Accelerating Convergence of Linear Iterative Solvers using Machine Learning

Laurea Magistrale in Mathematical Engineering - Computational Science and Computational Learning

Author: Luca Saverio

Advisor: Prof. Nicola Parolini

Academic year: 2022-2023

Introduction

Iterative Numerical Methods (INMs) are widely used to solve complex mathematical problems in various scientific and engineering applications. However, these methods can be computationally expensive and time-consuming, especially when dealing with large-scale systems. To overcome this challenge, researchers have been exploring the use of Machine Learning (ML) techniques to accelerate the convergence of INMs [4].
ML algorithms can learn patterns from large amounts of data and exploit them to make informed predictions or decisions. In the context of INMs, ML models can learn the relationships between the input parameters and the convergence behavior of the iterative solver.
The Generalized Minimal Residual (GMRES) [3] method is a widely used INM for solving non-symmetric, non-definite positive large linear systems of equations. It was introduced in 1986 by Saad and Schultz [8] as an extension of the Minimal Residual (MINRES) Method to non-symmetric matrices.
However, the GMRES method can be computationally expensive, especially for large-scale problems.
This thesis studies the possibility of using ML techniques to predict an optimal initial guess to then be fed into a GMRES solver, aiming to obtain an acceleration in the convergence.
The use of ML for accelerating INMs is an active area of research, with promising results in various fields such as fluid dynamics, structural mechanics, and computational electromagnetics. As the field of ML expands and continues to advance, it is plausible to expect further developments and improvements in this area, making it possible to look forward to faster and more accurate solutions to complex mathematical problems.

1. The Prediction Algorithm

In the field of computational mathematics, a system of partial differential equations can be approximated to a linear system in the form:

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where A represents the matrix of coefficients, of dimension $n \times n$, in this study only square linear systems will be considered, $\boldsymbol{b}$ is the second member vector, or right hand side (RHS), of size n and $\boldsymbol{x}$ represents the vector of unknowns to be determined, of size n.
The ML algorithm developed in this thesis was

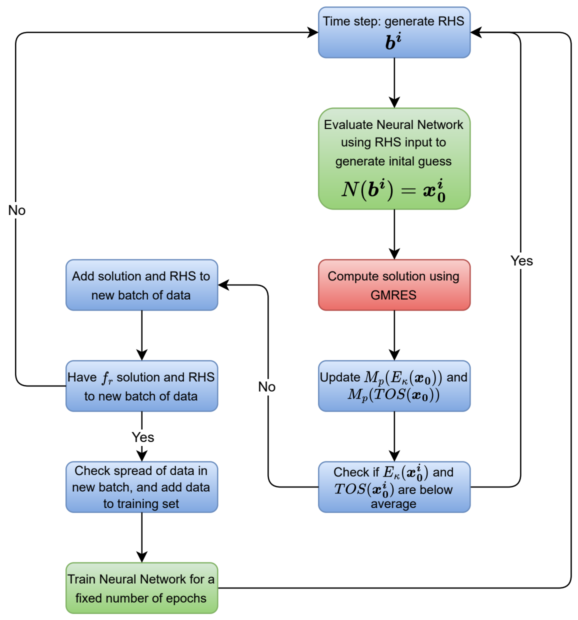
Time step: generate RHS b^i
Evaluate Neural Network using RHS input to generate inital guess $N(b^i) = x_0^i$
Compute solution using GMRES
Update $M_p(E_\kappa(x_0))$ and $M_p(TOS(x_0))$
Check if $E_\kappa(x_0^i)$ and $TOS(x_0^i)$ are below average
Add solution and RHS to new batch of data
Have f_r solution and RHS to new batch of data
Check spread of data in new batch, and add data to training set
Train Neural Network for a fixed number of epochs
No
Yes
No
Yes

Figure 1: Scheme of the ML workflow [4]. In red the sections ran with CPU and in green the ones ran with GPU.

inspired by the work described in: 'Accelerating GMRES with Deep Learning in Real-Time' written by Kevin Luna, Katherine Klymko and Johannes P. Blaschke [4, 5]. In the paper the implementation of a ML accelerated GMRES solver in Python, and using Pytorch, is defined. Figure 1 details the ML workflow. The idea behind the algorithm is to train in real-time a Neural Network (NN), denoted as $N(\boldsymbol{b})$ and taking as input the RHS vector, as linear problems are solved by GMRES and therefore using an Online Learning strategy. This is accomplished by having the NN provide an initial guess $N(\boldsymbol{b}) = \boldsymbol{x_0}$ to the GMRES solver.
The loss function used to train the model is defined as the Mean Squared Error (MSE) between the predicted initial guess and the true solution:

$$L(\boldsymbol{x_0}, \boldsymbol{x}) = \|\boldsymbol{x_0} - \boldsymbol{x}\|^2 = \|N(\boldsymbol{b}) - \boldsymbol{x}\|^2 , \quad (2)$$

where $\|\cdot\|$ denotes the Euclidean norm, which measures the squared distance between $\boldsymbol{x_0}$ and $\boldsymbol{x}$. Minimizing this loss function encourages the network to provide initial guesses that are closer to the numeric solutions.
The first training of the model is performed after a certain *initial set* of problems is solved, that is done in order to generate a first dataset to train on. After the first training the model is retrained after each f_r (*retrain frequency*) additions to the dataset. The size of the initial set and f_r are both inputs of the user.
The dataset used to train the model consists of RHS-solution pairs $\{(\boldsymbol{b^i}, \boldsymbol{x^i})\}$. However, unlike traditional deep learning approaches, the goal here is to train the network in real-time while data is being generated from the simulation. This naturally leads to an online supervised learning problem since at a given time during the simulation, only a finite number of $(\boldsymbol{b^i}, \boldsymbol{x^i})$ pairs are available. In order to ensure a high-quality dataset some time steps are discarded, while only 'high-quality' ones are kept. This operation is performed by computing two quality metrics once a system is resolved. The first is the time to reach convergence, or to obtain the solution, starting from $\boldsymbol{x_0^i}$, $TOS(\boldsymbol{x_0^i})$, while the second metric is the residual at the end of the first restart, $E_\kappa(\boldsymbol{x_0^i})$. These two values are then compared with the averages of the previous p iterations $(M_p(TOS(\boldsymbol{x_0})),\ M_p(E_\kappa(\boldsymbol{x_0})))$. Now, if the new values are worse than the averages, specifically if both are greater, the system is saved into the dataset, as this indicates that the model is not apt to provide good predictions for this type of data.

2. Implemented Neural Network Architectures

Since the Neural Networks implemented in [4] were defined to work on a different class of data and for problems of small dimension overall, it was pivotal to develop new models.
Three different architectures were implemented and tested:

1. Dense NN based Model;
2. Convolutional NN based Model;
3. Mixed model [9]: Combination of Convolutional NNs to process the $\boldsymbol{b}$ vector and Graph NNs to process the matrix A.

2.1. Dense Neural Network based Model

As shown by Figure 2 the implemented DNN-based architecture is generated as a sequance of linear operations and activations functions, in this case the ELU function was used. The iterative application of these layers and activations enables the network to learn and represent the

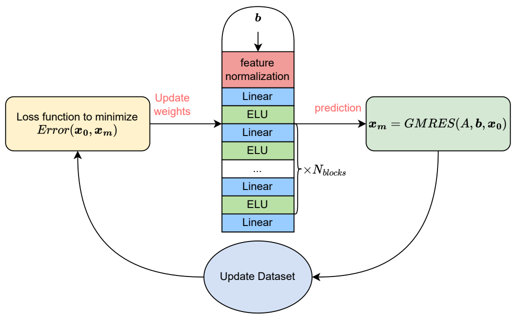

Figure 2: Structure of the Dense Neural Network used.

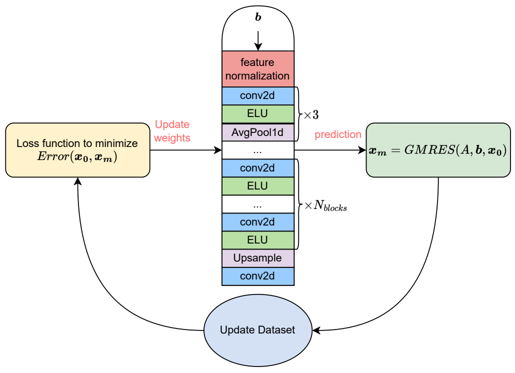

Figure 3: Structure of the Convolutional Neural Network used.

data in a progressively more expressive manner. The final linear layer reshapes the output vector to match the original size.

2.2. Convolutional Neural Network based Model

As shown by Figure 3 the implemented CNN-based architecture comprised a sequence of convolutional layers and ELU functions, with the introduction of pooling. Specifically, average pooling is applied, which reduces the spatial dimensions of the feature maps while retaining their depth. Then, following each average pooling operation, a series of convolutional layers and activation functions are applied iteratively. Finally, the smaller-sized feature maps are upsampled to match the size of the original feature map. All the obtained vector are concatenated together and a final convolution is applied.

2.3. Mixed Model

As shown by Figure 4 the last implemented network architecture is a hybrid model that combines CNNs and GNNs. The CNN component takes as input the $\boldsymbol{b}$ vector and it is as described

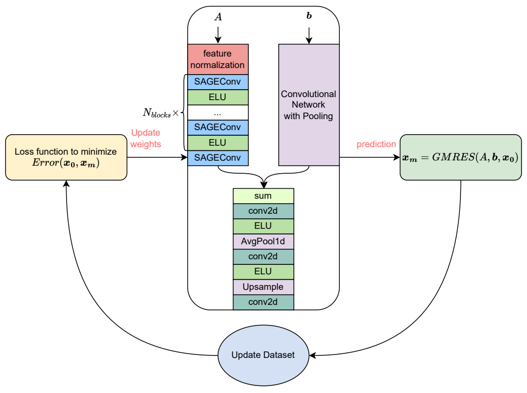

Figure 4: Structure of the Graph Neural Network used.

in Section 2.2, while the GNN component takes A as input, transformed into a graph representation, as each node has a certain number of features and the edges capture the connectivity of the matrix. The normalized features undergo a series of graph convolutional layers (SAGEConv) and activation functions, to introduce non-linearity. The graph convolutional layers and activation functions are iterated. This iterative process enables the GNN to extract and refine features from the graph data. Finally, the features from the last graph convolutional layer are reshaped to a vector.
Once both branches have finished, a combination operation is performed. The resulting vector represents the combined features from both branches of the network. Finally, the obtained vector is fed into a two-layered CNN architecture with pooling.
This final model, combining both the CNN and GNN components, leverages the strengths of both approaches.

3. Numerical Experiments on Simple Problems

In order to verify the capabilities of the implemented solver, it was applied to the same test cases described in the original paper [4]. First, the real-time deep-learning methodology is applied to the discretized 1D Poisson problem and consequently it was used to accelerate the convergence of the discretized 1D time-dependant Advection-Diffusion problem.

3.1. Laplace Equation

The algorithm was tested on a set of 1000 systems each of size 20×20, fixing the residual tolerance to 10^{-10} and the Krylov space, $m = 4$, also in order to compare the results with the ones obtained by [4]. The simulation is run using the DNN model defined in Section 2.1, with 800 trainable parameters and with the initial set size fixed to 32 (*i.e.*, the training of the network, will wait until at least 32 systems are in the database to begin the online training procedure). At the end of the simulation the size of the dataset is equal to 234 (*i.e.*, 202 sets of $\boldsymbol{b}$ and $\boldsymbol{x}$, where added to the database since the system they represented was above the average with respect to the time to solution or the error, *i.e.*, see Section 1). It is possible to plot the residual at the end of the first restart ($E_\kappa(\boldsymbol{x}_\mathbf{0}^{\boldsymbol{k}})$) with respect to the number of iterations. Figure 5 shows the comparison between the values obtained by the classic GMRES method with the zeros vector as initial guess and the ML enhanced version.

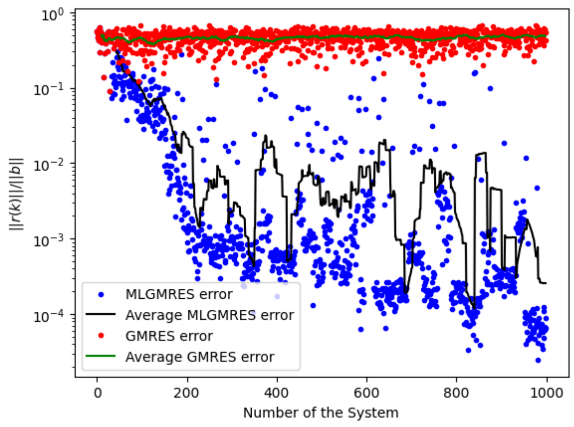

Figure 5: Residual at the end of the first restart ($E_\kappa(\boldsymbol{x}_\mathbf{0}^{\boldsymbol{k}})$) w.r.t. the number of systems (according to the x-axis label).

From Figure 5 one can see that the initial guesses predicted by the model are already closer to the numerical solution at the first restart than for the classic algorithm. Moreover, it becomes evident that as the number of systems increases, there is a discernible learning process occurring, as evidenced by the decreasing residual, depicted by the black line in the figure.
Furthermore, one can take the last system of the simulation and see how the standard and ML-powered GMRES behave with respect to the norm of the normalized residual $\|\boldsymbol{r}\|/\|\boldsymbol{b}\| = \|\boldsymbol{b} - A\boldsymbol{x}\|/\|\boldsymbol{b}\|$, as shown in Figure 6. From the figure it is possible to observe that already the norm initial residual $\|\boldsymbol{r}_\mathbf{0}^{\mathbf{1000}}\| = \|\boldsymbol{b} - A\boldsymbol{x}_\mathbf{0}^{\mathbf{1000}}\|$ is much smaller than the initial residual with the classic choice of taking $\boldsymbol{x}_\mathbf{0} = [0, ..., 0]^T$.

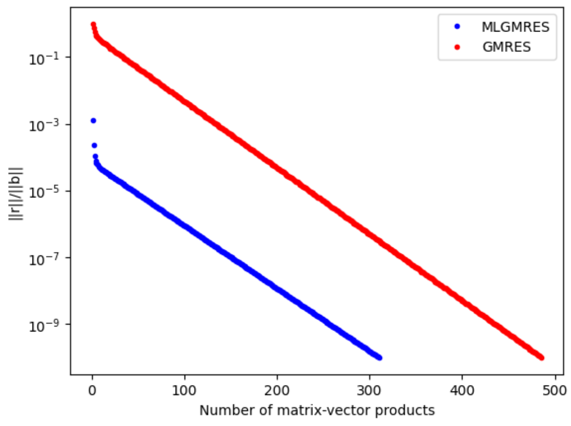

Figure 6: Behaviour of the norm of the normalized residual for the last system of the sequence w.r.t. the number of matrix-vector products.

When all the N_S are completed, it is also possible to observe the number of matrix-vector products needed to reach convergence for each single system solved by the GMRES method. Figure 7 displays the behaviour of the ML enhanced version of the GMRES code, confirming that indeed a learning is taking place, indeed the number of matrix-vector products decreases as the simulation goes on, after the first training is performed.

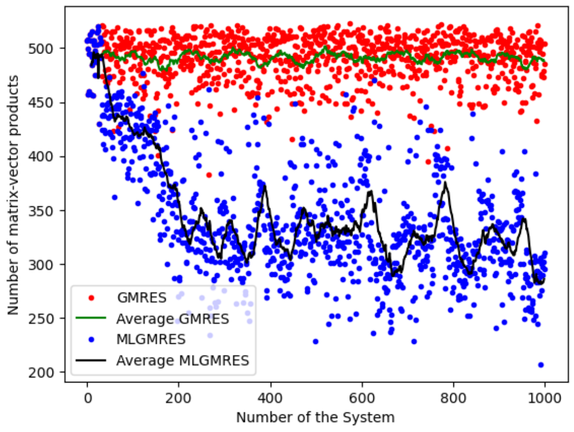

Figure 7: Number of matrix-vector products to reach convergence w.r.t. the number of systems.

Finally, one can observe the plot of the iterations' speed-up, as shown by Figure 8. The speed-up is computed as the ratio between the time taken to resolve the system by the classic GMRES solver (T_GMRES) and the one taken by the MLGMRES solver (T_MLGMRES).

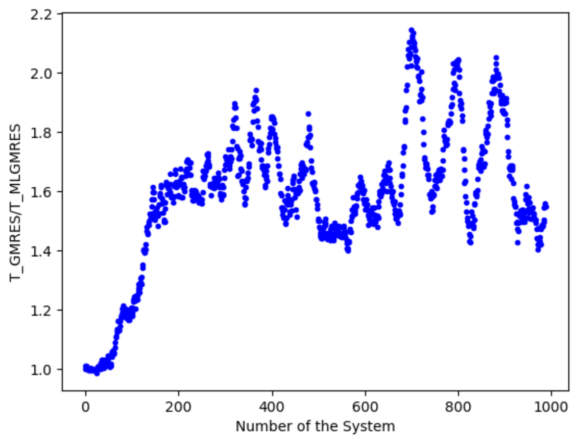

Figure 8: Iteration speed-up.

Figure 8 shows that the implemented algorithm recovered approximately the same speed-up observed by [4], considering that for the demo for the Laplace equation, found at [5], showed a speed-up of approximately 2.

3.2. The Time-Dependent Advection-Diffusion Equation

In order to test the implemented algorithm on a larger dimension n is set to 100. In this case, $N_S = 1000$ is the number of time iterations, hence, the resulting procedure corresponds to solving a set of 1000 systems each of size 100×100, fixing the tolerance to 10^{-10} and $m = 4$. This choice of parameters allows to compare the results with the ones obtained by [4], despite that the authors in [4] used a maximum dimension of $n = 40$. Moreover, $\Delta t = 0.05$ and $T = 50$. The simulation is run using the DNN model defined in Section 2.1, with 60000 parameters and with the initial set size fixed to 32. At the end of the simulation the size of the dataset is equal to 186. It is possible to plot the residual at the end of the first restart $(E_\kappa(\boldsymbol{x}_0^k))$ with respect to the number of iteration. Figure 9 shows the comparison between the values obtained by the classic GMRES method with the zeros vector as initial guess and the ML enhanced version.

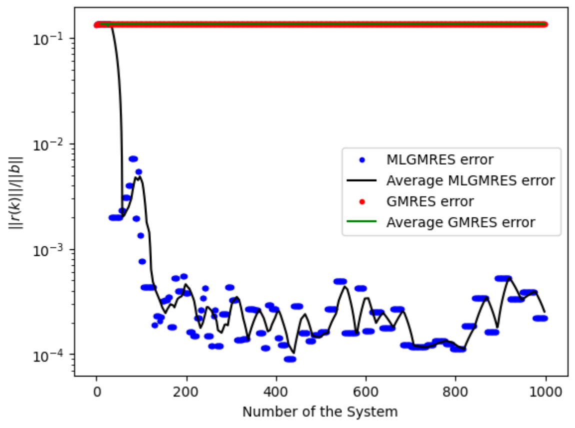

Figure 9: Residual at the end of the first restart $(E_\kappa(\boldsymbol{x}_0^k))$ w.r.t. the number of systems.

From Figure 9 it is possible to acknowledge that also with a larger dimension and with a time-dependent problem the ML decorator is able to predict a better $\boldsymbol{x}_0$ in order to minimize the residual at the end of the first restart of the GMRES code.
In Figure 10 it is possible to see that the speed-up in this case is not as good as in the previous test, however it is still comparable to [4].

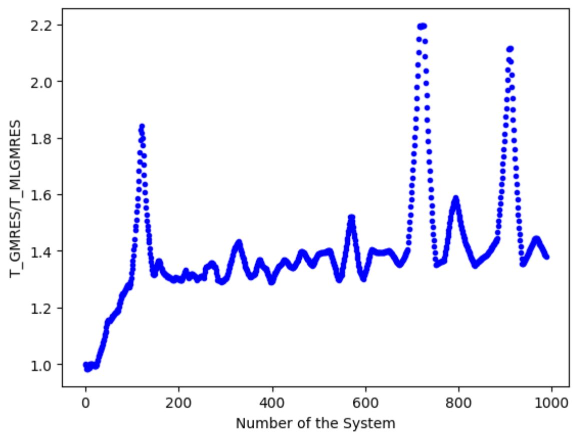

Figure 10: Iteration speed-up.

4. Recycling of the Previous Solution

In order to make a comparison between the ML enhanced algorithm and other more classical heuristics with regards to the initial guess used with GMRES, the recycling of the previous time step's solution is applied to strengthen the classic GMRES algorithm.

4.1. Advection Diffusion Problem with Increasing time step

The solution recycling method is applied to the time-dependent Advection-Diffusion equation with an increasing time step. This is done in order to make the system stiffer as the simulation progresses. The number of systems for this case is fixed as $N_S = 2000$. The time step assumes values in the interval $[2.5, 150]$.
By, investigating the number of matrix-vector products required in order to reach the stopping criterion, one can notice, as depicted in Figure 11, that until half of the simulation the average number of iterations by the implemented solver is higher than the number required by using as initial guess the solution of the previous time iteration. However, the figure also presents a decreasing trend with the black line, and an increasing one with the green line, the two moving averages. In fact, after a certain number of systems the ML algorithm requires less iterations to reach convergence, in average.

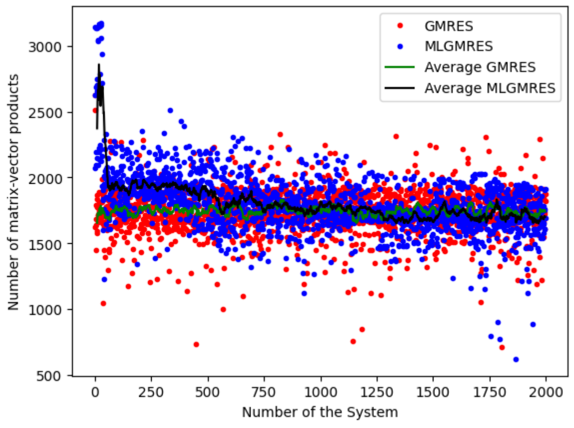

Figure 11: Number of matrix-vector products w.r.t. the number of systems.

The behaviour shown in Figure 11 is also sustained by Figure 12. In fact it is possible to see that the residual at the end of the first restart of recycling solver keeps approximately constant at around 10^{-2}, while the one of the ML solver shows indeed that the learning produces better results with each new training.

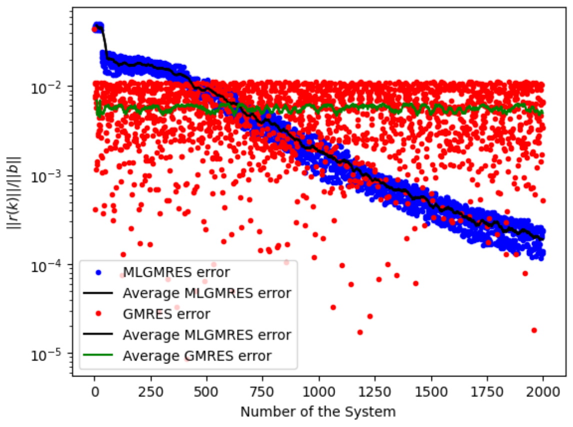

Figure 12: Residual at the end of the first restart ($E_\kappa(\boldsymbol{x}_0^{\boldsymbol{k}})$) w.r.t. the number of systems.

Finally, the speed-ups show that indeed, before the training, feeding a vector of zeros in much worse than recycling the last computed solution, as the first values in Figure 13 are below 1. However, as the simulation continues the same time as the recycling version is reached and even reduced by the ML counterpart, as showed.

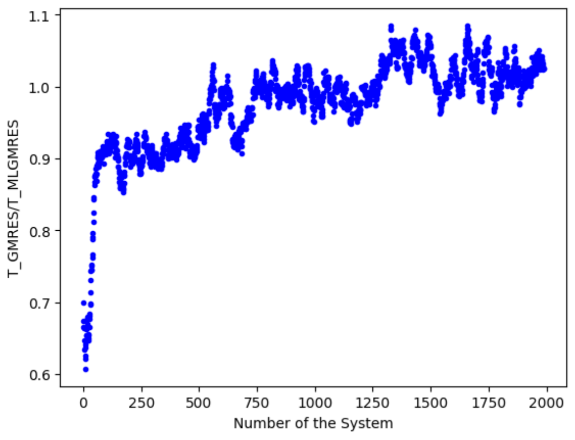

Figure 13: Iteration speed-up w.r.t. recycling the previous solution.

5. Numerical Experiments on Representative Test Cases

Due to the substantially larger dimensions of the upcoming test cases compared to the previously examined ones, employing the previously tested DNN-based model is impractical. The reason is that the DNN model demands a considerable number of parameters. Consequently, using the DNN-based approach for testing the forthcoming systems is not feasible within the current computational constraints.

To address this issue and facilitate the evaluation of the larger-scale systems, an alternative approach will be adopted. Specifically, the testing will be conducted using the CNN-based architecture, as introduced and described in Section 2.2. Unlike the DNN model, the CNN architecture is known for its ability to efficiently process and extract relevant features from high-dimensional data, making it more suitable for handling the increased complexity of the upcoming test cases.

5.1. Laminar Flow around a Cylinder at Low Reynolds Number

Understanding the flow characteristics around a cylinder is of great importance in various engineering applications. This work focuses on a 2D laminar flow over a cylinder using a Discontinuous Galerkin (DG) approach. The fluid dynamics are modeled using the RANS equations [6], with the one-equation turbulence model of Spalart-Allmaras [7]. The flow operates at a Mach number (M_∞) of 0.15 and a Reynolds number (Re) of 80.

The mesh discretization comprises 1028 triangular elements. The flow operates at a laminar state, and the specified Mach and low Re ensure a subsonic flow. These values do not produce an unsteady wake, therefore, a steady solution is searched.

The spatial discretization is performed using the DG method, with a modal basis and order of the discretization equal to 3 and 4.

The convective flux is approximated using the Roe numerical flux, which handles shock waves and discontinuities accurately. The viscous flux is treated using the BR2 method [1, 2], ensuring accurate representation of the viscous effects near the cylinder surface.

The implicit Backward Euler scheme is employed for time integration, using a local time step.

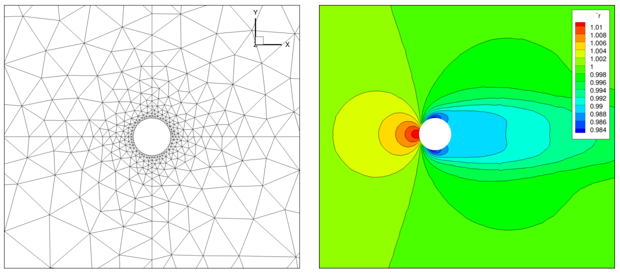
(a) Unstructured mesh used. (b) Steady-state density field.

Figure 14: Cylinder test case.

The implemented algorithm is tested on a set of 98 systems each of size 30840×30840, with 3621600 non-zeros, fixing the GMRES residual tolerance to 10^{-3}, $m = 50$ and imposing a maximum of 2 restarts. The simulation is run using the CNN model, with 1500929 parameters and with the initial set size fixed to 32. At the end of the simulation the size of the dataset is equal to 42.

In this specific test case, convergence is not reached within the researched tolerance without the use of a preconditioner, therefore it would not make sense to analyse the speed-up results. Still it is of interest to observe the behaviour of the residuals.

Studying the last system, it is interesting to see the values of the norm of the normalized residual at the end of the two restarts. Indeed, the ML algorithm presents better values than the non enhanced version. This is quite interesting since it was possible to reach better convergence values without the use of a preconditioner for complex and large linear systems.

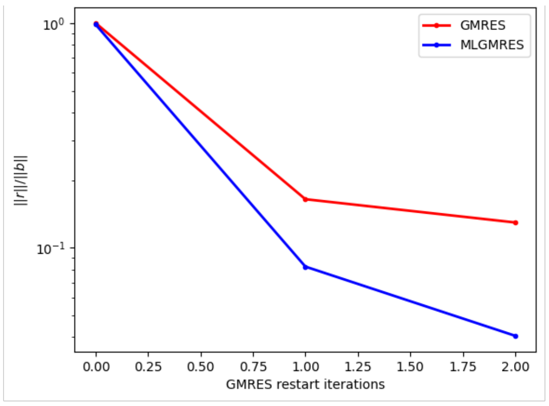

Figure 15: Behaviour of the normalized residual w.r.t. the number restarts for the last problem of the sequence.

Figure 16 shows the behaviour of the residual at the end of the first restart for each system of the simulation. It can be easily extracted that after the training the values of $E_\kappa(\boldsymbol{x}_0^k)$ are better using a ML predicted initial guess.

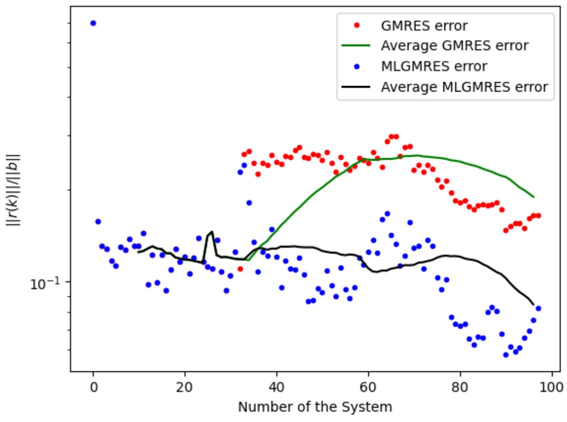

Figure 16: Residual at the end of the first restart ($E_\kappa(\boldsymbol{x}_0^k)$) w.r.t. the number of systems.

5.2. Taylor-Green Vortex

The Taylor-Green Vortex (TGV), a well-known benchmark problem in fluid dynamics, serves as an ideal test case for studying turbulent flows. This work focuses on a 3D turbulent flow within a box with periodic boundary conditions. The periodicity allows the simulation to evolve without any external influences, maintaining a constant total energy level throughout the decay process.

The simulation employs Direct Numerical Simulation (DNS) of a flow with M_∞ set at 0.1 and models the fluid dynamics using the Navier-Stokes (NS) equations. The computational domain consists of a Cartesian mesh of 64 hexahedral elements. The simulation employs the fourth-order Rosenbrock-like scheme (ROS44) for time discretization. This choice guarantees high temporal accuracy and stability, enabling a precise representation of the flow's transient behavior during the decay process. This case, being outside of the considered framework, is used as a last validation of the ML approach.

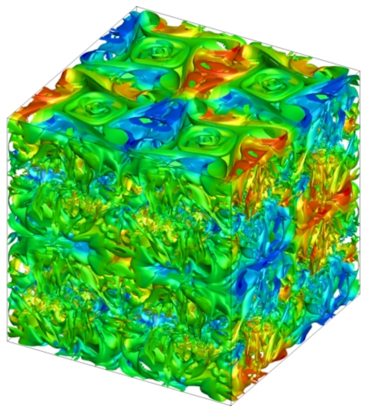

Figure 17: Visualisation of the solution of the Taylor-Green Vortex.

As final test case, the implemented algorithm is tested on a set of 4000 systems each of size 2560×2560, with 716800 non-zeros, fixing the GMRES residual tolerance to 10^{-3}, $m = 50$ and imposing a maximum of 2 restarts. The simulation is run using the CNN model, with 265729 parameters and with the initial set size fixed to 32. At the end of the simulation the size of the dataset is equal to 695.

This final test case does reach tolerance without the use of a preconditioner. Therefore, all the previous metrics can be used to analyze the results.

First, Figure 18 shows the number of matrix-vector products needed to reach the desired tolerance. It is possible to see that, since after the first training, and except from a very small number of systems, the number of products to reach convergence with the optimized $\boldsymbol{x}_0$ is smaller than with the classic $\boldsymbol{x}_0$.

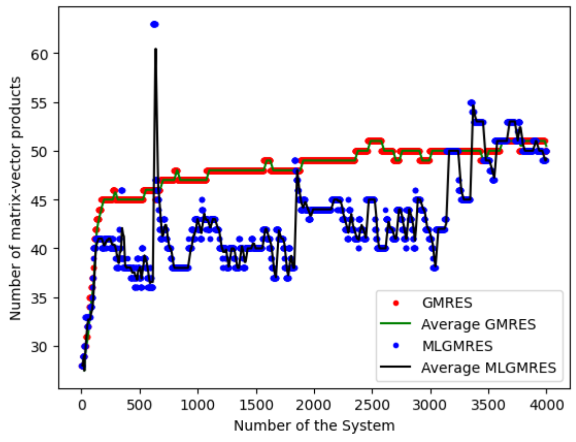

Figure 18: Number of matrix-vector products w.r.t. the number of systems.

Figure 19 shows instead the speed-ups, since this time it is an interesting result to investigate. It can be seen that the value of the speed-ups is mostly above 1, showing that indeed the time to reach convergence with the ML implementation is smaller than using $\boldsymbol{x_0} = \mathbf{0}$. Despite our expectations for speed-up improvements, there are instances where we observe speed-up values below 1, indicating sub-optimal performance. However, the retraining mechanism of the model comes to the rescue in such situations. This retraining process works diligently to address the issues, ultimately resulting in better speed-up values.

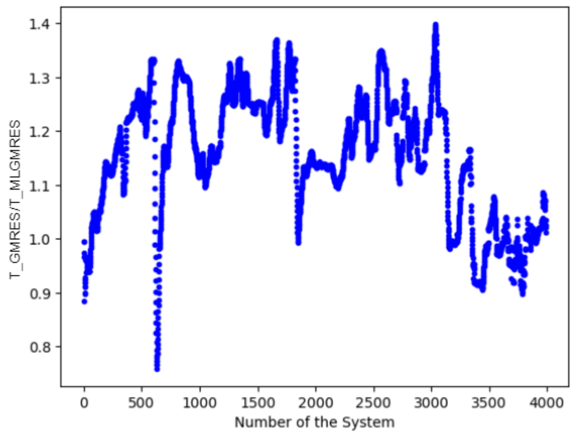

Figure 19: Iteration speed-up.

5.2.1 Results on the TGV using GNNs

Until now the initial guess was predicted only from the RHS $\boldsymbol{b}$ vector, therefore it is interesting to investigate what happens when the hybrid model defined in Section 2.3 is deployed, in other words, to see the effect of using matrix features to extrapolate the matrix importance in the choice of the initial guess. The features used are very simple to extract, the values on the principal diagonal of the matrix and the value of diagonal dominance, defined as:

$$dd_i = \frac{|a_{ii}|}{\sum_{i \neq j} |a_{ij}|}\ , \tag{3}$$

where a_{ij} is the element of A in position (i, j).
As detailed in Section 2.3, the implemented GNN-based architectures uses a combination of GNN and CNN layers.
The size of the Krylov subspace is changed to $m = 30$, in order to better study the value of the residual at the end of the first restart. The model is composed of 1320899 trainable parameters and with the initial set size fixed to 32. At the end of the simulation the size of the dataset is equal to 102.
By studying the behaviour of $E_\kappa(\boldsymbol{x_0})$, it is possible to see that the values of the ML prediction solver are below the values of the normal implementation. Moreover, it is possible to observe that, even having decreased m, some values start to be quite close to the imposed tolerance.

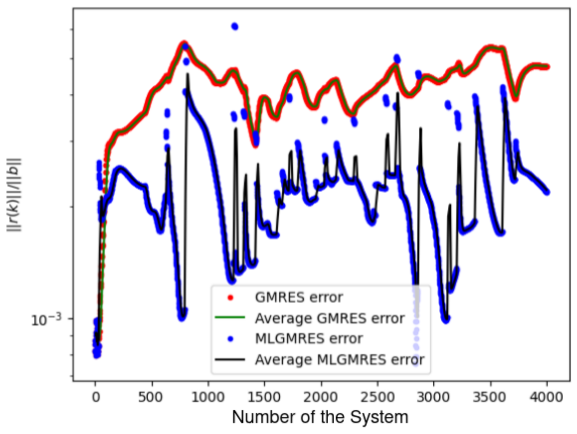

Figure 20: Residual at the end of the first restart ($E_\kappa(\boldsymbol{x_0^k})$) w.r.t. the number of systems.

Finally, observing the plot of the speed-ups in Figure 21, it is possible to see that better values are obtained with respect to the case where only CNNs where being used. This confirms the assumption that capturing matrix features in the model is essential to obtain better results.

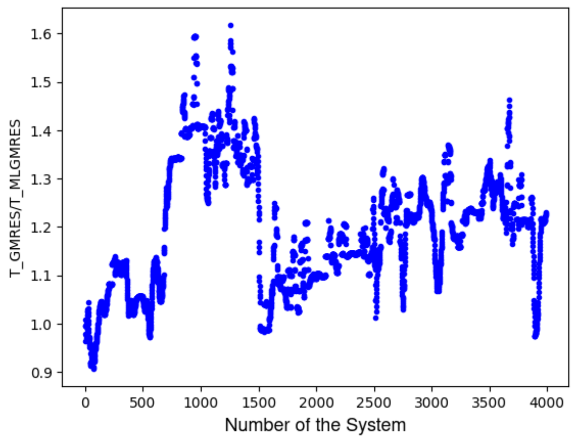

Figure 21: Iteration speed-up using the hybrid model.

However, when considering the total times, the one of the ML simulation is much larger than the time spent by the classic GMRES method. This is a result of the fact that the trainings of the GNN-based model require large computational times, resulting in a longer simulation. Furthermore, CNNs seem to be more optimized within the Pytorch environment.

6. Conclusion

This thesis has presented an approach, based on Machine Learning techniques, that aims to deliver an adequate initial guess for the iterative Krylov methods in the contest of a series of linear systems, where A and $\boldsymbol{b}$ can change significantly. Furthermore, one of the main concerns of this work is to explore its application to representative cases arising from compressible flows in compressible computational fluid dynamics.
The prediction algorithm was presented and applied to simple problems, including the Laplacian equation and the Advection-Diffusion equation. Furthermore, the proposed ML-based algorithm is compared to a classic heuristic used to obtain an initial guess, which consists in recycling the previous solution during an time-iteration scheme. The results obtained are quite promising. The ML predicted initial guesses accelerate the convergence for most of the considered cases. Moreover, the algorithm demonstrated to have a better behaviour with respect to the recycling algorithm for the advection-diffusion equation with an increasing time step.
The algorithm's application was extended to representative test cases governed by the Navier-Stokes and RANS equations, precisely the flow around a cylinder with low Reynolds number and the Taylor-Green vortex test case. The results obtained demonstrate that the developed algorithm leads to shorter times to reach the stopping conditions of the GMRES solver, unfortunately, the training of the network causes the computational time and memory constraints for large systems to be still prohibitive with the current implementation. Hopefully, advancements of algorithms and GPU hardware could lead to address these issues in the future.
While we have achieved interesting results, this work also opens avenues for future research. Further exploration of hybrid approaches, combining iterative methods and machine learning, could lead to even more robust and efficient solvers for fluid dynamics simulations.

References

[1] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In *Turbomachinery - Fluid Dynamics and Thermodynamics, European Conference, 2*, pages 99–108, 1997.

[2] Lorenzo Botti and Luca Verzeroli. BR2 discontinuous Galerkin methods for finite hyperelastic deformations. *Journal of Computational Physics*, 463:111303, 2022.

[3] M. Jadoui, C. Blondeau, E. Martin, F. Renac, and F. Roux. Comparative study of inner–outer Krylov solvers for linear systems in structured and high-order unstructured CFD problems. *Computers & Fluids*, 244, 2022.

[4] K. Luna, K. Klymko, and J. P. Blaschke. Accelerating GMRES with Deep Learning in Real-Time. `https://doi.org/10.48550/arXiv.2103.10975`, 2021.

[5] K. Luna, K. Klymko, and J. P. Blaschke. GMRES-Learning, 2021.

[6] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[7] NASA Christopher Rumsey. The Spalart-Allmaras Turbulence Model. `https://turbmodels.larc.nasa.gov/spalart.html`, 2020. Accessed: 2023-07-03.

[8] Youcef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[9] Z. Tang, H. Zhang, and J. Chen. Graph Neural Networks for Selection of Preconditioners and Krylov Solvers, 2022.