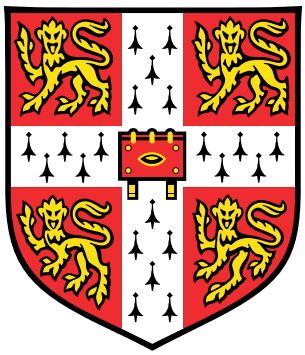
Physics-Informed Graph Neural Networks with Divergence-Free Constraints for Incompressible Flow Simulations



Adam Dray

Department of Physics
University of Cambridge

This dissertation is submitted for the degree of

Master of Philosophy

August 2025

Declaration

This project report is substantially my own work and conforms to the University of Cambridge's guidelines on plagiarism. Where reference has been made to other research this is acknowledged in the text and bibliography.

Adam Dray
August 2025

Acknowledgements

Thank you to James Rowbottom and Stefania Fresca for supervising this project and for their guidance and feedback throughout.

Abstract

This work develops Graph Neural Network (GNN) surrogate models for incompressible fluid simulations. Unlike conventional machine learning approaches, GNNs operate directly on unstructured meshes prevalent in industrial computational fluid dynamics (CFD). A complete pipeline for dataset generation, preprocessing, and distributed training was constructured – leveraging OpenFOAM and PyTorch – and Li et al.'s Finite-Volume-Graph-Network (FVGN) baseline is reproduced on both the *CylinderFlow* benchmark and a custom *EllipseFlow* dataset. The absence of hard conservation constraints in FVGN reduces the reliability and physical interpretability of its predictions, motivating us to draw on finite-volume principles to design explicitly conservative alternatives. To this end, conservation-inspired variants are introduced, some embedding hard divergence-free constraints and others reducing divergence through conservative message passing. These physics-inspired architectures improve conservation but also reveal optimisation fragility, while complementary strategies such as pushforward training and physically consistent integration provide more robust gains. Results show that conservation constraints alone do not guarantee stability or robustness; effective surrogates emerge from a balance between physical fidelity and learnability. These findings establish GNN surrogates as a promising framework for fast, mesh-native CFD, while emphasising that broader applicability will depend on addressing challenges in robustness, out-of-distribution generalisation, and integration into real-world engineering workflows.

Contents

Part I

First Report

1. Introduction to Report I

Computational fluid dynamics (CFD) underpins a breadth of scientific and engineering applications, from automotive and aerospace design to environmental modelling [1, 2, 3, 4]. In the majority of these contexts, flow velocities remain considerably below the speed of sound, justifying the assumption of constant fluid density and thereby reducing the governing Navier–Stokes equations to their incompressible form. Traditional approaches to solving these equations, such as the Finite Volume Method (FVM), are computationally intensive, typically using iterative schemes to resolve the pressure–velocity coupling and enforce incompressibility [5, 6, 7]. This motivates the development of new methods that can deliver solutions to the incompressible Navier-Stokes equations at a reduced computational cost.

While advances in computational hardware – including multi-core CPUs, GPUs and multi-node supercomputers – have accelerated traditional computational fluid dynamics (CFD) solvers, the inherent complexity and limited parallel scalability of the numerical algorithms often prevent them from fully exploiting modern chip architectures [8]. In contrast, deep learning algorithms are designed to maximise the power of parallel hardware since they rely almost exclusively on dense matrix operations [9]. This has catalysed interest in deep learning-based approaches to performing physical simulations, especially in the context of CFD [10, 11, 12, 13].

This research focuses on the use of deep learning models, specifically Graph Neural Networks (GNNs), as *surrogate models* for incompressible flow simulations. Surrogate modelling refers to the approximation of high-fidelity numerical solvers by machine learning models trained on data generated from classical simulations. These models aim to replicate the mapping from input parameters (e.g., geometry, boundary conditions) to physical quantities of interest, at significantly reduced computational cost. In the context of CFD, GNNs are particularly well-suited to this task as they operate natively on graph-structured data, making them ideal for unstructured meshes that are prevalent in

industrial simulations.

A persistent challenge in surrogate modelling is the incorporation of physical knowledge into the learned models. Standard data-driven approaches risk overfitting to training distributions and may fail to generalise to unseen domains or violate key physical constraints. To mitigate this, physics-informed neural networks (PINNs) includes physics priors into the training objective by penalising the residuals of the governing partial differential equations (PDEs) [14]. This encourages the network to learn solutions that are not only consistent with data but also approximately satisfy the underlying physics.

For incompressible flows, a defining constraint is the divergence-free condition on the velocity field, which ensures mass conservation. Classical FVM solvers satisfy this condition exactly through their integral formulations [15]. In contrast, physics-informed surrogate models only minimise deviations from the governing equations and do not enforce the constraints exactly. As a result, these are referred to as "soft" or "weak" constraints. In the absence of exact conservation, errors can accumulate over time, undermining the stability and reliability of the simulation.

To address these limitations, recent research has explored "hard" constraint enforcement, where physical conditions such as incompressibility are built directly into the network architecture. Techniques including divergence-free projections, stream or vector potentials, and constraint-preserving neural operators have shown promise in convolutional and mesh-free models [16, 17, 18, 19]. These approaches will be examined in greater depth in the context of GNNs in the second report.

This first report lays the theoretical and methodological foundation for developing graph neural network (GNN) surrogates for incompressible flow simulations. It reviews recent advances in CFD surrogate modelling, with a focus on GNN architectures and techniques for enforcing hard physical constraints, and identifies a research gap at their intersection. As a step toward addressing this gap, a recently proposed GNN incorporating a weak divergence-free constraint – Li et al.'s Finite-Volume-Graph-Network (FVGN) [20] – is implemented. A full pipeline for dataset generation, model training, and evaluation is also established. The model is evaluated on both a standard benchmark dataset (CylinderFlow) [11] and a custom-generated dataset created using OpenFOAM (LaminarEllipse).

These preliminary results assess the model's ability to produce accurate, stable and conservative simulations and reveal key limitations associated with the weak constraint formulation. They motivate the introduction of a hard continuity constraint and guide

the design of architectural modifications. These findings form the basis for the central contribution of Report 2: the development of a graph-based architecture with a hard divergence-free constraint aimed at improving long-term rollout stability and generalisation to unseen flow conditions.

2. Background

2.1 Fluid Dynamics

The behaviour of fluids can be described by the Navier-Stokes equations, which represent the conservation of mass, momentum and energy. When deriving these equations, it can be useful to consider a discrete "control volume" V, with a boundary surface ∂V (see Figure 2.1A). The general conservation law for a fluid property ϕ within a control volume V can be formulated as a balance between the rate of change of ϕ, the flux through the boundary ∂V, and any internal source terms:

$$\frac{d}{dt}\int_V \phi \,\mathrm{d}V = -\int_{\partial V} F \,\mathrm{d}S + \int_V s \,\mathrm{d}V, \tag{2.1}$$

where the face flux is $F = \phi\mathbf{u}\cdot\mathbf{n}$, $\mathbf{u}$ is the fluid velocity, $\mathbf{n}$ is the surface normal and s represents any source or sink terms.

Applying the divergence theorem to convert surface integrals into volume integrals, and assuming that V is infinitesimally small (leading to a local, differential form), the conservation equations for mass ρ, momentum ρu and total energy E, can be written as:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0 \tag{2.2}$$

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v}) = -\nabla P + \nabla\cdot\boldsymbol{\tau} + \rho\mathbf{f} \tag{2.3}$$

$$\frac{\partial E}{\partial t} + \nabla\cdot((E+P)\mathbf{u}) = \nabla\cdot(\kappa\nabla T) + \nabla\cdot(\mathbf{u}\cdot\ \tau) + \rho\mathbf{f}\cdot\mathbf{u} \tag{2.4}$$

where f are the body forces, κ the thermal conductivity, τ the viscous stress tensor and P is the pressure determined by an equation of state. Equations 2.2-4 are the general "conservation" form of the Navier-Stokes equations [21].

2.1.1 The Incompressible Limit

For low-speed flows, it is reasonable to assume that variation in fluid density is negligible ($\nabla\rho = \partial_t\rho = 0$). Typically this occurs when the ratio of u and sound speed c_s (the Mach number) is less than 0.3. With additional assumptions of constant viscosity, negligible bulk viscosity and no external forcing, the Navier-Stokes equations simplify significantly:

$$\nabla \cdot \mathbf{u} = 0 \tag{2.5}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = -\nabla \cdot (\nu \nabla \mathbf{u}) - \frac{1}{\rho}\nabla p \tag{2.6}$$

where ν is kinematic viscosity ($\nu = \mu/\rho$). Equation 2.5 and 2.6 are the incompressible continuity and momentum equations, respectively. Future references to the continuity and momentum equations pertain to their incompressible form.

Note that the energy equation is decoupled from the mass and momentum equations because, under the assumption of constant density, the equation of state is no longer applicable and pressure is not thermodynamically determined. Consequently, the energy equation can be solved independently once the flow field is known.

The momentum equation contains a convective term, $\nabla \cdot (\mathbf{u} \otimes \mathbf{u})$, representing the transport of momentum by the fluid motion itself, and a viscous term, $\nabla \cdot (\nu \nabla \mathbf{u})$, accounting for the diffusion of momentum due to molecular viscosity. To assess the relative importance of inertial (convective) and viscous effects in a flow, a dimensionless ratio known as the Reynolds number is introduced. By defining a characteristic velocity scale U and length scale L, the Reynolds number is given by

$$\mathrm{Re} = \frac{UL}{\nu},$$

where ν is the kinematic viscosity. This ratio quantifies the relative magnitude of inertial forces ($\sim \rho U^2$) to viscous forces ($\sim \mu U/L$) in the flow. At low Re, viscous effects dominate, damping velocity fluctuations and promoting laminar, stable flow. As Re increases past a critical value, defined by geometry, convective effects overcome viscous damping, leading to amplification of perturbations and the onset of instabilities. At sufficiently high Re, these instabilities undergo non-linear interactions, leading to a transition to turbulent flows.

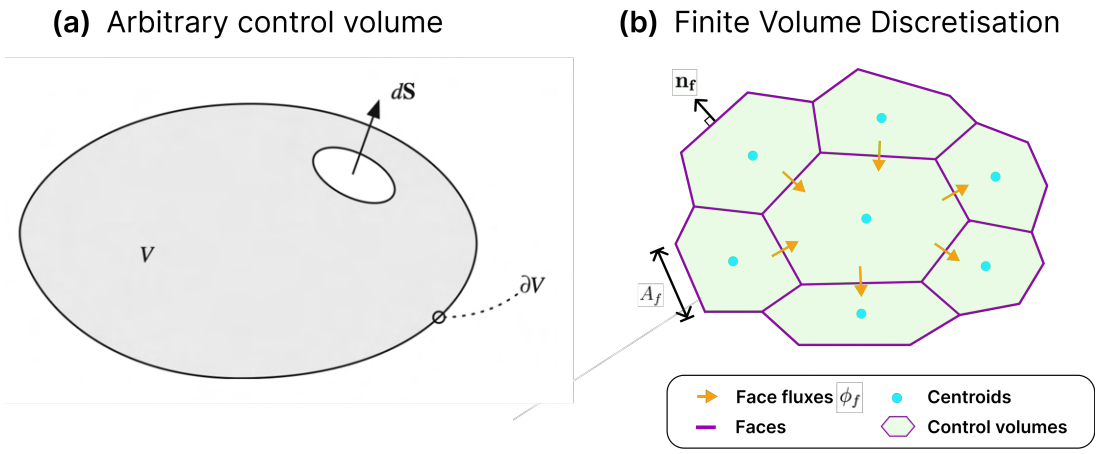


Figure 2.1: **(a)** Arbitrary control volume V with boundary surface ∂V. $d\mathbf{S} = \mathbf{n}\,\mathrm{d}S$ is the surface area element vector. This diagram is credited to [22]. **(b)** Finite volume discretisation of a 2D geometry into control volumes or "cells". The face fluxes at defined at the face centroids and solution variables are defined at the control volume centroids. In finite volume literature, 3D terminology is often used even for 2D discretisation – for instance, control volumes are technically areas, and face areas are lengths.

2.2 Finite Volume Method

Many partial differential equations (PDEs), such as the Navier-Stokes equations, cannot be solved analytically and therefore require numerical methods. Aside from spectral methods [23], most numerical strategies involve spatial discretisation: finite difference methods discretise derivatives on grid points; finite volume methods apply conservation laws over control volumes; and finite element methods use local basis functions within a variational framework [24]. In each case, the continuous PDEs are ultimately transformed into systems of algebraic equations.

The Finite Volume Method (FVM) is widely used for physical simulations, due to its inherent conservation properties. The domain is divided into non-overlapping control volumes (see Figure 2.1B), and the governing equations are integrated over each control volume. In this work, we adopt a cell-centred finite volume method, where each control volume is associated with a single computational "cell" – the terms cell and control volume will be used interchangeably. For incompressible flows, the integral form of the Navier-Stokes equations over a control volume V are:

$$\int_V \nabla \cdot \mathbf{u}\,\mathrm{d}V = 0, \tag{2.7}$$

$$\frac{\partial}{\partial t}\int_V \mathbf{u}\,\mathrm{d}V = -\int_V \nabla \cdot (\mathbf{u} \otimes \mathbf{u})\,\mathrm{d}V - \frac{1}{\rho}\int_V \nabla p\,\mathrm{d}V + \nu \int_V \nabla^2 \mathbf{u}\,\mathrm{d}V. \tag{2.8}$$

The divergence theorem can be used to convert volume integrals into surface integrals over the boundary ∂V:

$$\int_{\partial V} \mathbf{n} \cdot \mathbf{u} \, \mathrm{d}S = 0, \tag{2.9}$$

$$\frac{\partial}{\partial t} \int_V \mathbf{u} \, \mathrm{d}V = -\int_{\partial V} \mathbf{n} \cdot (\mathbf{u} \otimes \mathbf{u}) \, \mathrm{d}S - \frac{1}{\rho} \int_{\partial V} \mathbf{n} p \, \mathrm{d}S + \nu \int_{\partial V} \mathbf{n} \cdot \nabla \mathbf{u} \, \mathrm{d}S, \tag{2.10}$$

These integrals can be approximated with discrete sums over the cell faces:

$$\sum_f \mathbf{u}_f \cdot \mathbf{S}_f = \sum_f F_f = 0, \tag{2.11}$$

$$\frac{\partial}{\partial t} \int_V \mathbf{u} \, \mathrm{d}V = -\sum_f (\mathbf{u}_f \otimes \mathbf{u}_f) \cdot \mathbf{S}_f - \frac{1}{\rho} \sum_f p_f \, \mathbf{S}_f + \nu \sum_f (\nabla \mathbf{u})_f \cdot \mathbf{S}_f. \tag{2.12}$$

where F_f is the face flux, $\mathbf{S}_f = \mathbf{n}_f A_f$ is the face area vector, $\mathbf{n_f}$ is the face normal vector and A_f is the face area. Since each face is shared by exactly two neighbouring control volumes and the flux through a face is equal and opposite for each volume, this formulation guarantees strict local conservation – any quantity leaving one volume enters the adjacent one.

The properties at the faces centroids, such as velocity $\mathbf{u}_f$, ∇p_f and $\nabla(\mathbf{u})_f$ must be interpolated from the cell-centred solution. For example, central differencing interpolatation is defined as:

$$\phi_f = \frac{x_N - x_f}{x_N - x_P} \phi_P + \frac{x_f - x_P}{x_N - x_P} \phi_E$$

where x_P, x_N, and x_f are the centroids of adjacent control volumes P and N, and their shared face f, respectively. The choice of interpolation scheme governs spatial accuracy and stability. The span of neighbouring cells involved defines the *numerical stencil*.

To advance the solution in time, the temporal term must also be discretised. While second-order time integration schemes offer improved accuracy and stability, a first-order scheme is often used due to its lower computational cost and sufficient accuracy in many cases. The temporal discretisation takes the form:

$$\frac{\partial}{\partial t} \int_V \mathbf{u} \, \mathrm{d}V \approx \frac{|V|}{\Delta t} \left(\mathbf{u}^{n+1} - \mathbf{u}^n \right), \tag{2.13}$$

where $|V|$ is the control volume. Substituting this into the left-hand side of Equation 2.12 requires a choice of whether to evaluate face-interpolated values on the right-hand side using the solution at t^n or t^{n+1}:

- **Explicit schemes** use known solutions at t^n. This enables fully decoupled updates of each control volume, allowing simple and parallel implementations. However, stability requires that the time step Δt satisfies a Courant–Friedrichs–Lewy (CFL) condition:

$$\Delta t \leq \frac{C\Delta h_{\min}}{|\mathbf{u}_{\max}|}, \tag{2.14}$$

  where C is the Courant number (typically $C \leq 1$), $\Delta h_{\min}$ is the smallest mesh spacing, and $\mathbf{u}_{\max}$ is the maximum velocity magnitude in the domain.

- **Implicit schemes** evaluate terms using the unknown solution at t^{n+1}, leading to a more stable formulation not limited by the CFL condition. Because fluxes at cell faces depend on unknown values from neighbouring cells – introduced through interpolation – the resulting discretised equations are coupled. The resulting algebraic system must be solved simultaneously at each time step. For incompressible flows, this fully coupled system takes the block form:

$$\begin{bmatrix} \mathbf{A} & (\nabla) \\ (\nabla\cdot) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{s} \\ 0 \end{bmatrix}, \tag{2.15}$$

  where $\mathbf{A}$ includes contributions from the time derivative and discretised momentum equations, (∇) and $(\nabla\cdot)$ are discrete gradient and divergence operators, and $\mathbf{s}$ includes source terms and boundary conditions from the previous time step.

Implicit methods are generally preferred for engineering flow simulations due to their robustness and unconditional stability, though they require more complex and computationally intensive solvers.

2.2.1 Segregated Methods

The challenge for an implicit numerical method is to solve Equation 2.15. Rather than monolithically solving the fully coupled system, segregated methods decouple the momentum and continuity equations. An initial guess for the pressure field p^* is used to solve the momentum equation for a provisional velocity field [25]:

$$a_P\mathbf{u}^* = H(\mathbf{u}) - \nabla p^*, \tag{2.16}$$

where $\mathbf{u}^*$ is the provisional velocity, a_P represents the diagonal coefficients of $\mathbf{A}$ and $H(\mathbf{u}) = \mathbf{s} - \sum_N a_N^{\mathbf{u}}\mathbf{u}_N$ contains the off-diagonal coefficient a_N and source contributions s. Note that the convention of defining p as kinematic pressure has been followed Equations

2.16 and 2.17.

This provisional velocity generally violates the continuity equation. To enforce incompressibility, the momentum equation is substituted into the continuity equation, yielding a pressure Poisson equation:

$$\nabla \cdot \left(a_P^{-1} \nabla p'\right) = \nabla \cdot \left(a_P^{-1} H(\mathbf{u})\right), \tag{2.17}$$

where $p^{'}$ is the corrected pressure. Solving this equation provides a corrected pressure field, which is used to construct face-normal fluxes. These fluxes are exactly conservative by construction, ensuring zero net flux through each control volume. When used to update the velocity field, they yield a velocity that is discretely divergence-free, thereby enforcing incompressibility at the discrete level.

The approach to updating the velocity depends on the pressure–velocity coupling strategy. In SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) [7], corrected fluxes are used to compute a divergence-free velocity without re-solving the momentum equation. Since the initial pressure guess p^* is only approximate, the process is iterated until convergence, making it well-suited to steady-state problems. Under-relaxation is typically applied to pressure and velocity updates to enhance stability. In contrast, the PISO (Pressure Implicit with Splitting of Operators) algorithm [5] applies multiple pressure corrections per time step, often re-solving the momentum equation between corrections. This yields a more accurate velocity field within each step, making PISO more suitable for transient simulations.

Iterative linear solvers, such as Gauss–Seidel or conjugate gradient methods, are used to solve the momentum and pressure equations [15]. Alternatively, more robust solvers, such as algebraic multigrid (AMG) can be employed to solve the fully block-coupled system (Equation 2.15) directly, eliminating the need for iterations over pressure–velocity coupling [26]. However, segregated solvers remain widely used due to their computational efficiency, lower memory requirements and algorithmic flexibility.

2.2.2 Turbulence Modelling

Resolving turbulence accurately requires 3D simulations capturing the cascade of kinetic energy from large-scale flow structures down to the Kolmogorov scale, where viscous dissipation dominates and the energy is irreversibly dissipated as heat. Direct Numerical Simulation (DNS), which resolves all turbulent scales without modelling, is computationally prohibitive for most practical flows due to the vast range of scales involved. Therefore,

turbulence modelling approaches are used to approximate the effects of unresolved small-scale motions, enabling the use of coarser computational meshes [27].

Reynolds-Averaged Navier–Stokes (RANS) methods model the time-averaged velocity field, $\overline{u} = \frac{1}{T}\int_0^T u(t), \mathrm{d}t$, by averaging out the time-dependent turbulent fluctuations. Alternatively, Large Eddy Simulation (LES) resolves large-scale motions using a spatial filter, $\overline{u}(\mathbf{x}, t) = \int_V \psi(\mathbf{x} - \mathbf{x}')u(\mathbf{x}', t), \mathrm{d}V'$, and models the smaller, subgrid-scale effects. A widely used subgrid-scale model in LES is the Smagorinsky model [28], which introduces an effective viscosity based on the local strain rate and grid scale to represent energy dissipation by unresolved small eddies. Both RANS and LES introduce additional transport equations to represent the impact of unresolved turbulent fluctuations, typically solving for quantities such as turbulent kinetic energy (k), dissipation rate (ϵ), specific dissipation rate (ω) or turbulent viscosity.

The choice between approaches depends on the trade-off between computational cost and accuracy: RANS is well-suited for statistically steady flows, while LES is the preferred method for capturing transient, unsteady turbulence, offering greater fidelity at a higher computational cost.

2.3 Divergence-Free Vector Fields

The continuity equation dictates that the velocity field must be divergence-free to ensure conservation of mass. Since the divergence-free constraint is a major focus of this study, this section briefly introduces concepts relevant to divergence-free vector fields.

2.3.1 Constructing Divergence-Free Vector Fields

Any sufficiently smooth vector field $\mathbf{v}$ defined on a suitable domain in $\mathbb{R}^2$ can be decomposed into two orthogonal components: an irrotational (scalar curl-free) part and a solenoidal (divergence-free) part. This result is known as the *Helmholtz–Hodge decomposition*, and in two dimensions it takes the form [29]:

$$\mathbf{v} = \nabla\phi + \nabla^{\perp}\psi, \tag{2.18}$$

where ϕ and ψ are scalar potentials, and the operator $\nabla^{\perp} = (\partial_y, -\partial_x)$ denotes the rotated gradient. The term $\nabla\phi$ is scalar curl-free, while $\nabla^{\perp}\psi$ is divergence-free.

This decomposition is justified by basic identities in vector calculus. Specifically, for

any scalar field ϕ, the scalar curl of its gradient vanishes:

$$\partial_x(\partial_y\phi) - \partial_y(\partial_x\phi) = 0,$$

and for any scalar field ψ, the divergence of its rotated gradient also vanishes:

$$\nabla \cdot (\nabla^\perp\psi) = \partial_x(\partial_y\psi) - \partial_y(\partial_x\psi) = 0.$$

These properties ensure that the components $\nabla\phi$ and $\nabla^\perp\psi$ are, respectively, scalar curl-free and divergence-free [30].

Proofs of these results will be presented in Report 2, where a more detailed discussion is given in the context of incorporating a hard divergence-free constraint into the architecture of a neural network.

2.3.2 Measuring the Divergence on a Discrete Mesh

Since this study focuses on enforcing the divergence-free constraint, it is important to define how divergence is computed on an unstructured mesh. Two common approaches are summarised below [31]:

- **Least-squares reconstruction** estimates gradients by minimising the error between predicted and actual values at neighbouring points. It is robust to mesh skewness and non-orthogonality, and applicable to both vertex- and cell-centred data.
- **Green–Gauss integration** applies Equation 2.11 to compute divergence from face fluxes. This method is standard in FVM solvers, where divergence-free constraints are enforced to machine precision. Accuracy may degrade on skewed or non-orthogonal meshes without correction.

As the datasets used for training are generated using FVM, the Green-Gauss method is the natural choice. However, interpolation and non-orthogonal corrections require care on unstructured meshes. For a given cell c, the local divergence is defined as

$$(\nabla \cdot \mathbf{u})_c \approx d_c = \frac{1}{V_c}\sum_f F_f, \tag{2.19}$$

where $_f = \mathbf{u}_f \cdot \mathbf{n}_f|A_f|$ is the face-normal flux. First-order interpolation of $\mathbf{u}_f$ from adjacent cell values is generally adequate on good quality meshes. When the angle between $\mathbf{n}_f$ and the vector connecting neighbouring cell centroids exceeds 70°, a non-orthogonal correction

is applied [25].

A global measure of divergence is given by the volume-weighted root-mean-square:

$$\mathrm{RMS}_{\mathrm{div}} = \sqrt{\frac{\sum_c d_c^2 V_c}{\sum_c V_c}}. \tag{2.20}$$

Since for incompressible flow we expect $\nabla \cdot v = 0$, $\mathrm{RMS}_{\mathrm{div}}$ will also be referred to as the *divergence error*.

To compare across simulations and domain sizes, the *dimensionless* local divergence can be constructed:

$$\hat{d}_c = \frac{d_c L_{\mathrm{sim}}}{|U_{\mathrm{sim}}|}, \tag{2.21}$$

where L_{sim} and U_{sim} are characteristic length and velocity scales, e.g., the cylinder diameter and inlet velocity in canonical flow around a cylinder.

2.4 Scientific Machine Learning

Scientific Machine Learning integrates domain knowledge from physics-based models to develop Machine Learning (ML) methods for complex engineering problems in science and engineering. The goal is not solely to fit data, but to incorporate known physical laws and constraints into the learning process [32].

The objective of machine learning is generally to learn a mapping $f_\theta(x)$ from inputs x to outputs y. In the context of physical systems, the target output may be the solution to a governing partial differential equation (PDE). For example, consider a general time-dependent PDE of the form:

$$\mathcal{F}(u, \partial_t u, \partial_x u, \partial_{xx} u, \ldots, x, t) = 0, \quad x \in \Omega \subset \mathbb{R}^d, \ t \in [0, T]$$

subject to appropriate initial and/or boundary conditions. Here, $u(x,t)$ denotes the unknown solution field, $\mathcal{F}$ is a differential operator characterising the physical system and Ω is the spatial domain.

Traditionally, simulating the evolution of such systems involves discretising the spatial domain and applying numerical methods, such as Finite Volume Method (FVM), to march the solution forward in time. Instead of discretising the solution u directly, an alternative approach is to learn the parametric approximation of the time-stepping operator. Specifically, given the current state $u(x,t)$, we approximate the solution at the next

timestep as:

$$u(x, t + \Delta t) \approx f_\theta\big(x, u(x, t)\big)$$

where f_θ is a differentiable, parametric function – typically a neural network – that learns to emulate the time evolution of the system.

The parameters θ are obtained by minimising a loss function composed of:

- **Supervised loss** $\mathcal{L}_{\text{data}}$, based on data samples $(x_i, u_i^t, u_i^{t+\Delta t})$, enforcing agreement between the predicted and actual solutions at the next time step:

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(f_\theta(x_i, u_i^t) - u_i^{t+\Delta t} \right)^2 . \tag{2.22}$$

- **Physics-based loss** $\mathcal{L}_{\text{PDE}}$, which penalises violation of the governing PDE applied to the predicted state:

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left| \mathcal{F}\left(f_\theta(x_i, u_i^t), \partial_t f_\theta, \partial_x f_\theta, \dots \right) \right|^2 \tag{2.23}$$

The total loss is then given by:

$$\mathcal{L}(\theta) = \lambda_{\text{data}}\, \mathcal{L}_{\text{data}}(\theta) + \lambda_{\text{PDE}}\, \mathcal{L}_{\text{PDE}}(\theta)$$

where λ_{data} and λ_{PDE} balance data fidelity and physical constraint enforcement.

Minimisation of $\mathcal{L}$ is typically performed using gradient-based optimisation algorithms such as stochastic gradient descent (SGD) or Adam [33]. These methods require computing the gradient of the loss with respect to the model parameters, $\nabla_\theta \mathcal{L}$, a task typically performed using *automatic differentiation* (AD). AD computes gradients by decomposing model computations into a sequence of elementary operations and systematically applying the chain rule to these. It is available as a core feature in libraries such as TensorFlow and PyTorch [34, 35]. Optimisation proceeds iteratively by updating the parameters in the direction of the negative gradient, $-\eta \nabla_\theta \mathcal{L}$, where η is the *learning rate* that controls the step size.

The flexibility of AD allows f_θ to be any composition of differentiable operations. Neural networks have become a popular choice for f_θ in scientific machine learning due to their universal approximation capability and computational efficiency [36, 37]. The simplest type of neural network is a Multi-Layer Perceptron (MLP). Figure 2.2 illustrates

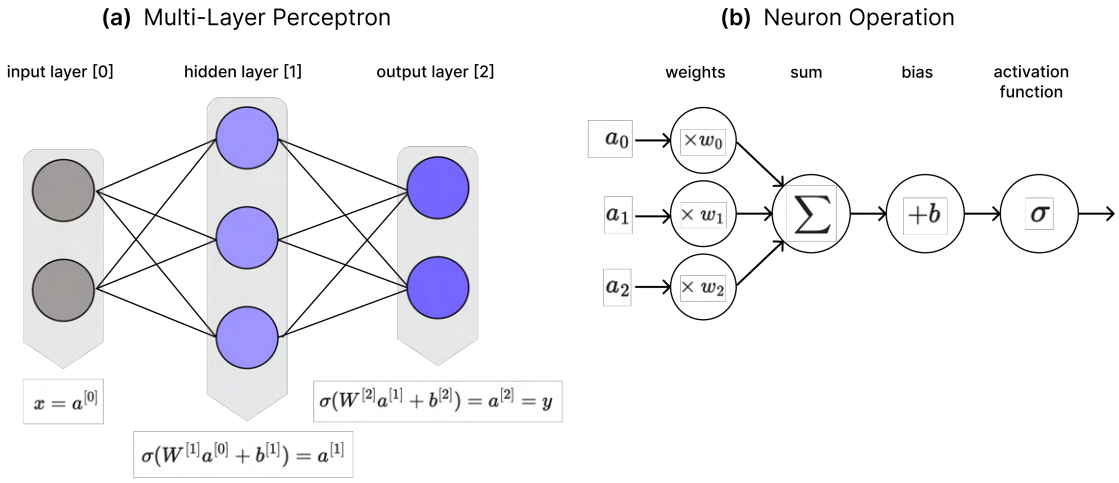


Figure 2.2: **(a)** Architecture of a three-layer multilayer perceptron (MLP) with input layer (0), one hidden layer (1), and output layer (2). Neural networks with more than one hidden layer are referred to as *deep* networks. The input vector $x = a^{(0)}$ is transformed through weight matrices $W^{(l)}$, bias vectors $b^{(l)}$ and activation function σ, yielding a new vector of "activations" $a^{(l)}$ at each layer. Circles represent individual *neurons* (components of $a^{(l)}$), each of which receives input from all activations in the previous layer $a^{(l-1)}$. The parameters W and b together constitute the model parameters θ. **(b)** Operation of a single neuron: each input activation a_i from the previous layer is multiplied by its own weight w_i, summed, shifted by a bias b, and passed through σ, producing the output activation $a = \sigma\big(\sum_i w_i\, a_i + b\big)$.

how the operation of a MLP can be represented using matrix operations applied to an input vector x. In practice, multiple input vectors are grouped together into a matrix X, known as a *mini-batch*, where each column of X corresponds to a single training example. This allows matrix-matrix operations that are highly parallelisable and well-suited to modern hardware, such as GPUs. Batching also improves optimisation stability by reducing gradient variance through averaging over mini-batches.

Factors affecting model accuracy and convergence – such as loss weighting, learning rate, and mini-batch size – are known as *hyperparameters*. Other notable hyperparameters include the number and width of hidden layers in the neural network (see Figure 2.2).

2.4.1 Graph Neural Networks

Real-world data is often relational or exists on irregular domains, and can be naturally represented as graphs (see Figure 2.3a). Mathematically, a graph $G(N, E)$ is defined by a set of nodes N and a set of edges E representing relationships between pairs of nodes. Unlike Cartesian grids with a fixed spatial adjacency, graphs can flexibly represent complex geometries and relationships, making them well-suited for representing unstructured meshes. Unstructured meshes are popular for CFD simulations due to their ability to

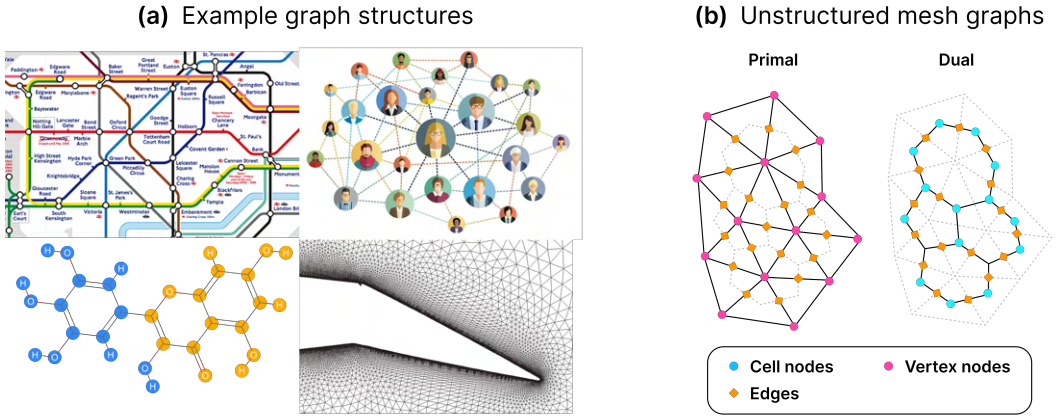


Figure 2.3: **(a)** Examples of real-life graph structure. Graphs can represent physical, spatial relationships such as molecules structure (bottom left) and meshes (bottom right) as well-as relational information such as the transport networks (top left) and social networks (top right). **(b)** The primal graph has nodes at vertices; the dual, related by a Voronoi transformation, has nodes at cell centroids connected across shared edges.

conform to complex geometries and allow flexible, localised refinement. Depending on whether the data is vertex- or cell-centred, these meshes admit two related graph representations, as illustrated in Figure 2.3b.

GNNs operate on graph-structured data by accepting feature vectors defined on nodes and, optionally, on edges. Each node encodes properties such as physical states or measurements, while edge features capture relationships between nodes. Graph connectivity — defined by the edges — guides how information flows through the network. This is achieved through *message passing*, where nodes update their features based on information from their neighbours (nodes directly connected by an edge). Formally, for a node j, the message passing typically involves [38]:

- **Compute messages** from each neighbouring node $i \in \text{Neighbourhood}(j)$ using a learnable function g_m:

$$m_{i \to j} = g_m(n_i, n_j, e_{ij}),$$

  where n_i and n_j are the feature vectors of nodes i and j, respectively, and e_{ij} denotes the feature vector of the edge between them.

- **Aggregate messages** received at node j using a permutation-invariant operation:

$$m_j = \bigoplus_{i \in N(j)} m_{i \to j}.$$

  Common choices for the aggregation operator $\bigoplus$ include element-wise summation, mean, or maximum, all of which are invariant to the ordering of neighbours.

- **Update node state** by applying a learnable function g_u to the current node features and the aggregated message:

$$n_j^{\text{new}} = g_u(n_j^{\text{old}}, m_j).$$

Note that learnable functions g_m and g_u are typically implemented as MLPs. Additionally, multiple (k) message-passing steps allows nodes to update their features based on information from its k-hop neighbourhood, capturing broader structural dependencies.

3. Literature Review

This chapter explores neural networks as surrogates for incompressible Navier-Stokes simulations, highlighting a research gap: the hard enforcement of the continuity equation in GNNs for transient simulations. Broader machine learning applications in CFD such as closure modelling, inverse problems, flow control, and reduced-order modelling are not addressed [39, 40, 41].

3.1 Deep Learning for CFD

Early applications of deep learning in CFD primarily focused on accelerating specific subcomponents of traditional numerical solvers, rather than replacing them entirely. For example, neural networks were employed to expedite iterative procedures such as the pressure Poisson solver in Chorin's projection method [42]. These hybrid approaches demonstrated that machine learning could be effectively integrated with established numerical frameworks, paving the way for more ambitious surrogate modeling strategies. Subsequently, the field progressed toward developing full surrogate models that directly map initial and boundary conditions to entire flow fields, encompassing both steady-state solutions [43] and unsteady, time-evolving dynamics through autoregressive architectures [11]. Notably, recent advances have seen a renewed interest in hybrid methodologies that incorporate insights from traditional methods into the architecture aiming to enhance stability [44, 20, 45].

Through the use of Convolutional Neural Networks (CNNs), Jin et al. demonstrated the critical role of model architecture in surrogate performance [46]. CNNs capture local flow features via convolutional filters, which aligns with the localised interactions expected in physical fluid systems. This introduces a physics-informed inductive bias into the model, which subsequent studies have shown leads to consistently more robust performance and generalisation across various flow scenarios [42, 47, 48]. However, a key limitation of CNNs remains their requirement for structured, grid-like data, necessitat-

ing interpolation of mesh-based inputs onto uniform grids – a step that can introduce numerical errors and increase preprocessing time.

The introduction of physics-based regularization in the loss function by Raissi et al. has enabled surrogate models to be constrained by the governing equations and conservation laws [14]. By embedding physical priors into training, Physics-Informed Neural Networks (PINNs) can learn effectively from limited data and have shown strong generalisation to unseen conditions [49, 50]. PINNs mark a major advancement in surrogate modelling by learning the underlying physics, rather than merely fitting the empirical distribution of training data. However, physics-informed loss terms can only encourage approximate conservation – they do not ensure strict conservation as achieved by classical FVM solvers.

3.2 Graph Neural Networks for Unstructured Meshes

Unlike CNNs, GNNs naturally handle arbitrary domain sizes and unstructured meshes, making them well-suited for industrial CFD applications. GNNs generalize convolution operations to graph structures by passing messages along edges and updating node states based on local neighbourhoods [38, 51]. Building on this concept, Battaglia et al. introduced the Encoder–Processor–Decoder (EPD) architecture [52], where node features are first encoded into a latent space, leveraging *representation learning* to capture complex physical phenomena in a higher-dimensional feature space [53]. These latent features are iteratively refined in the latent space through message passing in the processor blocks, before being decoded back into physical space.

The EPD architecture has been effectively applied in CFD surrogate models, notably MeshGraphNets, which performed well on steady and unsteady cylinder flows using unstructured meshes [11]. Li et al. iterated on the MeshGraphNets design to build Finite-Volume-Graph-Network (FVGN), which uses finite-volume-inspired face-centre predictions to update cell-centred velocities and employs a novel message passing routine, leveraging both the dual and primal graph connectivity [20]. FVGN demonstrated superior rollout stability and generalisation. Unlike MeshGraphNet, which lacks any enforcement of physics constraints, FVGN weakly incorporates them, but neither guarantees strict conservation.

3.3 Physics-Constrained Neural Architectures

Incorporating conservation laws directly into model architectures remains an active research area. For incompressible flows, the key constraint is a divergence-free velocity field, ensuring exact mass conservation.

Several methods have been developed to enforce hard divergence-free constraints in neural networks. Rao et al. (2020) and Wandel et al. (2020) construct divergence-free velocity fields by taking the curl of scalar potentials, satisfying the constraint by design [18, 12]. This can be seen as a simplification of the differential forms framework used by Richter-Powell et al. (2022), who apply a mesh-free method based on neural operators and exterior calculus [17]. These approaches outperform conventional weak penalties in both accuracy and constraint enforcement. However, they have been applied only to structured grids or mesh-free settings, and are yet to graph neural networks (GNNs).

More recently, Horie and Mitsume proposed FluxGNN, which enforces conservation during message passing using a FVM-inspired approach, rather than projecting outputs onto a divergence-free manifold post-decoding [54]. While effective on simple cases, FluxGNN has not yet been scaled to the complexity of models like MeshGraphNets or FVGN. This offers a promising avenue for advancing GNN architectures by incorporating strict divergence-free constraints to guarantee exact conservation.

4. Methodology

This chapter details the model architecture, dataset generation, preprocessing pipeline and training setup used to develop a GNN surrogate model. In this first report, the model architecture is based on Finite-Volume-Graph-Network (FVGN), which combines data-driven learning with a FVM-inspired integration step [20]. The FVGN architecture only weakly enforces a divergence-free velocity field; the second report will present modifications to ensure strong conservation. Models are implemented using PyTorch [34] and PyG (providing graph-specific features) [55].

4.1 Model Architecture

The model uses both the primal and dual graph representations of an unstructured mesh (see Figure 2.3B). Although these graphs are undirected – typically implemented with bidirectional edges – only one directed edge per connected node pair is stored. This reduces memory usage and computational cost by avoiding redundancy in the adjacency structure. To preserve permutation invariance, half of the directed edges are randomly flipped at each training epoch. This ensures the model remains insensitive to edge orientation and effectively treats the graph as undirected. Boundary faces in the dual graph are represented as self-loops on their corresponding owner cells (see Figure 4.1A).

Since FVM solution data is defined at cell centres, input features are defined at the nodes of the dual graph:

$$\mathbf{c}_i = \text{concat}(\mathbf{u}_{c,i},\, \mathbf{t}_{c,i})^\top \tag{4.1}$$

where $\mathbf{u}_{c,i}$ are the cell-centre velocities and $\mathbf{t}_{c,i}$ is a one-hot vector indicating the type of cell (e.g. normal, inlet, outlet, boundary).[1]

[1] If the cell types normal, inlet, outlet and boundary are labelled with integers 1–4, respectively, the one-hot vector is then of length four, with a 1 in the position of the type and 0 elsewhere. For example, an outlet type is represented as $\mathbf{t}_{c,i} = (0,\, 0,\, 1,\, 0)^\top$.

The input edge features – shared between the primal and dual graphs – are chosen to be:

$$\mathbf{e}_{i\to j} = \text{concat}(\mathbf{u}_{c,j} - \mathbf{u}_{c,i},\, \mathbf{x}_{v,j} - \mathbf{x}_{v,i},\, \|\mathbf{x}_{v,j} - \mathbf{x}_{v,i}\|,\, \mathbf{t}_{i\to j})^\top, \tag{4.2}$$

where $\mathbf{x}_v$ are the adjacent vertex positions (i.e. nodes of the primal graph), and $\mathbf{t}_{i\to j}$ is the one-hot vector indicating the face type. No feature vectors are defined at the nodes of the primal graph.

Following the Encode-Process-Decode (EPD) approach, input node and edge features are embedded into an n-dimensional space (where n is a tunable hyperparameter) using two learnable *encoders* – one for nodes and one for edges. All learnable functions in FVGN are implemented as a MLPs with two hidden layers and SiLU activation functions.

Processor blocks perform message passing to update the embeddings with information from the neighbouring nodes and edges. The FVGN approach to message passing, shown in Figure 4.1B-D, is as follows:

1. Update cell-centre features using message passing on the primal graph.
   1.1. Use edge feature embeddings on the primal graph as messages.
   1.2. Aggregate messages at vertex nodes via summation and then transfer messages to dual graph by vertex aggregations at the corresponding cell centres.
   1.3. Update cell-centre (dual node) embeddings with the messages using the node processor MLP.[2] These updated cell centre features serve as messages for the next step.
2. Aggregate messages at each edge by summing the features of its two adjacent cell-centre nodes.
3. Update edge feature embeddings with the messages using the edge processor MLP.

The processor block is applied m times to extend the receptive field. The weights of the node and edge processor MLPs are shared across all iterations.

Decoder MLPs map the updated node and edge embeddings to physical quantities. Instead of directly predicting cell-centre velocities, FVGN decodes edge features into three face-centred quantities: face velocity $\mathbf{u}_f$, face pressure p_f, and compensation flux $\mathbf{Q}_f = \nu(\nabla \mathbf{u_f}) \cdot \mathbf{n_f} A_f$. These are used in a FVM-inspired integration step, the *spatial*

[2]The node and edge processor MLPs include residual connections: the output is added to the original embedding, improving gradient flow and training stability.

integration layer, that applies the discretised integral form of the momentum equation (Equation 2.12) to predict the cell-centre velocity update:

$$\mathbf{u}^{t+\Delta t} - \mathbf{u}^t = \frac{\Delta t}{V}\left[-\sum_{f\in\text{cell}} A_f(\mathbf{u}_f \otimes \mathbf{u}_f)\mathbf{n}_f - \frac{1}{\rho}\sum_{f\in\text{cell}} p_f A_f \mathbf{n}_f + \sum_{f\in\text{cell}} \mathbf{Q}_f\right] \tag{4.3}$$

where A_f is the scalar face area, $\mathbf{n}_f$ is the outward-pointing unit normal at face f, and V is the cell volume. The compensation flux $\mathbf{Q}_f$ is predicted directly as a face-centred vector quantity, bypassing the need to compute $(\nabla\mathbf{u})_f$ explicitly on the unstructured grid, which would necessitate costly weighted least squares regression. Since all face-centred values on the right-hand side are predicted at t^{n+1}, the formulation is implicit.

Several aspects of this model architectural resemble traditional finite volume methods (FVM). As discussed, the construction of face-centred fluctuations at time t^{n+1} resembles an implicit FVM scheme. However, Equation 4.3 permits time stepping without assembling a coupled matrix system, akin to explicit FVM formulations. Consequently, this method blends the stability advantages of implicit integration with the computational simplicity of explicit schemes. Furthermore, the number of message passing steps plays a role similar to the numerical stencil width in classical solvers—at increased computational cost, information from a broader neighbourhood can be incorporated into the update of each cell.

An overview of the FVGN architecture is shown in Figure 4.1.

4.2 Dataset Generation

The CylinderFlow dataset, introduced by Pfaff et al. and derived from COMSOL simulations, is a widely used benchmark in the GNN literature [11, 56, 20, 57]. It is valuable for comparing model performance across studies. The dataset includes both steady and unsteady flow past a cylinder at Reynolds numbers ranging from 10 to 250. Across training examples, variations are introduced in the position within the pipe, the radius of the cylinder, and the inflow velocity. However, this dataset has certain limitations for the present study. The FVGN architecture requires cell-centred inputs, whereas the data in CylinderFlow is stored at mesh vertices. This discrepancy necessitates interpolation to face and cell centres, which can introduce error. Additionally, the conservative fluxes used in the FVM simulations are not stored, which may limit the dataset's utility for analysing exact divergence-free properties.

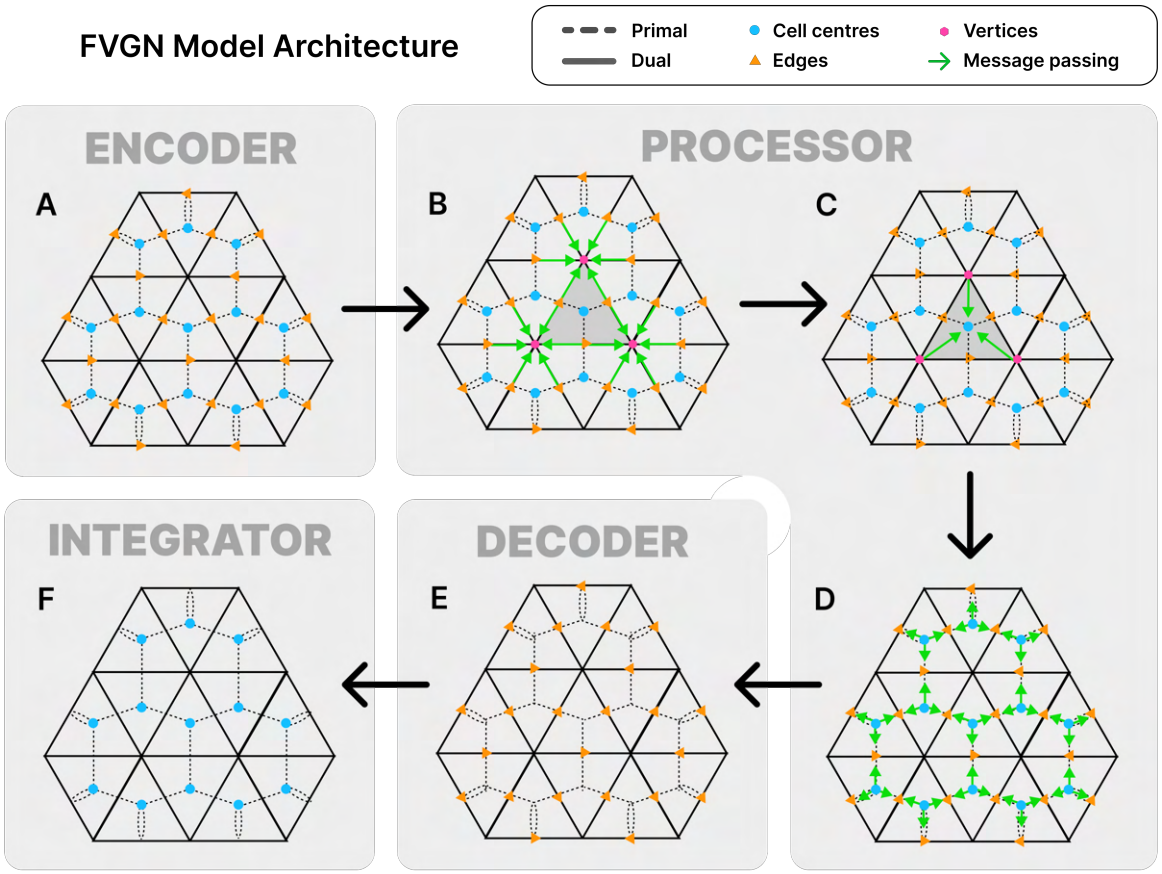
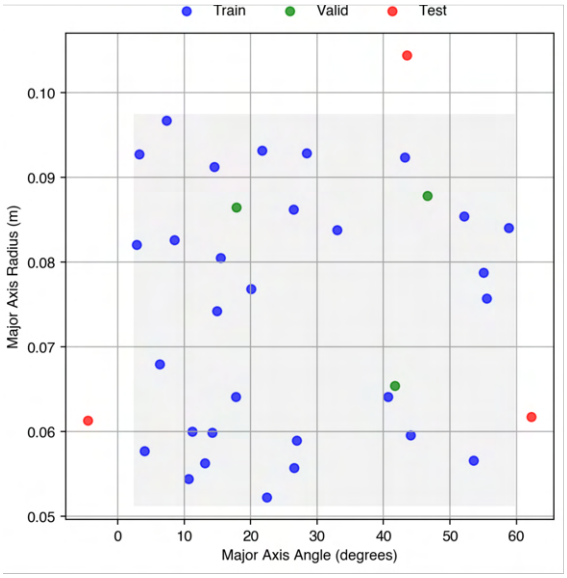


Figure 4.1: Overview of the FVGN model architecture [20]. **(A)** Encodes node and edge features into embedding space. **(B)** Aggregate (summation) edges features at vertices. **(C)** Average vertex aggregations to corresponding cell centres and update cell-centre embeddings with "node processor" MLP. **(D)** Aggregate (summation) cell embeddings at edges and update edge embeddings with "edge processor" MLP. **(E)** Decode edge embeddings into face-centred physical properties $(\mathbf{u}_f, \mathbf{p}_f, Q_f)^T$. **(F)** Integrate face fluxes for each cell using discretized momentum equation to predict velocity change $\Delta\mathbf{u}_c$ at cell centres.

To address these issues, a custom dataset was generated using OpenFOAM [15]. The simulation domain consists of flow through a pipe past a parametrized obstacle, with potential for variation in shape, size, boundary conditions and mesh refinement. Mesh quality is assessed using OpenFOAM's native `checkMesh` tool. Ensuring high-quality meshes – characterised by low non-orthogonality and minimal skewness – is important for the accuracy of Green-Gauss gradient calculations (see Section 2.3.2). Simulations are carried out using the segregated PIMPLE solver, which combines the inner PISO and outer SIMPLE loops for enhanced stability in transient flows. A uniform timestep, corresponding to the minimum stable CFL condition across all cases, is applied. For turbulent scenarios (e.g. Figure 4.2b), LES with a Smagorinsky model is employed [28] and wall functions approximate near-wall turbulence, enabling reduced mesh refinement in the boundary layer [58]. Although the results are presented in 2D, turbulent simulations

(a) Geometric distribution in LaminarEllipse



(b) Example of generated turbulent simulation

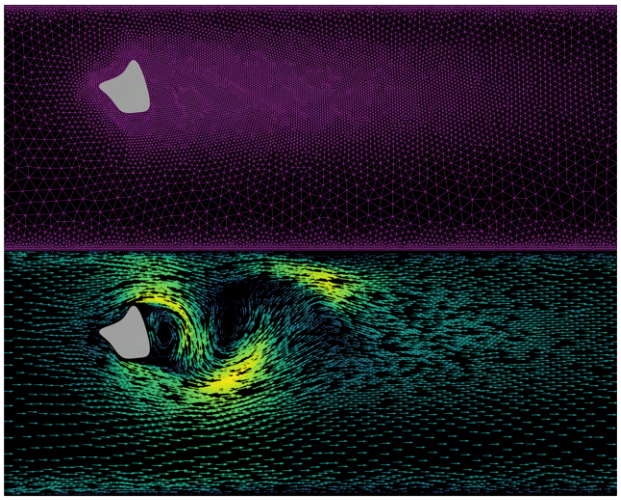


Figure 4.2: **(a)** Distribution of geometric properties of training samples in LaminarEllipse. **(b)** Example of the mesh (top) and solution (bottom) of a turbulent simulation (Re $\sim$ 3000). Note that the wake field is automatically scaled and positioned based on the obstacle geometry.

are conducted in 3D to capture vortex stretching effects essential for physically meaningful solutions.

To demonstrate the utility of this data generation pipeline, a simple dataset of laminar flow past elliptical obstacles was generated, which is referred to as *LaminarEllipse*. Using a fixed position and constant aspect ratio, variation in the system geometry was created by rotating and rescaling the ellipse. The distribution in geometric properties across 36 simulations (30:3:3 for training:validation:testing) are shown in Figure 4.2a. Test examples are chosen from beyond the distribution of training and validation examples so that the model's ability to generalise to unseen geometries can be measured.

To efficiently generate large datasets, a hierarchical parallelisation strategy is employed. At the top level, SLURM job arrays are used to distribute independent simulations across CPU nodes. Within each node, OpenFOAM's native MPI-based domain decomposition enables intra-node parallelism by partitioning the computational domain across CPU cores.

4.3 Preprocessing

Geometric properties and interpolated values required for input features, targets (see Section 4.4), and the integration layer are precomputed and stored before training. This preprocessing speeds up training by avoiding redundant calculations during each epoch. The data is stored in an HDF5 file, which records the static geometry only once for a given simulation. Unlike legacy VTK data formats that duplicate the geometry at every simulation timestep, this approach avoids redundancy and significantly reduces disk space usage.

In addition, dataset-wide statistics are computed to normalise input features to zero mean and unit variance. This normalisation is standard practice and essential for model generalisation, preventing overfitting to the raw magnitudes of input features [59].

An overview of how the preprocessing steps fit into data generation and training pipeline is shown in Figure 4.3.

4.4 Training

Each training *epoch* consists of showing the model all available training examples, where each example is drawn from a randomly selected timestep and geometry. The total number of training examples is equal to the number of timesteps in each simulation multiplied by the total number of simulations.

An individual training example consists of the input graph with node and feature vectors at time t^n and targets for three predicted quantities: the cell-centred velocity change $\hat{\Delta u}_c = \hat{u}_c^{t+\Delta t} - \hat{u}_c^t$, the face velocity $\hat{u}_f^{t+1}$ and the face pressure $\hat{p}_f^{t+1}$. Target values for face-centred values are obtained by interpolating the simulation data.

The total loss is composed of multiple mean squared error (MSE) terms, each targeting a specific physical quantity or constraint:

$$\mathcal{L}_u = \frac{1}{N_f} \sum_{f \in \text{graph}} \|\hat{u}_f - u_f\|_2^2 \qquad \text{(face velocity loss)} \tag{4.4}$$

$$\mathcal{L}_p = \frac{1}{N_f} \sum_{f \in \text{graph}} \|\hat{p}_f - p_f\|_2^2 \qquad \text{(face pressure loss)} \tag{4.5}$$

$$\mathcal{L}_{\text{mom}} = \frac{1}{N_c} \sum_{c \in \text{graph}} \left\| \hat{\Delta u}_c - \Delta u_c \right\|_2^2 \qquad \text{(cell acceleration loss)} \tag{4.6}$$

$$\mathcal{L}_{\text{cont}} = \frac{1}{N_c} \sum_{c \in \text{graph}} \left\| \sum_{\text{cell faces}} (u_f \cdot n_f A_f) \right\|_2^2 \qquad \text{(continuity loss)} \tag{4.7}$$

The total loss is defined as:

$$\mathcal{L}_{\text{total}} = \lambda_u \mathcal{L}_u + \lambda_p \mathcal{L}_p + \lambda_{\text{mom}} \mathcal{L}_{\text{mom}} + \lambda_{\text{cont}} \mathcal{L}_{\text{cont}} \tag{4.8}$$

where $\lambda_u, \lambda_p, \lambda_{\text{mom}}, \lambda_{\text{cont}}$ are weighting coefficients controlling the contribution of each loss term. N_f and N_c are the number of faces and cells in the graph, respectively, and $\mathbf{n}_f$, A_f denote the face normals and areas.

Although $\mathcal{L}_{\text{mom}}$ appears as a supervised loss term, it also serves as a physics-inspired penalty for momentum conservation due to the integration layer (see Equation 4.3). Similarly, $\mathcal{L}_{\text{cont}}$ encourages adherence to the continuity equation. However, since these constraints are only weakly enforced (the optimiser is not expected to drive $\mathcal{L}_{\text{mom}}$ and $\mathcal{L}_{\text{cont}}$ exactly to zero), strict conservation is not guaranteed.

4.4.1 Hyperparameter Selection

Hyperparameters were primarily chosen based on those used to train Finite-Volume-Graph-Network (FVGN) in the original study [20], given the identical model architecture and similar datasets. The cost function weights $\lambda_u = \lambda_p = \lambda_{\text{cont}} = 1$ and $\lambda_{\text{mom}} = 10$ were retained, as they yielded optimal performance in that work. Other choices include $n = 128$ for embedding space dimensionality and $m = 15$ message passing blocks, which defines the spatial range of local information propagation, analogous to a numerical stencil. These choices may be revisited in Report 2 when modifications are made to enforce a hard divergence-free constraint.

Other hyperparameters such as mini-batch size are hardware dependent. Large mini-batches generally improve gradient estimates and training speed by incorporating more

data per training step. In this study, training was performed on Nvidia A100 GPUs – this allowed for a maximum mini-batch size of 15 for the CylinderFlow dataset.

The AdamW optimiser [60] was used, as in the original FVGN work, but with a OneCycle learning rate schedule [61] instead of the original step/exponential decay hybrid – we observed faster convergence with OneCycle. This schedule warms up to a peak learning rate of 10^{-3} over the first $\sim 15\%$ of training steps, followed by slow cosine decay to 10^{-6}. The warm-up stabilises early training, while the gradual decay maintains higher learning rates longer than exponential decays, leading to faster convergence.

4.4.2 Performance Considerations

Two components of the training pipeline were parallelised to reduce overall training time:

- **Data loading.** This CPU task reads simulation data from disk and constructs graph inputs for the model, which are then transferred to the GPU. PyTorch supports multi-core data loading via multiple CPU worker processes. A custom dataloader was implemented to assign each worker a subset of geometries, allowing efficient caching of geometry data and minimising reads from disk when loading new timesteps.
- **Gradient computation.** Gradients are computed over mini-batches, but batch size is limited by single-GPU memory. To scale training, multiple GPUs process mini-batches in parallel, and their gradients are averaged, yielding an effective batch size of (number of GPUs) $\times$ (batch size per GPU). This is implemented using PyTorch's MPI-enabled distributed training library, which provides tools for synchronisation of model parameters and mini-batch distribution across GPUs.

4.5 Rollout

Starting from a graph encoding the initial conditions and geometry at time t^0, the model simulates dynamics autoregressively. At each step t^n, it takes the current graph as input and predicts cell-centered velocity changes at t_f^{n+1}, which are used to update the graph state. This updated graph is then passed forward to the next step. The process, known as a *rollout*, is repeated for a fixed number of timesteps.

A central challenge in rollouts is the compounding nature of prediction errors: since the model's outputs are used to update the input graph at each step, any inaccuracies in prediction become part of the next input. As rollouts progress, these errors can accumulate,

with degraded inputs leading to increasingly inaccurate outputs. To improve robustness, zero-mean Gaussian noise (standard deviation 2×10^{-2}) is added to the normalised velocity inputs during training. This helps the model learn to make stable predictions even when the inputs deviate from the true solution [10].

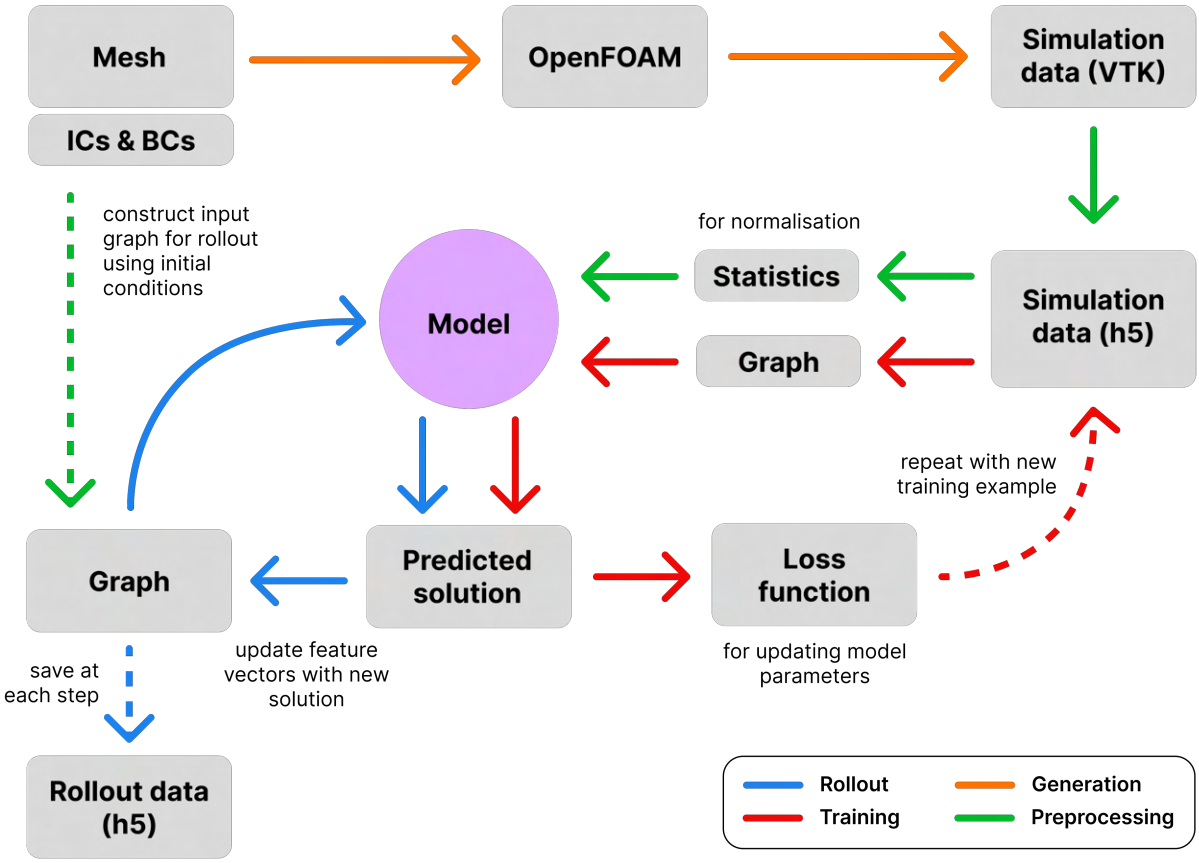


Figure 4.3: An overview of the pipeline used to generate data, train and rollout results.
(Generation) Geometry is defined as 2D `gmsh` data with initial/boundary conditions in a text file. Meshes are extruded to 3D and converted to OpenFOAM PolyData. Conditions are parsed into control dictionaries. Simulation results are converted to VTK and reduced to 2D.
(Preprocessing) VTK data is used to compute all properties for graph input construction, stored in HDF5 format. Dataset statistics are accumulated and passed to the model for feature normalization.
(Training) The dataloader reads `h5df` simulation data, constructs graphs, transfers them to GPU, and passes them through the model. Outputs inform the cost function, and backpropagation updates model parameters across mini-batches.
(Rollout) Mesh and initial conditions generate the first graph input. The model predicts the next state, updates the graph, and re-feeds it, progressing autoregressively through the simulation.

5. Initial Results

Analysis of datasets properties and rollouts performance using the FVGN model architecture are presented in this chapter. These initial results will help guide future development of the model architecture and training strategy.

5.1 Divergence of Dataset Velocity Fields

Figure 5.1 shows local velocity divergence fields at the final timestep of simulations from the CylinderFlow (A) and LaminarEllipse (B–C) datasets. In A and B, face-centred fluxes ϕ_f are computed by interpolating cell-centred velocities to face centres, projecting onto face normals, and scaling by face area. In contrast, C uses F_f directly from OpenFOAM's finite-volume solution, which guarantees local conservation, enforcing $\sum_f F_f = 0$ to machine precision. Although B and C are based on the same simulation, the divergence fields differ markedly due to the non-conservative nature of the interpolated fluxes in B. Divergence errors in A and B are substantial (41.2 and 32.0), while for C it effectively vanishes (1.06×10^{-9}).

This discrepancy highlights a key limitation of the current FVGN-based architecture: it is trained on interpolated face velocities, such as those used in schemes A and B. These interpolated values are subject to error, particularly on unstructured meshes where face centres are not aligned with the vectors connecting adjacent cell centroids. Even higher-order interpolation schemes with non-orthogonal corrections cannot guarantee local conservation. As a result, the model learns to reproduce fluxes that may not satisfy discrete mass conservation, leading to divergence errors in its predictions. In Report 2, we address this limitation by directly incorporating the conservative FVM fluxes (as used in scheme C) into the training process. This is made possible by our ability to generate custom datasets with direct access to the solver-computed fluxes, ensuring that the model learns from physically consistent, locally conservative targets.

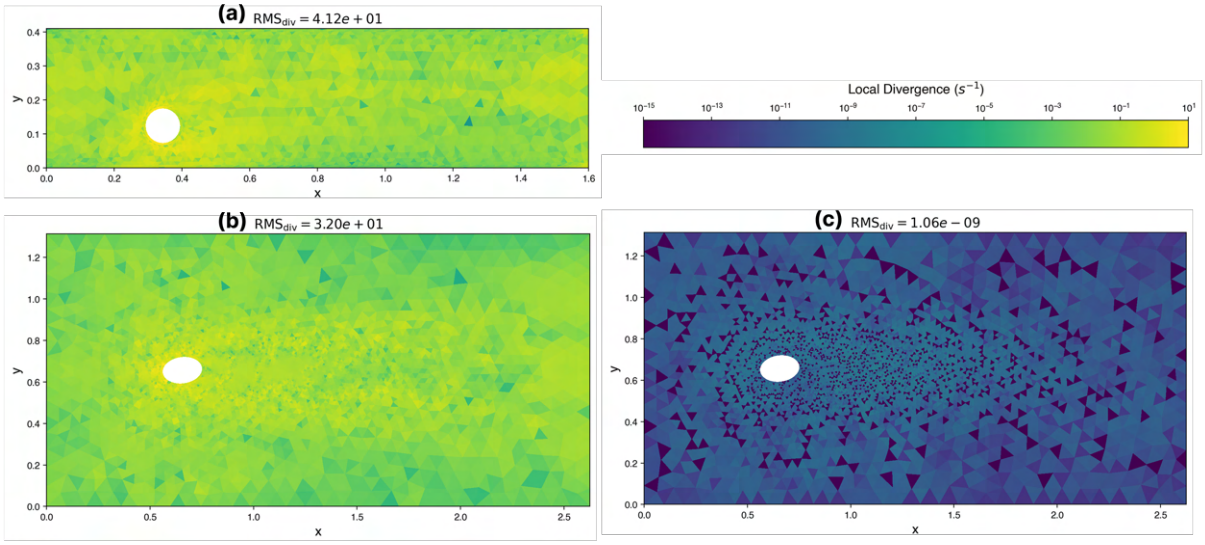


Figure 5.1: Colour maps of local divergence $d_c = \frac{1}{V_c} \sum_{f \in c} F_f$ at the final timestep of the simulations. **(a)** and **(b)** (from CylinderFlow and LaminarEllipse, respectively) use fluxes computed from face-centred velocities interpolated from cell centres. **(c)** shows the same LaminarEllipse simulation as (b), but computes divergence using conservative fluxes from the OpenFOAM solver. Divergence is shown in physical units, as the simulations have comparable inlet velocity and characteristic scale.

5.2 Rollouts with CylinderFlow

Training was performed over 23 epochs using the first 300 timesteps from 1,000 CylinderFlow simulations, yielding 3×10^5 unique training examples. The examples span a range of cylinder sizes, positions and inlet speeds. Since the dataset includes 600 timesteps per simulation, the model's temporal generalisation can be evaluated by performing extended rollouts up to timestep 600 — significantly beyond the 300 timesteps seen during training.

5.2.1 Performance

Training time was approximately 16 hours using four NVIDIA A100 GPUs. In comparison, Li et al. reported a 78-hour training time for the original FVGN implementation on a single NVIDIA V100. Even accounting for a 60–100% speedup when moving from a V100 to an A100, this represents a substantial reduction in training time, highlighting the efficiency of the multi-GPU implementation.

A 600-timestep rollout on a single A100 GPU averaged 5.22 ± 0.03 seconds. For reference, the numerical simulations used to generate CylinderFlow data required approximately 500 seconds on a single CPU. This corresponds to a nearly 100× speedup, demonstrating the model's potential for fast surrogate simulation.

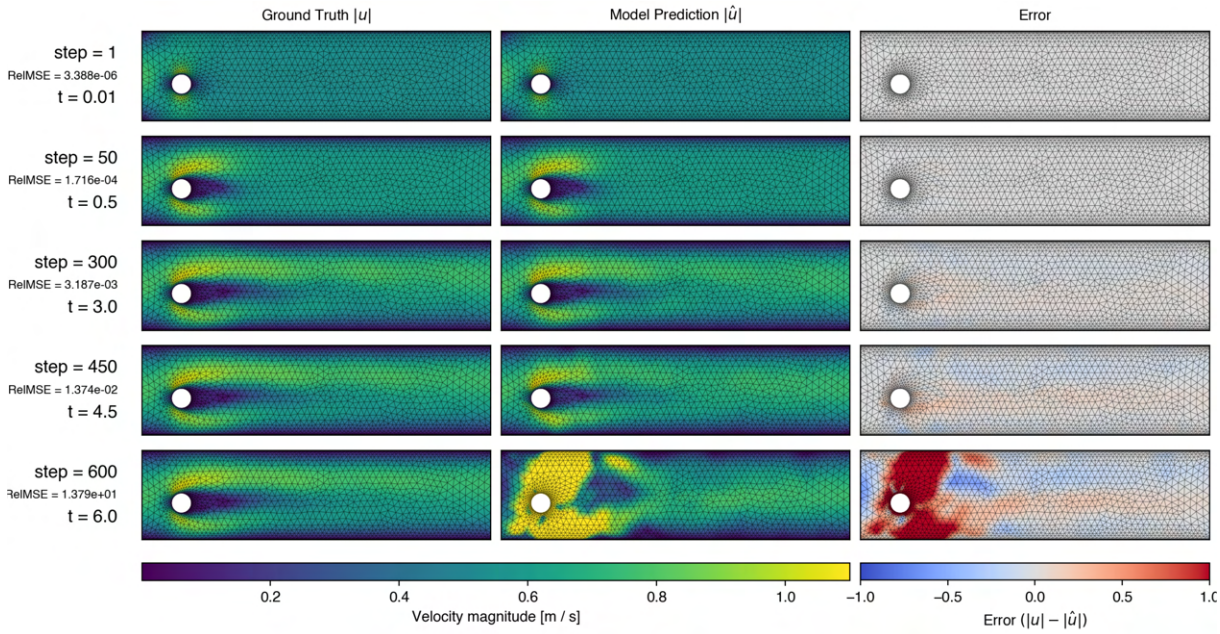


Figure 5.2: Example rollout of 600 timesteps for steady cylinder flow (Re $\sim$ 40), using a test geometry unseen during training. Ground truth (left), model prediction (middle), and error residual (right) are shown. The simulation remains stable until around timestep 470, after which it diverges.

5.2.2 Steady Simulations

Figure 5.2 shows an example rollout on an unseen test geometry. This system has Re ~ 40 and is expected to reach steady-state by $t = 3$. So that comparison can be made to existing literature results, the "Relative Mean Square Error" from [11] is chosen as a global measure of predictive error at each timestep. For velocity, this is:

$$\text{RelMSE} = \frac{\sum(u_x - \hat{u}_x)^2 + (u_y - \hat{u}_y)^2}{\sum(u_x^2 + u_y^2)}. \tag{5.1}$$

For scalar fields, such as pressure, only one component needs to be considered.

Rollouts begin with low error, achieving an average relative MSE of $2.7 \pm 0.3 \times 10^{-6}$ for velocity predictions at the first timestep. This is significantly lower than the single-step error reported for MeshGraphNets ($2.3 \pm 0.1 \times 10^{-3}$) [11], highlighting the effectiveness of the physics-informed FVGN architecture in capturing a step of the Navier–Stokes equations. However, error accumulates over time, and most simulations become unstable between timesteps 400 and 500. By the final timestep, the solution is typically unphysical, as shown in Figure 5.2. In contrast, the original FVGN paper – despite using the same architecture and training subset – reports stable rollouts to 600 timesteps [20]. This suggests that stability is sensitive to minor implementation differences – such as data

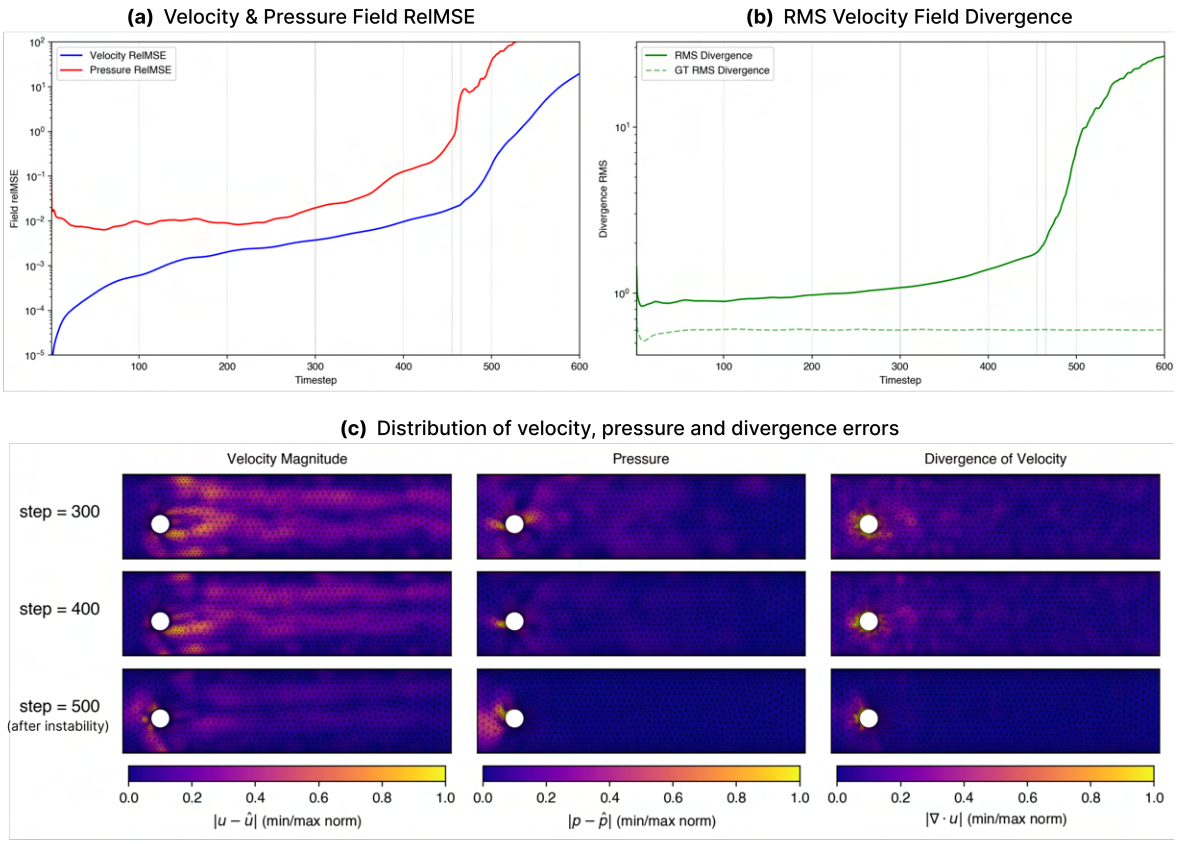


Figure 5.3: **(a)** RelMSE errors during the rollout for pressure (red) and velocity (blue). Note that pressure error starts significantly higher and begin to rise rapidly first. **(b)** Evolution of RMS_{div} for the model prediction (solid line) relative to the dataset (dashed line). Note that pressure and divergence errors increase rapidly as the same time. Velocity follows soon after but there is likely a delay of a few timestep since it is being updated with a residual prediction (Δu). **(c)** Spatial maps of the absolute errors per cell for velocity, pressure and divergence normalised to min-max for each field at each timestep. High divergence errors consistently near front-region of cylinder – it appears that pressure errors then concentrates here too. After pressure predictions diverge, the velocity errors are largest in this region too.

preprocessing, training strategy, or hyperparameter choices – and indicates that stronger performance is achievable without altering the underlying FVGN architecture. Nevertheless, the following analysis focuses on the limitations of the current implementation and motivates the decision to pursue architectural modifications as a means of addressing them.

A closer examination of how errors in velocity, pressure, and divergence accumulate in Figure 5.3 reveals key mechanisms that lead to instability. An initial inflection point occurs around timestep 300, marking the model's extrapolation beyond its training regime, where error growth begins to accelerate. A second, more critical point arises near timestep 460, where the rollout becomes unstable and errors escalate sharply. This loss of stability appears to be triggered by a sudden increase in pressure error. Note that these trends

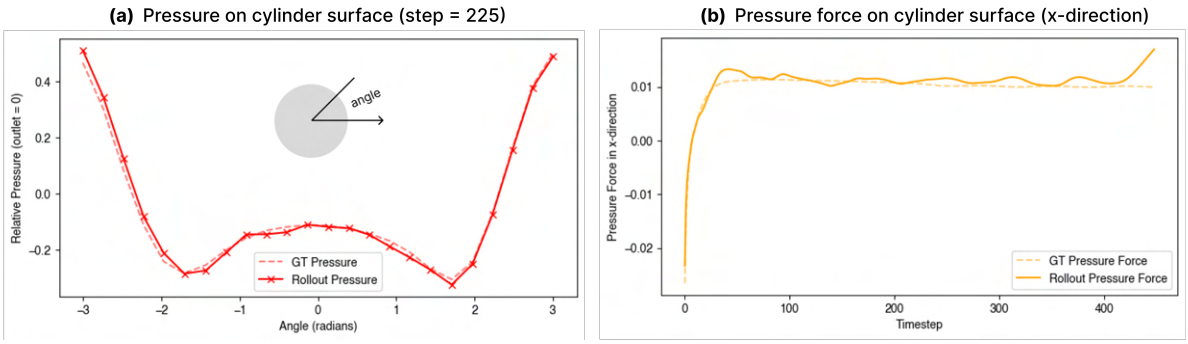


Figure 5.4: **(a)** Pressure on faces on the cylinder surfaces at $t = 225$ with respect to the angle the radius (in radians) to that face makes with the positive x-direction. **(b)** The x-direction pressure force $F_p = \sum_{f \in \text{cylinder surface}} n_f p_f$ integrated on the cylinder surfaces. Small oscillations are visible, but generally a good prediction until the simulation diverges.

were observed across the test cases. Although pressure is not included among the model's inputs, it plays a role in the integration step (Eq. 4.3); once pressure predictions become unphysical, significant velocity errors follow, ultimately destabilising the rollout.

The spatial distribution of errors, shown in Figure 5.3c, indicates that pressure error is largest near the stagnation point—an expected outcome, as steep pressure gradients in this region make the model particularly sensitive to perturbations. Notably, the divergence error is also consistently largest in this area. This suggests that divergence error may be a primary driver behind the breakdown of physically meaningful pressure predictions. A model that is conservatively formulated and strictly divergence-free may help prevent this degradation.

Beyond enforcing a divergence-free constraint, improving pressure prediction accuracy would further slow the accumulation of errors. At the final training epoch, the pressure loss (see Equation 4.8) is 1.3×10^{-1}, which is significantly higher than those for cell-centre velocity (3.0×10^{-4}) and face-centre velocity (2.4×10^{-3}), indicating that pressure is not being optimised as effectively. Increasing its weighting coefficient, β, in the loss function is a straightforward adjustment worth considering in Report 2.

Regarding divergence and velocity errors, their relationship appears weak. For example, in Figures 5.3a-b, between timesteps 0 and 300, velocity error increases approximately linearly, while divergence error grows exponentially. Furthermore, Figure 5.3c shows no clear spatial correlation between velocity and divergence errors before the onset of instability. This suggests that reducing velocity error alone is unlikely to improve divergence error, reinforcing the importance of explicitly enforcing the divergence-free constraint.

Despite instabilities during extended rollouts, the simulations yield useful estimates

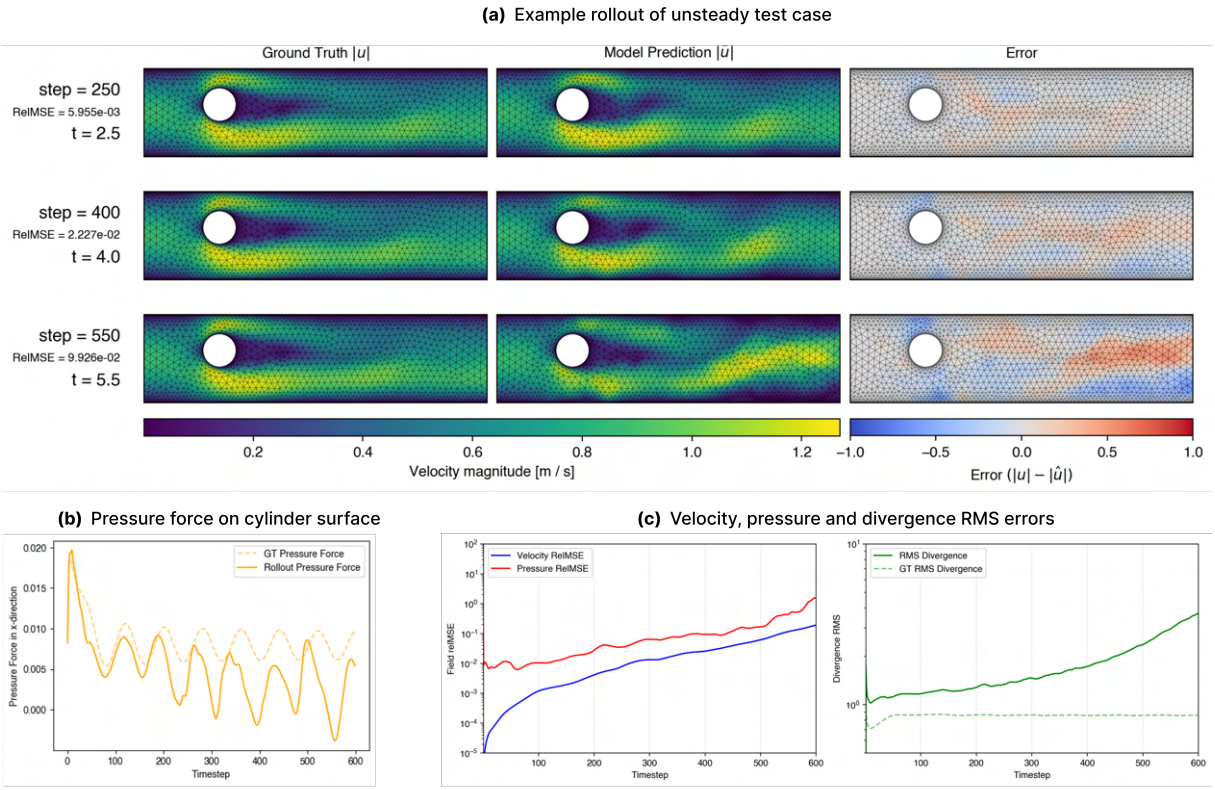


Figure 5.5: Rollout of an example unsteady test case with Re $\sim$ 125. **(a)** Snapshots of predicted and ground truth velocity fields and the difference between them (error) **(b)** Time series of pressure force on the cylinder surface. The unstable wake field causes oscillation which grow out of phase. **(c)** Evolution of velocity, pressure and divergence error during the rollout.

of the steady-state solution, typically reached by timesteps 200 to 300. For instance, Figure 5.4a shows surface pressures at $t = 225$ with an average relative error of $9 \pm 2\%$. Figure 5.4b presents the x-direction pressure force, which remains accurate until the onset of instability. While stability is poor for extended rollouts, these simulations still provide approximate steady-state metrics at significantly lower computational cost than traditional solvers.

5.2.3 Unsteady Simulations

In unsteady test cases, instability can arise earlier – sometimes before timestep 300 at high *Re*. Unlike steady cases, this is often caused by unphysical oscillations in the time-dependent velocity field rather than pressure errors. Figure 5.5a shows a mild unsteady case (Re $\sim$ 125) where the simulation doesn't diverge, providing an insight into the model's handling of transient flows. The pressure force (Figure 5.5b) captures wake oscillations, but the prediction gradually drifts out of phase and diverges in magnitude.

A similar trend is also observed for high Re cases, although these typically become highly unphysical after two to three oscillations.

Figure 5.5c shows a steady rise in divergence error, indicating that the growing velocity error is not merely due to phase drift – the predicted oscillations are increasingly unphysical. This suggests that physical violations remain a significant source of error and may be contributing to its accumulation, indicating that stronger constraints could help mitigate this effect.

However, ensuring physical consistency alone may not suffice for accurate transient rollouts. Capturing features like oscillation frequency requires further modifications to the training process to better model temporal dynamics. Potential approaches include strategies proposed by Brandstetter et al. [62]: the "pushforward" trick, which perturbs inputs using the model's own predictive distribution, and temporal bundling, where the model predicts multiple future steps to encourage temporal consistency. Additionally, curriculum learning could be effective, training the model on short rollouts initially and gradually increasing sequence length to improve temporal stability and long-horizon performance.

5.3 Rollouts with LaminarEllipse

After training for 100 epochs on the 30 LaminarEllipse training examples, 200-timestep rollouts were performed on the validation geometries. The full 200 timesteps were included in training to avoid instability observed in extended rollouts and to isolate the effect of mesh geometry on prediction error.

5.3.1 Dependence on Mesh Resolution

Unlike CylinderFlow, LaminarEllipses exhibits variation in maximum mesh refinement levels, as the smallest cell sizes scale with the size of the ellipse. This revealed that the model performs worse on geometries with finer meshes, sometimes failing to produce stable rollouts, as shown in Figure 5.6c.

One contributing factor is that in highly refined regions, the spatial extend of message passing is reduced, limiting information exchange to small, local areas. This can degrade prediction accuracy, particularly for pressure, which depends on the global flow field [1].

[1] This reflects the fundamentally different roles of pressure and velocity: pressure is obtained by solving a global elliptic equation (2.17), and thus depends on the flow field throughout the domain, while velocity is advanced locally through advection and diffusion, making it primarily influenced by nearby conditions

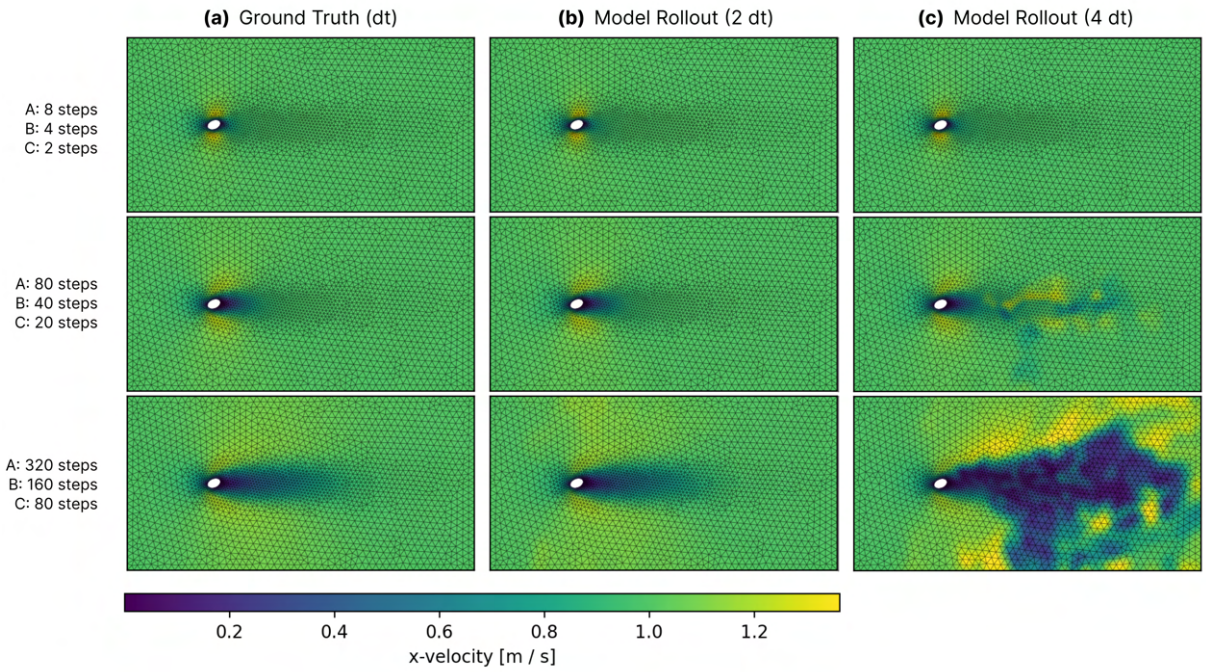


Figure 5.6: Predicted velocity field from rollout on an validation test case with a fine mesh. Results are produced **(a)** the CFD solver with timestep dt, **(b)** the model with timestep $2\times$dt and **(c)** the model with timestep $4\times$dt.

Another factor is that smaller cells exhibit more abrupt state changes due to reduced numerical diffusion, making the mapping harder to learn. One solution is to train on data generated with a smaller timestep, allowing the state evolution in fine cells to appear more gradual. As shown in Figure 5.6b, halving the timestep enabled a stable rollout. To isolate this effect from dataset size, an alternative dataset was created using the original timestep but with twice as many training examples. This model still failed on fine meshes, indicating a timestep stability condition related to mesh size. Notably, this condition is less restrictive than the traditional CFL limit, as stable rollouts were achieved with timesteps twice the CFL-bound value.

However, reducing the timestep doubles the computational cost, and the increased number of prediction steps leads to greater error accumulation in velocity, pressure, and divergence (see Figure 5.7). One approach is to address the limited message-passing range by using multi-scale message passing—a technique that augments the model's receptive field by aggregating information across multiple spatial resolutions, allowing better context integration in highly refined regions [63]. Integrating this with a hard divergence-free constraint may offer a promising path toward improved stability and accuracy on heterogeneous meshes.

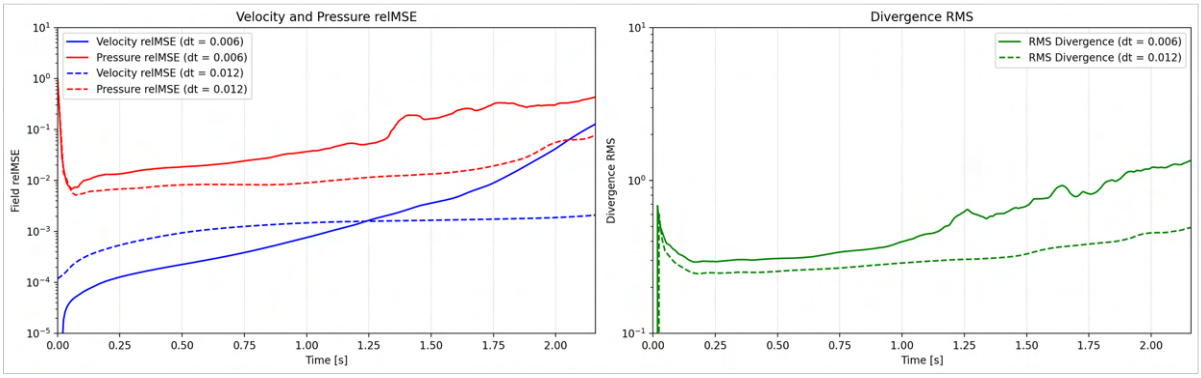


Figure 5.7: Evolution of velocity, pressure and divergence errors during two different rollouts, using timesteps sizes of 0.006 (solid) and 0.012 (dashed). Error accumulation is quicker for the smaller timestep model (solid) as twice as many prediction steps are required per simulation.

5.3.2 Testing Geometric Generalisation

Since ellipse size – and consequently mesh refinement – strongly influenced rollout performance, it was challenging to assess the model's generalisation to new geometries independently. For instance, the model performed well on a test case with an ellipse 8% larger than any seen during training, achieving a lower final-timestep velocity RelMSE (3.23×10^{-3}) than that of the largest training ellipse (9.07×10^{-3}). However, this likely reflects the advantage of a coarser mesh rather than true generalisation.

Nonetheless, a test case with a negative orientation angle showed comparable performance to a training example of similar size with the same angle magnitude but positive sign, suggesting the model respects geometric symmetry.

6. Conclusion to Report I

This report demonstrated the potential of GNNs as surrogate models for fluid simulations and identified a potential research gap: the hard enforcement of the incompressibility constraint into a GNN surrogate model. A complete training pipeline for GNNs was developed, including custom dataset generation, preprocessing, and distributed multi-GPU training. This provides a solid foundation for future model development.

Initial results on CylinderFlow highlighted limitations in rollout stability. While short-term predictions were accurate and could recover physical quantities such as pressure force, extended rollouts exhibited instabilities: pressure blow-up in steady flows and phase drift of oscillation in unsteady flows. These findings motivate the use of hard physical constraints to improve long-term stability. Additional strategies were proposed to enhance rollout performance, including stricter pressure loss handling and curriculum learning. Further experiments on a custom dataset, LaminarEllipse, highlighted sensitivity to mesh resolution, indicating that future divergence-free architectures may benefit from global message passing, particularly when handling datasets with heterogeneous mesh resolutions.

Report 2 will focus on enforcing the divergence-free constraint exactly through modifications to the GNN architecture. This inductive bias aims to enhance stability, reduce data dependence and improve generalisation.

Part II

Second Report

7. Introduction to Report II

This work directly extends Report I, aiming to improve the FVGN model through architectural and training modifications to enhance the accuracy and stability of the CFD simulations.

Before progressing to new architectures, further validation was conducted for FVGN. Section 9.1 reports the successful validation of my FVGN implementation against the original results, establishing a dependable baseline from which to build new architectures.

Section 8.1 then introduces a new training dataset, *EllipseFlow*, designed to address key limitations in *LaminarEllipse*, particularly those related to spatial and temporal scale variance. To adjust to this new training distribution, a series of sweeps are performed over hyperparameters, such as message-passing depth and noise injection levels, which are detailed in Section 9.2.

The primary research effort was directed at designing conservative architectures, particularly those that impose hard divergence-free constraints. Architectures are proposed in Section 8.2 and their viability and performance evaluated in Chapter 10. In-depth validation of several architectures revealed important failure modes and design limitations, ultimately yielding a deeper understanding of the challenges involved in training physically constrained models. In response to some of these challenges, other architectural and training changes, unrelated to conservation, were explored; these are evaluated in Chapter 11.

Finally, Chapter 12 reflects on the successes – and shortcomings – of the explored approaches, highlighting the trade-off between physical fidelity and architectural simplicity in GNN-based physics prediction. Directions for future research are outlined across the model design, training strategy, and application domains.

8. Methodology Refinement

8.1 Dataset Generation

In Section 5.3.1, it was shown that *LaminarEllipse* suffered from inconsistent spatial and temporal scales, which limited GNN surrogate models without multi-scale message passing. Specifically, mesh refinement varied with Reynolds number (Re) and geometry, altering the effective receptive field of message passing. This introduces an implicit distribution shift: the model must learn different relational patterns for different mesh resolutions, increasing sample complexity and impairing generalisation. To address this, *EllipseFlow* was developed, preserving the same basic setup – an elliptic cylinder with variations in angle of attack, inlet velocity, and size (Figure 8.1b). Scale-related issues are explicitly resolved by fixing the minimum and maximum mesh spacings according to the smallest ellipse diameter and the maximum Re across the dataset, ensuring a uniform message-passing scope across all training and test cases.

Additionally, greater attention was paid to temporal scales, with the simulation length informed by the characteristic shedding frequency f for each geometry. The oscillation period is given by

$$T = \frac{1}{f} = \frac{L}{U \times St}, \tag{8.1}$$

where L and U are characteristic length and velocity, and St is the Strouhal number. For intermediate angles of attack, the ellipse approaches bluff-body behaviour, with St potentially as low as 0.12. For a consistent runtime between simulations, the geometry with the largest L/U ratio determines the largest shedding period, T, and simulations are run for $100T$ to allow initial transient dynamics to decay and to capture a sufficient number of steady oscillation cycles for reliable learning. Timestep size is conversely set by the system with the shortest timescale, determined through the CFL condition, and applied uniformly across all simulations to ensure consistent temporal sampling.

The *EllipseFlow* dataset comprises 50 training, 5 validation, and 5 test simulations

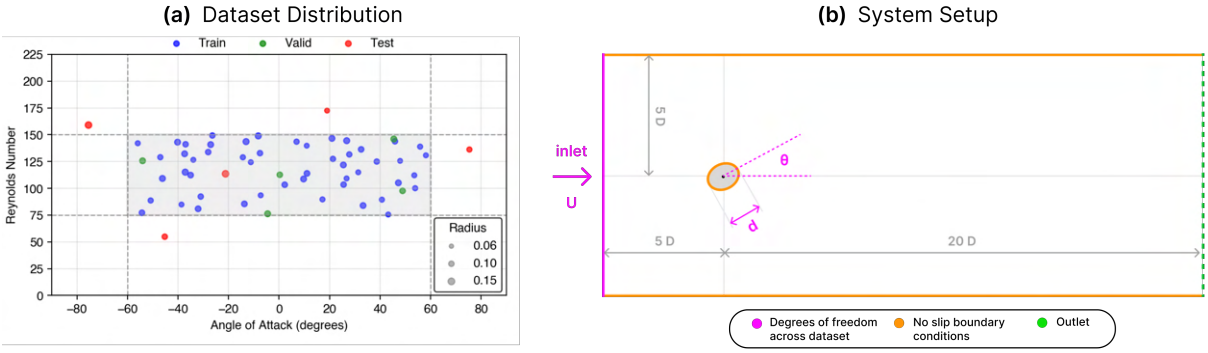


Figure 8.1: **(a)** Distribution of Reynolds number vs. angle of attack for training, validation, and test examples from the *EllipseFlow* dataset. Test examples are chosen to be from outside the training distribution to assess model generalisation. Note that one test example appears in-distribution on these axis, but is out-of-distribution with respect to major axis radius. **(b)** Schematic of the *EllipseFlow* setup showing inlet velocity U, angle of attack θ and major axis diameter d. Note that D is the largest major axis diameter from across the dataset. Boundary conditions: fixed velocity at inlet with zero-gradient pressure; outlet with fixed pressure and zero-gradient velocity; and no-slip walls and obstacle with fixed velocity and zero-gradient pressure. The dynamics viscosity is chosen to be $\nu = 1e-3$, and so with $d \sim 0.2$ and $Re \sim 10^2$, U is ~ 0.5

(Figure 8.1a), each running for 600 timesteps. Although its size is only 5% of the *CylinderFlow* benchmark dataset – and smaller datasets can limit model performance [64, 65] – this reflects a deliberate compromise between dataset scale and training efficiency. The reduced size shortens training to under 6 hours on a single GPU, compared to over 70 GPU hours for *CylinderFlow*, enabling faster and more cost-effective model architecture iteration.

An additional benefit of generating *EllipseFlow* is complete control over solver configurations, allowing direct verification of solver convergence and physical consistency. The solver was first validated through spatial convergence testing against the analytical Taylor–Green Vortex problem, then tested on *EllipseFlow* simulations by monitoring residuals and physical quantities (lift and drag coefficients).

Full configuration details, convergence study results, and temporal–spatial scale considerations are provided in Appendix A.1.

8.2 Conservative Model Architectures

This section presents a set of architectures that build on the FVGN framework introduced in Section 4.1. [1] Each design incorporates conservation-inspired mechanisms, aiming to enhance rollout stability and long-term accuracy.

[1] Further details on the original FVGN model's normalisation method, MLP configuration, and parameter count are provided in Appendix A.2.

8.2.1 Vertex Potential (VertPot)

In addition to the existing FVGN face predictions, VertPot aggregates latent edge features to vertices and predicts a scalar potential ϕ_v at each vertex. For every face, the flux F_f is defined as the signed difference between the ϕ_v values at its two vertices (see Figure 8.2). Using a fixed anticlockwise vertex ordering for each cell, the sum of these differences around the N_f faces of a cell satisfies:

$$\sum_{i=0}^{N_f-1} \left(\phi_{i+1} - \phi_i\right) = \sum_{i=0}^{N_f-1} F_f = 0 \quad (\phi_{N_f} := \phi_0),$$

which (by the telescoping property) enforces local flux conservation for each cell.

These locally conserved F_f values are then used directly in the integration equation, together with the existing $\mathbf{u}_f$, p_f, and $\mathbf{Q}_f$ predictions at the faces, by updating the convective term in Equation 4.3:

$$\mathbf{u}^{t+\Delta t} - \mathbf{u}^t = \frac{\Delta t}{V}\left[-\sum_{f\in\text{cell}} \mathbf{u}_f F_f - \frac{1}{\rho}\sum_{f\in\text{cell}} p_f A_f \mathbf{n}_f + \sum_{f\in\text{cell}} \mathbf{Q}_f\right]. \tag{8.2}$$

Although VertPot ensures locally conserved fluxes, it is not obvious that this property improves accuracy when used in FVGN's normalized-space integration, where the fluxes have no physical interpretation (see Appendix A.2). To investigate this, we independently developed a physical-space integration model, discussed in Section 11.1, and combined this with the VertPot flux predictions.

8.2.2 Streamfunction Construction (StreamFunc)

StreamFunc reverses the FVGN processor order: the final update and decoding occurs at cell-centre (see Figure 8.3). The model now predicts two scalar values at each cell-centre,

$$\mathbf{c}_i = (\psi_i, p_i)^\top$$

where ψ_i is the streamfunction and p_i is the cell-centre pressure. [2] For a two-dimensional system, the rotated gradient (Section 2.3.1) of the streamfunction yields a divergence-free

[2] Note that sometimes the streamfunction is defined so that $\mathbf{u} = (\partial\psi/\partial y, -\partial\psi, \partial x)$, but here we define it in terms of the rotated gradient convention.

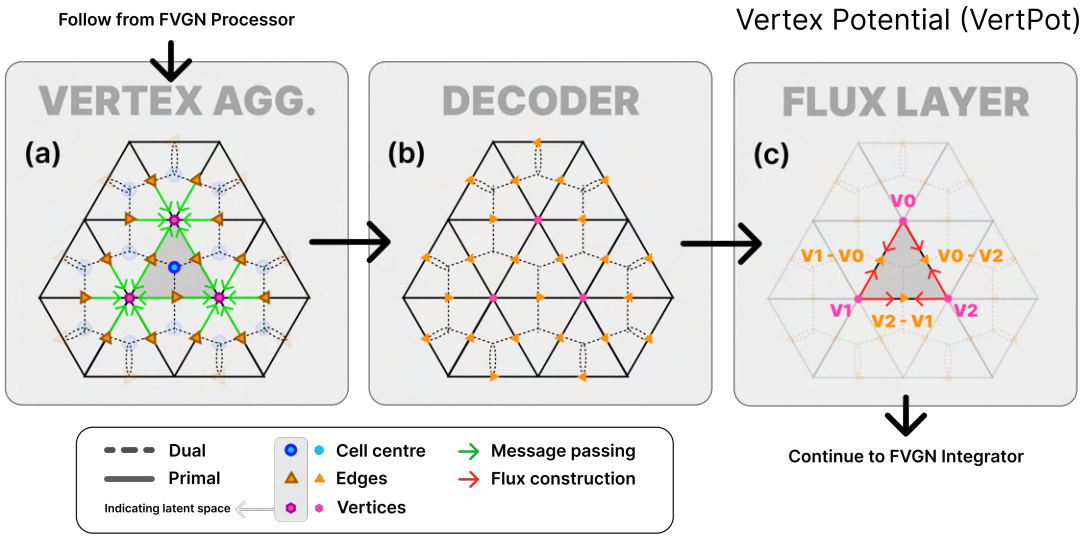


Figure 8.2: Schematic of the Vertex Potential (VertPot) architecture. **(a)** Aggregates (by summation) the latent face features from the FVGN processor to the vertices. **(b)** Decodes $\mathbf{e}_{i\to j} = \text{concat}(\mathbf{u}_f, p_f, \mathbf{Q}_f)^\top$ at faces (as in FVGN) and additionally decodes a scalar ϕ_v at each vertex. **(c)** For each face f connecting vertices v_a and v_b, the face flux F_f is computed as $\phi_{v_b} - \phi_{v_a}$, with a consistent anticlockwise ordering applied globally. This makes F_f antisymmetric for adjacent cells and ensures the per-cell fluxes sum to zero. F_f is used together with $\mathbf{u}_f$, p_f, and $\mathbf{Q}_f$ in the FVGN integration layer, modified to use Equation 8.2.

velocity field:

$$\mathbf{u}_c = \left(-\frac{\partial \psi_c}{\partial y}, \frac{\partial \psi_c}{\partial x} \right)$$

Appendix B.1 derives this result and demonstrates how this formulation naturally generalises to higher dimensions, enabling application to other conservation laws [17].

Inspired by Liu et al. [19], the streamfunction gradients at cell centres are computed via the Moving Least Squares (MLS) method [66]. MLS reconstructs values by fitting a local polynomial of degree p to neighbouring cell data using geometry-dependent weighted least squares. The choice of p determines the formal accuracy of the reconstructed gradients, which in turn affects how well divergence-free constraints are satisfied. More details on the MLS method are provided in Appendix B.2.

Additionally, a revised denormalisation approach is required for the velocity field. Using the component-wise z-score denormalisation from FVGN would rescale the x and y velocity components differently, distorting the field and breaking the divergence-free property. The same denormalisation scaling must therefore be applied to both components. The simplest approach is to normalise both velocity components with a single scaling factor, such as the standard deviation of the velocity magnitude across the dataset. Alternatively, the anisotropic z-score is retained for input normalisation, while the *denor-*

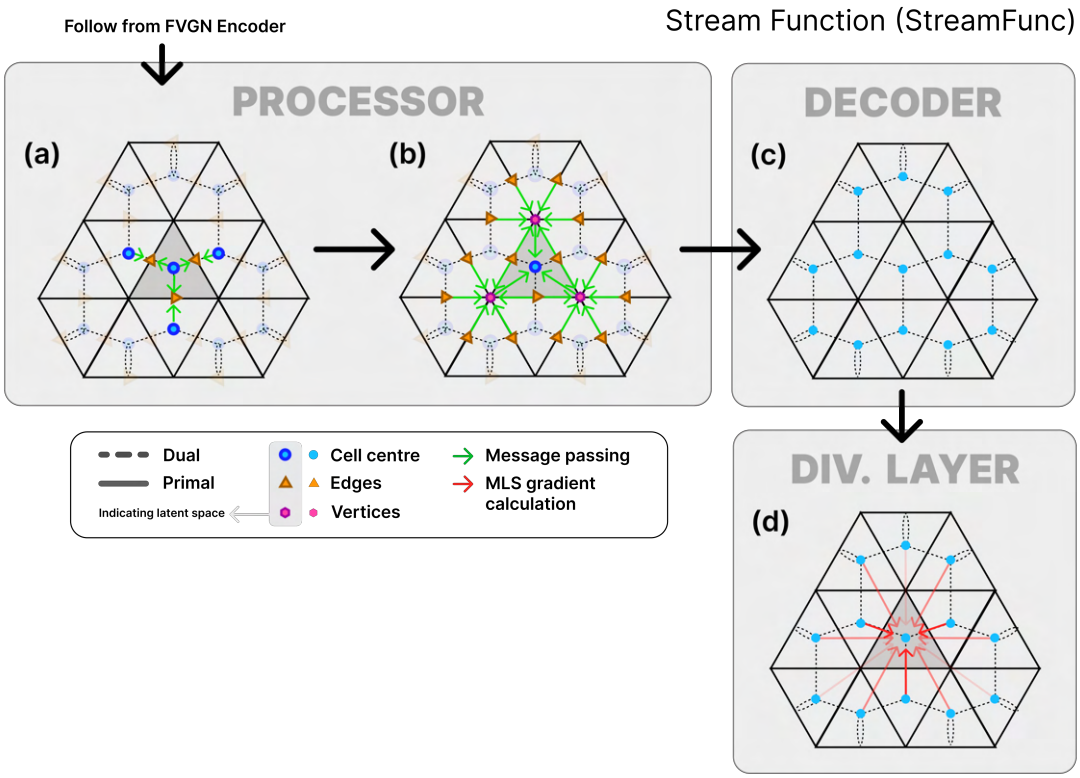


Figure 8.3: Schematic of the streamfunction (StreamFunc) approach. **(a-b)** This is the reverse of the FVGN processor of Figure 4.1b–d: edge features are updated first, followed by cell feature updates via the FVGN's "twice message passing" method [20]. **(c)** Decode the streamfunction ψ and pressure p at cell centres. **(d)** Compute cell-centre velocities by evaluating the rotated gradient of ψ using Moving Least Squares (MLS): $\mathbf{u}_c = \left(-\frac{\partial \psi_c}{\partial y}, \frac{\partial \psi_c}{\partial x} \right)$.

malisation step is modified: instead of rescaling the velocity components directly, ψ is scaled by a learnable parameter so that its gradients produce velocities with the correct physical scale. Given the strong anisotropy of the *EllipseFlow* system – where the y-component mean is typically 10^{-3} times smaller than the x-component – the latter approach is adopted, as isotropic input scaling would hinder learning from the y-velocity.

A final consideration is that of gauge freedom. Since the velocity field depends only on spatial gradients of ψ, any constant offset in ψ produces the same velocity. In other words, the space of constant functions forms a null space of the gradient operator. If left unconstrained, the model may drift arbitrarily in this null direction, which can make training unstable. To eliminate this degree of freedom, the mean value of ψ across the mini-batch is subtracted, which forces ψ to have zero spatial average and removes the null-space ambiguity.

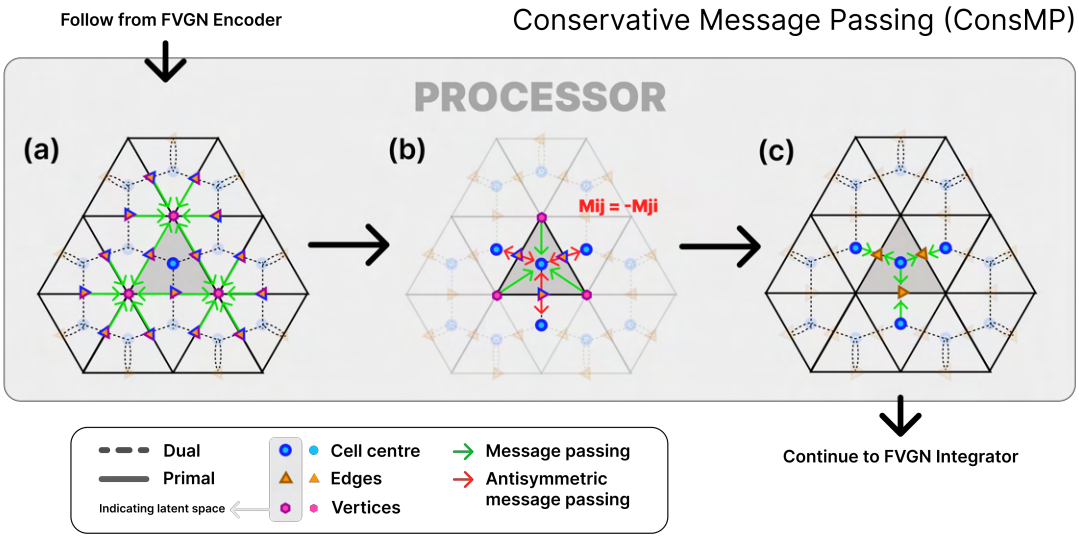


Figure 8.4: Schematic of the conservative message passing (ConsMP) approach. **(a)** Split edge features into antisymmetric and symmetric parts. Symmetric parts aggregated at vertices (as in FVGN). **(b)** Aggregate message as cell centres: symmetric from vertex aggregations and antisymmetric directly from edges. Ensure that messages to adjacent cells are oppositely signed to encourage conservative message passing. **(c)** Update edge features and continue with normal FVGN integration.

8.2.3 Conservative Message Passing (ConsMP)

This approach is not guaranteed to yield a divergence-free velocity field but incorporates an inductive bias inspired by flux conservation in traditional CFD methods and by Horie et al.'s work on conservative message passing [54]. In FVGN, edge-to-cell message passing follows a two-step process – first aggregating at nodes, then at cell centres – which neglects the equal and opposite exchange of fluxes between adjacent cells (i.e. $m_{i\to j} = -m_{j\to i}$). As shown in Figure 8.4, the conservative message passing (MP) method addresses this by decomposing edge features into *symmetric* and *antisymmetric* components. Antisymmetric edge features form antisymmetric messages (i.e. equal and opposite) for adjacent cells, while symmetric information can undergo the standard FVGN two-step aggregation.

The key design choice is how to split edge features. One option is a heuristic split, dividing latent edge encoding in half to represent symmetric and antisymmetric features: the model must implicitly learn to disentangle symmetric and antisymmetric information during training. This model is named ConsMP.

A more rigorous approach – named ConsMP-Rig – is to encode feature symmetry as an explicit physical prior:

- Edge direction-dependent quantities (e.g., velocity difference, face normal) form the antisymmetric features $\mathbf{e}_{\text{asym}}$.

- Edge direction-independent quantities (e.g., face area, distance between cells) form the symmetric features $\mathbf{e}_{\text{symm}}$.

These two edge features are maintained throughout the model computation so that their symmetries are preserved. This ensures that if the graph edge direction was reversed, $\mathbf{e}_{\text{asym}} \to -\mathbf{e}_{\text{asym}}$ for all layers. Consequently, although the roles of owner (i) and neighbour (j) cells are swapped when edge direction is reversed, the flux-like messages exchanged between cells remain unchanged.

Aggregation and updates are formulated in a way to preserve the features symmetries. Cell latent features are updated by,

$$\mathbf{c}_{\text{updated}} = \mathbf{c} + \text{MLP}\left(\mathbf{c}, \sum_{\text{faces}} \mathbf{e}_{\text{symm}}, \sum_{\text{faces}} \mathbf{e}_{\text{asym}}\right),$$

and edge updates follow,

$$\begin{aligned}\mathbf{e}_{\text{symm,updated}} &= \mathbf{e}_s + \text{MLP}(\mathbf{e}_{\text{asym}}, \mathbf{c}_i + \mathbf{c}_j),\\ \mathbf{e}_{\text{asym,updated}} &= \mathbf{e}_{\text{asym}} + \text{MLP}_{\text{asym}}(\mathbf{e}_{\text{asym}}, \mathbf{c}_i - \mathbf{c}_j)\end{aligned}$$

where MLP_{asym} employs tanh activation, no bias weights, and no normalization layers to preserve antisymmetry. For decoding, we note that $\mathbf{u}_f$, p_f, and $\mathbf{Q}_f$ are independent of the graph edge direction and so are symmetric. Hence the decoder was chosen to be,

$$\mathbf{e}_{\text{decoded}} = \text{MLP}(\mathbf{e}_{\text{symm}}, \mathbf{e}^2_{\text{asym}})$$

where the antisymmetric features have been symmetrised through taking the square.

8.3 Assessing Model Performance

The Relative MSE (RelMSE) was introduced in Section 5.2.2 as a metric to compare the error between the prediction and ground-truth CFD data at a given timestep. While RelMSE is useful for assessing how well the model emulates the precise time series of simulation data, it does not necessarily reflect whether the model attains a physically correct steady-oscillation regime: phase discrepancies introduced in the transient start-up can obscure an otherwise accurate prediction of vortex shedding. To address this limitation, several phase-insensitive metrics are introduced. These are time-averaged over the "steady" oscillation portion of the simulation and, where relevant, restricted to a subdomain Ω in the wake region behind the obstacle. These metric are:

- **Recirculation Length** L_r: the distance from the trailing edge of the obstacle to the downstream centerline location where time-averaged $\langle u_x \rangle_t = 0$ (zero-crossing). This provides a simple characterization of the wake bubble structure.

- **Total Kinetic Energy (E_K)** per unit mass is defined as

$$\overline{E_K} = \left\langle \int_\Omega \frac{1}{2} \|\mathbf{u}(\mathbf{x}, t)\|^2 \, \mathrm{d}V \right\rangle_t ,$$

  where $\langle \cdot \rangle_t \equiv \frac{1}{T} \int_0^T (\cdot) \, \mathrm{d}t$ denotes the temporal average. E_K serves as a global diagnostic for simulation stability.

- **Fluctuation Kinetic Energy ($E_{\mathbf{fluc}}$)** quantifies the energy of velocity fluctuations in the vortex-shedding regime:

$$\overline{E_{\text{fluc}}} = \left\langle \int_\Omega \frac{1}{2} \|\mathbf{u}'(\mathbf{x}, t)\|^2 \, \mathrm{d}V \right\rangle_t , \qquad \mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle_t,$$

  with $\langle \mathbf{u} \rangle_t$ as the time-averaged velocity field. It provides a measure of the intensity of wake-field oscillations associated with vortex shedding.

- **Drag C_D and lift C_L coefficients** are computed from the pressure distribution along the ellipse boundary ∂b. The total hydrodynamic force is

$$\mathbf{F}(t) = \int_{\partial b} (-p\mathbf{I} + \tau_{\mathbf{w}}) \, \mathbf{n} \, \mathrm{d}L$$

  where p is the pressure on the surface, $\mathbf{n}$ the outward normal, and $\tau_{\mathbf{w}}$ the viscous stress tensor at the wall. For $Re \sim 10^2$, viscous forces are assumed to be small relative to pressure forces [67], and are therefore neglected.[3] The non-dimensional force coefficients are,

$$C_D(t) = \frac{F_x(t)}{\frac{1}{2}\rho U_\infty^2 D}, \quad C_L(t) = \frac{F_y(t)}{\frac{1}{2}\rho U_\infty^2 D}.$$

[3] Estimate for ratio of viscous to pressure force: the wall shear stress is $\tau_w \sim \mu U/\delta$, where δ is the characteristic boundary-layer thickness. For laminar flow, $\delta \sim x/\sqrt{Re}$, with x a characteristic length (taken here as D). The dynamic viscosity is $\mu = \rho U x/Re$ from the definition of Re. The characteristic pressure scale is ρU^2. Combining these:

$$\frac{\tau_w}{\rho U^2} \sim \frac{\mu U/\delta}{\rho U^2} = \frac{(\rho U x/Re)\, U/(x/\sqrt{Re})}{\rho U^2} = \frac{1}{\sqrt{Re}}.$$

Thus, viscous forces are smaller than pressure forces by a factor $\mathcal{O}(Re^{-1/2})$ [68, 67]. Given our simulations have $Re \sim 100$, pressure forces dominate.

In line with CFD practice, the mean drag $\overline{C_D}$ and RMS lift $C_{L,\mathrm{RMS}}$ are reported, since these are the standard quantities used to characterise average drag and unsteady lift in vortex-shedding flows.

- **Strouhal number** St is obtained from the dominant oscillation frequency in $C_L(t)$. Instead of working directly with the Fourier transform – which yields complex-valued amplitudes – analysis is performed using the power spectral density (PSD) $S_{xx}(f)$, which expresses the distribution of signal power across frequencies and facilitates quantitative comparison of peak amplitudes. The PSD is computed as

$$S_{xx}(f_k) = \frac{\Delta t}{N} \left|\mathrm{FFT}[x(t)]_k\right|^2$$

  The dominant frequency f_1 in $S_{xx}(f)$ is taken as the vortex shedding frequency, and St is given by $St = f_1 D/U_\infty$, with U_∞ chosen to be the inlet velocity and D the ellipses' major axis diameter.

- **Additional spectral metrics** derived from $S_{xx}(f)$ include the magnitude-squared coherence γ^2 between predicted (i) and reference (j) signals at vortex shedding frequency:

$$\gamma^2_{C_L}(f_1) = \frac{|S_{ij}(f_1)|^2}{S_{ii}(f_1)\, S_{jj}(f_1)}$$

  where S_{ij} is the "cross-PSD". For $\gamma^2 > 0.7$, indicating that a stable linear relationship exists between the predicted and reference signals at f_1, the phase offset between the signals can estimated as $\phi_{C_L}(f_1) = \arg(S_{ij}(f_1))$.

Estimates for St and other spectral metrics are also computed from $v_y(t)$ at wake probes, which serve both to confirm correlation between velocity and pressure fields, and to provide a fallback when pressure-based C_L predictions are unreliable.

To ensure that these metrics are comparable between simulations with different velocity and pressure scales, we use the normalised RMSE (nRMSE) compared to the ground truth:

$$\mathrm{nRMSE} = \frac{\sqrt{(x_{\mathrm{pred}} - x_{\mathrm{gt}})^2}}{\overline{x}_{\mathrm{gt}}},$$

where $\overline{x}$ represents the average over all points on the spatial domain. It follows that the "mean" nRMSE refers to the time-average of this value over simulation steps (excluding the transient start-up).

9. Validation Results

9.1 Validation of FVGN

In Report I, the accuracy and stability reported for FVGN in the original study could not be reproduced. Before exploring new architectures, the focus was placed on resolving this issue. An error in how graph connectivity was constructed during dataset preprocessing was identified as a key source of the degraded performance, among other subtle differences in preprocessing and dataloader steps. After aligning my implementation more closely with the original literature implementation [20], the model successfully produced stable rollouts on the *CylinderFlow* dataset (presented in Figure 9.1). Despite training for only 5 out of the 23 epochs used in the original study,[1] the steady-state test case from [20] achieved a RelMSE of 9×10^{-5}, approaching the value of 6×10^{-5} reported in the original paper [20]. This represents an improvement of roughly two orders of magnitude over the results shown in Report I.

This investigation underscored the sensitivity of performance to dataset preparation; no faults were identified in the model architecture itself. This finding was a primary motivator behind a greater focus on aspects beyond model architecture such as the dataset construction (Section 8.1) and exploring different normalisation and training approaches (Chapter 11).

Ultimately, these results show that FVGN performance can be replicated within our codebase. With a validated FVGN reference implementation, this provided a solid basis for comparing new model architectures against the FVGN baseline.

[1] A full 23 epoch training run with the literature codebase revealed that validation rollout performance plateaued after the first few epochs, with minimal improvement thereafter. This compromise enabled faster debugging and iteration by avoiding the $\sim$ 70-hour cost of a full training run.

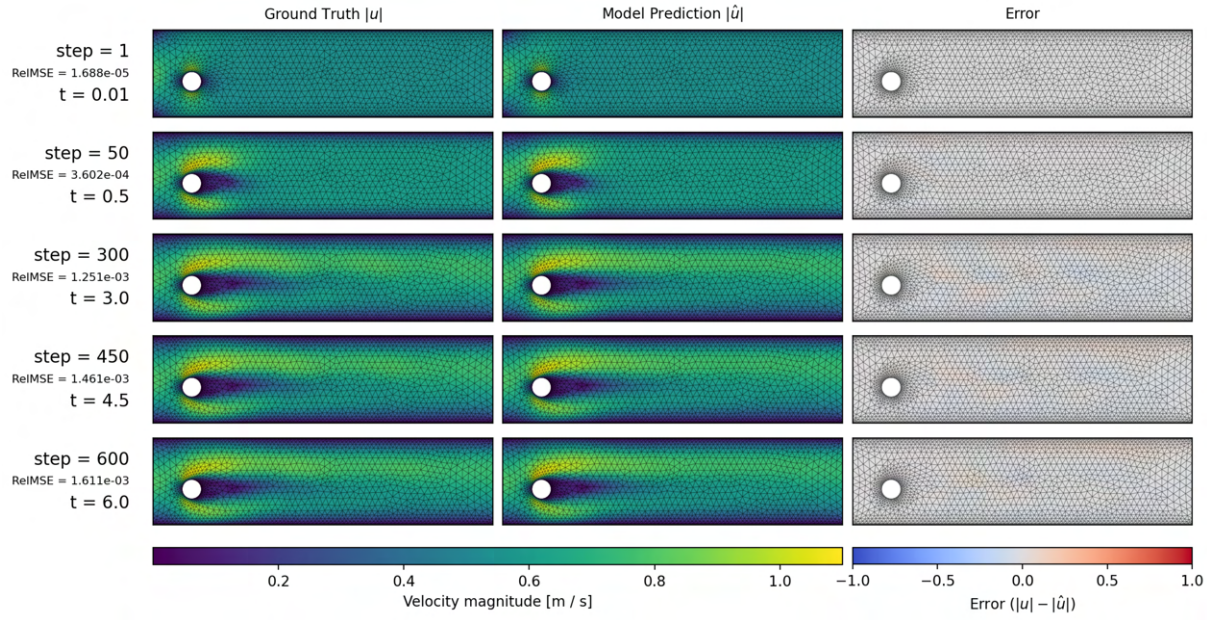


Figure 9.1: Velocity field snapshots from a rollout with the corrected FVGN implementation. This can be compared directly to Figure 5.2 in Report I to see the improved stability.

9.2 Hyperparameter Tuning

Given that the *EllipseFlow* dataset differs from the *CylinderFlow* dataset used in the original study, the hyperparameters were re-tuned for this new dataset distribution. Hyperparameter sweeps were conducted on 40% subset of training dataset, and run for 15 epochs. Performance was assessed across the five in-distribution validation simulations using the metrics outlined in Section 8.3.

To restrict the search space, certain hyperparameters – such as the MLP depth, MLP width (latent vector size), and loss term weights – were fixed to the original FVGN configuration. A summary of selected hyperparameters following this analysis is presented in Table 9.1.

9.2.1 Message Passing Depth

Beyond MLP depth and width, the primary hyperparameter controlling model capacity is the number of message passing (MP) layers, since each layer has its own learnable weights. Increasing the MP depth therefore increases the model expressivity, as well as enlarging the receptive field of message passing.

Sweeping over MP layers $\in 1, 2, 4, 8, 16$ yields the expected trend of improved valida-

Table 9.1: Summary of hyperparameters and training configuration.

Category	Details
Dataset	
Train set	50 geometries, 400 timesteps
Validation set	5 geometries, 700 timesteps
Test set	5 geometries, 700 timesteps
Model	
MLP depth (hidden layers)	3
MLP width (latent size)	128
MLP activation	SiLu
Message passing layers	15
Training	
Batch size	4
Noise level	0.06
Learning schedule	hybrid
Max learning rate	1×10^{-3}
Min learning rate	1×10^{-6}
Number of epochs	30

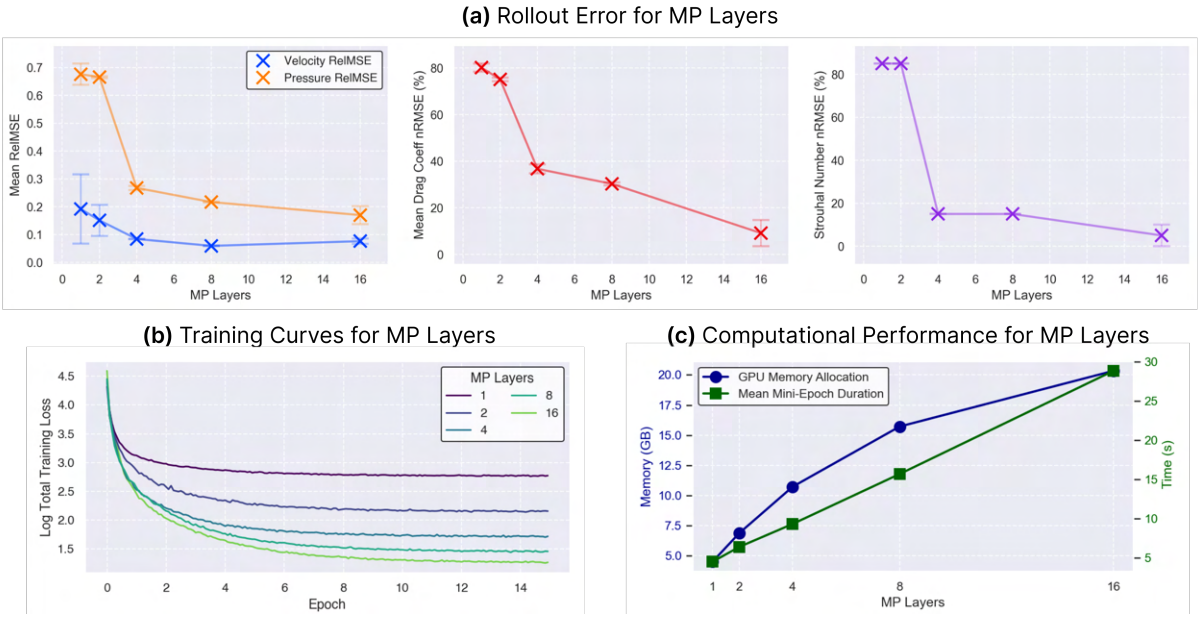


Figure 9.2: **(a)** Rollout error compared to ground truth for different MP layers, showing RelMSE for velocity and pressure, mean drag coefficient error, and Strouhal number error. Discrete jumps in Strouhal error arise from the finite frequency bin width $1/dt$. Error bars represent the standard error in the mean (SEM $= \sigma/\sqrt{n}$) across validation simulations. **(b)** Training loss curves for different MP counts, illustrating diminishing improvements at high depth. **(c)** GPU memory allocation and mean mini-epoch duration vs. MP layers. A mini-epoch is 1000 training updates plus one validation step. Both metrics increase with depth, though memory allocation appears sub-linear, likely due to backend memory management effects.

tion performance with depth. For example, Figure 9.2a shows decreasing errors for the velocity and pressure RelMSE, drag coefficient, and Strouhal number. Note that the dis-

crete jumps in Strouhal error are due to the finite widths of frequency bands from the Fourier nalysis. Velocity predictions plateau beyond 8 layers – likely because velocity's local dependencies are sufficiently captured within smaller receptive fields – whereas pressure predictions benefit from the larger receptive fields enabled by deeper message passing due to pressure's stronger global coupling. An additional contribution to the saturation in error is the combined effect of oversmoothing – where depth-driven aggregations homogenise node features – and the optimisation and gradient flow challenges inherent in deeper architectures [69]. Figure 9.2b supports this interpretation: additional layers yield only smaller training loss reductions despite large capacity increases.

Figure 9.2c shows that mini-epoch time and GPU memory allocation both increase with depth. Here, a mini-epoch is defined as 1000 training updates plus one validation step, capturing the runtime of both forward/backward passes during training and inference during validation. We observe the expected linear scaling for runtime, but not for memory growth – this is potentially explained by the fact that the reported memory corresponds to allocation rather than the true usage, and so is impacted by PyTorch's memory allocator, kernel overhead, and fragmentation.

The scaling of runtime and memory constrains training, restricting batch sizes and network depth, and creating a trade-off between computational cost and predictive accuracy when choosing message passing depth. For this study, 15 MP layers are chosen to prioritise high accuracy, and also to match the model capacity originally used by Li et. al [20].

9.2.2 Learning Rate

The original FVGN implementation uses a multi-stage learning schedule: a step decay of factor 0.1 at $\sim 21\%$, hold until $\sim 50\%$, and then a final exponential decay until the minimum learning rate [20].

This schedule was compared against a cosine annealing schedule, motivated by its widespread ML use due to its ability to retain a higher learning rate for longer without destabilising training [70]. While a reduced training loss was reached (see Figure 9.3b), rollout performance showed limited improvement in RelMSE accuracy and degraded physical consistency in metrics such as kinetic energy fluctuations and velocity divergence (Figure 9.3c). This suggested that the sharp learning rate drop in the original schedule may act as a form of regularisation, biasing optimisation toward flatter minima that better preserve physical structure during autoregressive rollouts [71].

Figure 9.3: **(a)** Learning rate schedules for the original, cosine annealing, and hybrid approaches. **(b)** Log total training loss over epochs. **c)** Validation metrics box plots: total RelMSE, kinetic energy fluctuation nRMSE, and RMS divergence. The whiskers give full range and boxes give the IQR.

To address this trade-off, a hybrid schedule was introduced. It consists of an initial plateau, then a rapid cosine decay to locate robust flat minima, and finally slower cosine decay for finer refinement. This design preserves the training stability provided by smooth learning rate schedules while reintroducing the regularisation-like benefits of a rapid learning rate reduction. Empirically, the hybrid schedule converged faster and was superior in all validation metrics (Figure 9.3c).

Additionally, learning rate sensitivity analysis confirmed that the maximum learning rate of 1×10^{-3} from the original paper represents an optimal balance for predictive accuracy, with values 50% above and below showing degraded performance, as illustrated by Figure 9.4.

9.2.3 Batch Size

Larger batch sizes perform fewer gradient updates per epoch, and so learning rates were scaled proportionally to $\sqrt{\text{batch size}}$ to account for this [71]. Despite this adjustment, smaller batch sizes both converged more rapidly across all loss terms (Figure 9.5a) and achieved lower validation RelMSE in rollout evaluation (Figure 9.5b). This advantage likely arises not solely from the higher gradient update frequency but also from the stochastic noise intrinsic to small batches, which perturbs the optimisation trajectory,

Figure 9.4: Validation metrics for different maximum learning rates for the hybrid learning schedule. From left to right: total relative MSE, mean drag coefficient nRMSE, and mean RMS divergence. The optimal learning rate of 1×10^{-3} minimizes all three metrics.

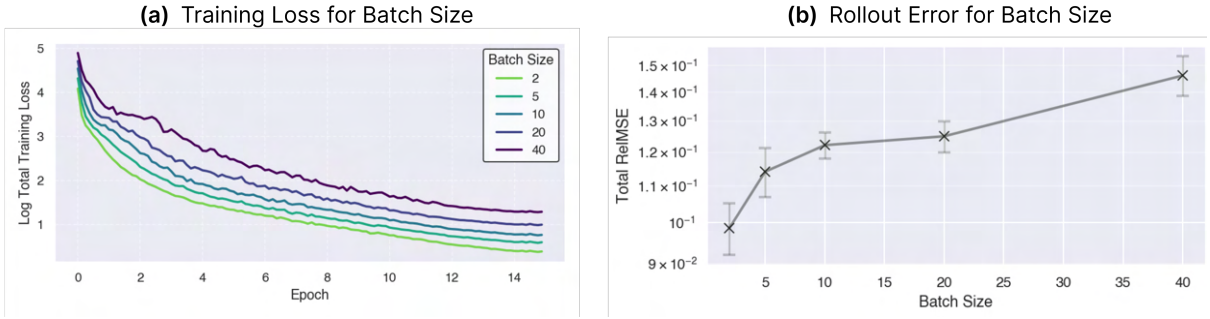


Figure 9.5: **(a)** Evolution of log-scaled total training loss across epochs for batch sizes $2, 5, 10, 20, 40$. Smaller batch sizes yield faster convergence and lower final loss. **(b)** Total validation rollout RelMSE versus batch size, showing an increase in error for larger batches, suggesting improved generalisation for smaller batches.

promotes exploration of broader parameter regions, and biases the model toward flatter minima. This effect acts as implicit regularisation and improved robustness, particularly in autoregressive rollout settings where early errors can compound over time.

However, increasing the number of gradient updates increases training time due to additional backpropagation. For instance, with the same number of epochs, training with a batch size of 40 was 2.3 times faster than with a batch size of 2. A batch size of 4 was therefore selected as a compromise, balancing the beneficial update frequency and noise regularization of smaller batches against acceptable training speed.

9.2.4 Noise Scale

During training, zero-mean Gaussian noise with standard deviation $\sigma = \eta \cdot \bar{u}$ was independently injected into each node's velocity input, where η is the noise level and $\langle \bar{u} \rangle_t$ is the mean velocity across the simulation. Experiments were conducted for $\eta \in 0, 0.015, 0.03, 0.045, 0.06, 0.09$. Figure 9.6a-b shows that increasing η reduced rollout error

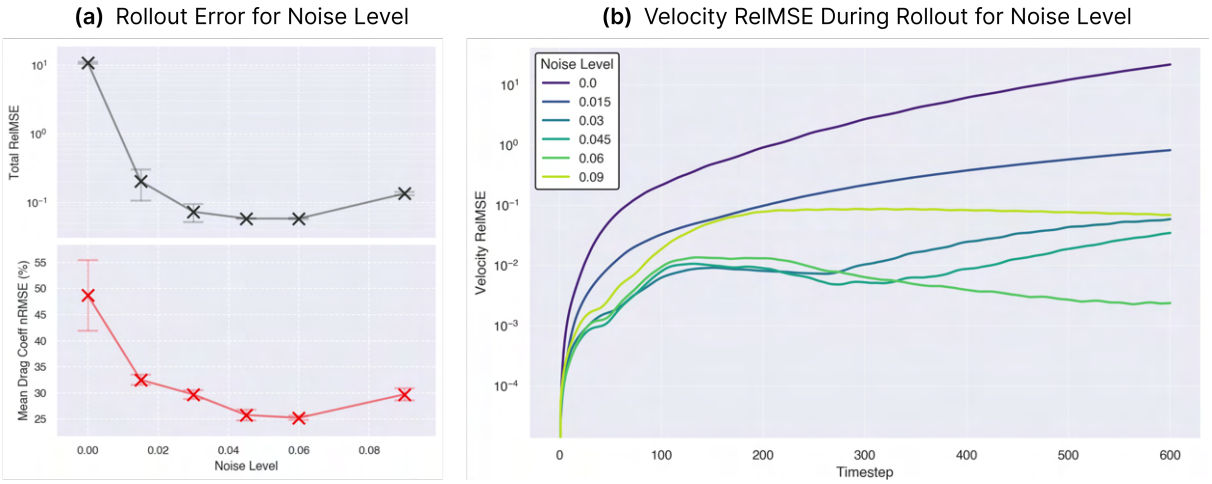


Figure 9.6: **(a)** Validation metrics across noise levels. The optimal noise level is observed near 0.06, where the total RelMSE (mean of velocity and pressure errors) and the mean drag coefficient nRMSE reach their minima. **(b)** Time evolution of velocity RelMSE for different noise levels. For $\eta \leq 0.06$, increasing noise dampens long-term error growth after the initial transient. At 0.09, the error reaches a stable – but higher – value, reflecting an incorrect learned distribution.

and improved stability up to $\eta = 0.06$. This was the selected noise level.

Beyond $\eta \approx 0.06$, performance degraded because the Gaussian perturbations injected into node states reached amplitudes comparable to or exceeding the characteristic neighbour-to-neighbour velocity differences $(u_i - u_j)$ or timestep-to-timestep changes $(u^{t+1} - u^t)$. This reduced the local signal-to-noise ratio to $\lesssim \mathcal{O}(1)$. As Gaussian noise does not respect the spatial–temporal correlations or conservation laws of the flow, the model increasingly learned to fit unphysical variability rather than true dynamics, leading to poor rollout accuracy.

10. Conservative Model Results

This section presents the development and evaluation of new conservative architectures. It is assumed that the models considered here are sufficiently similar in capacity and design to the baseline FVGN that consistent hyperparameters can be used across all architectures (Table 9.1). However, it is acknowledged that marginal performance gains may be obtained with architecture-specific tuning.

10.1 Vertex Potential

As an intermediate step to a full Vertex Potential (VertPot) model, a model called FVGN-Flux was implemented that predicts face fluxes directly, rather than constructing them from a vertex potential. A loss term supervises face flux predictions using the OpenFOAM ground truth. The purpose of this model is to demonstrate the ability to learn by predicting values for the integration step given by Equation 8.2. While the training loss curves appeared promising (Figure 10.1), rollout error with respect to the CFD ground truth was large, prompting further investigation.

Examining the evolution of model parameters during training pointed to a failure mode linked to `BatchNorm` (see Appendix A.2 for details) in the integration layer. `BatchNorm`'s learnable scaling parameter was suppressing the convective term by a factor of 1×10^{-5} (see Figure 10.1b). As a result, the cell velocity update from the integration step only depended on p_f and Q_f, with the convective dynamics – the dominant physics at $Re \sim 100$ – effectively bypassed. Since $\mathbf{u}_f$ and F_f were free to be optimised on their respective losses, independently of any constraint from the integration layer, the model gave the illusion of learning progress while in reality undermining physical consistency [72].

This issue likely originates from using the product of two predicted quantities ($\mathbf{u}_f F_f$) in the convective term, a formulation prone to high variance and weak gradients as errors compound multiplicatively. As a result, the model learns to suppress this term through

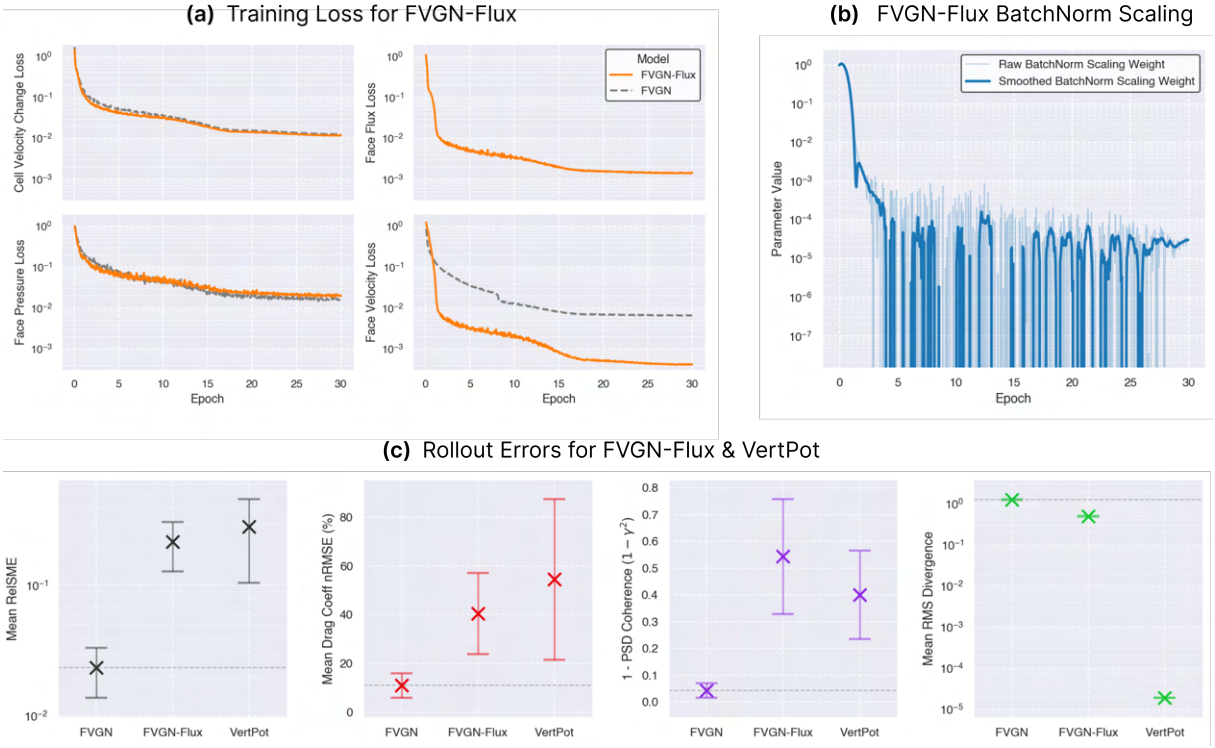


Figure 10.1: **(a)** Comparison of FVGN-Flux and FVGN training losses for cell-centre velocity, pressure, face velocity, and face flux. Cell-centre velocity and face pressure losses (left) remain similar between models, while face velocity and face flux losses (right) converge more rapidly for FVGN-Flux. This behaviour arises because the `BatchNorm` scaling minimises u_f and F_f in the integration step, reducing the constraint from the integration loss. **(b)** Training dynamics of the `BatchNorm` scaling parameter. A sharp reduction aligns with faster convergence of face velocity and flux losses. Despite low values, large fluctuations persist even after smoothing, indicating instability in the convective term. **(c)** Time-averaged rollout errors across example metrics. FVGN-Flux and VertPot show significantly reduced rollout accuracy compared to FVGN. The divergence plot (right) confirms that VertPot enforces a strict divergence-free constraint.

the `BatchNorm` scaling parameter, ignoring the convective dynamic predictions and relying instead on more stable terms formed from p_f and Q_f.

This suppression of the convective term can be prevented by (i) clipping the learnable weight at a minimum value or (ii) moving the integration into physical space where the `BatchNorm` scaling is not required. The latter performed better empirically, likely due to the added physical fidelity of integrating with the physical momentum equation. However, this approach did not resolve the high variance arising from the product of two predictions, and FVGN-Flux rollout RelMSE was still an order of magnitude higher than FVGN.

Ultimately, as shown in Figure 10.1c, extending FVGN-Flux to VertPot degraded predictive performance further, despite successfully enforcing a hard divergence-free constraint: the mean RMS divergence was 1.15×10^{-5}, four order of magnitude lower than that of the CFD ground truth data (1.28×10^{-1}). The potential-based flux construction

only deepens the computational graph, weakening gradient flow and compounding existing optimisation challenges. As a result, the model satisfies local conservation by design but fails to translate this advantage into improved rollout performance due to training difficulty.

10.2 Streamfunction Construction

Unlike the original FVGN, this model uses a cell-centred prediction strategy instead of face-centred updates. Therefore, as a first step, an intermediate architecture was tested to confirm that (i) cell-centre predictions and (ii) direct next-step velocity outputs, rather than increments, can be trained effectively on the dataset. This model is referred to as FVGN-Direct.

Empirically, FVGN-Direct's rollout accuracy was poor compared to FVGN, with a RelMSE 4.3 times higher. However, the rollout was stable, and physical enough to suggest this approach of direct prediction is viable. The inferior accuracy is not surprising: the increment-learning approach, used in FVGN, maintains a direct path from u^t to u^{t+1}, which aids optimisation by improving gradient flow, keeping the learning problem better conditioned, and stabilising autoregressive rollouts [73]. However, direct prediction is necessary here because divergence-free constraints can be imposed cleanly on u^{t+1} itself, whereas increment-based updates do not inherently guarantee that the resulting field remains divergence-free.

Validating the Divergence-Free Construction

The next step was to validate whether the MLS gradient calculation method could accurately reconstruct the velocity field from the streamfunction on an unstructured mesh. The Taylor–Green vortex system was used for this validation, as it admits an analytical solution. A regular 64^2 mesh was tested first, with derivatives of the analytical streamfunction computed using the MLS method and compared to the exact analytical solution. The test was repeated on an unstructured triangular mesh of 7500 cells, chosen to match the cell count, and emulate the refinement pattern, of the *EllipseFlow* meshes.

Figure 10.2a shows that the analytical velocity fields are reproduced accurately on both meshes. Accuracy on the structured grid was approximately four times higher, since each cell has a uniform distribution of neighbours; the error rises rapidly near boundaries due an anisotropic neighbour distribution. On the unstructured mesh, error patterns reflect mesh resolution (Figure 10.2c). Coarse zones accumulate larger errors due to

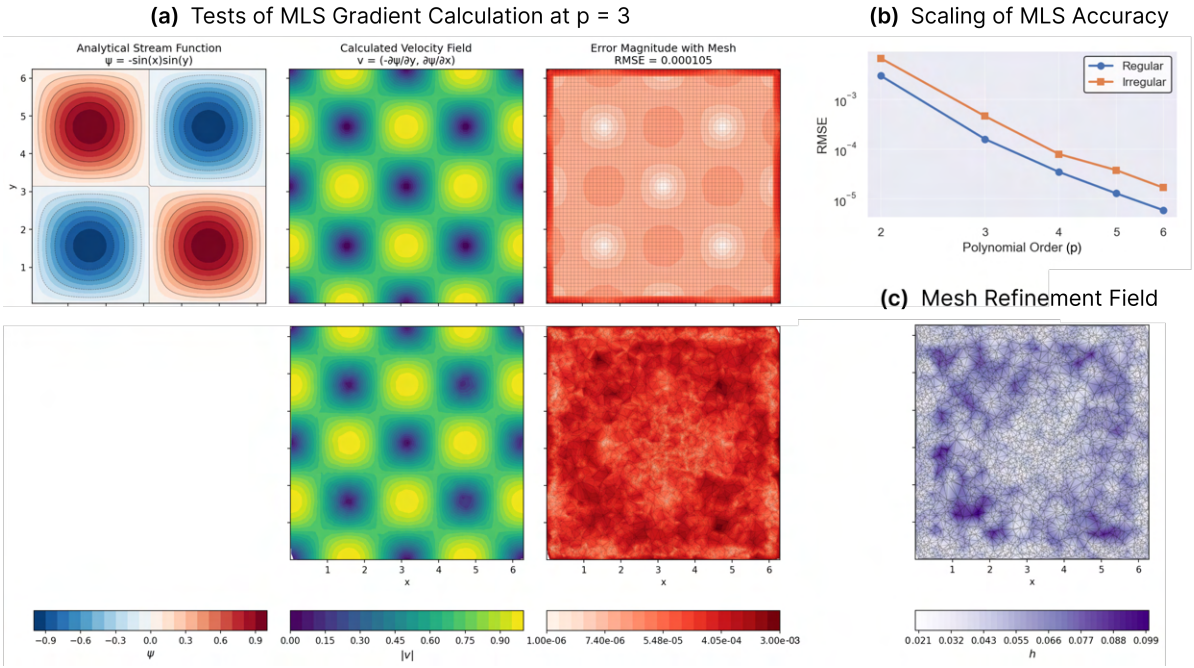


Figure 10.2: Validation of the MLS gradient calculation method using the Taylor–Green vortex, which admits an analytical streamfunction $\psi = -\sin(x)\sin(y)e^{-2\nu t}$ ($t = 0$ is used for this test). **(a)** Comparison between the analytical streamfunction (left), MLS-computed gradient magnitude (middle), and pointwise error magnitude overlaid with the computational mesh (right) for a regular mesh (top row) and an irregular triangular mesh (bottom row). Both meshes accurately reproduce the analytical gradients, with the regular mesh achieving lower RMSE and boundary-localised errors. **(b)** Scaling of RMSE with polynomial order p for regular and irregular meshes, showing near-exponential convergence consistent with MLS theory [74]. **(c)** Mesh refinement field of irregular mesh (darker = coarser). The darker "ring" of coarser mesh spacing is spatially similar to the ring of higher errors visible in the adjacent error plot (bottom right) in panel (a) .

increased neighbour spacing leading to weaker gradient estimation. Unlike the structured grid, boundary errors are less pronounced since local refinement near the boundaries counteracts the increasing anisotropy.

Furthermore, the theoretically-predicted exponential convergence of error with increasing polynomial order is observed for both regular and irregular cases (Figure 10.2b) [74]. This is further evidence that the MLS implementation is stable, and can reliably compute gradients on the *EllipseFlow* domain.

Next, the reconstructed velocity field was compared to the CFD "ground truth" solution shown in Figure 10.3. These results highlight a challenge: while the reconstruction is divergence-free (up to the accuracy of the gradient calculation), the CFD velocity field is not strictly divergence-free, since in finite-volume methods the divergence-free property is defined in terms of fluxes. As a result, the learning target is displaced from the divergence-free manifold. Even with a perfect gradient operator, a residual error will always exist

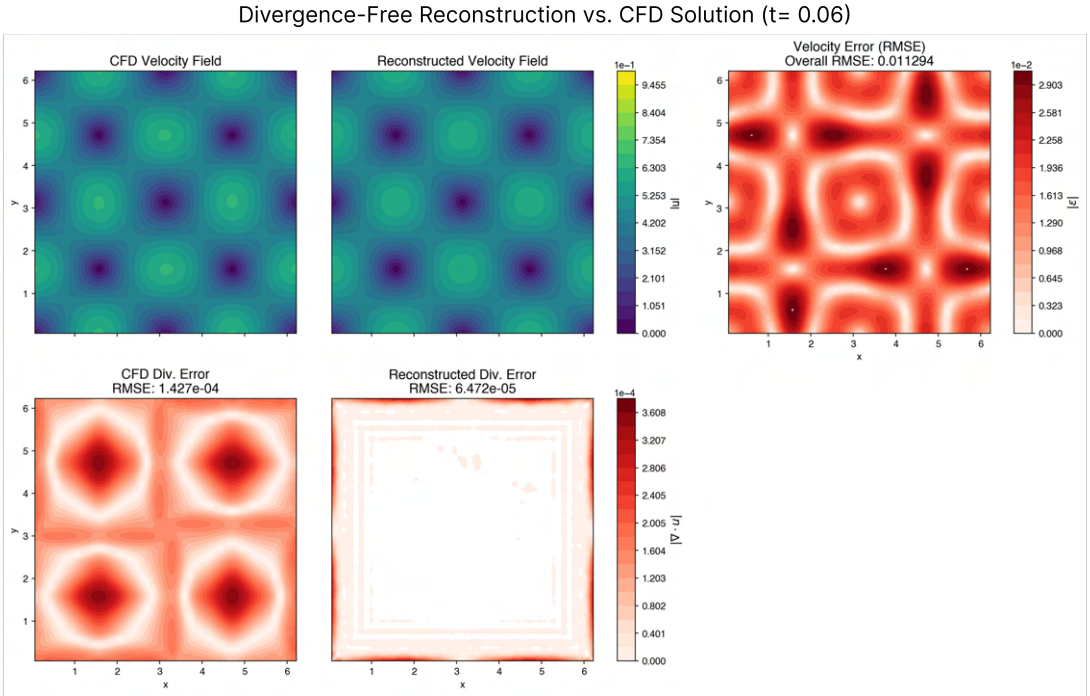


Figure 10.3: Comparison of OpenFoam solution and constructed velocity field (from the analytical stream-function using MLS) at viscous time $t_\nu = \nu t/L^2 = 0.06$ where $L = 2\pi$ and $\nu = 10^{-3}$.

relative to the ground truth. This error corresponds to the irrotational component of the CFD field and cannot be eliminated through training: the optimisation gradient therefore contains an off-manifold component, which may slow or prevent convergence.

Training with Taylor Green Vortex

To evaluate the model's learning capacity, we trained on a single Taylor–Green trajectory of 8 timesteps. Larger timesteps were used so that $|u^{t+\Delta t} - u^t|$ exceeded the noise from divergence-free mismatches. The model was trained for 2000 epochs.

Figure 10.4a shows that StreamFunc's training loss is about three orders of magnitude higher than FVGN-Direct. Its rollouts diverge after only a few steps (Figure 10.4b), whereas FVGN-Direct errors grow slowly. At the first timestep, StreamFunc already underestimates the velocity field (Figure 10.4c). This suggests that enforcing a divergence-free velocity field prevents the model from locally correcting peak errors. Instead, the optimiser reduces loss by globally damping velocity magnitudes. The unresolved local discrepancies manifest as high-frequency noise, which the gradient calculation amplifies.

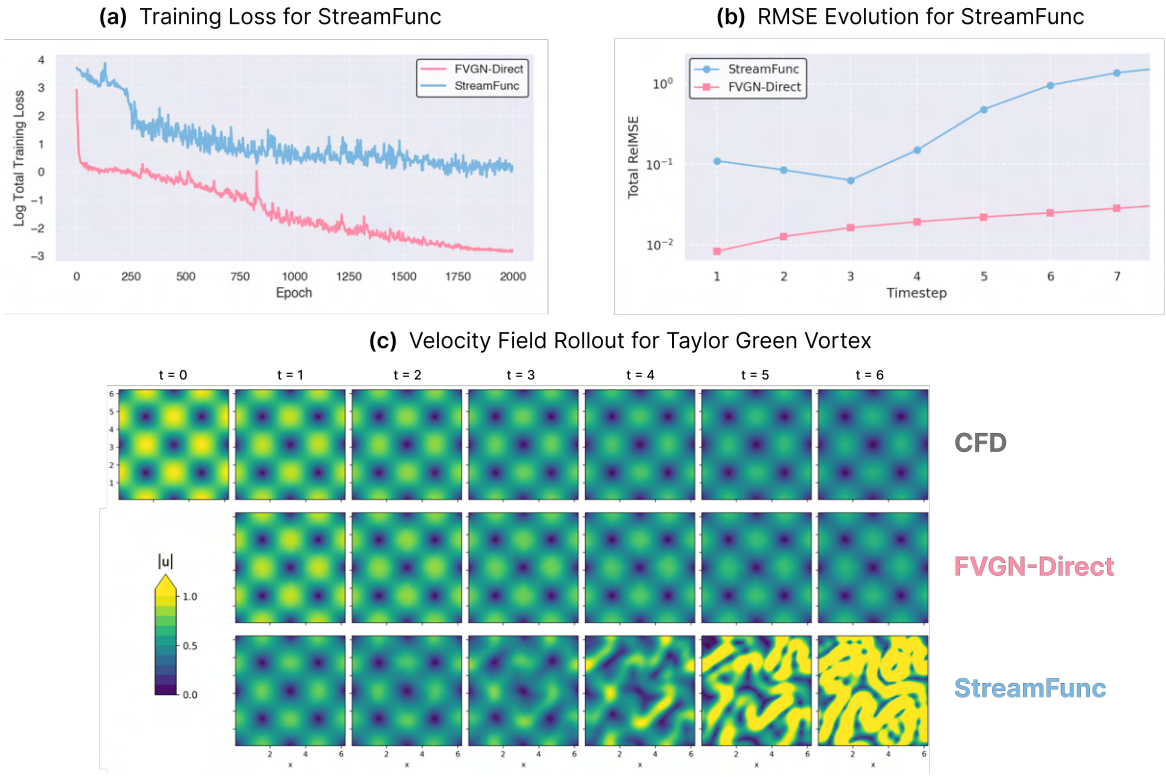


Figure 10.4: **(a)** Training loss for StreamFunc when overfitting to a single Taylor–Green trajectory. Loss values are three orders of magnitude larger than for FVGN-Direct, indicating difficulty in enforcing incompressibility while matching the ground truth. **(b)** RMSE growth over rollout timesteps. StreamFunc errors grow rapidly beyond $t = 4$, whereas FVGN-Direct grows slowly. **(c)** Rollouts on Taylor–Green flow ($t = 0$ to $t = 5$) for ground truth, FVGN-Direct, and StreamFunc. StreamFunc underestimates velocity magnitude from $t = 1$ onward, suppressing overall scale to minimise loss. Localised high-frequency noise, amplified by the gradient layer, likely causes the blow-up.

In Fourier space, the rotated gradient is,

$$\widehat{\mathbf{u}}(\mathbf{k}, t) = i\,\mathbf{k}^{\perp}\,\psi(\mathbf{k}, t),$$

where $\psi(\mathbf{k}, t)$ is the spectral streamfunction and $\mathbf{k}^{\perp} = (k_y, -k_x)$. Since each Fourier mode is weighted by $|\mathbf{k}|$, high-wavenumber errors in ψ amplify disproportionately in $\mathbf{u}$, making the system stiff and leading to the high-frequency artifacts observed in the final rollout frames.

Training with LaminarEllipse

Given the significant training challenges encountered with StreamFunc on the Taylor Vortex system, a full training run on *LaminarEllipse* was not pursued. However, the model was trained for a few epochs to verify that these difficulties persisted on a larger dataset:

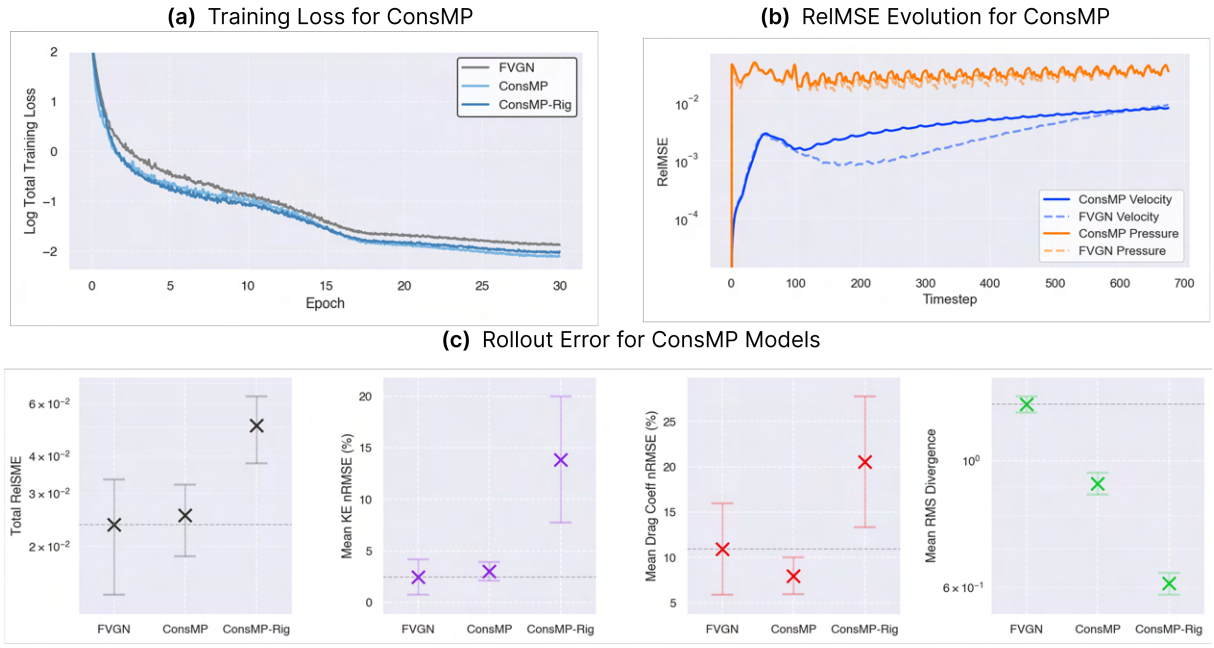


Figure 10.5: **(a)** Training curves showing slightly faster convergence for ConsMP models, with minimal difference between ConsMP and ConsMP-Rig. **(b)** Evolution of RelMSE for velocity and pressure for ConsMP (solid) and FVGN (dashed). ConsMP exhibits slower error growth during the periodic oscillation regime, although it handles the transient start-up less effectively. **(c)** Performance comparison of FVGN, ConsMP, and ConsMP-Rig across key metrics. ConsMP matches FVGN on mean RelMSE and kinetic energy, slightly outperforms in drag coefficient prediction, and achieves lower divergence.

the same high-frequency artifacts emerged in the predicted fields. This confirmed that – despite accurate gradient calculations on unstructured meshes – the StreamFunc's formulation inherently amplifies small-scale errors, making it unsuitable for stable simulations of this system.

10.3 Conservative Message Passing

The performance of both ConsMP, ensuring simple $m_{ij} = -m_{ji}$ for edge messages, and Cons-Rig, extending ConsMP to rigorously preserve symmetry throughout message-passing, are analysed in this section.

Starting with ConsMP, Figure 10.5a shows that ConsMP achieves lower training loss than FVGN, indicating that conservation-inspired message passing introduces beneficial inductive bias for learning. This improved training translates to better rollout performance: ConsMP maintains stable velocity and pressure fields consistent with CFD solution, extending beyond its 400 training steps without instability (see Figure 10.6). The RelMSE evolution (Figure 10.5b) confirms this stability – while ConsMP exhibits higher

initial errors during transient start-up, its error level remains more stable throughout the periodic oscillation regime, whereas FVGN shows faster error accumulation. With respect to global metrics of accuracy and physical consistency, shown in Figure 10.5c, ConsMP also performs well: it matches FVGN in mean RelMSE and kinetic energy; marginally outperforms it in drag coefficients; and yields lower divergence, evidence that conservation-inspired message passing produces more conservative predictions.

While ConsMP-Rig has almost indistinguishable training performance from ConsMP, its rollout stability is noticeably worse, as reflected in its poor validation metrics (Figure 10.5c). It does achieve the lowest divergence error, indicating that its more rigorous conservation enforcement has been impactful, yet, this has not translated to better predictions. A possible explanation is that the additional antisymmetric feature constraints may heighten sensitivity to perturbations, making the model more reliant on maintaining precise feature interactions. These interactions can degrade under distribution shift or when noisy intermediate states arise during rollout, allowing small errors to compound and likely contributing to the poorer long-term performance. Given this context, ConsMP represents a practical compromise – balancing the flexibility of FVGN with the physical fidelity of stricter conservation enforcement – leading to stable and accurate rollouts.

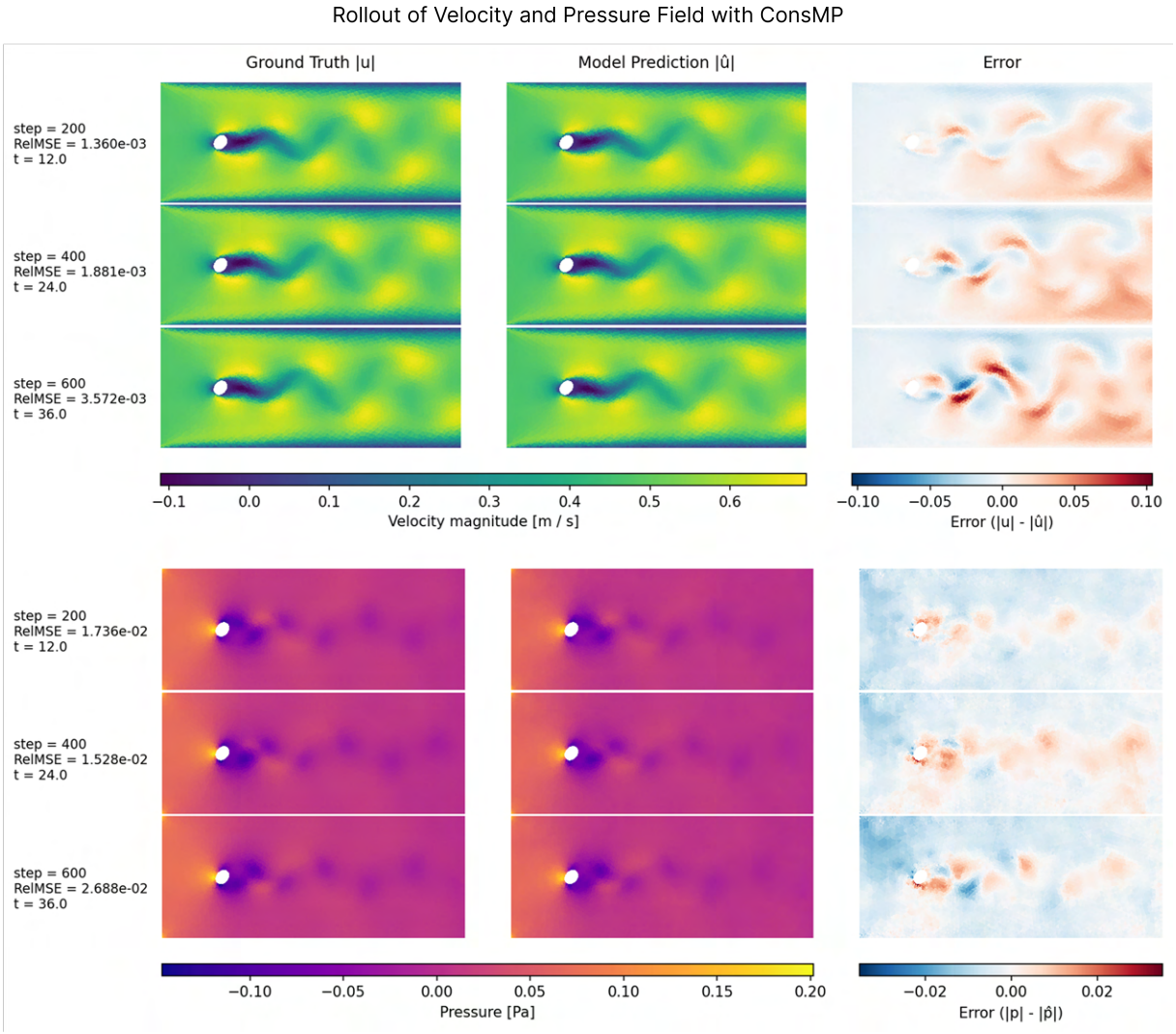


Figure 10.6: Snapshots of ground truth and ConsMP predictions for velocity and pressure fields at timesteps 200, 400, and 600 from simulation rollout. The model maintains stable and produces physically plausible fields beyond its 400-step training horizon without blow-up. Note that pressure is defined relative to zero pressure at the outlet.

11. Further Results

Following the pursuit of conservation-related architectural modifications, alternative adjustments to the baseline model and training setup were explored as potential avenues for improvement. Several strategies were tested – including input feature engineering, temporal bundling, pushforward methods, and normalisation – but the discussion here focuses on a few representative techniques. Additionally, results from evaluating and comparing several models on out-of-distribution geometries are presented to examine generalisation.

11.1 Physical Integration

One limitation of the original FVGN architecture is that it computes the integration step (Equation 4.3) in normalised space. This breaks physical consistency, as the integration no longer meaningfully represents the momentum equation. In contrast, performing the integration in real space would correspond to directly solving the momentum equation and may yield better alignment with physical dynamics.

The main challenge for this approach is the denormalisation of the diffusion term Q_f: the dataset lacks explicit Q_f values, preventing the use of dataset-wide statistics that were applied to other outputs. This motivates three alternative strategies, which we refer to as PhysIntA, PhysIntB, and PhysIntC:

- (PhysIntA) Compute $Q_f = \nu \nabla \mathbf{u}_f \cdot \mathbf{n}_f$ directly, using Moving Least Squares (MLS) to approximate $\nabla \mathbf{u}_f$ from predicted u_f values. Predictions for $\mathbf{u}_f$ and p_f can still be denormalised using z-score statistics.
- (PhysIntB) Apply a dimensionless denormalisation framework, under the assumption the model outputs are dimensionless. For example, if we define characteristic length L and velocity U: $\mathbf{u}_f$ is scaled by U, p_f by ρU^2 and Q_f by $U^2 L^2$.
- (PhysIntC) Introduce learnable parameters for denormalisation for all face-level quantities, including Q_f, enabling the model to discover an appropriate scaling.

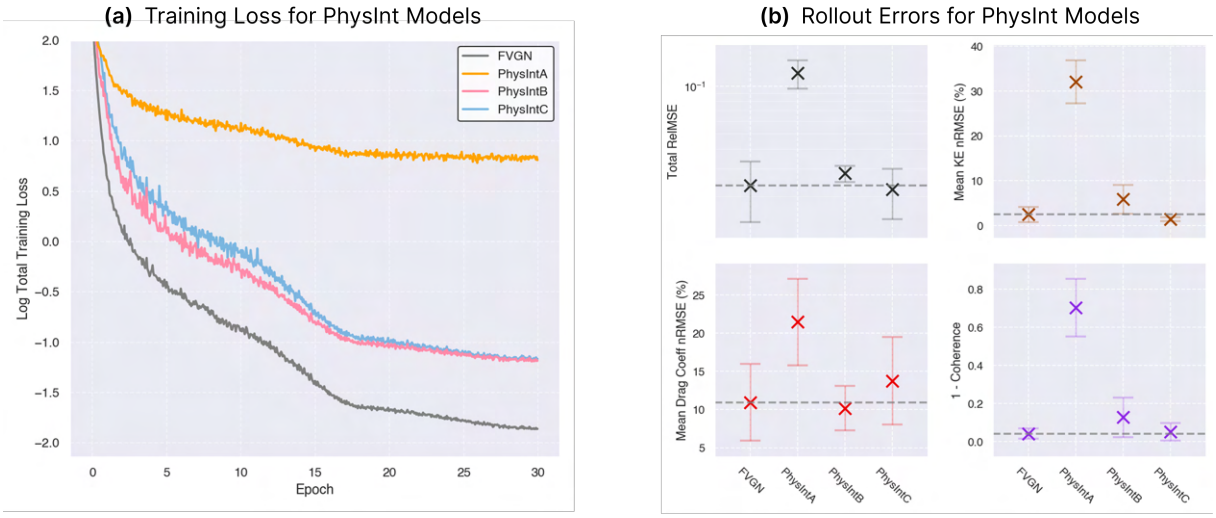


Figure 11.1: **(a)** Training curves for different PhysInt models, compared to FVGN. **(b)** Rollout errors. Overall, PhysIntC appears well matched, if not better, with FVGN for most metrics (not all shown here). PhysIntB also performs well on metrics such as the drag coefficient prediction. PhysIntA produces highly unphysical predictions, shown by large error to ground truth on these metrics.

The integration in physical space is expected to hinder convergence because it introduces numerical errors that are amplified during re-normalisation. This effect is particularly severe for variables with low variance, where the signal-to-noise ratio is inherently poor. These perturbations propagate through the loss function, increasing prediction noise, destabilising gradients, and slowing optimisation. Figure 11.1a illustrates this effect, showing that PhysInt models have higher training loss than FVGN. PhysIntA trains especially poorly, likely due to additional noise from the MLS-based gradient estimation.

Despite this, a modest level of noise can improve rollout stability and overall performance [11]. Consistent with this, PhysIntC achieves performance comparable to, and sometimes better than, FVGN across several metrics, as shown in Figure 11.1b. While PhysIntB is theoretically the most appealing due to its scale-invariant normalisation, it does not consistently perform as well across the metrics. By contrast, PhysIntA shows pronounced underperformance, likely because excessive noise in MLS-based gradient estimates prevented convergence. These results reinforce a central theme of this work: balancing physical fidelity with learning flexibility is critical for surrogate model performance.

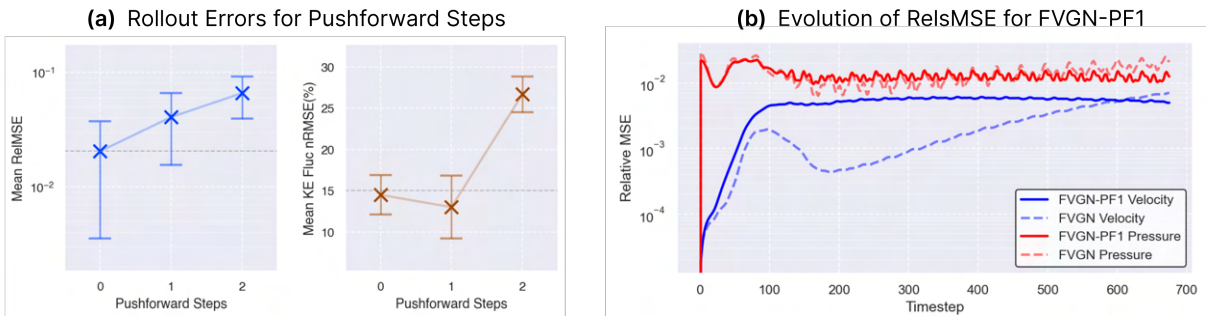


Figure 11.2: **(a)** Increasing pushforward steps during training leads to higher rollout RelMSE but improved preservation of global dynamics, as shown by lower nRMSE in fluctuation kinetic energy. **(b)** Relative MSE over time shows that pushforward (solid lines) yields stable errors during the periodic regime compared to the baseline (dashed lines), despite degraded accuracy during the initial transient regime. This trade-off reflects the denoising effect of pushforward training, which suppresses high-frequency errors in favour of long-term stability.

11.2 Pushforward Training

Introduced by Brandstetter [62], the "pushforward" technique perturbs training inputs using the model's own prediction errors. Let the model $\hat{f}_\theta$ estimate the next state from an input u_t, yielding $\hat{u}t+1 = \hat{f}_\theta(u_t)$. This prediction is then used as input to compute $\hat{u}^{t+2} = \hat{f}_\theta(\hat{u}^{t+1})$, and the loss is computed with the target at that timestep. Gradients are cut at $\hat{u}^{t+1}$, so backpropagation flows only through the final step. The model thus learns to make predictions from inputs which have noise injected through its own rollout. Increasing the number of pushforward steps means chaining additional forward passes – e.g., using $\hat{u}^{t+k} = \hat{f}\theta(\hat{u}^{t+k-1})$ – before computing the loss at $t+k+1$, and cutting gradients at $t+k$. This injects stronger, temporally accumulated model error into the training inputs. For simplicity, this model is referred to as FVGN-PFx, where x is the number of pushforward steps used.

Empirically, a single pushforward (FVGN-PF1) step improves rollout stability but reduces pointwise prediction accuracy. As shown in Figure 11.2a, the relative mean squared error (RelMSE) increases with the number of pushforward steps. However, for phase-independent metrics such as the turbulent kinetic energy error, FVGN-PF1 matches or slightly outperforms FVGN, indicating better preservation of global physical properties.

These findings are supported by examining temporal evolution of RelMSE, an example of which is shown in Figure 11.2b. While FVGN-PF1 yields less accurate predictions during the initial transient start-up, it quickly stabilises to a steady-state error. In contrast, the baseline model accumulates error gradually across both velocity and pressure fields

throughout the rollout. Hence, while the global mean RelMSE suggests that pushforward performs worse overall, this metric masks the improved stability.

An explanation for poor initial predictions during rollout can be attributed to the increased denoising property of the model: it is trained on more strongly perturbed inputs, it increasingly suppresses small-scale, high-frequency variations, effectively acting as a low-pass filter over the dynamics. This results in a loss of fine-grained accuracy – particularly relevant during the early transient phase – in exchange for improved long-term stability and robustness. As discussed in Section 9.2.4, if too much noise is added the denoising is too strong, leading to performance degradation; this is observed for FVGN-PF2 (see Figure 11.2a).

Finally, the computational overhead introduced by pushforward steps during training is considered. Each pushforward step adds an extra forward pass per training sample; however, since backpropagation – the dominant cost in each training iteration – is only applied to the final prediction, the total increase in training time is modest, around 20% for adding a single pushforward step.

11.3 Model Comparisons

In this section, we compare the performance of the three best models developed in this study against the baseline FVGN:

- ConsMP: conservative message passing with antisymmetric messages between adjacent cells.
- PhysIntC: physical-space integration step with flux terms denormalised using learnable scale factors.
- FVGN-PF1: vanilla FVGN trained with a single pushforward step.

Figure 11.3 presents the predicted x-velocity fields at timestep 600 for a geometry from the est dataset. The predictions from all models appear visually similar to the ground truth, aside from phase discrepancies, and recover the correct shedding regime and scale.

The differences are captured quantitatively in Table 11.1, which reports averages over both validation and test datasets and thus covers both in- and out-of-distribution cases. Although FVGN attains lower RelMSE, this metric is sensitive to phase shifts and does not reliably reflect physical fidelity. The models developed in this study show advantages for predictions of physical quantities such as drag and lift coefficients, which more directly

Table 11.1: Error metrics for models in this study compared with FVGN. Reported are RelMSE for velocity and pressure, RMS divergence, and standard CFD quantities: fluctuation kinetic energy, Strouhal number, mean drag coefficient, and RMS lift coefficient. The uncertainty reflects the standard error in the mean.

Model	Vel RelMSE ($\times 10^2$)	Press RelMSE ($\times 10^2$)	Div RMS ($\times 10^0$)	Fluc KE nRMSE (%)	Strouhal nRMSE (%)	Drag nRMSE (%)	Lift RMS nRMSE (%)
FVGN	**2.3±0.7**	**5.8±1.5**	6.3±1.0	30.2±10.6	2.5±1.3	15.8±4.1	37.3±8.0
ConsMP	3.6±1.0	7.1±1.8	4.7±0.8	25.7±5.5	3.4±1.6	**13.8±3.6**	49.3±13.7
PhysIntC	4.2±1.6	8.4±2.3	**2.9±0.4**	**23.2±7.3**	**2.4±1.5**	20.5±6.1	53.9±9.9
FVGN-PF1	4.2±1.4	9.4±2.8	4.5±0.9	32.0±13.7	7.5±3.1	19.0±5.3	**28.8±6.2**

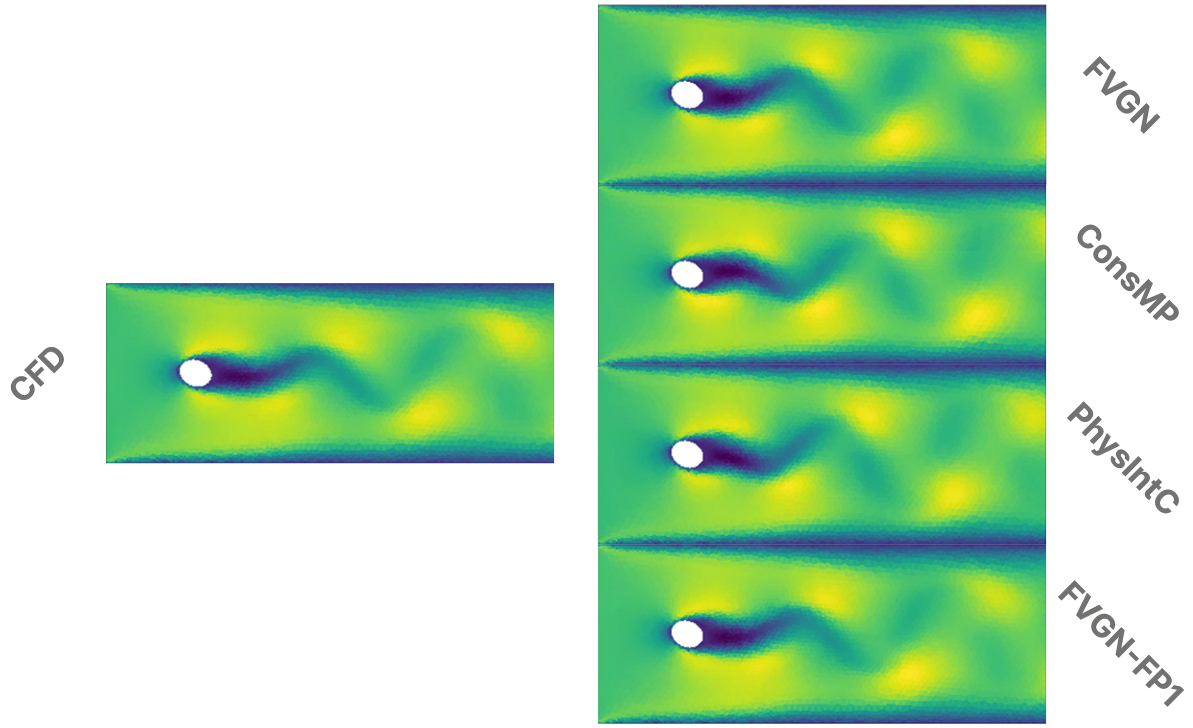


Figure 11.3: Comparison of x-velocity fields at timestep 600 with CFD ground truth for a simulation from the test dataset. The vortex shedding regime is captured well by all models, but, for this simulation, ConsMP and PhysIntC show phase offsets to the ground truth CFD solution.

capture the industrial utility of the simulations. However, the differences between the models generally aren't significant given the uncertainty for these values.

11.3.1 Computational Performance

Rolling out five simulations simultaneously on a single Nvidia A100 GPU with FVGN has a runtime of 38.5 ± 1.8 seconds. Since FVGN-FP1 employs the same architecture,

differing only in training, its rollout time was identical. ConsMP adds two extra message-passing steps (splitting and antisymmetric passing), increasing inference time by about 6%. Although PhysIntC's only modification is to move denormalisation to before the integration, this requires $\mathbf{u}_f$, p_f, and Q_f to be denormalised instead of only the velocity update $\Delta\mathbf{u}_c$, leading to an 8% increase in runtime relative to FVGN.

These differences are small relative to the runtime of the CFD ground truth: a single simulation on an Intel Xeon Platinum 8368Q CPU core had a mean runtime of approximately 300 seconds. Therefore, for five simulations, FVGN and FVGN-FP1 achieve a $\sim 40\times$ speedup, while ConsMP and PhysIntC reach $\sim 36\times$.

11.3.2 Out-of-Distribution Generalisation

Out-of-distribution (OOD) generalisation refers to a model's ability to maintain predictive accuracy when evaluated on data whose governing parameters differ from those seen during training. The variation in validation metrics with respect to the angle of attack (AoA), ellipse size, inlet velocity, and Reynolds number (Re) was analysed. No dependence on AoA was observed, indicating that the models did not overfit to specific geometries – the performance remains stable when extending from the 60° training range to larger angles of 80°. No clear pattern emerged for errors across ellipse size or inlet velocity individually. However, when these are combined into Re, clear and interpretable trends appear. This is expected, since Re determines the temporal and spatial scales of the flow.

Importantly, such trends are only visible when the evaluation metric reflects the underlying flow statistics rather than exact phase alignment. Metrics like mean drag coefficient demonstrate a clear rise in error as Re moves away from the training distribution, illustrating poor OOD generalisation. In contrast, RelMSE – being highly sensitive to exact phase and temporal alignment – fails to show meaningful trends or rank model performance reliably. Figure 11.4 presents these results, with the shaded region indicating the training distribution of Re.

The variation of mean drag coefficient error, across different models provided an insight into the relative OOD generalisation capabilities. FVGN and FVGN-PF1 exhibit symmetric error profiles across Re, reflecting their low physical fidelity; these models primarily learn statistical correlations from the training data rather than underlying physical laws, so their accuracy depends mainly on whether Re is within the training range, with little sensitivity to the physical changes associated with different Re. In contrast, the physics-informed models (PhysIntC and ConsMP) display anisotropic accuracy. For PhysIntC, large errors at low Re likely arise from denormalisation amplifying prediction errors on

small velocity scales, making the system stiffer; at higher Re, the benefit of physically consistent integration dominates, providing an inductive bias that improves generalisation compared to FVGN. ConsMP shows the opposite trend – stronger generalisation at low Re but weaker at high Re. A possible explanation is that the enhanced message passing provides greater benefits when velocity gradients are weak and length scales are long, where effective use of the receptive field is critical. At high Re, gradients are sharp and all relevant information lies within the existing receptive field, so the advantage of refined message passing diminishes; instead, the superior single-step prediction of FVGN becomes more valuable for capturing high-frequency oscillations and hence the drag.

Finally, the dimensionless RMS divergence is considered, presented in bottom plot of Figure 11.4. Although this metric does provide significant insights into OOD generalisation, it reveals a consistent and somewhat unexpected ranking in divergence error. All modified models improved on FVGN. ConsMP's improved performance aligns with its conservation-based design, but FVGN-PF1's reduced divergence is less expected and may result from its stronger denoising effect, yielding smoother (though, not necessarily more accurate) predictions with lower measured divergence. PhysInt performs best, which is plausible since directly applying the physical momentum equation enforces a strong bias toward the true, divergence-free solution, embedding physical consistency more explicitly than the other approaches.

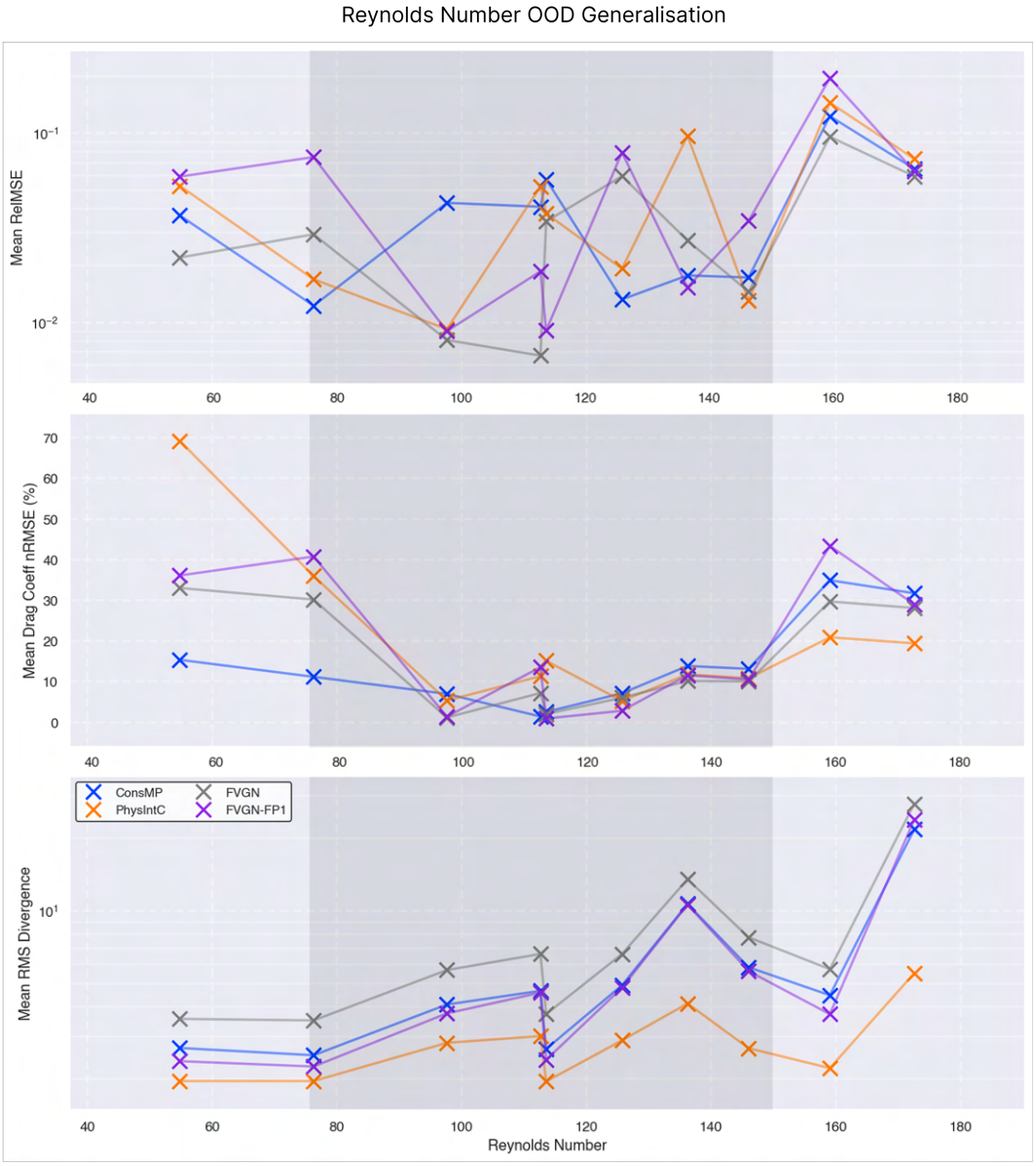


Figure 11.4: Model performance across Reynolds numbers (Re). Shaded region shows the training range. Top: RelMSE is noisy and phase-sensitive, obscuring generalisation trends. Middle: Drag coefficient nRMSE reveals anisotropic generalisation – PhysIntC improves at high Re, ConsMP at low Re, while FVGN variants degrade symmetrically. Bottom: dimensionless RMS divergence across Re.

12. Conclusion

This work developed graph neural network (GNNs) surrogate models for incompressible flow, with a focus on physics-inspired architectural changes, particularly the hard enforcement of the divergence-free continuity constraint. GNNs were selected for their ability to operate directly on unstructured meshes, maintaining geometric fidelity without interpolation. The central question was whether such constraints could improve stability and accuracy compared to an established baseline, Li et al.'s Finite-Volume-Graph-Network (FVGN) [20].

A complete development pipeline was built, including dataset generation with OpenFOAM, data preprocessing, and a distributed PyTorch/PyG training framework. The FVGN baseline was reproduced on the *CylinderFlow* benchmark [11], and then applied to custom datasets. This provided a reproducible foundation for comparing architectural and training modifications.

Three new conservative architectures were explored. The Vertex Potential model directly predicted face fluxes, but instability arose from the convective term ($\mathbf{u}_f \cdot F_f$) in Equation 8.2. Because this term multiplies two predicted quantities, even minor inaccuracies in $\mathbf{u}_f$ and F_f combined multiplicatively, yielding unstable updates and unreliable training dynamics. The Stream Function model relied on divergence-free reconstruction from a streamfunction via MLS gradients and showed strong validation, yet faced persistent rollout failure due to mismatch between its divergence-free predictions and the slightly non-divergence-free CFD targets. This off-manifold supervision introduced irreducible errors and the gradient formulation amplified high-frequency noise. Finally, the Conservative Message Passing model incorporated antisymmetric message updates and yielded modest improvements in metrics such as the drag coefficient prediction and divergence of velocity field. Its more physically-consistent variant, ConsMP-Rig, was measurably more conservative, but also more brittle, indicating that enforcing physical structure too rigidly can hinder robustness during rollout.

Non-architectural refinements generally showed greater promise. Pushforward training, where rollout-generated noise was injected into training inputs, improved robustness without changing the model structure. Changing the point of denormalisation to before the integration step ensured the momentum equation operated on dimensional variables, increasing physical consistency. Both techniques had trade-offs – pushforward could oversmooth dynamics, and early denormalisation could amplify errors – but, together, these models reinforced the conclusion that training strategy can have as much, if not more, influence on rollout performance as architectural design.

The GNN surrogates introduced in this study exhibit two principal limitations. First, physics-inspired architectural modifications can degrade optimisation by worsening problem conditioning. Deep learning models typically rely on well-conditioned gradients for effective training, but introducing elements such as strict antisymmetry or divergence-free constraints alters the geometry of the loss landscape in already highly non-linear, non-convex settings. This disrupts gradient flow, increases sensitivity to inputs – forming stiff systems – and results in unstable rollouts which must be robust to noisy inputs. Such degradation reflects a broader machine learning trade-off: strong inductive biases can enforce physical consistency but may also increase bias and fragility under noise or when the target distribution deviates from the imposed constraints. The second limitation is that the models show weak OOD generalisation across Reynolds numbers, suggesting an inability to learn scale-invariant dynamics. This undermines their effectiveness as CFD surrogates, as reliable deployment requires robustness across varying flow conditions – not merely interpolation within the training distribution, and their lack of stability remains a weakness compared with the robustness of traditional finite-volume methods.

Despite these limitations, surrogate models offer substantial advantages over many traditional CFD methods with respect computational speed, scalability to unstructured meshes without interpolation, and differentiability, which enables gradient-based optimisation and control tasks that are impractical with classical solvers. Within this study, speedups of up to 40$\times$ were obtained, validating the potential computational gains. In industry or research, these surrogates are most useful where approximate but rapid predictions are sufficient, such as parametric sweeps, design space exploration, and flow control, while high-fidelity CFD remains necessary for final verification and safety-critical applications.

Future work falls into three broad directions that aim to address these fundamental limitations and position GNN surrogates for more practical deployment in engineering workflows and novel applications:

- **Architectural refinement** could focus on balancing physical fidelity with optimisation robustness, seeking constrained architectures that can preserve favourable conditioning for the optimisation problem. Multi-scale message passing is a promising avenue, allowing the model to independently handle local and global constraints [75].
- **Dataset expansion** spanning wider flow conditions and geometries would better support generalisation and benchmarking. Scaling datasets to levels comparable with benchmarks like CylinderFlow enables more rigorous evaluation, yielding truer performance assessment and results that can be more reliably applied in industrial contexts.
- **Shifting beyond forward prediction** to exploit model differentiability for optimisation tasks – such as inverse design and closed-loop flow control – represents a promising longer-term application, using the surrogate not only for fast field prediction but also as an active tool for design and control.

In summary, this work developed, validated, and extended a class of physics-informed GNN surrogates for incompressible flow, integrating finite-volume principles with the aim of improving physical fidelity. While hard divergence-free constraints did not deliver the expected stability gains, they provided a clearer understanding of the interplay between constraint enforcement, optimisation behaviour, and model robustness. These insights establish a baseline for future attempts to design architectures that are both physically faithful and resilient, and highlight the potential for GNN surrogates to transform simulation and design once that balance is achieved.

Appendix A

Implementation Details

A.1 EllipseFlow Dataset

Flow Configuration. The elliptic cylinder is chosen over the canonical flow around a cylinder as it serves as a more application-relevant geometry, capturing lift and drag behaviours representative of a wider range of engineering structures and setups [76]. The characteristic length is defined as the major-axis diameter, following standard airfoil CFD conventions where the chord length is used. In contrast to circular cylinders, the laminar–turbulent transition for elliptic cylinders is sensitive to the angle of attack. For the aspect ratio of 0.8 used in this study, the transition Reynolds number decreases from $\mathrm{Re} \approx 60$ at 0° to $\mathrm{Re} \approx 40$ at 90° [77].

The training dataset spans Reynolds numbers from 75 to 150, while test cases extend this range down to 50 and up to 200. For circular cylinders, laminar–turbulent transition occurs near $Re \approx 250$, but elliptical bodies – particularly at elevated angles of attack – can transition earlier due to geometry-driven separation effects [78, 79]. By limiting the maximum Re to 200, all simulations remain in the stable, well-characterised Kármán vortex shedding regime [80], circumventing the substantial computational overhead associated with modelling turbulent or transitional flow phenomena.

OpenFOAM Solver Verification. We verified the OpenFOAM case using the Taylor–Green vortex with periodic boundaries [81] [1]. Figure A.1a reports the sensitivity of

[1] Taylor Green vortex details: square domain with size $[0, 2\pi]$; characteristic velocity of 1.0; periodic boundary conditions; and field solutions are given by,

$$
\begin{aligned}
u(x, y, t) &= \sin(x)\, \cos(y)\, e^{-2\nu t}, \\
v(x, y, t) &= -\cos(x)\, \sin(y)\, e^{-2\nu t}, \\
p(x, y, t) &= -\tfrac{1}{4}\big(\cos(2x) + \cos(2y)\big)\, e^{-4\nu t}.
\end{aligned}
$$

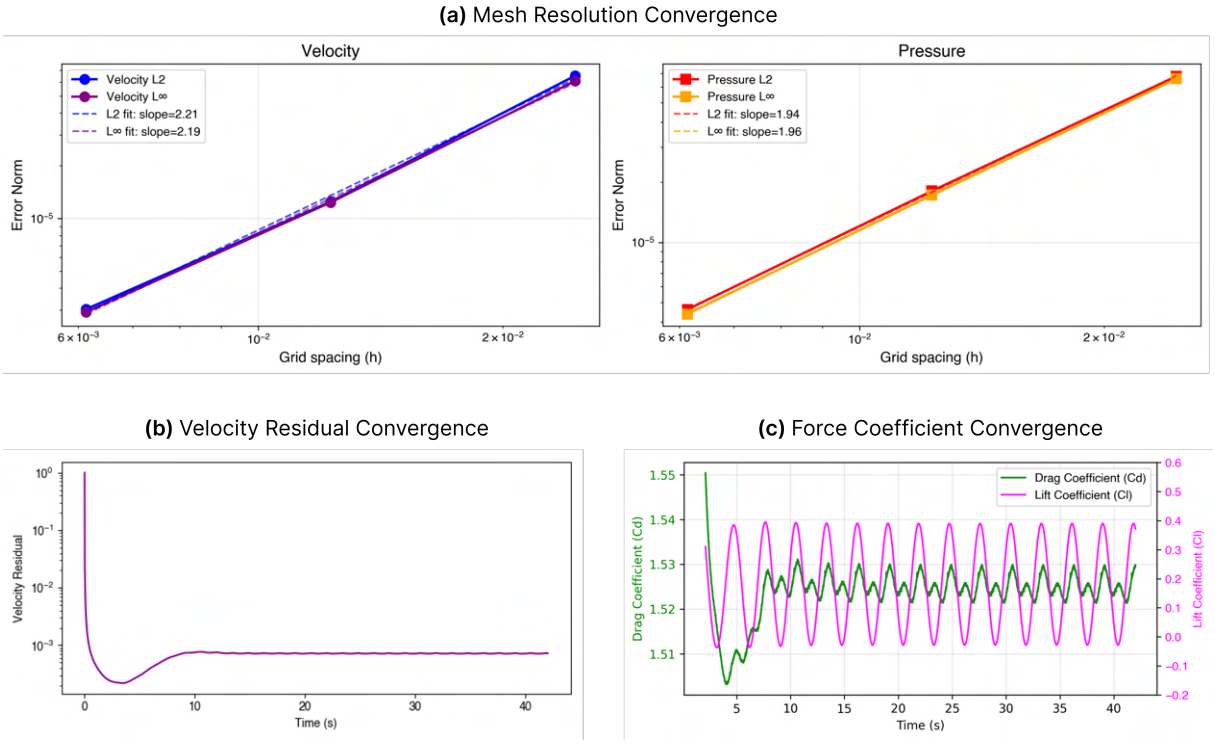


Figure A.1: **(a)** Convergence study for mesh resolutions $(1/h)$ of 256^2, 512^2 and 1024^2. Both $L2$ and $L\infty$ norms are plotted. Taylor Green vortex system is used with a final simulation time of 1 second, chosen to be significantly shorter than the viscous time scale of 500 seconds to avoid decaying amplitude obscuring the true solver error. **(b)** Time evolution of the velocity residual from the first solver iteration for a single geometry. After an initial transient decay, the residual stabilizes, indicating a statistically steady state characterised by fully developed periodic oscillations. **(c)** Evolution of lift and drag coefficients, calculated from integral of pressure of ellipse surface. Steady oscillations suggest a stable solution.

the L_2 and L_∞ norms of velocity and pressure to mesh resolution. The `pimpleFoam` solver was used with first-order Euler time stepping and predominantly second-order spatial discretisation (`Gauss linear` for divergence, `leastSquares` gradients, and `Gauss linear corrected` laplacians). A small timestep was chosen such that the temporal truncation errors were negligible relative to spatial discretisation errors. The apparent super-second-order convergence observed for velocity, even on a perturbed uniform grid, likely arises from partial cancellation of dominant error terms enabled by the highly smooth, symmetric, and periodic nature of the flow.

Mesh Resolution. Maximum y^+ from trial runs was analysed per geometry, and mesh spacing refined iteratively until the viscous sublayer criterion $y^+ \leq 1$ was met. The most restrictive case required $\Delta x \approx D/15$, with D as the ellipse major axis of the limiting geometry; this spacing was used for all simulations, yielding mean y^+ of 0.78 and 0.51 at the obstacle and domain walls. All meshes passed OpenFoam's `checkMesh` test of mesh

quality, which checks for detrimentally high skewness and non-orthogonality.

Timestep Selection. The transient `pimpleFoam` solver requires the timestep to satisfy the CFL stability criterion, with $C < 0.5$ widely used to ensure stability. Under the assumption that maximum velocity would be approximately twice the highest inlet velocity, a uniform timestep of $dt = 0.006$ was calculated. Courant numbers were monitored during runtime and remained consistently below this threshold $C < 0.5$. Note that our GNN surrogate, which is not subject to CFL constraints, is trained to predict every 10th timestep. Hence, throughout this study, a timestep refers to $dt = 0.06$.

Solution Verification and Validation. Numerical convergence was evaluated by tracking the initial residuals of all solved fields. As shown for the velocity residual in Figure A.1b, residuals decreased by several orders of magnitude before plateauing, indicating attainment of a statistically stationary state. Final residuals for each field remained below the prescribed solver tolerances at all time steps.

Physical validation was performed against established literature. The temporal evolution of the drag and lift coefficients (Figure A.1c) exhibited the periodic oscillations characteristic of this flow regime. Mean drag and lift coefficients agreed with reported values for flow over elliptic cylinders in comparable regimes [82]. A Fast Fourier Transform (FFT) of the lift coefficient signal yielded a shedding frequency f, from which the Strouhal number $St = fD/U$ was computed. The resulting range, 0.15–0.25, matched published values across various angles of attack [82].

A.2 Baseline FVGN Model

Each MLP, used in the encoder, message-passing blocks, and decoder, has two hidden layers of width 128 with SiLU activations and a linear output. All outputs, except in the decoder, are normalised to zero mean and unit variance for stability. With 15 message-passing blocks each with their own weights, FVGN contains 2.2 million parameters.

All input features undergo z-score normalisation (zero mean, unit variance). While this element-wise operation obscures the directionality of vector inputs, it mitigates input bias and conditions the optimisation problem – a worthwhile compromise for the highly anisotropic velocity fields in this study.

The integration layer uses normalised decoded outputs, with timestep, cell volume, and face area combined and passed through a `BatchNorm` layer (z-score normalisation over the batch with learnable scaling). This disrupts the strict physical relationships of

the momentum equation, making the step an approximate numerical stencil rather than a physically exact integration. The choice likely encourages implementation simplicity and learning efficiency; the viability of performing this step in physical space is discussed in Section 11.1.

Appendix B

Mathematical Derivations

B.1 Differential Forms and Divergence-Free Fields

This section derives the fundamental result that the rotated gradient of a scalar potential yields a divergence-free velocity field, using the language of differential forms. For a more comprehensive derivation, see [17].

A differential form generalizes functions and vector fields. In our context, 0-forms are scalar functions $f(x, y)$, 1-forms are vector fields $\mathbf{v} = v_x dx + v_y dy$, and 2-forms are area densities $\omega = f(x, y)\, dx \wedge dy$. The exterior derivative d acts on k-forms to produce $(k + 1)$-forms:

$$df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy \qquad \text{(gradient)} \tag{B.1}$$

$$d(v_x dx + v_y dy) = \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) dx \wedge dy \qquad \text{(curl in 2D)} \tag{B.2}$$

The key property of the exterior derivative is $d^2 = 0$.

The Hodge star operator $\star$ maps k-forms to $(n-k)$-forms. In 2D: $\star dx = dy$, $\star dy = -dx$, $\star(dx \wedge dy) = 1$, and $\star 1 = dx \wedge dy$. For a 1-form $v_x dx + v_y dy$, applying $\star$ yields:

$$\star(v_x dx + v_y dy) = v_x dy - v_y dx$$

This corresponds to the vector $(-v_y, v_x)$ – a 90° anticlockwise rotation of (v_x, v_y).

For any $(n - 2)$-form μ in n dimensions, the vector field $\mathbf{v} = \star d\mu$ is divergence-free. Since divergence equals $\star d \star$ applied to 1-forms:

$$\operatorname{div}(\star d\mu) = \star d \star (\star d\mu) = \pm \star d^2 \mu = 0$$

In 2D, let streamfunction ψ be a 0-form. Then:

$$d\psi = \frac{\partial \psi}{\partial x} dx + \frac{\partial \psi}{\partial y} dy$$

$$\star d\psi = \frac{\partial \psi}{\partial x} dy - \frac{\partial \psi}{\partial y} dx$$

This 1-form corresponds to velocity field $\mathbf{u} = \left(-\frac{\partial \psi}{\partial y}, \frac{\partial \psi}{\partial x}\right)$ which is the rotated gradient. Direct verification shows that this is indeed divergence-free:

$$\nabla \cdot \mathbf{u} = \frac{\partial}{\partial x}\left(-\frac{\partial \psi}{\partial y}\right) + \frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial x}\right) = 0$$

by equality of mixed partial derivatives.

Note that in 3D, the construction $\mathbf{B} = \star d\mathbf{A}$ for vector potential $\mathbf{A}$ yields the familiar identity $\mathbf{B} = \nabla \times \mathbf{A}$, with $\nabla \cdot \mathbf{B} = \nabla \cdot (\nabla \times \mathbf{A}) = 0$.

It follows that constructing a divergence-free field in n dimensions is as simple as predicting the components of μ and forming $\star d\mu$. In 2D, this is the single component of a scalar potential ψ, and in 3D it is the components of a vector potential A.

Richter–Powell et al. demonstrated that divergence-free constraints can be constructed for general conservation laws in augmented space–time [17]. A conservation law in n spatial dimensions,

$$\frac{\partial u}{\partial t} + \nabla_x \cdot \mathbf{f}(u) = 0, \tag{B.3}$$

can be written as the divergence-free condition

$$\nabla_{t,x} \cdot (u, \mathbf{f}(u)) = 0, \tag{B.4}$$

in $(n+1)$-dimensional space–time, where u is interpreted as flux in the temporal direction. Hence, a neural network only needs to predict components of μ in the $(n+1)$-dimensional space, and then obtain the divergence-free solution from $\star d\mu = (u, \mathbf{f}(u))$, where the time component gives the conserved quantity u and the spatial components give its flux $\mathbf{f}(u)$, ensuring the conservation law is satisfied automatically.

B.2 Moving Least Squares for Gradient Estimation

This section outlines the approach to estimate gradients on an unstructured mesh using the Moving Least Squares (MLS) method. For a detailed derivation, refer to [74].

The objective is to approximate $\nabla \mathbf{u}$ at a point $\mathbf{x}_0$ from scattered data samples by fitting a polynomial of degree p to nearby data points using weighted least squares.

The polynomial expansion around $\mathbf{x}_0$ is

$$\mathbf{u}(\mathbf{x}) \approx \sum_{|\alpha| \leq p} c_\alpha (\mathbf{x} - \mathbf{x}_0)^\alpha, \tag{B.5}$$

where α represents a multi-index for the polynomial terms. In d dimensions, this expansion contains $n_t = \binom{p+d}{d}$ terms. For example, in 2D with $p = 2$, there are $n_t = 6$ terms: constant, linear (x, y), and quadratic (x^2, xy, y^2).

To determine the coefficients c_α, the method minimizes the weighted residual over k neighbouring points:

$$J = \sum_{i=1}^{k} w_i \left\| \mathbf{u}_i - \sum_{|\alpha| \leq p} c_\alpha (\mathbf{x}_i - \mathbf{x}_0)^\alpha \right\|^2. \tag{B.6}$$

The number of neighbours k must exceed n_t to create an overdetermined system that provides stability against noise. For this implementation, $k = 2n_t$ is chosen.

The weights w_i give more influence to points closer to $\mathbf{x}_0$:

$$w_i = \frac{1}{(\|\mathbf{x}_i - \mathbf{x}_0\| + \epsilon)^2}, \tag{B.7}$$

where ϵ prevents division by zero. These weights are recomputed for each data point $\mathbf{x}_0$.

For implementation, define displacements $\boldsymbol{\delta}_i = \mathbf{x}_i - \mathbf{x}_0$ and construct the basis matrix $\mathbf{P}$ where row i contains all polynomial basis functions evaluated at $\boldsymbol{\delta}_i$. With diagonal weight matrix $\mathbf{W}$ containing $W_{ii} = w_i$,the least squares solution is found via the normal equations:

$$(\mathbf{P}^T \mathbf{W} \mathbf{P} + \epsilon \mathbf{I})\, \mathbf{c} = \mathbf{P}^T \mathbf{W}\, \mathbf{u}, \tag{B.8}$$

where $\epsilon \mathbf{I}$ provides regularisation for numerical stability.

The gradient at $\mathbf{x}_0$ is obtained by differentiating the polynomial. Since evaluation occurs at $\mathbf{x}_0$ where $\mathbf{x} - \mathbf{x}_0 = 0$, only the linear term coefficients contribute to the gradient. The gradient selection matrix

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & \cdots \end{bmatrix} \tag{B.9}$$

extracts these coefficients (e.g., in 2D, the first row selects the x-derivative coefficient).

Combining gradient extraction with the solution yields weights that directly map neighbour values to gradient components:

$$\mathbf{w}_{\text{grad}} = \mathbf{G}(\mathbf{P}^T\mathbf{W}\mathbf{P} + \epsilon\mathbf{I})^{-1}\mathbf{P}^T\mathbf{W}. \tag{B.10}$$

The final gradient estimate is

$$\nabla u(\mathbf{x}_0) = \mathbf{w}_{\text{grad}}\, \mathbf{u}_{\text{neighbour}}. \tag{B.11}$$

For each data point (in our case, the cell or face centres), the method requires constructing and solving a small $n_t \times n_t$ linear system. While weights w_i and matrix $\mathbf{P}$ must be recomputed for each $\mathbf{x}_0$, applying $\mathbf{w}_{\text{grad}}$ to neighbour values is just a matrix-vector multiplication. Also note that since the mesh geometry is fixed for each simulation these weights can be *precomputed* before training. The method exactly reproduces gradients of polynomials up to degree p.

Bibliography

[1] Hans Petter Langtangen, Kent-Andre Mardal, and Ragnar Winther. "Numerical Methods for Incompressible Viscous Flow". In: *Advances in Water Resources* 25.8–12 (Aug. 2002), pp. 1125–1146. ISSN: 03091708. DOI: 10.1016/S0309-1708(02)00052-0.

[2] Dochan Kwak, Cetin Kiris, and Chang Sung Kim. "Computational Challenges of Viscous Incompressible Flows". In: *Computers & Fluids* 34.3 (Mar. 2005), pp. 283–299. ISSN: 00457930. DOI: 10.1016/j.compfluid.2004.05.008.

[3] Dochan Kwak and Cetin Kiris. "CFD for Incompressible Flows at NASA Ames". In: *Computers & Fluids* 38.3 (Mar. 2009), pp. 504–510. ISSN: 00457930. DOI: 10.1016/j.compfluid.2008.06.010.

[4] Michael Schäfer. *Computational Engineering: Introduction to Numerical Methods*. Berlin New York: Springer, 2006. ISBN: 978-3-540-30686-3.

[5] R.I Issa. "Solution of the Implicitly Discretised Fluid Flow Equations by Operator-Splitting". In: *Journal of Computational Physics* 62.1 (Jan. 1986), pp. 40–65. ISSN: 00219991. DOI: 10.1016/0021-9991(86)90099-9.

[6] Alexander N. Brooks and Thomas J.R. Hughes. "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations". In: *Computer Methods in Applied Mechanics and Engineering* 32.1–3 (Sept. 1982), pp. 199–259. ISSN: 00457825. DOI: 10.1016/0045-7825(82)90071-8.

[7] S.V Patankar and D.B Spalding. "A Calculation Procedure for Heat, Mass and Momentum Transfer in Three-Dimensional Parabolic Flows". In: *International Journal of Heat and Mass Transfer* 15.10 (Oct. 1972), pp. 1787–1806. ISSN: 00179310. DOI: 10.1016/0017-9310(72)90054-3.

[8] Paul Fischer et al. *Scalability of High-Performance PDE Solvers*. 2020. DOI: 10.48550/ARXIV.2004.06722. URL: https://arxiv.org/abs/2004.06722. Pre-published.

[9] Terence Parr and Jeremy Howard. *The Matrix Calculus You Need For Deep Learning*. 2018. DOI: 10.48550/ARXIV.1802.01528. URL: https://arxiv.org/abs/1802.01528. Pre-published.

[10] Alvaro Sanchez-Gonzalez et al. *Learning to Simulate Complex Physics with Graph Networks*. Sept. 14, 2020. DOI: 10.48550/arXiv.2002.09405. arXiv: 2002.09405. URL: http://arxiv.org/abs/2002.09405. Pre-published.

[11] Tobias Pfaff et al. "Learning Mesh-Based Simulation with Graph Networks". In: (2020). DOI: 10.48550/ARXIV.2010.03409.

[12] Chengping Rao, Hao Sun, and Yang Liu. "Physics-Informed Deep Learning for Incompressible Laminar Flows". In: *Theoretical and Applied Mechanics Letters* 10.3 (Mar. 2020), pp. 207–212. ISSN: 2095-0349. DOI: 10.1016/j.taml.2020.01.039.

[13] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.1 (Jan. 5, 2020), pp. 477–508. ISSN: 0066-4189, 1545-4479. DOI: 10.1146/annurev-fluid-010719-060214.

[14] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 00219991. DOI: 10.1016/j.jcp.2018.10.045.

[15] Hrvoje Jasak. "OpenFOAM: Open Source CFD in Research and Industry". In: *International Journal of Naval Architecture and Ocean Engineering* 1.2 (Dec. 2009), pp. 89–94. ISSN: 20926782. DOI: 10.2478/IJNAOE-2013-0011.

[16] Ekhi Ajuria Illarramendi et al. "Towards an Hybrid Computational Strategy Based on Deep Learning for Incompressible Flows". In: *AIAA AVIATION 2020 FORUM*. AIAA AVIATION 2020 FORUM. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, June 15, 2020. ISBN: 978-1-62410-598-2. DOI: 10.2514/6.2020-3058.

[17] Jack Richter-Powell, Yaron Lipman, and Ricky T. Q. Chen. "Neural Conservation Laws: A Divergence-Free Perspective". In: (2022). DOI: 10.48550/ARXIV.2210.01741.

[18] Nils Wandel, Michael Weinmann, and Reinhard Klein. *Learning Incompressible Fluid Dynamics from Scratch – Towards Fast, Differentiable Fluid Models That Generalize*. 2020. DOI: 10.48550/ARXIV.2006.08762. URL: https://arxiv.org/abs/2006.08762. Pre-published.

[19] Ning Liu et al. *Harnessing the Power of Neural Operators with Automatically Encoded Conservation Laws*. 2023. DOI: 10.48550/ARXIV.2312.11176. URL: https://arxiv.org/abs/2312.11176. Pre-published.

[20] Tianyu Li et al. "Predicting Unsteady Incompressible Fluid Dynamics with Finite Volume Informed Neural Network". In: *Physics of Fluids* 36.4 (Apr. 1, 2024), p. 043601. ISSN: 1070-6631, 1089-7666. DOI: 10.1063/5.0197425.

[21] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. 3rd ed. Dordrecht New York: Springer, 2009. ISBN: 978-3-540-49834-6.

[22] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*. 1st ed. 2016.

Cham: Springer International Publishing : Imprint: Springer, 2016. ISBN: 978-3-319-34864-3.

[23] John P. Boyd. *Chebyshev & Fourier Spectral Methods*. Lecture Notes in Engineering 49. Berlin Heidelberg New York London Paris Tokyo Hong Kong: Springer, 1989. 798 pp. ISBN: 978-3-540-51487-9 978-0-387-51487-1.

[24] Alfio Quarteroni. *Numerical Approximation of Partial Differential Equations*. Springer Series in Computational Mathematics Ser v.23. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2008. 1 p. ISBN: 978-3-540-85268-1.

[25] Hrvoje Jasak. "Error Analysis and Estimation for the Finite Volume Method With Applications to Fluid Flows". In: *Direct* M (Jan. 1, 1996).

[26] Tessa Uroić and Hrvoje Jasak. "Block-Selective Algebraic Multigrid for Implicitly Coupled Pressure-Velocity System". In: *Computers & Fluids* 167 (May 2018), pp. 100–110. ISSN: 00457930. DOI: 10.1016/j.compfluid.2018.02.034.

[27] David C. Wilcox. *Turbulence Modeling for CFD*. 3. ed., 2. print. La Cañada, Calif.: DCW Industries, 2010. 52 pp. ISBN: 978-1-928729-08-2.

[28] J. Smagorinsky. "GENERAL CIRCULATION EXPERIMENTS WITH THE PRIMITIVE EQUATIONS: I. THE BASIC EXPERIMENT*". In: *Monthly Weather Review* 91.3 (Mar. 1963), pp. 99–164. ISSN: 0027-0644, 1520-0493. DOI: 10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2.

[29] H. Bhatia et al. "The Helmholtz-Hodge Decomposition—A Survey". In: *IEEE Transactions on Visualization and Computer Graphics* 19.8 (Aug. 2013), pp. 1386–1404. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.316.

[30] Cristian Barbarosie. "Representation of Divergence-Free Vector Fields". In: *Quarterly of Applied Mathematics* 69.2 (Mar. 9, 2011), pp. 309–316. ISSN: 0033-569X, 1552-4485. DOI: 10.1090/s0033-569x-2011-01215-2.

[31] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. third, rev. edition. Berlin, Heidelberg s.l: Springer Berlin Heidelberg, 2002. 426 pp. ISBN: 978-3-642-56026-2.

[32] Zongyi Li et al. *Physics-Informed Neural Operator for Learning Partial Differential Equations*. 2021. DOI: 10.48550/ARXIV.2111.03794. URL: https://arxiv.org/abs/2111.03794. Pre-published.

[33] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980. Pre-published.

[34] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. DOI: 10.48550/ARXIV.1912.01703. URL: https://arxiv.org/abs/1912.01703. Pre-published.

[35] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. DOI: 10.48550/ARXIV.1603.04467. URL: https://arxiv.org/abs/1603.04467. Pre-published.

[36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[37] Salvatore Cuomo et al. "Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What's Next". In: *Journal of Scientific Computing* 92.3 (Sept. 2022), p. 88. ISSN: 0885-7474, 1573-7691. DOI: 10.1007/s10915-022-01939-z.

[38] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. DOI: 10.48550/ARXIV.1704.01212. URL: https://arxiv.org/abs/1704.01212. Pre-published.

[39] Xinghao Dong, Chuanqi Chen, and Jin-Long Wu. *Data-Driven Stochastic Closure Modeling via Conditional Diffusion Model and Neural Operator*. 2024. DOI: 10.48550/ARXIV.2408.02965. URL: https://arxiv.org/abs/2408.02965. Pre-published.

[40] Kelsey R. Allen et al. "Inverse Design for Fluid-Structure Interactions Using Graph Network Simulators". In: Advances in Neural Information Processing Systems. Oct. 31, 2022.

[41] Pin Wu et al. "Data-Driven Reduced Order Model with Temporal Convolutional Neural Network". In: *Computer Methods in Applied Mechanics and Engineering* 360 (Mar. 2020), p. 112766. ISSN: 0045-7825. DOI: 10.1016/j.cma.2019.112766.

[42] Jonathan Tompson et al. *Accelerating Eulerian Fluid Simulation With Convolutional Networks*. 2016. DOI: 10.48550/ARXIV.1607.03597. URL: https://arxiv.org/abs/1607.03597. Pre-published.

[43] Xiaoxiao Guo, Wei Li, and Francesco Iorio. "Convolutional Neural Networks for Steady Flow Approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, Aug. 13, 2016, pp. 481–490. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939738.

[44] Haiming Zhang et al. "Physics-Informed Graph Neural Network Based on the Finite Volume Method for Steady Incompressible Laminar Convective Heat Transfer". In: *Physics of Fluids* 37.1 (Jan. 1, 2025), p. 013625. ISSN: 1070-6631, 1089-7666. DOI: 10.1063/5.0250663.

[45] Sebastian K. Mitusch, Simon W. Funke, and Miroslav Kuchta. "Hybrid FEM-NN Models: Combining Artificial Neural Networks with the Finite Element Method". In: *Journal of Computational Physics* 446 (Dec. 2021), p. 110651. ISSN: 00219991. DOI: 10.1016/j.jcp.2021.110651.

[46] Xiaowei Jin et al. "Prediction Model of Velocity Field around Circular Cylinder over Various Reynolds Numbers by Fusion Convolutional Neural Networks Based on Pressure

on the Cylinder". In: *Physics of Fluids* 30.4 (Apr. 1, 2018). ISSN: 1070-6631, 1089-7666. DOI: 10.1063/1.5024595.

[47] Li-Wei Chen and Nils Thuerey. "Towards High-Accuracy Deep Learning Inference of Compressible Flows over Aerofoils". In: *Computers & Fluids* 250 (Jan. 2023), p. 105707. ISSN: 00457930. DOI: 10.1016/j.compfluid.2022.105707.

[48] A. Bormanis, C. A. Leon, and A. Scheinker. "Solving the Orszag–Tang Vortex Magnetohydrodynamics Problem with Physics-Constrained Convolutional Neural Networks". In: *Physics of Plasmas* 31.1 (Jan. 1, 2024), p. 012101. ISSN: 1070-664X, 1089-7674. DOI: 10.1063/5.0172075.

[49] Rui Wang et al. *Towards Physics-informed Deep Learning for Turbulent Flow Prediction*. 2019. DOI: 10.48550/ARXIV.1911.08655. URL: https://arxiv.org/abs/1911.08655. Pre-published.

[50] Nils Thuerey et al. "Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows". In: (2018). DOI: 10.48550/ARXIV.1810.08217.

[51] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 22, 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907. Pre-published.

[52] Peter W. Battaglia et al. *Relational Inductive Biases, Deep Learning, and Graph Networks*. 2018. DOI: 10.48550/ARXIV.1806.01261. URL: https://arxiv.org/abs/1806.01261. Pre-published.

[53] Yoshua Bengio, Aaron Courville, and Pascal Vincent. *Representation Learning: A Review and New Perspectives*. 2012. DOI: 10.48550/ARXIV.1206.5538. URL: https://arxiv.org/abs/1206.5538. Pre-published.

[54] Masanobu Horie and Naoto Mitsume. *Graph Neural PDE Solvers with Conservation and Similarity-Equivariance*. May 25, 2024. DOI: 10.48550/arXiv.2405.16183. arXiv: 2405.16183. URL: http://arxiv.org/abs/2405.16183. Pre-published.

[55] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. DOI: 10.48550/ARXIV.1903.02428. URL: https://arxiv.org/abs/1903.02428. Pre-published.

[56] Huayu Deng et al. *EvoMesh: Adaptive Physical Simulation with Hierarchical Graph Evolutions*. May 21, 2025. DOI: 10.48550/arXiv.2410.03779. arXiv: 2410.03779. URL: http://arxiv.org/abs/2410.03779. Pre-published.

[57] Brian R. Bartoldson et al. *Scientific Computing Algorithms to Learn Enhanced Scalable Surrogates for Mesh Physics*. Apr. 1, 2023. DOI: 10.48550/arXiv.2304.00338. arXiv: 2304.00338. URL: http://arxiv.org/abs/2304.00338. Pre-published.

[58] Ugo Piomelli and Elias Balaras. "Wall-Layer Models for Large-Eddy Simulations". In: *Annual Review of Fluid Mechanics* 34 (Volume 34, 2002 2002), pp. 349–374. ISSN: 1545-4479. DOI: 10.1146/annurev.fluid.34.082901.144919.

[59] J. Sola and J. Sevilla. "Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems". In: *IEEE Transactions on Nuclear Science* 44.3 (June 1997), pp. 1464–1468. ISSN: 0018-9499, 1558-1578. DOI: 10.1109/23.589532.

[60] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101. Pre-published.

[61] Leslie N. Smith. *A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 – Learning Rate, Batch Size, Momentum, and Weight Decay*. 2018. DOI: 10.48550/ARXIV.1803.09820. URL: https://arxiv.org/abs/1803.09820. Pre-published.

[62] Johannes Brandstetter, Daniel Worrall, and Max Welling. *Message Passing Neural PDE Solvers*. 2022. DOI: 10.48550/ARXIV.2202.03376. URL: https://arxiv.org/abs/2202.03376. Pre-published.

[63] Léonard Equer, T. Konstantin Rusch, and Siddhartha Mishra. *Multi-Scale Message Passing Neural PDE Solvers*. Feb. 7, 2023. DOI: 10.48550/arXiv.2302.03580. arXiv: 2302.03580. URL: http://arxiv.org/abs/2302.03580. Pre-published.

[64] Wen-Ping Tsai et al. "From Calibration to Parameter Learning: Harnessing the Scaling Effects of Big Data in Geoscientific Modeling". In: *Nature Communications* 12.1 (Oct. 13, 2021), p. 5988. ISSN: 2041-1723. DOI: 10.1038/s41467-021-26107-z.

[65] Tom Roeger et al. "Analysis of the Relationship between Training Data Volume and Model Quality for Surrogate Models in Physical Simulations". In: *Procedia CIRP* 126 (2024), pp. 745–750. ISSN: 22128271. DOI: 10.1016/j.procir.2024.08.302.

[66] P. Lancaster and K. Salkauskas. "Surfaces Generated by Moving Least Squares Methods". In: *Mathematics of Computation* 37.155 (1981), pp. 141–158. ISSN: 0025-5718, 1088-6842. DOI: 10.1090/S0025-5718-1981-0616367-1.

[67] G. K. Batchelor. *An Introduction to Fluid Dynamics*. 1st ed. Cambridge University Press, Feb. 28, 2000. ISBN: 978-0-521-66396-0 978-0-511-80095-5. DOI: 10.1017/CBO9780511800955.

[68] D J Acheson. *Elementary Fluid Dynamics*. Oxford University PressOxford, Mar. 15, 1990. ISBN: 978-0-19-859660-8 978-1-383-03095-2. DOI: 10.1093/oso/9780198596608.001.0001.

[69] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. *A Survey on Oversmoothing in Graph Neural Networks*. Version 1. 2023. DOI: 10.48550/ARXIV.2303.10993. URL: https://arxiv.org/abs/2303.10993. Pre-published.

[70] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. Version 5. 2016. DOI: 10.48550/ARXIV.1608.03983. URL: https://arxiv.org/abs/1608.03983. Pre-published.

[71] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train Longer, Generalize Better: Closing the Generalization Gap in Large Batch Training of Neural Networks". Version 2. In: (2017). DOI: 10.48550/ARXIV.1705.08741.

[72] Robert Geirhos et al. “Shortcut Learning in Deep Neural Networks”. Version 5. In: (2020). DOI: 10.48550/ARXIV.2004.07780.

[73] William Lotter, Gabriel Kreiman, and David Cox. *Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning*. Version 5. 2016. DOI: 10.48550/ARXIV.1605.08104. URL: https://arxiv.org/abs/1605.08104. Pre-published.

[74] Piotr Breitkopf, Alain Rassineux, and Pierre Villon. “An Introduction to Moving Least Squares Meshfree Methods”. In: *Revue Européenne des Éléments Finis* 11.7–8 (Jan. 2002), pp. 825–867. ISSN: 1250-6559. DOI: 10.3166/reef.11.825-867.

[75] Meire Fortunato et al. “MultiScale MeshGraphNets”. Version 1. In: (2022). DOI: 10.48550/ARXIV.2210.00612.

[76] Bofu Wang et al. “Active Control of Flow Past an Elliptic Cylinder Using an Artificial Neural Network Trained by Deep Reinforcement Learning”. In: *Applied Mathematics and Mechanics* 43.12 (Dec. 2022), pp. 1921–1934. ISSN: 0253-4827, 1573-2754. DOI: 10.1007/s10483-022-2940-9.

[77] Geunwoo Oh, Hyunwook Park, and Jung-Il Choi. “Drag, Lift, and Torque Coefficients for Various Geometrical Configurations of Elliptic Cylinder under Stokes to Laminar Flow Regimes”. In: *AIP Advances* 12.6 (June 1, 2022), p. 065228. ISSN: 2158-3226. DOI: 10.1063/5.0097916.

[78] Xiaoyu Shi, Mahbub Alam, and Honglei Bai. “Wakes of Elliptical Cylinders at Low Reynolds Number”. In: *International Journal of Heat and Fluid Flow* 82 (Apr. 2020), p. 108553. ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2020.108553.

[79] V.L. Nguyen, V.D. Duong, and L.H. Duong. “Transition in Vortex Wakes of Flow around an Elliptic Cylinder”. In: (2024). DOI: 10.24423/AOM.4556.

[80] T. v. Karmann. *Aerodynamics*. New York: McGraw-Hill, 1963. ISBN: 978-0-07-067602-2.

[81] “Mechanism of the Production of Small Eddies from Large Ones”. In: *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences* 158.895 (Feb. 3, 1937), pp. 499–521. ISSN: 0080-4630, 2053-9169. DOI: 10.1098/rspa.1937.0036.

[82] Alexander Radi et al. “From the Circular Cylinder to the Flat Plate Wake: The Variation of Strouhal Number with Reynolds Number for Elliptical Cylinders”. In: *Physics of Fluids* 25.10 (Oct. 1, 2013), p. 101706. ISSN: 1070-6631, 1089-7666. DOI: 10.1063/1.4827521.