ICCFD13-2026-184

Enhancing Production-Oriented Non-Linear PDE Solver Convergence with Learnable Models

J. Sitaraman*,J. Abras*,S, Hosseinverdi* and N. Hariharan**
Corresponding author: jayanarayanan.sitaraman.ctr@mail.mil
* HPCMP CREATE^{TM}-AV Helios Team, Vicksburg, MS, USA.
** HPCMP CREATE^{TM} Chief Scientist and Technologist, Vicksburg, MS, USA.

Abstract:
Accelerating implicit CFD solvers through machine-learned warm starts is an active research direction, but prior demonstrations have largely been limited to small Python prototypes on simple periodic problems. This paper presents the second step in a planned progression toward production-scale deployment: a fully performance-portable C++ implementation of the Reflex in-situ graph neural network (GNN) framework, executed on GPUs via the OCCA portability layer and the LibTorch C++ API. The solver is extended to viscous compressible flows with a Newton–Krylov GMRES linear solver, no-slip wall boundary conditions, and a Reynolds-driven boundary-layer mesh pipeline, enabling validation on wall-bounded external flows. Physical accuracy is demonstrated on the Re=40 steady cylinder, Re=100 unsteady cylinder (von Kármán shedding, $St = 0.157$), and NACA0012 at laminar Reynolds numbers, with drag and Strouhal results within 2–8% of published benchmarks. The nonlinear GNN model (smodel) consistently reduces the initial Newton residual by approximately one order of magnitude on both the two-dimensional isentropic vortex and the Re=100 cylinder, translating into a reduction of one to two Newton sub-iterations per time step. The linear model (lmodel) shows less consistent benefit, attributed to the wide dynamic range of training targets across Newton sub-iterations; amplitude scaling and a residual guard are essential for robustness. Results are reported in terms of iteration-count reduction rather than wall-clock time, as the small problem sizes used here are not representative of the GPU-dominated regime where end-to-end speedups will be meaningful.

Keywords: Numerical Algorithms, Computational Fluid Dynamics, Machine Learning.

1. Introduction

Computational Fluid Dynamics (CFD) simulations of aerodynamic phenomena are a core capability used extensively in the design and analysis of both fixed-wing and rotary-wing aircraft. Solving the compressible Reynolds-Averaged Navier–Stokes (RANS) equations, which form the foundation of these simulations, is computationally expensive. At the high-fidelity end of the spectrum, the Navier–Stokes equations are typically discretized using finite volume or finite element techniques on body-conforming unstructured mixed-element meshes. For problems involving moving bodies, sliding-mesh or overset-grid techniques are often employed, while Cartesian-grid-based high-order methods are used for wake capturing. Over decades, the need to accelerate these simulations has led to a wide range of low- to high-fidelity methods, each introducing different simplifying assumptions. However, despite these advances, the demand for higher-fidelity solvers remains strong, as many complex configurations can only be accurately predicted using such methods. This continuing need has motivated efforts to accelerate high-fidelity CFD simulations without introducing approximations that compromise accuracy.

The compressible Navier–Stokes equations are non-linear, hyperbolic partial differential equations that can be written compactly as $\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = 0$, where $\mathbf{q}$ is the vector of conserved variables (mass, momentum, and energy), and $\mathbf{F}(\mathbf{q})$ is the non-linear flux function governing advection and diffusion consistent with conservation laws and equations of state. For typical aircraft and rotorcraft flow conditions, the Reynolds

numbers are large, meaning viscous effects are significant only near solid surfaces. Fully resolving all turbulent scales at such Reynolds numbers is computationally intractable, so the RANS equations—with suitable turbulence closures—are employed to obtain accurate engineering solutions.

RANS simulations require highly stretched, anisotropic grids near solid boundaries, making explicit time-stepping inefficient due to the severe time-step restriction imposed by the smallest grid spacing. Implicit time-stepping overcomes this limitation by linearizing the governing equations about the current solution state and using Newton's method to drive the residual to zero. This linearization leads to a large, sparse system of equations of the form $\mathbf{A}\,\Delta\mathbf{q} = \mathbf{B}$, where $\Delta\mathbf{q}$ is the solution update, $\mathbf{B}$ is the discrete residual evaluated at the current state, and $\mathbf{A}$ is the sparse Jacobian matrix of the residual, typically augmented with a damping term to enhance diagonal dominance. In modern RANS solvers, the majority of computational time is spent solving this linear system using iterative methods (often preconditioned or multigrid-accelerated). Consequently, improved initial approximations for these iterative solvers can substantially reduce the total time to solution.

In unsteady problems—such as those involving aeroelasticity or rotorcraft flows—the compressible Navier–Stokes equations are solved in a time-accurate manner using the unsteady RANS (URANS) formulation. Each time step requires non-linear iterations to solve the fully discrete algebraic system, and each non-linear iteration, in turn, involves solving the large linear system described above.

Helicopter rotor and turbomachinery flows represent particularly demanding instances of this challenge. A helicopter rotor simulation typically requires hundreds of time steps per revolution and must be advanced through five to twenty revolutions before the aerodynamic loads reach a periodic steady state—implying tens of thousands of implicit linear solves per simulation. Turbomachinery blade-row interactions present a similar profile: long transient phases followed by a sustained quasi-periodic regime. Once the flow enters this periodic regime, successive time steps produce residuals and solution updates that repeat with each rotor revolution or blade-passing period. This temporal self-similarity is precisely the condition under which an in-situ trained surrogate can be most effective: the model trains on the first few cycles and thereafter provides high-quality warm starts for every subsequent linear solve, compounding savings over the remainder of the run. The present work is motivated by the long-term goal of exploiting this structure to accelerate production-scale rotorcraft and turbomachinery simulations, with an on-device GPU training loop that adds negligible overhead relative to the solver cost.

There have been several research efforts in the last five years to use learning-based iterative numerical methods to complement or replace iterative linear solvers [1, 2, 3, 4, 5, 6, 7, 8]. Our work is inspired by Luna et al. [5], who explored the concept of using machine learning for initial guesses in the context of Krylov subspace methods. They developed an *in situ*, real-time machine learning framework that accelerates the GMRES(k) algorithm for solving linear systems from second-order PDEs. Saverio [8] extended this concept to fluid dynamics problems including flow over a NACA0012 airfoil.

In our prior work [9], we introduced an in-situ graph neural network (GNN) framework—referred to as *Reflex*—that learns to predict updates for both the non-linear and linear stages of an implicit CFD solver. Two independent GNN models are maintained: a non-linear model (smodel) that maps the unsteady residual to a full-step solution update, and a linear model (lmodel) that maps the linear right-hand side to an initial guess for the iterative linear solver. Data are curated online during the simulation, training is performed in-situ, and predictions are applied as warm starts. The approach was demonstrated on canonical one-dimensional and two-dimensional test cases using a Python prototype solver, achieving up to 26% reduction in overall wall-clock time on the two-dimensional Euler equations. The long-term goal is to integrate Reflex into the GPU-accelerated Rapidus solver [10].

The present paper is the second in this series and advances the Reflex framework in four key areas:

1. **Performance-portable C++ implementation.** The Python two-dimensional demonstrator has been replaced by a full C++ finite-volume solver built on the OCCA performance-portability layer [10], enabling execution on CPUs (Serial, OpenMP) and GPUs (CUDA, HIP) from a single

source.

2. **GPU-resident GNN inference and training.** The LibTorch C++ API is used to keep both smodel and lmodel on the GPU throughout the simulation, eliminating host–device data transfers for the ML components and enabling fully GPU-resident inference and online training.

3. **Viscous flow capability and Newton–Krylov linear solver.** The solver is extended with full compressible viscous fluxes, no-slip wall boundary conditions, and a Reynolds-layer mesh generation pipeline, enabling viscous validation against published results. A matrix-free Newton–Krylov GMRES solver with Fréchet-derivative matrix–vector products and a block-Jacobi left preconditioner replaces the Gauss–Seidel scheme of the prior work, delivering substantially faster convergence for both steady and unsteady configurations.

4. **Improved Reflex hooks.** Several robustness improvements are made to the online GNN integration: (a) a residual guard that detects when a warm-start prediction overshoots the zero-start residual and iteratively rescales the prediction until it is beneficial, (b) improved lmodel amplitude calibration, and (c) consistent data curation across sub-iterations.

The physical accuracy of the C++ solver is validated on the Re = 40 steady cylinder, the Re = 100 unsteady cylinder (von Kármán shedding), and the NACA0012 airfoil across a range of laminar Reynolds numbers. Reflex GNN augmentation is then evaluated on the two-dimensional isentropic vortex (carried over from the Python prototype) and on the Re = 100 cylinder, where the quasi-periodic shedding provides the stationary training distribution the framework is designed to exploit.

2. Methodology

The core concept of the present method is to use neural networks to warm-start both stages of an implicit unsteady solver, reducing the total number of iterations required at each time step.

At the *nonlinear* level, the outer Newton iteration solves $\mathbf{R}(\mathbf{q}) = 0$ for the solution update $\Delta\mathbf{q}$. Rather than beginning each time step from the previous solution, a nonlinear surrogate (smodel) predicts $\Delta\mathbf{q}$ directly from the current residual $\mathbf{R}$, providing a warm start that reduces the residual entering the first Newton sub-iteration.

At the *linear* level, each Newton sub-iteration requires solving the sparse system $\mathbf{A}\mathbf{x} = \mathbf{B}$, which is typically initialized from zero. A linear surrogate (lmodel) predicts an improved initial guess $\mathbf{x}_0$ given $\mathbf{B}$, reducing the number of Krylov or stationary iterations needed to reach convergence.

Both surrogates are trained in-situ from data $(\mathbf{B}, \mathbf{x})$ and $(R, \Delta\mathbf{q})$ collected during the simulation itself, requiring no a priori knowledge of the flow configuration.

2.1 Governing Equations and Discretization

The compressible Navier–Stokes equations in two spatial dimensions can be written in conservation law form as

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}_{\text{inv}}(\mathbf{q}) - \nabla \cdot \mathbf{F}_{\text{vis}}(\mathbf{q}, \nabla\mathbf{q}) = 0, \tag{1}$$

where $\mathbf{q} = [\rho,\ \rho u,\ \rho v,\ E]^\top$ is the vector of conserved variables (density, momentum components, and total energy). The inviscid flux $\mathbf{F}_{\text{inv}}$ is the standard compressible Euler flux, and the viscous flux $\mathbf{F}_{\text{vis}}$ consists of the compressible Newton stress tensor and the Fourier heat flux,

$$\boldsymbol{\tau} = \mu\Big(\nabla\mathbf{u} + (\nabla\mathbf{u})^\top - \tfrac{2}{3}(\nabla\cdot\mathbf{u})\mathbf{I}\Big), \qquad \mathbf{q}_h = -k\nabla T,$$

with dynamic viscosity μ set from the Reynolds number and thermal conductivity $k = \mu\gamma/[(\gamma - 1)\,\mathrm{Pr}]$.

The spatial domain is partitioned into N non-overlapping control volumes on a two-dimensional unstructured mesh of mixed triangular and quadrilateral elements. A finite-volume formulation is employed, and second-order accurate gradients at control-volume centroids are reconstructed using a weighted least-squares (WLS) procedure. Inviscid interface fluxes are computed using the Roe approximate Riemann solver, and the viscous fluxes use an alpha-damped directional-derivative correction to suppress the even–odd decoupling mode common in cell-centered schemes. Body-conforming boundary-layer meshes are generated using an advancing-front quadrilateral extruder driven by the local boundary-layer thickness, providing adequate near-wall resolution for viscous computations. No-slip wall boundary conditions are enforced via a mirror-cell approach in which the halo cell carries the reflected velocity and an adiabatic temperature condition.

For temporal integration, the second-order implicit backward differentiation formula (BDF2) is employed. Applying the spatial and temporal discretizations to Eq. (1) yields the fully discrete non-linear algebraic system

$$\frac{3Q^{n+1} - 4Q^n + Q^{n-1}}{2\Delta t} = -S(Q^{n+1}), \tag{2}$$

where $S(Q)$ denotes the spatial residual. A non-linear pseudo-time iteration indexed by s with a stabilizing damping term leads to

$$\frac{3(Q^{s+1} - Q^s)}{2\Delta t} + \frac{Q^{s+1} - Q^s}{\Delta\tau} = -S(Q^{s+1}) - \frac{3Q^s - 4Q^n + Q^{n-1}}{2\Delta t}, \tag{3}$$

where $\Delta\tau$ is the pseudo-time step. Linearizing the spatial residual S about the current iterate Q^s via Newton's method yields the sparse linear system

$$\underbrace{\left(\frac{I}{\Delta\tau'} + \frac{\partial S}{\partial Q}\right)}_{A} \Delta Q = \underbrace{-R(Q^s)}_{B}, \tag{4}$$

where $\Delta\tau' = 2\Delta t\,\Delta\tau/(3\Delta\tau + 2\Delta t)$ and $R(Q^s) = S(Q^s) + (3Q^s - 4Q^n + Q^{n-1})/(2\Delta t)$ is the unsteady residual combining spatial and temporal contributions.

2.2 Iterative Linear Solvers

The linear system Eq. (4) is written as $(D + A_{\text{off}})\,\Delta Q = B$, where D collects the 4×4 block-diagonal entries corresponding to each cell's self-contribution and A_{off} contains the off-diagonal coupling terms. Two iterative solvers are employed, described below.

2.2.1 Block-Jacobi Sweeps

The simplest solver performs N_{sw} stationary iterations in which the off-diagonal coupling is lagged:

$$D\,\Delta Q^{(k+1)} = B - A_{\text{off}}\,\Delta Q^{(k)}, \qquad \Delta Q^{(0)} = 0. \tag{5}$$

Each sweep requires one pass over all edges to form $A_{\text{off}}\,\Delta Q^{(k)}$ followed by a 4×4 block-diagonal solve via Gaussian elimination with partial pivoting. The block-diagonal Jacobian D is assembled once per sub-iteration using a finite-difference edge-Jacobian approach and remains frozen across sweeps. The convergence rate is bounded by the spectral radius $\rho(D^{-1}A_{\text{off}})$, which is kept below unity by the diagonal dominance enforced through the pseudo-time damping term $I/\Delta\tau'$ in Eq. (4).

The baseline time-marching algorithm using block-Jacobi sweeps is summarised in Algorithm 1, and the resulting hierarchical iteration structure is illustrated in Figure 1.

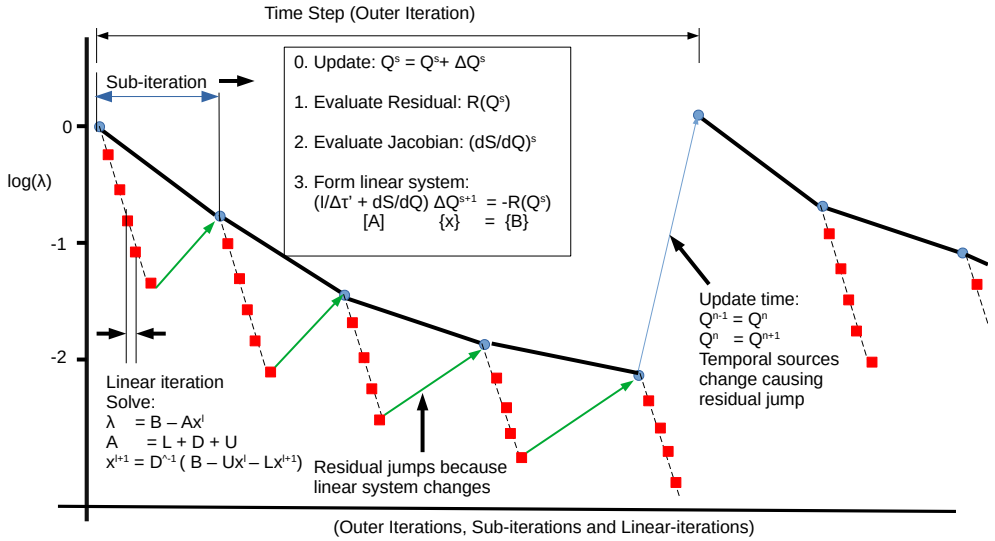

Figure 1: Hierarchical iteration structure of the baseline block-Jacobi solver. Within each time step, several non-linear sub-iterations are performed (large blue circles); within each sub-iteration, block-Jacobi sweeps reduce the linear residual (red squares). Green arrows indicate residual jumps due to Jacobian and right-hand-side updates between sub-iterations.

Algorithm 1 Baseline Time-Marching with Block-Jacobi Linear Solver

Initialize $Q^0 \leftarrow \text{InitialState}(t = 0)$; $Q^n \leftarrow Q^0$; $Q^{n-1} \leftarrow Q^0$
for $n = 1$ to N_{steps} **do**
  for $s = 1$ to N_{subiter} **do**
    Compute unsteady residual: $R \leftarrow S(Q) + \text{TemporalSource}(Q, Q^n, Q^{n-1})$
    if $\|R\|/\|R_0\| < \epsilon_{\text{nl}}$ **break**
    Assemble block-diagonal Jacobian: $D \leftarrow \text{DiagJacobian}(Q)$
    $B \leftarrow -R$; $X \leftarrow 0$
    for $k = 1$ to N_{sw} **do** ▷ Block-Jacobi sweeps
      $X \leftarrow D^{-1}(B - A_{\text{off}}\, X)$
      if $\|B - AX\| < \epsilon_l$ **break**
    $Q \leftarrow Q + X$
  $Q^{n-1} \leftarrow Q^n$; $Q^n \leftarrow Q$

2.2.2 Newton–Krylov GMRES

Block-Jacobi sweeps are inexpensive per iteration but their convergence rate is limited by $\rho(D^{-1}A_{\text{off}})$, which degrades on fine meshes and at high Reynolds numbers. To mirror the linear solver architecture of production CFD codes and to obtain deeper convergence within each Newton sub-iteration, the solver also supports a left-preconditioned GMRES(m) Krylov method in a matrix-free framework. Rather than assembling A explicitly, matrix–vector products $A\mathbf{v}$ are approximated via a Fréchet finite difference of the unsteady residual operator,

$$A\mathbf{v} \approx \frac{R(Q + \varepsilon\mathbf{v}) - R(Q)}{\varepsilon}, \qquad \varepsilon = \frac{\varepsilon_0}{\|\mathbf{v}\|}, \tag{6}$$

with $\varepsilon_0 = 10^{-6}$. The block-diagonal D serves as the left preconditioner, giving the system $(D^{-1}A)\,\Delta Q = D^{-1}B$, solved by Arnoldi–GMRES(m) with restart parameter m chosen by the user.

The entire Arnoldi process runs on the GPU: dot products are computed via a per-cell reduction kernel whose partial sums are accumulated on the host; Gram–Schmidt axpy updates and vector normalization use dedicated device kernels; and the Krylov basis vectors reside in device memory throughout. Only the small $(m+1) \times m$ Hessenberg matrix and the Givens rotation scalars are maintained on the host. This design eliminates vector host–device transfers during the Arnoldi loop, leaving only an $O(n_{\text{owned}})$ scalar download per dot product.

For steady-state problems, a Switched Evolution/Relaxation (SER) CFL controller ramps the pseudo-time step automatically,

$$\text{CFL}_{n+1} = \min\left(\text{CFL}_{\max},\ \text{CFL}_n \times \left(\frac{\|R_{n-1}\|}{\|R_n\|}\right)^p\right), \tag{7}$$

allowing aggressive CFL growth when the residual is falling rapidly and automatic back-off when it stagnates. The exponent $p = 1$ and a ceiling $\text{CFL}_{\max} = 10^6$ are used throughout.

2.3 Graph Neural Network Models

Two independent GNN models are maintained during the simulation. The non-linear model (smodel) learns the mapping $R \mapsto \Delta Q$ and provides a warm-start update to the solution at the beginning of each time step. The linear model (lmodel) learns the mapping $B \mapsto X$ and provides a warm-start initial guess for the iterative linear solve at each non-linear sub-iteration.

Both models are implemented as graph-based solvers of the form

$$\text{GraphSolver}(B) = \text{Proj}_{\text{out}} \circ [\sigma \circ \text{GNNLayer}]^H \circ \text{Proj}_{\text{in}}(B), \tag{8}$$

where Proj_{in} and Proj_{out} are linear projections between the physical and hidden dimensions, $[\sigma \circ \text{GNNLayer}]^H$ applies H graph convolution layers each followed by a ReLU non-linearity, and each GNN layer aggregates neighbor messages as

$$y_i^{(l+1)} = y_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \text{MLP}(y_j^{(l)}). \tag{9}$$

The graph is defined on the finite-volume mesh: vertices are cell-centroids (where the degrees of freedom reside) and edges connect face-adjacent cells, mirroring the stencil connectivity of the discretization. The input dimension is four (one channel per conserved variable) and the output dimension matches the input. The graph structure and edge index are computed once from the mesh connectivity and remain fixed throughout the simulation.

Scaling is critical to training stability. The smodel uses a mean/standard-deviation normalization computed over the training dataset, while the lmodel uses an asinh scaling,

$$B_{\text{scaled}} = \text{asinh}\left(\frac{B}{\sigma_{\text{dataset}}}\right), \tag{10}$$

which handles the large dynamic range of the linear right-hand side across sub-iterations without clipping.

Both models are compiled with LibTorch and executed on the same GPU as the flow solver, eliminating host–device data movement for inference. The GNN graph structure, weights, and activations remain device-resident throughout training and inference.

2.4 Reflex Integration and Residual Guard

The GNN-augmented solver, referred to as Reflex, is summarized in Algorithm 2. GNN predictions are applied at lines 5 and 10 to initialize the nonlinear and linear solver stages respectively. Two distinct

stability mechanisms protect the solver against harmful warm-start predictions.

smodel amplitude scaling. The smodel prediction $\Delta\mathbf{q}_{\text{pred}}$ is applied before the Newton sub-iteration loop. Before acceptance, the predicted update is scaled by a factor $s \leq 1$ chosen so that the nonlinear residual after applying $s\,\Delta\mathbf{q}_{\text{pred}}$ does not exceed the unscaled baseline $\|R\|$. This iterative amplitude rescaling ensures the smodel can never increase the nonlinear residual, even when the prediction overshoots the true update.

lmodel residual guard. After the lmodel predicts an initial guess X_0 for the linear system, the true linear residual of this prediction is evaluated using one Fréchet matvec, $\|B - AX_0\|$. If this exceeds the zero-start baseline $\|B\|$ (i.e., the prediction makes the linear residual worse than starting from zero), X_0 is iteratively halved—using the same matvec to recheck after each halving—until the residual falls below the baseline or seven halvings have been applied. If no scaled prediction beats the zero-start baseline, the solver reverts to $X_0 = 0$. Both guards are fully device-resident and impose negligible overhead when predictions are accurate, while providing a robust fallback when the model is under-trained or predicts out-of-distribution inputs. The halving procedure also naturally corrects amplitude miscalibration: a qualitatively correct prediction with incorrect scale α satisfies $\|B - A(\alpha X^*)\| = |1 - \alpha|\|B\|$, which is reduced toward zero as $\alpha \to 1$ by the rescaling.

Algorithm 2 Reflex: ML-enhanced Time-Marching

Initialize Q^0, Q^n, Q^{n-1}; initialize smodel, lmodel
for $n = 1$ to N_{steps} **do**
  Compute $R \leftarrow S(Q) + \text{TemporalSource}(Q, Q^n, Q^{n-1})$
  if smodel.can_predict() **then**
    $\Delta Q \leftarrow \text{smodel}(R)$; $Q \leftarrow Q + \Delta Q$; recompute R
  for $s = 1$ to N_{subiter} **do**
    Recompute R; **if** $\|R\|/\|R_0\| < \epsilon_{\text{nl}}$ **break**
    Assemble $D \leftarrow \text{DiagJacobian}(Q)$; $B \leftarrow -R$
    if lmodel.can_predict() **then**
      $X \leftarrow \text{lmodel}(B)$
      Residual guard: while $\|B - AX\| > \|B\|$ and trials < 7: $X \leftarrow X/2$
      if guard failed: $X \leftarrow 0$
    else
      $X \leftarrow 0$
    Solve $(D^{-1}A)\,\delta X = D^{-1}(B - AX)$ via GMRES(m) or Block-Jacobi(N_{sw}); $X \leftarrow X + \delta X$
    $Q \leftarrow Q + X$
    Curate (B, X) for lmodel training
  $Q^{n-1} \leftarrow Q^n$; $Q^n \leftarrow Q$
  Curate $(R_0, \Delta Q_{\text{total}})$ for smodel training
  Retrain smodel and lmodel if error threshold exceeded

Data curation and training are managed online throughout the simulation. Examples are admitted to the training set using two complementary filters: a cosine-similarity filter that rejects examples too similar to those already stored, and a prediction-error filter that prioritises examples on which the current model performs poorly. A configurable fraction of the curated dataset is retained across training events to prevent catastrophic forgetting of earlier flow states.

Training is performed incrementally: a deeper initial phase (typically 200 epochs) trains the model from scratch once sufficient data have been collected, and lightweight maintenance updates (typically 10 epochs) are triggered thereafter whenever the prediction error exceeds a threshold. This fixed-epoch lightweight approach is appropriate for small datasets of order 15 nonlinear and 45 linear examples. An alternative deep-training mode increases the dataset size to allow a train/validation split, uses early

stopping based on validation loss, and saves only the best-performing checkpoint; this mode is used as a comparison baseline in the results section to demonstrate that the lightweight approach achieves equivalent solver convergence benefits.

All of these inputs are user adjustable knobs. When proceeding to investigate production level cases, many of the knobs built into Reflex will become more important. The typical knobs of learning rate, batch size, number of layers and layer size are present. However, additional knobs exist that control error and similarity thresholding for data curation and mainenance training events, the dataset sizes and retention criteria, training depth, and the overall start iteration. For the toy problems the cases demonstrated show differences but are relatively insensitive to these knobs. It is anticipated that production cases will be more sensitive to these features allowing for a deeper investigation into the logistical aspects of the present method.

3. Results

The results are organized in two parts. The first establishes the physical fidelity of the C++ solver on viscous external-flow benchmarks before any GNN augmentation is applied. The second evaluates the Reflex framework, beginning with the two-dimensional isentropic vortex case from the prior Python prototype [9] and then demonstrating the method on the Re=100 cylinder in the fully performance-portable C++ solver.

3.1 CFD Solver Validation

3.1.1 Re=40 Cylinder — Steady External Flow

The two-dimensional flow over a circular cylinder at Re = 40 and $M = 0.3$ provides a well-documented steady laminar validation case. The unstructured mixed-element mesh (boundary-layer quads + farfield triangles, 2436 nodes / 4001 cells) is generated with a Reynolds-driven first-cell height targeting $y^+ \approx 0.5$. Figure 2 shows the converged pressure and velocity magnitude fields with the mesh overlay. The stagnation region and the attached steady wake downstream of the cylinder are both well resolved on the quad-enriched boundary layer.

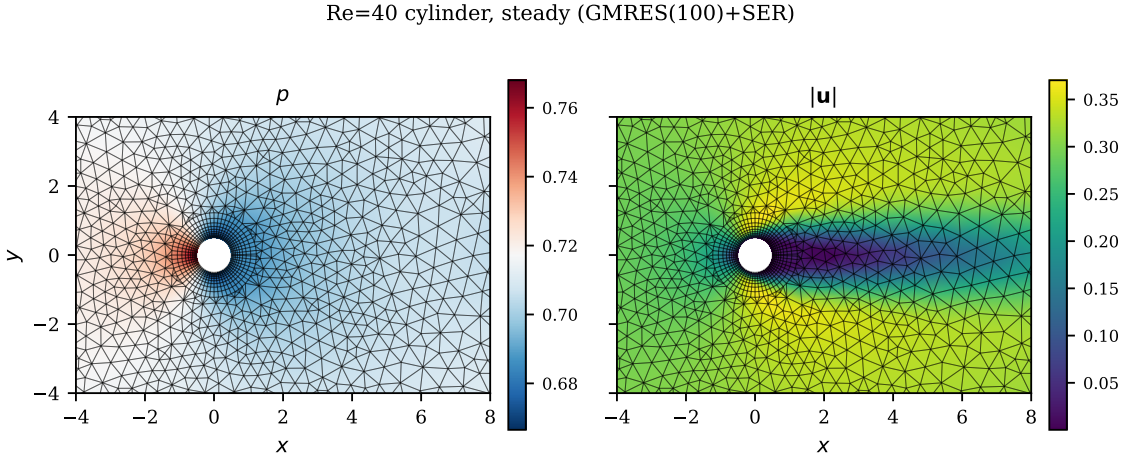

Figure 2: Re=40 cylinder, steady state: pressure (left) and velocity magnitude (right) with mesh overlay. BL quad layers are visible at the cylinder surface.

The converged drag coefficient is $C_D = 1.694$ ($C_{D,p} = 1.131$, $C_{D,v} = 0.563$), within 4% of the Henderson [11] incompressible correlation corrected by the Prandtl–Glauert factor $1/\sqrt{1 - M^2} \approx 1.05$, which gives $C_D^{\text{ref}} \approx 1.63$.

3.1.2 Re=100 Cylinder — Unsteady von Kármán Shedding

At Re = 100 the wake becomes unsteady, producing the von Kármán vortex street. The solver is advanced with BDF2 at $\Delta t = 0.05$ (5 Newton sub-iterations, GMRES(4) linear solver) for 5000 steps to $t = 250$ convective units. Figure 3 shows the instantaneous pressure and velocity at $t = 250$, where the alternating high/low-pressure lobes of the shed vortices are clearly visible in the wake.

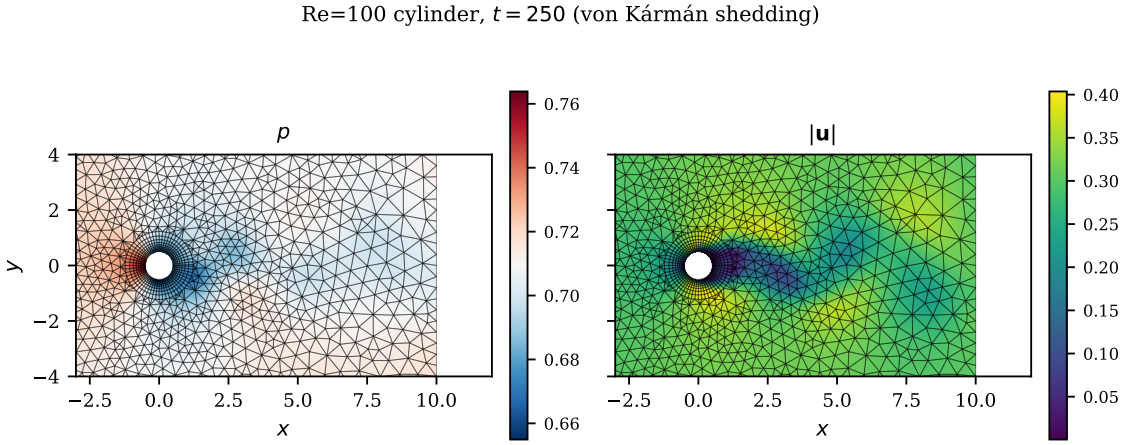

Figure 3: Re=100 cylinder at $t = 250$: pressure (left) and velocity magnitude (right) showing the saturated von Kármán vortex street.

Figure 4 plots the lift and drag histories. The lift grows from the symmetric initial condition, transitions to periodic shedding around $t \approx 150$, and saturates with amplitude $|C_L|_{\text{amp}} = 0.328$. The Strouhal number estimated from zero-crossings of C_L over the final third of the simulation is $St = 0.157$, in good agreement with the Williamson [12] benchmark value of $St = 0.166$. The mean drag in the saturated regime is $\overline{C}_D = 1.402$, consistent with literature values near 1.4.

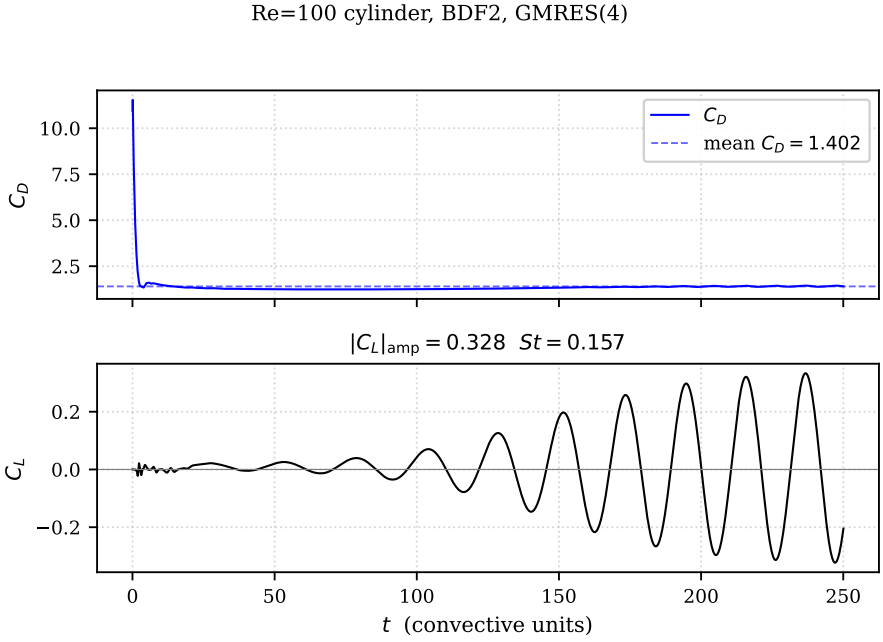

Figure 4: Re=100 cylinder: drag (top) and lift (bottom) histories. Saturated shedding establishes after $t \approx 150$; $St = 0.157$, $\overline{C}_D = 1.402$.

3.1.3 NACA0012 — Viscous Validation Against Di Ilio et al. (2020)

The NACA0012 airfoil at $\alpha = 0°$, $M = 0.2$ is solved for four Reynolds numbers to reproduce Table 1 of Di Ilio et al. (2020), which reports a hybrid lattice-Boltzmann method (HLBM) against XFOIL. Each case uses GMRES(100) with the SER CFL controller starting at CFL = 50 and advancing for 500 pseudo-time steps on the same mesh (boundary-layer quads + farfield triangles). Figure 5 shows the converged pressure and velocity fields at Re=1000.

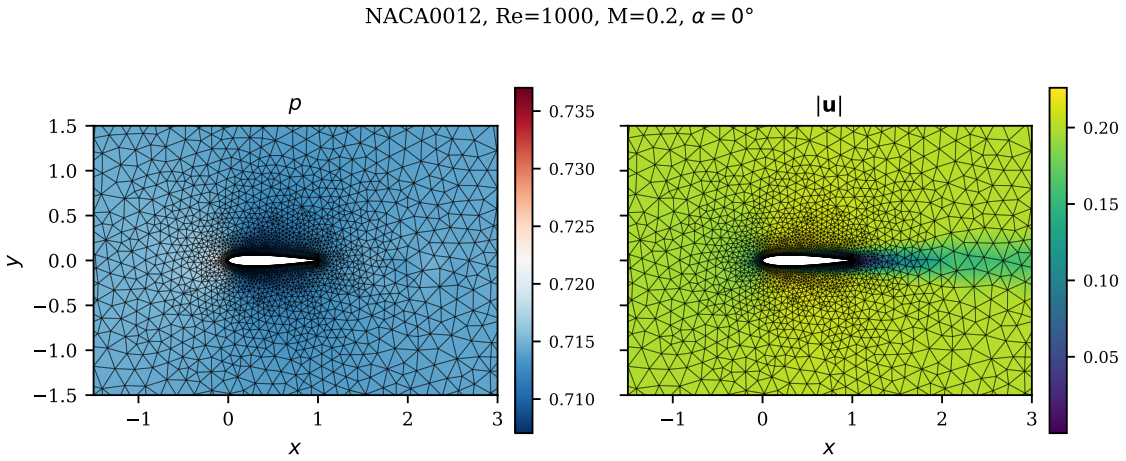

Figure 5: NACA0012, Re=1000, $M = 0.2$, $\alpha = 0°$: pressure (left) and velocity magnitude (right) with mesh overlay.

Table 1 compares the computed drag coefficients against Di Ilio et al. (2020) and XFOIL. The present solver agrees with HLBM to within 2–8%, with a small systematic over-prediction attributable to the blunt trailing edge (∼0.2% chord thickness) in the mesh generator, which adds form drag relative to the sharp-TE geometry used by XFOIL.

Table 1: NACA0012 C_D at $M = 0.2$, $\alpha = 0°$: present solver vs. Di Ilio et al. (2020).

Re	fv2d	HLBM [13]	XFOIL	Δ vs HLBM
1000	0.121	0.119	0.119	+2%
2000	0.085	0.084	0.084	+1%
5000	0.055	0.052	0.054	+5%
10000	0.040	0.037	0.040	+8%

Figure 6 shows the normalised residual histories for all steady cases on a single plot. The cylinder with GMRES(100)+SER reaches machine-zero precision in approximately 70 iterations, dropping 14 orders of magnitude. The block-Jacobi cylinder baseline stalls near 10^{-6} after 500 steps. The NACA cases converge 6–8 orders of magnitude; the residual plateau at higher Reynolds numbers reflects weak trailing-edge unsteadiness rather than solver failure.

3.2 Reflex GNN Results

The following subsections evaluate the Reflex framework. The 2-D isentropic vortex case reproduces results from the Python prototype of the prior work [9] to establish a baseline for iteration-count savings on a well-controlled periodic problem. The Re=100 cylinder then demonstrates the same methodology running in the C++ solver on a physically richer wall-bounded flow with a production-grade Newton–Krylov linear solver.

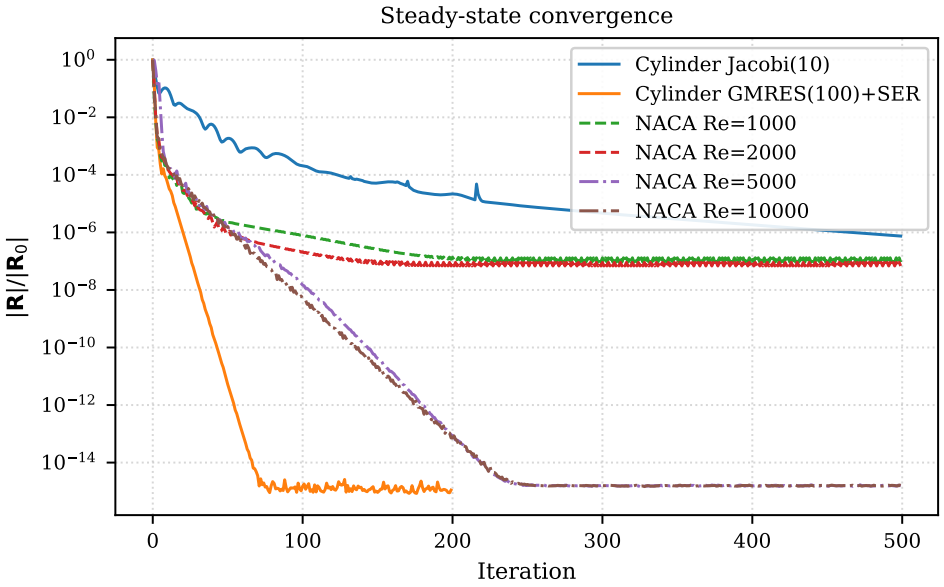

Figure 6: Normalised residual $\|\mathbf{R}\|/\|\mathbf{R}_0\|$ for all steady cases. GMRES(100)+SER (cylinder) reaches machine zero in ≈ 70 iterations; block-Jacobi and the higher-Re NACA cases stall at finite residual.

3.2.1 2-D Euler Equations

The 2-D Euler equations provide the first GNN-augmentation test case. The domain is initialized with an isentropic vortex that propagates along the x-axis; the initial and final vortex positions are illustrated in Figure 7 using density contour plots together with the unstructured triangular mesh overlay. The solver is executed with 20 sub-iterations and 100 linear iterations, with a tolerance cutoff of 0.1 on the ℓ_2-norm applied independently to the nonlinear and linear stages.

The GNN constructed for this equation has an input size of (4,23121) representing the four channels that correspond to the four equations and the 23121 cell centers that correspond to the solver degrees of freedom. This GNN has three hidden layers resulting in a total network size of five layers and utilizes 64 neurons per hidden layer. For the 2-D implementation the key network and training hyper-parameters were studied and optimized to investigate the balance of prediction quality and computational cost; only the key findings are reported here. The optimized settings use a dataset size of 15 examples. Because of the small size of this dataset the validation assessment has been turned off in favor of fixed epoch counts of 200 for initialization and 10 for retraining updates. 10% of the data is still preserved between retraining events, but this results in only 2 examples. The error threshold for data curation is 0.1 and the similarity threshold for data curation is 0.98. The error threshold for retraining is 0.7 and the learning rate is 0.001. All cases apply GNN predictions to the non-linear solver and all linear solver iterations.

Plots of the training and validation loss convergence are provided in Figures 8 and 9 for the non-linear and linear GNNs respectively. Two different versions of the GNN implementation are compared, the optimized version described above, and a deep version where the dataset size is increased to 100 examples and training uses 1500 epochs with early stopping based on validation loss and a patience of 20 epochs. The purpose of this comparison is to demonstrate that deep training is not required to achieve the same benefits for solver convergence. In the non-linear and linear plots the loss is convergent with either the optimized or deep training methods. The deep method trains to a lower loss level and reaches this level in fewer epochs than the optimized method.

The same early predictions at iteration 100 and the more mature results at iteration 400 are extracted for the 2-D discussion. The line plots in Figure 10 compare the baseline solver convergence and the optimized GNN solver convergence at iteration 100, a point early in the GNN training process. Only the

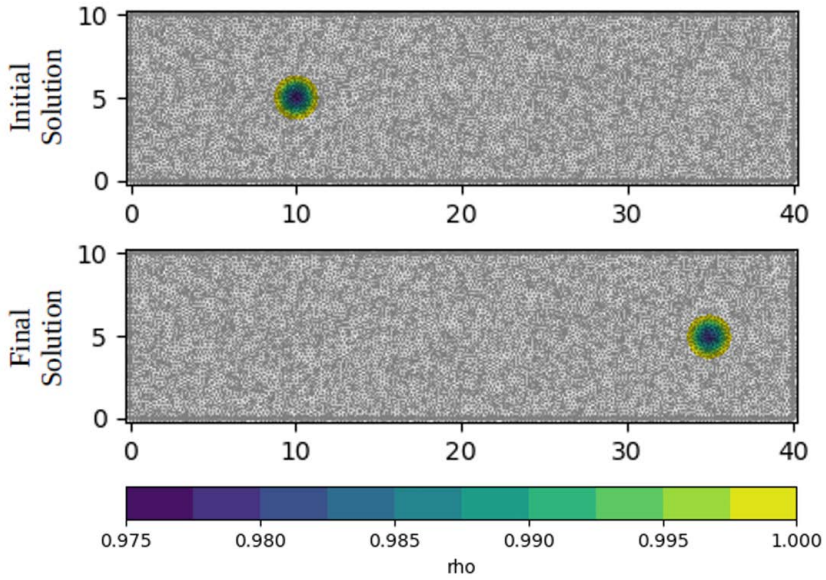

Figure 7: 2D Euler Equation: Initial and final 2-D Euler density contours with the unstructured grid included.

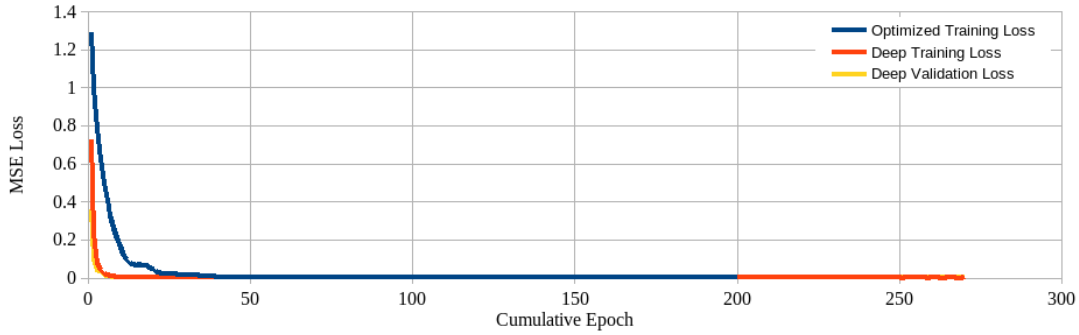

Figure 8: 2D Euler equation: GNN training loss for the non-linear GNN.

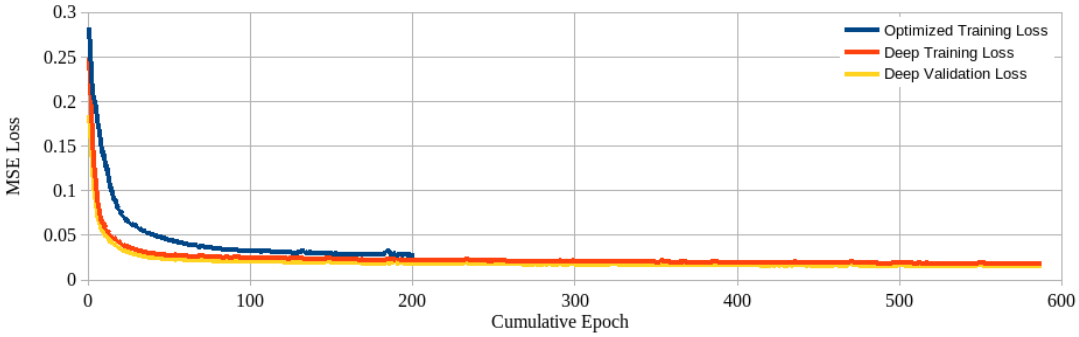

Figure 9: 2D Euler equation: GNN training loss for the linear GNN.

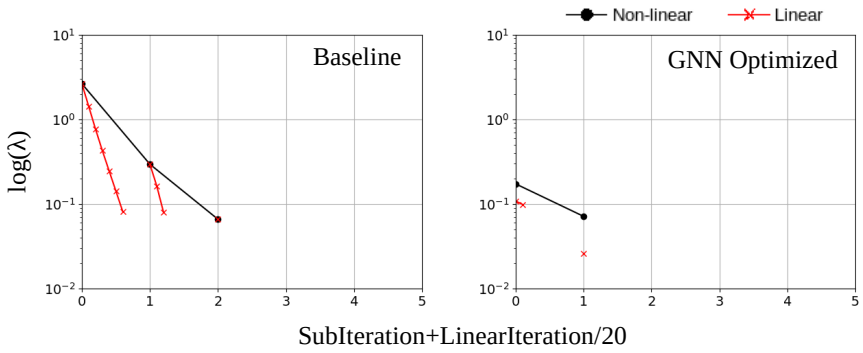

Figure 10: 2D Euler Equation: Comparison of 2-D non-linear and linear convergence between baseline and GNN accelerated methods at iteration 100.

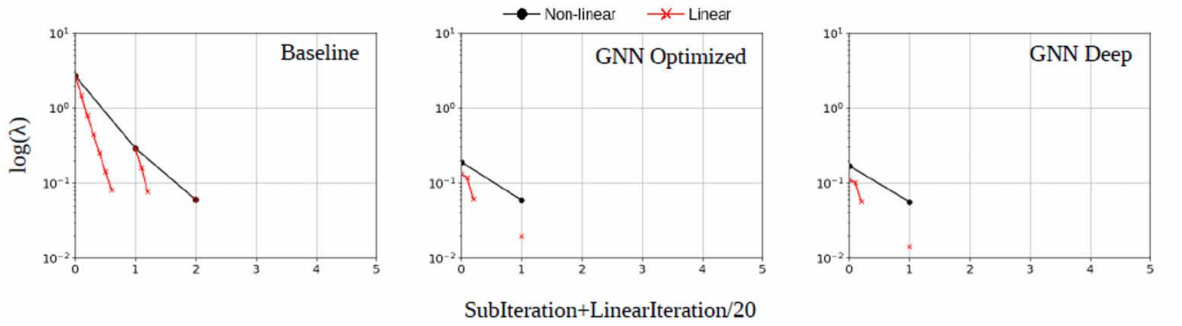

Figure 11: 2D Euler Equation: Comparison of 2-D non-linear and linear convergence between baseline and GNN accelerated methods at iteration 400.

optimized GNN is shown because the deep GNN has not gathered enough data by iteration 100 and has thus not started making predictions. The GNN-accelerated solver reduces the total linear solver iterations by 72.7% relative to the baseline at this snapshot.

The same is true at iteration 400, a later point in the simulation at which the deep GNN has also gathered sufficient examples to train and begin making predictions. The line plots in Figure 11 show the same 72.7% reduction in linear solver iterations. This is true for both the optimized GNN and the deep GNN method where identical convergence benefits are obtained, but at a much cheaper cost using the optimized GNN; thus, enforcing deep training requirements on the method is not required to obtain optimum solver convergence.

The contour plots that correspond to the line plots for the optimized GNN are provided in Figure 12 and Figure 13. These contours compare the instantaneous GNN prediction compared to the instantaneous final solution. The results show that the GNN predictions at both the early and later stages of the simulation are consistent with the non-linear and linear solver solutions. In these figures the linear solver results for all possible subiterations are included.

The per-step snapshots above show the benefit at individual iterations; Figures 14 and 15 provide the global view over the full 500-step simulation. Each bar shows the total linear solver iterations consumed at that time step; the coloured region is the GNN run and the black overlay is the baseline, so the visible black area represents iterations saved. The baseline executes a nearly constant 11–12 linear iterations per time step throughout.

The critical difference between the two configurations is *when* savings begin. The optimized GNN activates around iteration 60 and immediately drops the per-step count to 3–4, sustaining that reduction for the remaining ~440 steps. The deep GNN does not activate until around iteration 300, delivering

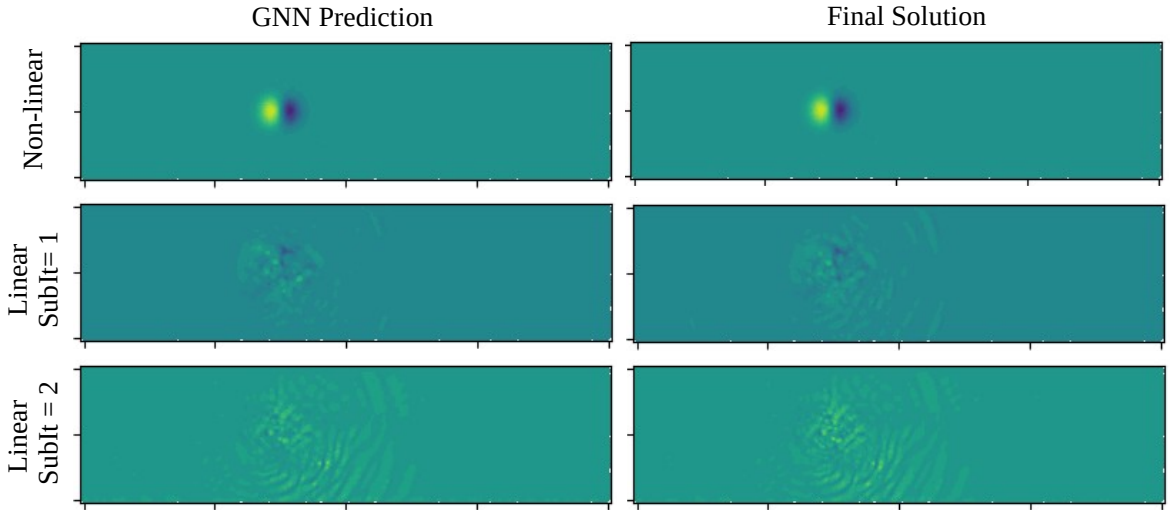

Figure 12: 2D Euler Equation: Comparison of 2-D non-linear and linear GNN predictions with the final solution at iteration 100.

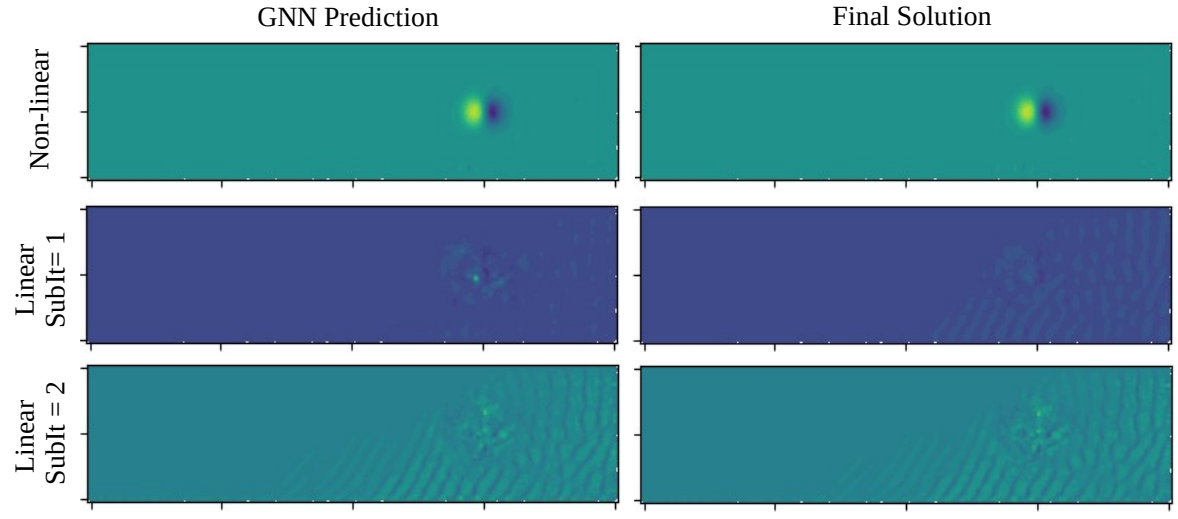

Figure 13: 2D Euler Equation: Comparison of 2-D non-linear and linear GNN predictions with the final solution at iteration 400.

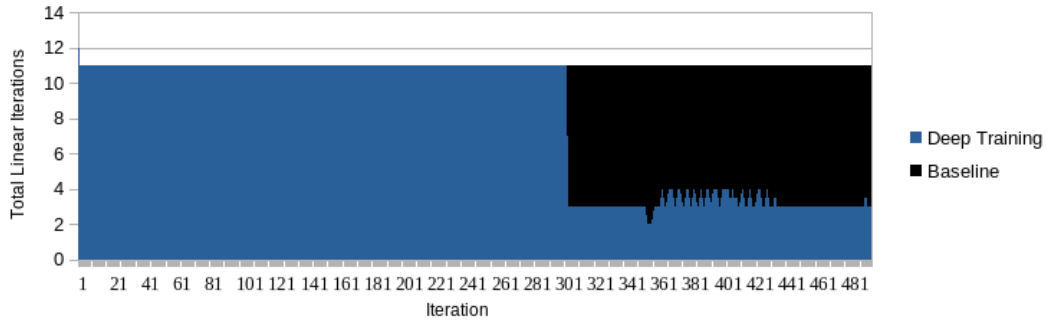

Figure 14: 2D Euler Equation: Total linear solver iterations executed as a function of iteration comparing the deep GNN with the baseline simulation.

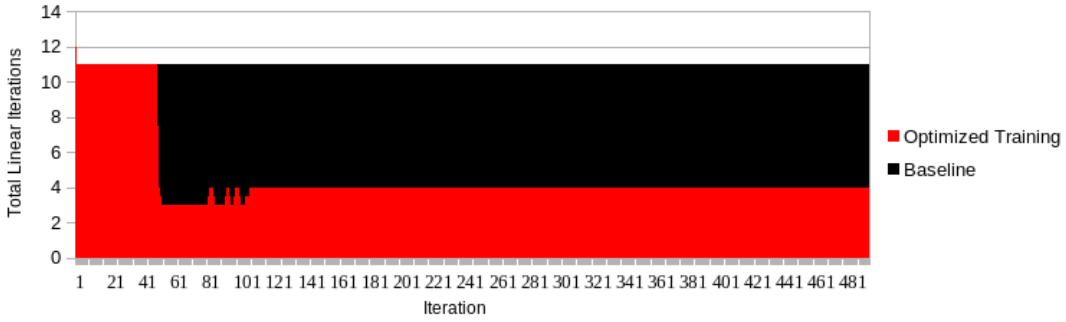

Figure 15: 2D Euler Equation: Total linear solver iterations executed as a function of iteration comparing the optimized GNN with the baseline simulation.

comparable per-step savings but only for the final third of the run. This timing asymmetry has direct implications for long unsteady simulations such as helicopter rotor or turbomachinery flows, where the run may span thousands of time steps across many shedding cycles or blade revolutions: a lightweight model that trains quickly and begins compounding savings early accumulates far greater benefit over the full simulation than a more deeply trained model that activates late. The optimized training strategy is therefore not merely a computational convenience — it is aligned with the operating regime of the production problems that motivate this work.

In order to add further perspective on the impact of GNN quality on the simulation, two additional cases were executed with shallow and very shallow GNN training settings of 50 and 10 epochs respectively. The area plot in Figure 16 illustrates that as the training quality decreases the solver savings also decrease; thus, the quality of the GNN cannot be continually compromised to save time. There is an optimum range of options that balance the quality of the GNN with the improvements to the solver convergence.

Overall, the 2-D Euler investigation confirms that the GNN methodology produces consistent iteration-count savings across a range of training configurations, and that deeper training beyond the optimized settings does not yield further improvement in solver convergence. The following subsection applies the same framework to the Re=100 unsteady cylinder in the C++ solver, extending the demonstration to a wall-bounded viscous flow with a Newton–Krylov GMRES linear solver.

3.2.2 Re=100 Cylinder — Reflex GNN Augmentation

The Re=100 unsteady cylinder is the primary test case for evaluating the Reflex framework in the C++ solver. Three configurations are studied: a coarse and a fine unstructured mesh both using a block-Jacobi linear solver, and the fine mesh with a GMRES linear solver to probe whether Reflex provides additional

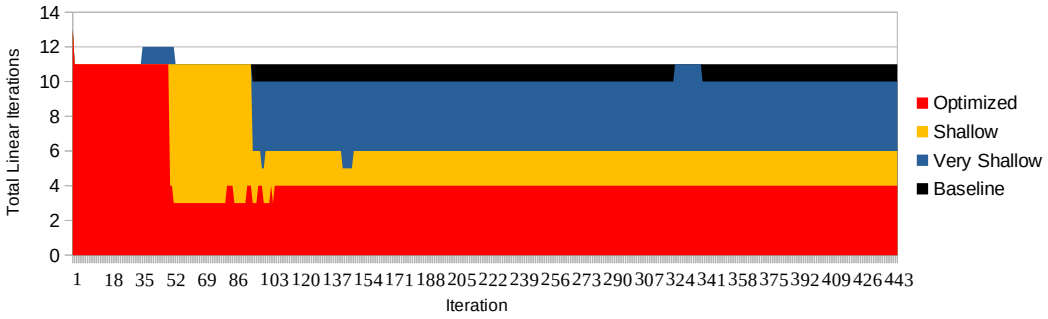

Figure 16: 2D Euler Equation: Total linear solver iterations executed as a function of iteration comparing GNN implementations of varying epoch count.

benefit when deeper linear convergence is sought. All runs use BDF2 time integration with $\Delta t = 0.05$ and 5 Newton sub-iterations per time step. Both smodel and lmodel use a three-layer GNN with 64 hidden neurons, the same architecture as the 2-D Euler case, with the graph defined on the cylinder mesh connectivity. Results are reported at time step 25000, well into the saturated von Kármán shedding regime where the flow is quasi-periodic and the GNN models have been trained on many representative examples.

Nonlinear model (smodel) prediction quality Figure 17 shows the density component of the nonlinear update ΔQ at step 25000 on the coarse grid: the ground-truth update (left), the smodel prediction (centre), and their difference (right). The smodel accurately captures the spatial structure of the von Kármán wake, reproducing the alternating positive and negative density perturbations shed from the cylinder. The prediction error (delta) is roughly an order of magnitude smaller than the signal and is concentrated near the cylinder body and in the vortex cores, where the flow is most sensitive to small phase differences. Similar prediction quality is observed on the fine grid (Figure 18), where the finer wake structure is also well reproduced, confirming that the smodel generalises to the larger, more resolved mesh.

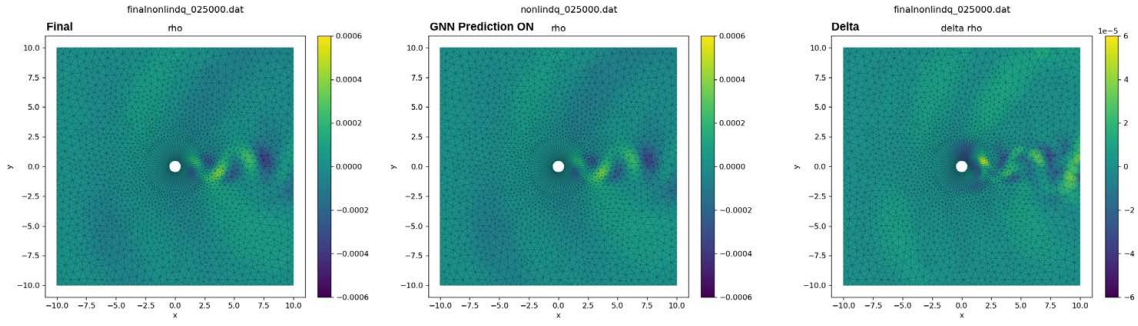

Figure 17: Coarse grid, step 25000: density component of the nonlinear update $\Delta\rho$. Left: ground truth; centre: smodel prediction; right: pointwise error. The GNN captures the large-scale wake structure; errors are concentrated in the vortex cores and are an order of magnitude smaller than the signal.

Newton sub-iteration convergence — block-Jacobi linear solver Figure 19 compares the Newton sub-iteration convergence at step 25000 for the coarse grid under three conditions: baseline (no GNN), nonlinear GNN only (smodel warm-start), and the combined nonlinear+linear GNN (smodel + lmodel warm-start). The x-axis is the Newton sub-iteration index and the y-axis is the nonlinear residual; the red crosses show individual linear solver iterations within each sub-iteration.

The most significant effect is from the smodel warm-start: the initial nonlinear residual entering the first

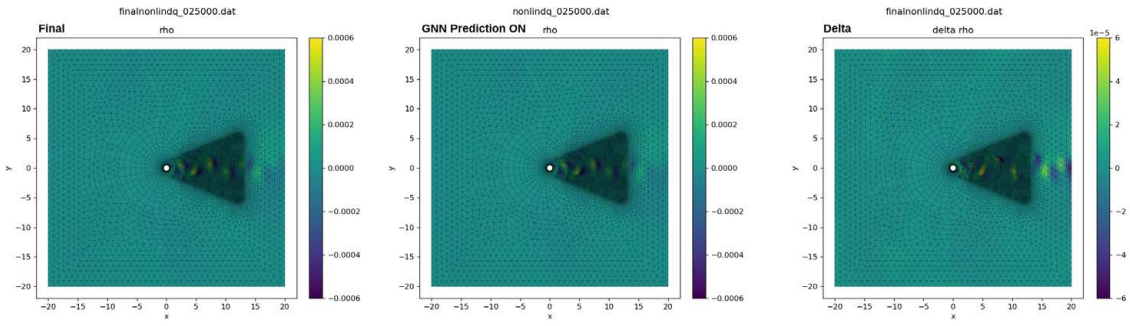

Figure 18: Fine grid, step 25000: density component of the nonlinear update $\Delta\rho$. The smodel reproduces the more finely resolved wake pattern with comparable accuracy to the coarse-grid case.

sub-iteration is reduced by approximately one order of magnitude relative to the baseline, from $O(1)$ to $O(0.1)$. This reflects the smodel providing a high-quality initial guess for the Newton step, effectively pre-solving the bulk of the nonlinear update before any sub-iterations begin. The nonlinear+linear GNN case reaches the same final residual level with one fewer sub-iteration, and the linear iterations within each sub-iteration converge somewhat faster due to the lmodel warm-start.

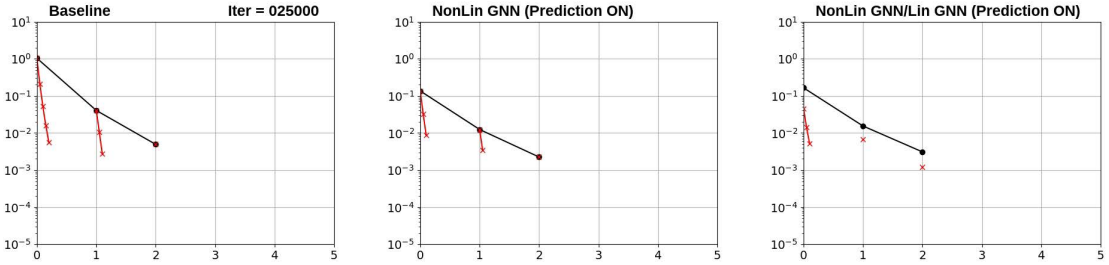

Figure 19: Coarse grid, step 25000: Newton sub-iteration convergence. Left to right: baseline, smodel only, smodel+lmodel. Black dots mark the nonlinear residual after each sub-iteration; red crosses are linear solver iterations. The smodel reduces the initial Newton residual ≈10×; the combined model reduces the required sub-iteration count by one.

The fine-grid results in Figure 20 show a consistent picture. The initial nonlinear residual is larger in absolute terms on the fine mesh (reflecting the higher resolution), and the smodel achieves an approximately 5× reduction rather than 10×. The baseline requires four to five sub-iterations to converge; the GNN-augmented cases reach the same level in three, with the lmodel providing a modest additional benefit in the linear iteration count.

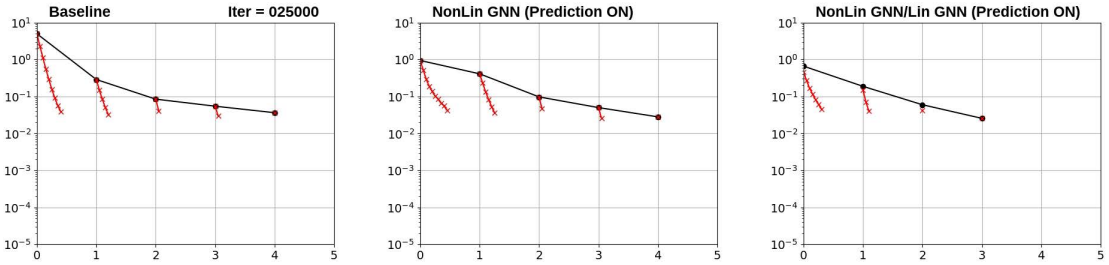

Figure 20: Fine grid, step 25000: Newton sub-iteration convergence for the three configurations. The smodel reduces the initial Newton residual ≈5×; the combined model converges in three sub-iterations versus four to five for the baseline.

Deep convergence with GMRES linear solver To assess whether Reflex provides benefit when the linear solver is converged more tightly, the fine-grid case is repeated with a GMRES linear solver. In addition a deeper training method is utilized for the GNN that uses a larger dataset size and validation loss control for both smodel and lmodel. Results are shown at step 10000, a point at which the GNN models are well-trained; the Jacobi cases above were shown at step 25000 for comparison. Figure 21 shows the Newton convergence at step 10000 for the baseline, the nonlinear-only GNN, and the combined GNN respectively. The GMRES solver drives each sub-iteration's linear residual several orders lower than the block-Jacobi cases, and the nonlinear residual drops correspondingly further per sub-iteration.

The smodel warm-start again provides a substantial reduction in the initial Newton residual, with the combined GNN case converging in two sub-iterations compared to three for the baseline. The benefit of the lmodel is less consistent in the GMRES setting: while the linear iterations converge faster in some sub-iterations, the improvement is not uniform. This is attributed to the wider range of residual magnitudes encountered across sub-iterations — the lmodel training data spans several orders of magnitude in scale, making it a harder regression target than the smodel, which operates on examples of more comparable magnitude.

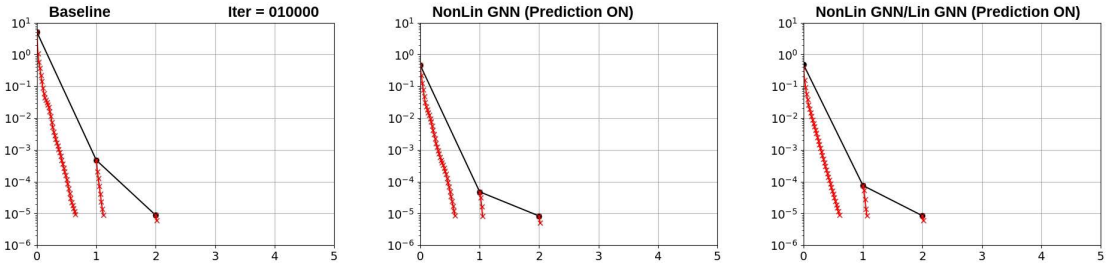

Figure 21: Fine grid GMRES, step 10000: Newton sub-iteration convergence for the three configurations. The smodel reduces the initial Newton residual ≈10×; the addition of the lmodel in the combined model does not show any benefit in this case.

However, comparing the prediction accuracy of smodel and lmodel in Figures 22–24, the prediction quality of both models acceptable, even though lmodel is trained on data that is not only an order of magnitude smaller in scale than the smodel targets, but also spans several orders of magnitude across Newton sub-iterations. Note that the predictions plotted for lmodel are the raw GNN predictions prior to any potential warm-start scaling. In the lmodel plots all three panels utilize the same min/max range, while the smodel plots the delta on a scale that is an order of magnitude smaller to make it easier to distinguish the variations.

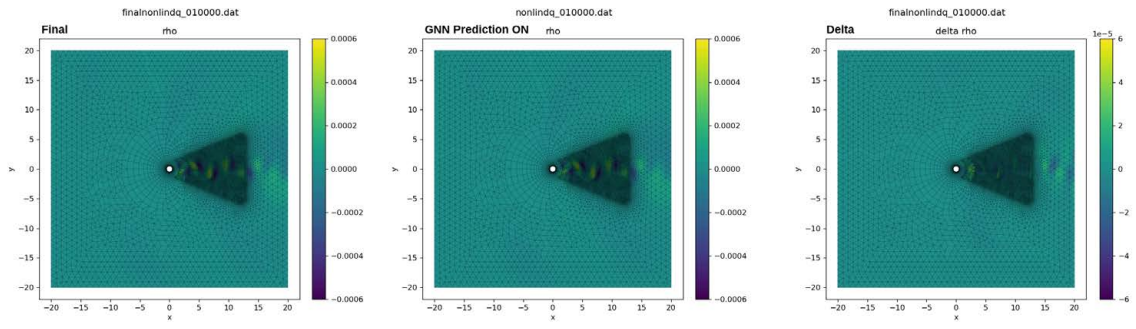

Figure 22: Fine grid GMRES, step 10000: density component of the nonlinear update $\Delta\rho$. The more deeply trained smodel reproduces the wake pattern with improved accuracy compared to the Jacobi case.

A key requirement for any warm-start method is that it must not alter the converged solution. Figure 25 compares the lift and drag histories between the baseline and the combined GNN run. The results are

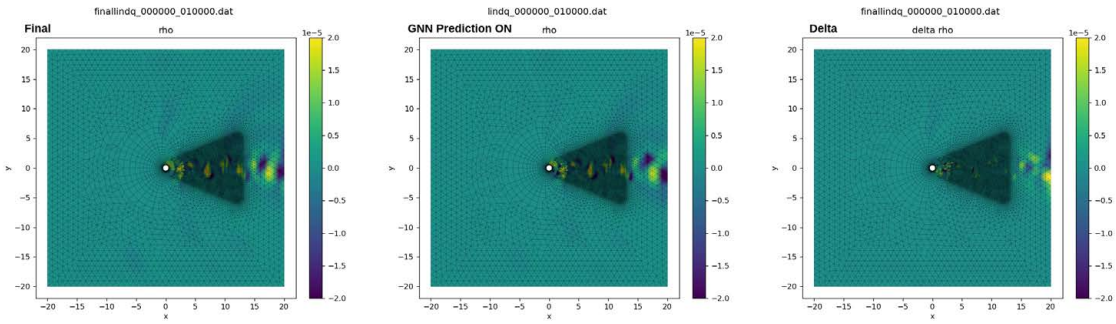

Figure 23: Fine grid GMRES, step 10000: density component of the linear update at subiteration zero $\Delta\rho$. The lmodel reproduces the wake pattern for this reduced order of magnitude data.

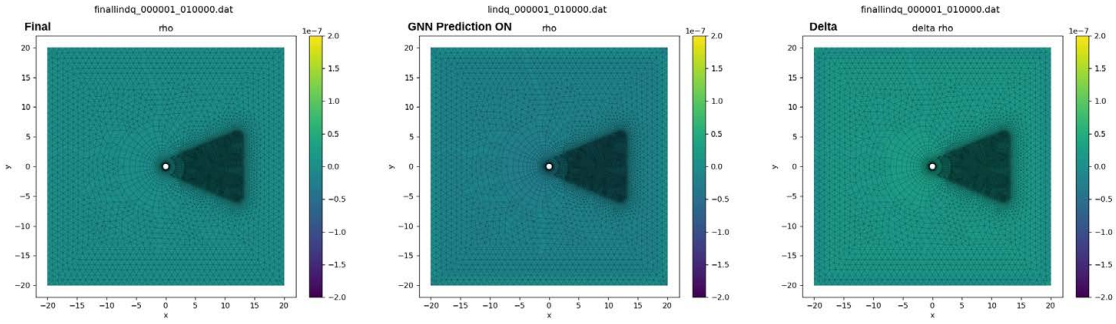

Figure 24: Fine grid GMRES, step 10000: density component of the linear update at subiteration one $\Delta\rho$. The order of magnitude of the data is very small and nearly constant for this more deeply converged case. The lmodel is able to capture this state with little error.

effectively identical, with a maximum pointwise difference of 10^{-5} in C_D and 3×10^{-6} in C_L over the full simulation, confirming that Reflex acts as a pure accelerator with no effect on solution accuracy.

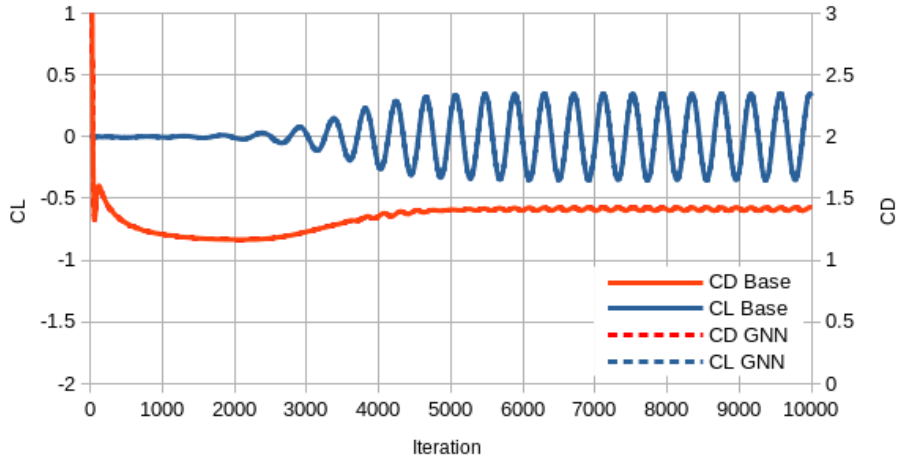

Figure 25: Fine grid GMRES: lift and drag histories for the baseline and combined GNN runs. Results are effectively identical; maximum difference is 10^{-5} in C_D and 3×10^{-6} in C_L.

Total linear iteration savings Table 2 summarises the total linear solver iterations accumulated over the full run for each configuration. The smodel-only case delivers the bulk of the savings across all three mesh and solver combinations; the combined model provides further reduction in all cases to varying

degrees. The savings are cumulative and would continue to grow with additional time steps. The start iteration at which GNN predictions are first applied (*mlstart*) is a tunable parameter: an earlier start increases total savings at the cost of less mature initial predictions, while a later start yields more reliable warm starts but defers the benefit; the relative ordering of configurations remains consistent regardless of this choice.

Table 2: Total linear solver iterations for the Re=100 cylinder cases.

Configuration	Baseline	NonLin GNN	NonLin+Lin GNN
Coarse grid, Jacobi	209K	150K	126K
Fine grid, Jacobi	388K	370K	290K
Fine grid, GMRES	474K	425K	418K

4. Conclusion and Future Work

This paper presents an intermediate step in a deliberate progression toward deploying the Reflex in-situ GNN framework in production-scale three-dimensional CFD simulations. Rather than attempting a direct leap from the one- and two-dimensional Python prototype of the prior work [9] to a multi-million-cell solver, the present effort establishes the method on a fully physics-validated, performance-portable C++ platform that mirrors the structure of production codes in all essential respects.

The solver infrastructure is now entirely C++ and OCCA-based, executing on CPUs and GPUs from a single source. GNN inference and online training are performed via the LibTorch C++ API and run device-resident throughout the simulation, eliminating host–device transfers for the ML components. The linear solver framework has been upgraded to a matrix-free Newton–Krylov GMRES scheme with a block-Jacobi preconditioner and a Fréchet-derivative matrix–vector product — a structure directly comparable to that used in production unstructured-mesh solvers. Viscous compressible fluxes, no-slip wall boundary conditions, and a Reynolds-driven boundary-layer mesh pipeline extend the solver to wall-bounded external flows, validated against published drag and Strouhal number benchmarks for the Re=40 steady cylinder, Re=100 unsteady cylinder, and NACA0012 at laminar Reynolds numbers.

The central result for the Reflex framework is that the nonlinear model (smodel) delivers substantial and reliable improvement across all cases tested. On the two-dimensional isentropic vortex and on both the coarse and fine Re=100 cylinder meshes, the smodel warm-start reduces the initial Newton residual entering the first sub-iteration by approximately one order of magnitude. This translates directly into a reduction of one to two Newton sub-iterations per time step — the dominant cost in an implicit unsteady simulation. The prediction quality is high: the smodel accurately reproduces the large-scale spatial structure of the nonlinear update field, with prediction errors concentrated in the vortex cores and an order of magnitude below the signal.

The linear model (lmodel), by contrast, shows mixed results. In cases where the lmodel is effective it accelerates the convergence of individual linear solves, providing a further reduction in iteration count beyond what the smodel alone achieves. However, the benefit is not consistent: the lmodel training data spans a wide range of residual magnitudes across Newton sub-iterations, making the regression problem significantly harder than for the smodel. The amplitude scaling and residual guard mechanisms introduced in this work — which detect and correct warm-start predictions that would increase rather than decrease the linear residual — were essential for numerical robustness, but they also mask cases where the lmodel prediction is out-of-distribution. Improving lmodel reliability, either through better normalisation of training targets, separate models per sub-iteration, or a more principled handling of the scale disparity, is identified as the key open problem before the combined framework can be relied upon in production.

Wall-clock time comparisons are intentionally deferred in this work. The test cases presented here — a few thousand cells on a single GPU — complete in seconds, a regime where GPU kernel launch overhead

and the fixed cost of the LibTorch training loop are comparable to or larger than the solver iteration time itself. Meaningful end-to-end speedups will only emerge at the mesh sizes and time-step counts representative of production problems, where the solver dominates the runtime and the amortised cost of online training is negligible. The present results therefore focus entirely on iteration-count reduction as the portable, hardware-independent measure of benefit.

With the solver infrastructure, physical validation, and GNN robustness mechanisms now in place, the path to a production implementation is well defined. The immediate next step is integration with the GPU-accelerated Rapidus solver [10] on rotationally periodic turbomachinery and helicopter rotor configurations, where the quasi-periodic nature of the flow over many blade revolutions provides precisely the stationary training distribution that the smodel exploits most effectively.

Acknowledgements

The material presented in this paper is a product of the CREATE-AV element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. Funding and support for the presented research is also provided by the US Army DEVCOM Aviation and Missile Center.

References

[1] E. Ajuria-Illarramendi, A. Alguacil, M. Bauerheim, A. Misdariis, B. Cuenot, and E. Benazera. Towards an hybrid computational strategy based on deep learning for incompressible flows. In *AIAA AVIATION 2020 Forum*, AIAA 2020-3058. AIAA, June 2020.

[2] K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 6111–6122, Red Hook, NY, 2020. Curran Associates, Inc.

[3] X. Huang, Z. Ye, H. Liu, S. Ji, Z. Wang, K. Yang, Y. Li, M. Wang, H. Chu, F. Yu, B. Hua, L. Chen, and B. Dong. Meta-auto-decoder for solving parametric partial differential equations. In *Advances in Neural Information Processing Systems*, volume 35, pages 23426–23438, Red Hook, NY, 2022. Curran Associates, Inc.

[4] Y. Vaupel, N. C. Hamacher, A. Caspari, A. Mhamdi, I. G. Kevrekidis, and A. Mitsos. Accelerating non-linear model predictive control through machine learning. *Journal of Process Control*, 92:261–270, August 2020.

[5] K. Luna, K. Klymko, and J. Blaschke. Accelerating gmres with deep learning in real-time. *arXiv preprint arXiv:2103.10975*, 2021.

[6] S. Nikolopoulos, I. Kalogeris, V. Papadopoulos, and G. Stavroulakis. Ai-enhanced iterative solvers for accelerating the solution of large scale parametrized systems. *arXiv preprint arXiv:2207.02543*, 2022.

[7] S. Arisaka and Q. Li. Principled acceleration of iterative numerical methods using machine learning. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, pages 1041–1059, Honolulu, Hawaii, 2023.

[8] L. Saverio. Accelerating convergence of linear iterative solvers using machine learning. Master's thesis, Politecnico di Milano, 2023. Tesi di Laurea Magistrale in Mathematical Engineering - Ingegneria Matematica.

[9] J. Abras, J. Sitaraman, S. Hosseinverdi, and N. Hariharan. Enhancing production-oriented non-linear PDE solver convergence with learnable models. In *AIAA SCITECH 2026 Forum*. AIAA, January 2026.

[10] S. Hosseinverdi, J. Sitaraman, D. Jude, and P. Premaratne. On the development of a performance-portable parallel flow solver for aerospace applications. In *AIAA SCITECH 2025 Forum*. AIAA,

January 2025.

[11] R. D. Henderson. Details of the drag curve near the onset of vortex shedding. *Physics of Fluids*, 7(9):2102–2104, 1995.

[12] C. H. K. Williamson. Defining a universal and continuous Strouhal–Reynolds number relationship for the laminar vortex shedding of a circular cylinder. *Physics of Fluids*, 31(10):2742–2744, 1988.

[13] G. Di Ilio, D. Chiappini, S. Ubertini, G. Bella, and S. Succi. Fluid flow around NACA 0012 airfoil at low-Reynolds-number and various angles of attack. *Fluids*, 5(1):6, 2020.