fully data-driven approaches designed to approximate the solution of parameterized partial differential equations (PDEs) using deep learning architectures. In this context, we will focus on a specific type of DL-ROM that utilizes autoencoders to reduce the dimensionality of the problem and learn the parameter-to-solution map efficiently.

Given a parameter space $\Theta \subset \mathbb{R}^p$ associated with a parameterized PDE and a high-fidelity space $\mathbb{R}^{N_h}$, we aim to find a reduced order representation with a latent dimension $N \ll N_h$. To achieve this, we define three neural networks:

$$\Psi' : \mathbb{R}^{N_h} \to \mathbb{R}^n,$$
$$\Psi : \mathbb{R}^n \to \mathbb{R}^{N_h},$$
$$\phi : \mathbb{R}^p \to \mathbb{R}^n,$$

where Ψ' is the encoder, Ψ is the decoder, and ϕ is the parameter-to-latent space map.

The networks are trained to satisfy the following conditions:

$$\mathbf{u}_\mu \approx \Psi(\Psi'(\mathbf{u}_\mu)), \quad \text{and} \quad \phi(\mu) \approx \Psi'(\mathbf{u}_\mu),$$

where $\mathbf{u}_\mu$ is the high-fidelity solution corresponding to the parameter $\mu \in \Theta$.

The training process involves two main phases:

1) **Autoencoder Training**: Train the autoencoder, composed of the encoder Ψ' and the decoder Ψ, to minimize the reconstruction error of the high-fidelity solutions. This step ensures that the autoencoder can effectively compress and reconstruct the high-fidelity data.
2) **Latent Space Mapping Training**: Freeze the encoder and decoder networks. Train the parameter-to-latent space map ϕ to minimize the error in mapping the parameters to the corresponding latent representations obtained from the encoder.

After these two training phases, the final model is expressed as:

$$\mathbf{u}_\mu \approx \Psi(\phi(\mu)).$$

In the case of time-dependent problems, the parameter vector is extended to include the time variable t. That is, $\mu = [\tilde{\mu}, t]$, where $\tilde{\mu}$ represents the PDE parameters and t is treated as an additional parameter.

III. Application 1: Navier Stokes Boundary Value Problem

We are modelling the Navier-Stokes equations:

$$\begin{cases} \rho\left(\dfrac{\partial v}{\partial t} + v\cdot\nabla v\right) - \varepsilon\Delta v + \nabla q = 0 & \text{in } \Omega\times(0,T], \\ \nabla\cdot v = 0 & \text{in } \Omega\times(0,T], \\ v = g & \text{on } \Gamma_{\text{in}}\times(0,T] \cup \Omega\times\{0\}, \\ v = 0 & \text{on } \partial\Omega\setminus(\Gamma_{\text{in}}\cup\Gamma_{\text{out}})\times(0,T], \\ q = 0 & \text{on } \Gamma_{\text{out}}\times(0,T] \end{cases}$$

Here, $\varepsilon > 0$ is the dynamic viscosity and $\rho > 0$ is the fluid density, whereas $v : \Omega\times[0,T] \mapsto \mathbb{R}^2$ is the fluid velocity field. For the sake of simplicity, we shall focus on the scalar field.

$$u := |v|$$

describing the velocity magnitude across Ω, where $|\cdot|$ is the Euclidean norm. The solution is approximated using a $P2$ discretization in space, with the results returned for $N_t = 141$ equispaced timesteps in the interval $[0,T] = [0,3.5]$. Due to the computational expense of the FOM solver, we initialize our study with a dataset comprising 41 simulated trajectories, generated using randomly sampled parameters within the ranges $\log_{10}\varepsilon \in [-3.5,-2.5]$ and $\rho \in [0.5,1]$.

IV. Training Data

For the Navier-Stokes problem, the dataset consisted of 41 trajectories, each spanning 141 time steps. Out of these, 25 trajectories (totalling $25\times 141 = 3525$ snapshots) were used for training, and the remaining 16 trajectories, totalling $16\times 41 = 2256$ snapshots, were reserved as a validation set .

V. Choice of the dimension of the latent space

To determine the optimal dimension of the latent space, we rely on the theoretical foundation provided by the following theorem. Prior to presenting the theorem, we first establish several key definitions:

Continuous Nonlinear Kolmogorov n-width This measure, denoted as $\delta_n(S)$, quantifies how well a set S can be approximated by an n-dimensional nonlinear manifold:

$$\delta_n(S) = \inf_{\substack{\Psi'\in C(S,\mathbb{R}^n) \\ \Psi\in C(\mathbb{R}^n,\mathbb{R}^{N_h})}} \sup_{u\in S} \|u - \Psi(\Psi'(u))\|$$

Where Ψ' is an encoding function and Ψ is a decoding function.

Minimal Latent Dimension The minimal latent dimension, denoted as $n_{\min}(S)$, represents the smallest number of parameters needed to fully characterize a system's behaviour:

$$n_{\min}(S) := \min\{m \mid \delta_m(S) = 0\}$$

Theorem V.1 (Franco, Manzoni, Zunino, 2023)**.** *[2] Assume the parameter space is compact. Then,*

a) $n_{\min}(S) \geq p$ *if the parameter-to-solution map is continuous and locally injective at some point.*
b) $n_{\min}(S) \leq 2p+1$ *if the parameter-to-solution map is Lipschitz continuous.*
c) $n_{\min}(S) = p$ *if the parameter-to-solution map is continuous and injective.*

Here, p represents the dimension of the parameter space. This theorem establishes conditions relating the minimal latent dimension $n_{\min}(S)$ to the parameter space dimension p, based on properties of the parameter-to-solution mapping.

In our Navier-Stokes problem, we have three parameters: ε (dynamic viscosity), ρ (fluid density), and time t. Thus, the total number of parameters p in our system is 3.

To determine the appropriate dimension for our latent space, we refer to point b of the theorem presented earlier:

$$n_{\min}(S) \leq 2p+1$$

where $n_{\min}(S)$ is the minimal latent dimension and p is the number of parameters.

Assuming the parameter-to-solution map in our Navier-Stokes problem is Lipschitz continuous, we apply this upper bound:

$$n_{\min}(S) \leq 2(3)+1 = 7$$

This result suggests that a 7-dimensional latent space is sufficient to capture the essential features of our system while maintaining a compact representation. By choosing this upper bound, we ensure that we have enough dimensions to represent the complexity introduced by our three parameters, balance model expressiveness with computational efficiency, and adhere to the theoretical guarantees provided by the theorem.

While a smaller latent space might be sufficient, choosing 7 dimensions gives us confidence that we are not underfitting the problem, allowing our deep learning-based reduced order model (DL-ROM) to capture potential nonlinear interactions between parameters and complex temporal dynamics in the Navier-Stokes system.

VI. Neural Networks Architecture

As previously anticipated, our Deep Learning-based Reduced Order Model (DL-ROM) consists of two main components: an autoencoder for dimensionality reduction and a feed-forward neural network (FFNN) for parameter-to-latent space mapping. Both networks are implemented using PyTorch and trained on GPU.

A. Autoencoder Architecture

The autoencoder is designed to reduce the high-dimensional solution space to a low-dimensional latent space. With *nh* we indicate the size of the original Finite Element space, and N the dimension of the latent space, set as 7 in this case.

Encoder

Layer	**Specification**
Input layer	Dense(*nh*, 60)
Output layer	Dense(60, N)

Decoder

Layer	**Specification**
Input layer	Dense(N, 50)
Hidden layer	Dense(50, 100)
Output layer	Dense(100, *nh*, activation=None)

The decoder component of the autoencoder is designed with a higher number of parameters compared to the encoder. This architectural choice reflects the inherent complexity of reconstructing the original high-dimensional

data from the compressed latent space representation, which is a more challenging task than the initial encoding process. The autoencoder's weights are initialized using the He initialization method to facilitate effective training. The model undergoes a training process spanning 1250 epochs, utilizing the L^2 error as the loss function for the training.

Loss Function

For training the autoencoder, we utilize the L^2 loss function rather than the more common Mean Squared Error (MSE). This choice is motivated by the irregular nature of our mesh. The mesh used in our Finite Element Method (FEM) solution is not uniform, with some areas of the domain having a denser mesh and others being coarser. Using MSE on the weights of the FEM basis functions would not accurately reflect the difference between the estimated and correct function in an integral sense, due to this mesh irregularity. The L^2 loss, defined as the integral of the squared difference between the true and predicted functions over the domain, provides a more accurate measure of the error for irregular meshes. While computationally more expensive to calculate, the L^2 loss ensures that our error metric is consistent with the underlying mathematical formulation of the problem, regardless of mesh density variations.

B. Feed-Forward Neural Network (FFNN) Architecture

The Feed-Forward Neural Network (FFNN) maps the input parameters to the latent space representation. It incorporates Fourier features to capture periodic patterns in the data. In the following tables, the variable p represents the number of parameters from the original Boundary Value problem. It's important to note that this count does not include time, which is treated as an additional parameter in our model.

Layer	**Specification**
Input layer	Fourier($fmodes$)
Hidden layer	Dense(p+1+2*$fmodes$, 750)
Output layer	Dense(750, N, activation=None)

Where:

- $fmodes = 15$ (number of Fourier modes)
- p is the number of input parameters

The FFNN is also initialized using the He method and trained for 2000 epochs using MSE loss with Euclidean distance.

C. Fourier Features in the Feed-Forward Neural Network

Our Feed-Forward Neural Network (FFNN) incorporates Fourier features as an embedding layer, specifically designed to enhance the model's ability to capture complex temporal patterns in the Navier-Stokes solution data. The Fourier features are applied exclusively to the time parameter, leaving the other input parameters (ε for dynamic viscosity and ρ for fluid density) unchanged.

For the time parameter t, we compute Fourier features as:

$$[\cos(0t), \sin(0t), \cos(1t), \sin(1t), \ldots, \cos(14t), \sin(14t)] \tag{1}$$

where the frequencies range from 0 to 14. These features, along with the original three inputs (ε, ρ, and t), are then fed into the dense layers. We use 15 Fourier modes, adding 30 features to the time parameter.

This approach results in a total of 33 features: 2 original parameters (ε and ρ), the original time parameter (t), and 30 Fourier features derived from time. By applying Fourier features solely to the time dimension, we enable the network to more effectively capture periodic and multi-scale temporal dynamics inherent in the Navier-Stokes equations, while maintaining the original representation of the physical parameters.

The use of Fourier features can potentially improve the model's ability to generalize across different time scales and capture complex temporal behaviors in fluid dynamics simulations.

Data normalization in the latent space

Before training the FFNN, we normalize the latent space representations obtained from the encoder:

$$\nu = \frac{\text{encoder}(u) - \text{mean}(\text{nu}_{\text{train}})}{\sqrt{\text{var}(\text{nu}_{\text{train}})}}$$

This normalization helps stabilize the training process and can lead to better convergence.

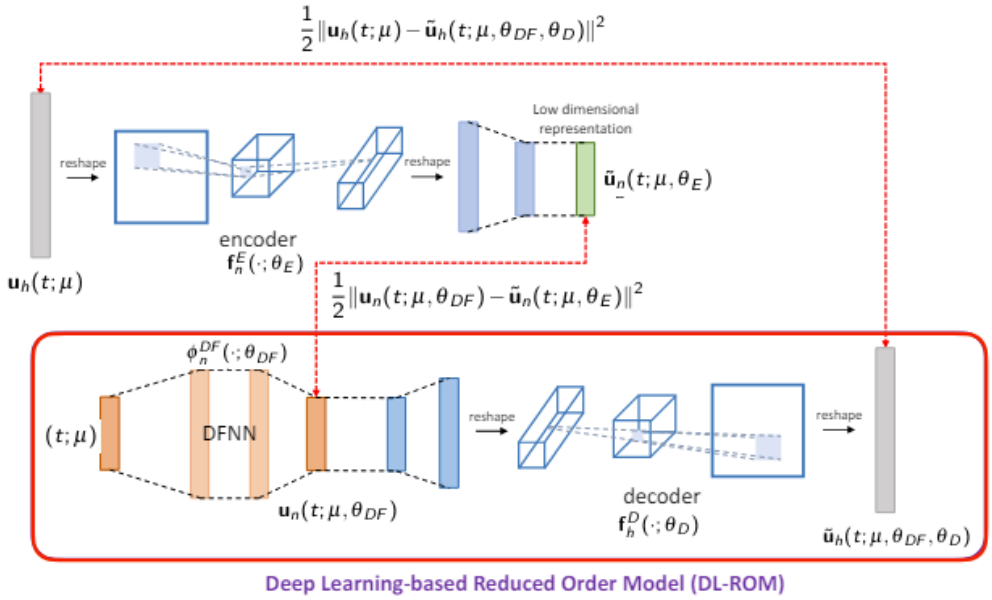

Fig. 1. Architecture of the DL-ROM[4]

VII. Accuracy and Computational Cost Comparison between the ROM and the DL-ROM

In the following figures, we present the simulation results for selected parameter pairs. These results comprise solutions derived from both the Finite Element Method (FEM) and the Deep Learning Reduced Order Model (DL-ROM). Additionally, we include the error visualization at the final time step of each simulation, as our analyses suggest that errors escalate in the later stages of the simulations.

Example 1

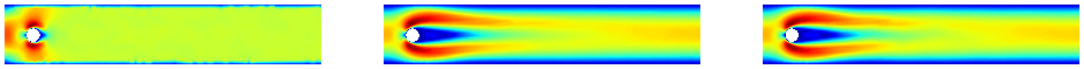

Fig. 2. **Example.1** Finite Element Solution at time $0s, 1.5s$ and $3.5s$

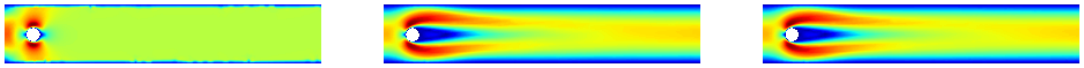

Fig. 3. **Example.1** DL-Rom Solution at time $0s, 1.5s$ and $3.5s$

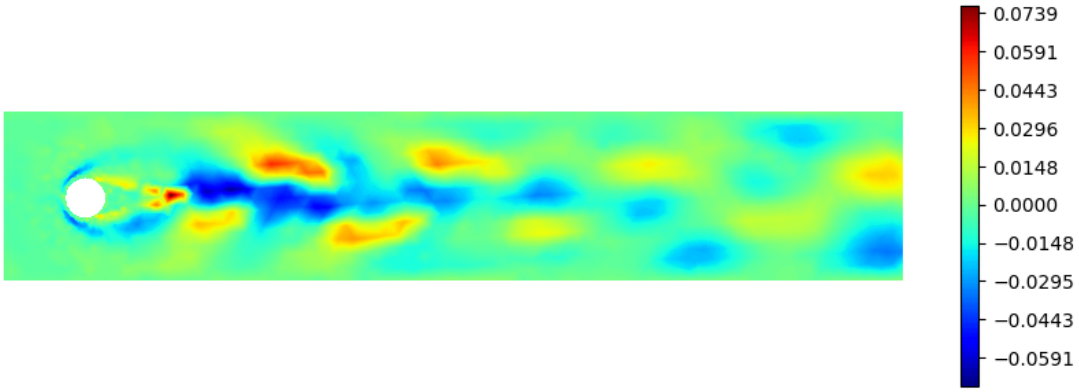

Fig. 4. **Example 1** Error at final time step $3.5s$

Example 2

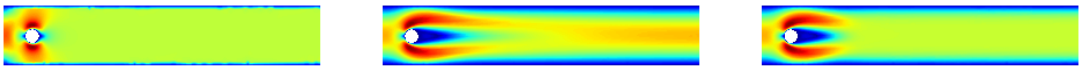

Fig. 5. **Example 2** Finite Element Solution at time $0s, 1.5s$ and $3.5s$

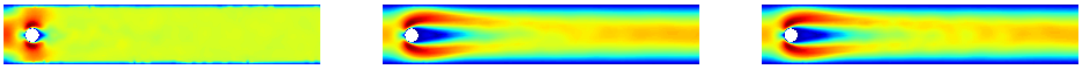

Fig. 6. **Example 2** DL-Rom Solution at time $0s, 1.5s$ and $3.5s$

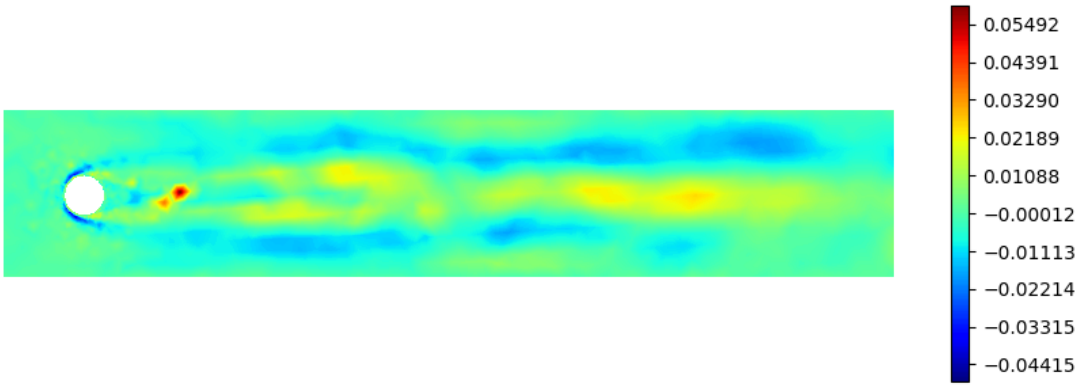

Fig. 7. **Example 2** Error at final time step $3.5s$

The following graph illustrates the average relative error of the model's predictions on the validation dataset across different time steps. The blue line represents the mean error, while the shaded blue area indicates the 25-75% quantile range.

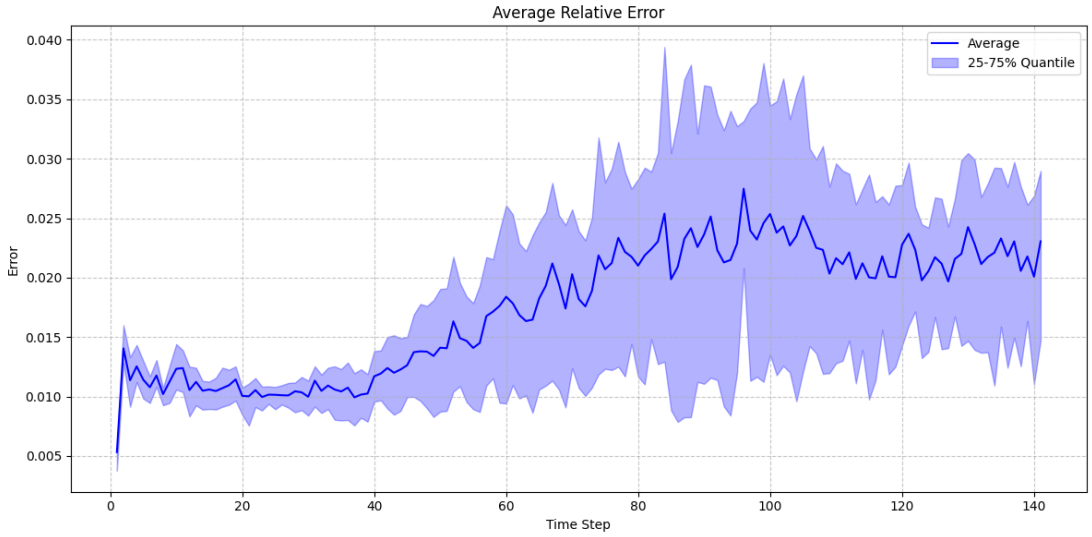

Fig. 8. Error as a function of time for the validation dataset

Key observations:

- **Error trend:** The average relative error generally increases over time, starting low (around 1%) and rising to approximately 2-2.5% by the end of the simulation.
- **Variability:** The error's variability, as shown by the quantile range, also increases with time, suggesting greater uncertainty in later predictions.
- **Peak error:** The highest average errors occur around time steps 80-100, reaching close to 3%.

The average error across all time steps is 1.67%.

VIII. Potential Improvements

Three key areas for improving our DL-ROM approach for Navier-Stokes are:

1) **Time-aware Architectures:** Explore recurrent neural network architectures (e.g., LSTMs) to better capture temporal dependencies in Navier-Stokes solutions, ensuring smoother time evolution. This approach could help address the temporal discontinuities inherent in our current model, which allows querying at specific time points without full time-stepping. LSTMs or similar architectures could provide a more continuous and physically consistent representation of the solution's temporal evolution.[1]
2) **Mesh-Informed Neural Networks (MINNs):** Implement mesh-informed neural network architectures to reduce the number of parameters and better capture the underlying physics of our problems. MINNs incorporate the mesh structure directly into the network design, allowing for the following:[3][7]
   - More efficient parameter usage by exploiting the known geometry and connectivity of the problem domain.
   - Better preservation of local conservation laws and boundary conditions, which is crucial for accurate fluid dynamics simulations.
3) **Introduction of POD Layer:** Incorporate a Proper Orthogonal Decomposition (POD) layer into our DL-ROM architecture to combine the benefits of traditional model reduction techniques with deep learning approaches.[5]
   - **Significant Parameter Reduction:** The primary achievement would be a substantial reduction in the number of neural network parameters. By implementing a POD-Galerkin layer as part of the encoder network, the input to subsequent neural network layers would be the reduced space obtained from the POD-Galerkin method, rather than the original high-dimensional space from the finite element method. This layer would perform POD on the input data, extracting the most important spatial modes and significantly reducing the dimensionality of the input space.

- **Physics-Informed Reduction:** The POD layer provides a physics-informed approach to dimensionality reduction, potentially improving the model's ability to capture the underlying physical behaviour of the system while working with a more compact representation.
- **Hybrid Model:** By combining POD with our existing autoencoder architecture, we create a hybrid model that leverages both data-driven and physics-based reduction techniques, potentially leading to more efficient and accurate representations.
- **Enhanced Interpretability:** The POD modes can offer insights into the dominant spatial patterns of the system, enhancing the interpretability of the reduced-order model while working in a lower-dimensional space.

The decision to forgo Convolutional Neural Networks (CNNs) in this approach warrants explanation. CNNs inherently impose a locality constraint that may not accurately reflect the underlying mesh structure, particularly when there exists no clear correspondence between node enumeration and mesh topology. In such scenarios, CNNs might erroneously establish connections between nodes that are proximate in numbering but topologically distant, while simultaneously neglecting connections between nodes that are adjacently situated in the mesh yet disparately numbered. While CNNs offer the advantage of reduced computational complexity, they may fail to faithfully represent the genuine spatial relationships inherent in complex mesh structures. This potential incongruence between CNN architecture and mesh topology could potentially compromise the neural network's efficacy in capturing and processing the pertinent structural information. Consequently, alternative neural network architectures such as the aforementioned Mesh Informed Neural Networks [3] that do not impose such rigid spatial assumptions may be more appropriate for tasks involving complex mesh geometries.

IX. Conclusion

Deep Learning-based Reduced Order Models (DL-ROMs) demonstrate significant computational advantages in solving high-dimensional PDEs, particularly for the Navier-Stokes equations. The ROM approach achieved a remarkable speedup of $\mathbf{3.21 \times 10^5}$ compared to the traditional FEM solver while maintaining high accuracy with a relative error of only $\mathbf{1.66}\%$ between the ROM and FEM solutions.

Note on CPU vs. GPU Comparison

It is important to note that this comparison regarding computational time involves computations performed on different hardware architectures. The FEM solver was executed on a CPU, while the ROM calculations were conducted on a GPU. This distinction is crucial as GPUs are generally more expensive than CPUs, both in terms of initial investment and operational costs (e.g., power consumption).

Despite potential hardware-related biases, the magnitude of the speedup strongly suggests that the ROM approach offers substantial computational advantages. As a curious note, the cost ratio between the CPU (used for the FEM solver) and the GPU (used for the DL-ROM) was approximately $1:30$ on Google Cloud. However, the observed speedup was on the order of 10^5. This significant disparity highlights the potential benefits of using reduced order models when exact solutions are not necessary.

Advantages and Applications of DL-ROMs

Deep Learning-based Reduced Order Models (DL-ROMs) offer a powerful and efficient approach to solving high-dimensional partial differential equations (PDEs). By leveraging deep learning techniques, these models excel where traditional methods like POD-Galerkin struggle, particularly in high-dimensional, non-linear systems. Unlike full-order models (FOMs), DL-ROMs allow for specific time-point queries without requiring full time-stepping procedures. In optimization problems involving PDEs, DL-ROMs serve as an efficient first-pass solution for parametric optimization. They can quickly narrow the parameter space before more computationally intensive methods are employed. This approach significantly reduces computational costs while maintaining acceptable accuracy for many applications. DL-ROMs can also serve as valuable initialization tools for full-order models (FOMs). By using DL-ROM solutions as initial conditions, the convergence of FOMs can be accelerated, potentially reducing the number of iterations required to reach a final solution.

A. Challenges and Limitations:

However, DL-ROMs face several significant challenges:

1) Lack of inherent physics enforcement: These data-driven models may not naturally respect physical laws or conservation principles without explicit constraints. This can lead to potentially unphysical solutions, especially when extrapolating beyond the training data.
2) Temporal discontinuities: The ability to query specific time points without full time-stepping, while computationally advantageous, may result in inconsistencies in the temporal evolution of solutions. This approach can lead to a lack of guaranteed smoothness, where solutions at different time points may not connect seamlessly, potentially violating the continuous nature inherent in many physical processes.

These drawbacks underscore the need for careful implementation and validation when employing DL-ROMs in scientific and engineering applications. While the computational efficiency and accuracy of DL-ROMs are impressive, addressing the challenges of physical consistency and temporal continuity remains crucial for their reliable application in complex systems.

X. Experimentation with Synthetic Data Training

Following the successful implementation and training of our Deep Learning-based Reduced Order Model (DL-ROM) for the Navier-Stokes equations, we conducted an additional experiment to explore the potential benefits of augmenting our training dataset with synthetically generated data.

Motivation and Approach

With our DL-ROM demonstrating promising results in solving high-dimensional PDEs efficiently, we hypothesized that expanding the parameter space coverage through synthetic data might lead to even better performance. Our approach involved:

1) Generating synthetic data points within the bounds of our original dataset
2) Combining this synthetic data with our original training set
3) Retraining our model on this augmented dataset

This approach is similar to that followed by AlphaFold[6], which also used synthetic data to improve model performance.

Data Generation and Model Retraining

We generated 100 new parameter sets and corresponding time series, increasing our training data volume by approximately five times. To accommodate this larger dataset, we adjusted our training process:

- Reduced autoencoder training epochs from 1250 to 400
- Reduced Feed-Forward Neural Network (FFNN) training epochs from 2000 to 750

Surprising Results

Contrary to our expectations, the incorporation of synthetic data led to a decrease in the DL-ROM's accuracy:

- Autoencoder error increased from 1.41% to 1.87%
- Overall system error rose from 1.66% to 2.14%

These results were particularly surprising given that the synthetic data had an average error of less than 2%, which we initially considered acceptable for training.

Analysis, Insights, and Conclusions

Our experiment on autoencoder resilience to input imperfections yielded unexpected results that challenge common assumptions in machine learning. We investigated how slight perturbations in input data affect the performance of autoencoders trained to learn an identity function—a task that should theoretically be robust to minor input variations. Contrary to expectations, we found that the autoencoder trained on an augmented dataset (which included slightly imperfect inputs) exhibited lower accuracy compared to the model trained on the original, unaugmented data. Specifically, the augmented model's reconstruction error increased across our test set. These findings suggest that in this case, the potential advantages of increased data quantity were outweighed by the drawbacks of lower data quality. The synthetic data appeared to introduce more noise than valuable information into the training process, ultimately degrading the model's performance instead of enhancing it.

REFERENCES

[1] Federico Fatone, Stefania Fresca, and Andrea Manzoni. *Long-time prediction of nonlinear parametrized dynamical systems by deep learning-based reduced order models*. 2022. arXiv: 2201.10215 [math.NA]. URL: https://arxiv.org/abs/2201.10215.

[2] N.R. Franco, A.M., and P. Zunino. "A Deep Learning approach to Reduced Order Modelling of Parameter Dependent PDEs". In: *Math. Comput.* 92.340 (2023), pp. 483–524. DOI: 10.1090/mcom/3781. URL: https://www.ams.org/journals/mcom/2023-92-340/S0025-5718-2022-03781-9/.

[3] N.R. Franco, A. Manzoni, and P. Zunino. "Mesh-Informed Neural Networks for Operator Learning in Finite Element Spaces". In: *J Sci Comput* 97 (2023), p. 35. DOI: 10.1007/s10915-023-02331-1. URL: https://doi.org/10.1007/s10915-023-02331-1.

[4] Simone Fresca, Luca Dede', and Andrea Manzoni. "A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs". In: *Journal of Scientific Computing* 87 (2021), p. 61. DOI: 10.1007/s10915-021-01462-7. URL: https://doi.org/10.1007/s10915-021-01462-7.

[5] Stefania Fresca and Andrea Manzoni. "POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition". In: *Computer Methods in Applied Mechanics and Engineering* 388 (2022), p. 114181. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.114181. URL: https://www.sciencedirect.com/science/article/pii/S0045782521005120.

[6] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2.

[7] Piermario Vitullo et al. "Nonlinear model order reduction for problems with microstructure using mesh informed neural networks". In: *Finite Elements in Analysis and Design* 229 (2024), p. 104068. ISSN: 0168-874X. DOI: https://doi.org/10.1016/j.finel.2023.104068. URL: https://www.sciencedirect.com/science/article/pii/S0168874X23001610.