A Study on the Stability Limits of Graph Neural Network Surrogates for Advection–Diffusion Problems

MSc Applied Mathematics Thesis Project

Mitchell Maassen van den Brink

Delft University of Technology

A Study on the Stability Limits of Graph Neural Network Surrogates for Advection-Diffusion Problems

by

Mitchell Maassen van den Brink

To obtain the degree of Master of Science
in Applied Mathematics
at the Delft University of Technology,
To be defended on: 11 June 2026

Student number: 4728068
Project duration: November 2024 – June 2026

Thesis committee:

Dr. Alexander Heinlein	Responsible Supervisor & Chair	Numerical Analysis - TU Delft
Prof. dr. ir Martin Verlaan	Core Member 2	Mathematical Physics - TU Delft
Dr. Richard P. Dwight	Daily Supervisor & Core Member 3	Aerodynamics - TU Delft

Abstract

Machine-learned surrogate time integrators promise large speed-ups over classical solvers, yet their performance is usually reported as a single aggregate error, leaving open the question of when they remain stable. This thesis determines the empirical stability limits of graph neural network (GNN) surrogates for the two-dimensional advection–diffusion equation on unstructured meshes with periodic boundary conditions, expressed directly in the dimensionless CFL and Fourier numbers.

Surrogates are trained to minimise the one-step error on fixed velocity and diffusion fields and evaluated autoregressively on unseen fields over horizons eight times the training window. Two families are compared at a fixed message passing budget: single-scale models, and multiscale models organised as V-cycles over predetermined coarsened graphs. For each model, a piecewise-linear fit of the final rollout error against the CFL and Fourier numbers yields empirical stability limits, defined by a blow-up threshold.

Within these limits the surrogates reproduce the finite element reference accurately on both seen and unseen fields and show no abrupt change beyond the training horizon, although the diffusion-dominated regime is consistently harder than the advection-dominated one. The single-scale CFL limit tracks the number of message-passing blocks and lies slightly above it. Adding coarse levels at a fixed total message passing layer budget broadens the advective stability range, decisively at the largest stride, but a two-level hierarchy trades diffusive stability and in-region accuracy for this gain. Only a three-level V-cycle removes the penalty, attaining zero blow-ups on both axes, and deeper models show no oversmoothing.

The diffusion-side limits carry large variance, traced partly to the dissipative backward-Euler reference, and should be read as indicative. The work delivers a concrete operational range for GNN surrogates and identifies how multiscale models can extend it.

Preface

This thesis is the result of a research project carried out at the Delft University of Technology, as the final component of the Master of Science in Applied Mathematics. This project was conducted externally at the faculty of Aerospace Engineering.

I would like to express my sincere gratitude to my supervisors, Dr. Alexander Heinlein and Dr. Richard P. Dwight, for their guidance and support throughout this research. Their expertise and encouragement have been instrumental in shaping the direction of this thesis and in helping me navigate the challenges encountered along the way. I am also grateful to Prof. dr. ir. Martin Verlaan for his insightful feedback and contributions as a core member of the thesis committee.

While the thesis itself was a massive challenge, the circumstances outside of it really shaped me as a person too, with the interplay between the two being especially tricky at times. I am more grateful than ever for my friends, family, housemates, and everyone close to me who helped me get through it. It is always one of those things you consider a cliché when you read it somewhere else, but I really could not have done it without you. And to Leo: I would not know where to begin. And since you do not like being corny, I will limit the torture this time and simply say: thank you.

I hope you enjoy reading this thesis as much as I enjoyed writing it. It has been a truly rewarding experience, and I am proud of the work that has been accomplished.

Mitchell Maassen van den Brink
Delft, June 2026

Declaration of use of AI

Artificial intelligence tools were employed selectively throughout this work. Specifically, GitHub Copilot with the Claude Opus model was utilised for code debugging and Python visualisation tasks (including the front cover), while Claude Sonnet assisted with proofreading and textual refinement. Additionally, code for TikZ diagrams was generated using Claude Opus and subsequently refined manually by the author.

The application of AI was confined to these auxiliary computational and editorial functions and did not encompass the core intellectual contributions of this thesis, namely the experimental design, results interpretation, and conclusions drawn. The author assumes complete responsibility for all content presented herein, including any potential errors or inaccuracies.

Contents

Chapter 1

Introduction

1.1 Motivation for surrogate modelling

Computational science and engineering is widely regarded as the third pillar of science, complementing theory and experiment by offering tools to model phenomena across disciplines ranging from quantum chemistry and fluid dynamics to epidemiology [1]. This discipline is so important because many research problems involve experiments that are either too costly or infeasible.

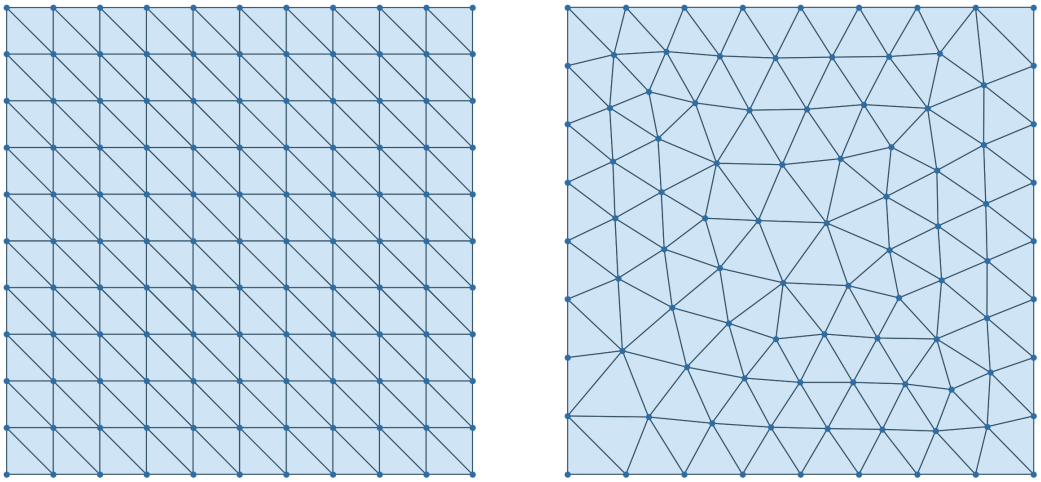
When a physical process evolves dynamically in space and time, it can often be described by a partial differential equation (PDE). The finite element method (FEM) is one of the most widely used numerical frameworks for solving such equations: it discretises the spatial domain into a mesh, constructs a finite-dimensional approximation space tied to that mesh, and solves the resulting discrete system [2]. FEM has become a cornerstone of simulation across fluid mechanics, structural mechanics, and electrodynamics [3]. A significant drawback, however, is the computational cost, which can be prohibitive for problems requiring fine spatial resolution or covering large domains. Iterative methods and multigrid methods reduce this cost substantially [4]. Parallelisation strategies, including domain decomposition [5], reduce wall-clock time and benefit from modern CPU and GPU hardware. The machine learning (ML) approach explored in this thesis, the surrogate model, shares this benefit.

Over the past years, ML-based surrogate time integrators have attracted considerable interest. For flood simulation, [6] reported speed-ups of one to two orders of magnitude relative to a classical solver, while [7] demonstrated broad applicability across compressible and incompressible flows as well as structural mechanics. Before graph neural networks (GNNs) became prominent for this task, convolutional neural networks (CNNs) were already used for surrogate modelling [8], benefiting from their inherent properties of locality and translation invariance [9]. GNNs have since emerged as a more flexible alternative that overcomes a fundamental limitation of standard CNNs.

Standard CNNs operate on regular, grid-structured data such as the uniform mesh in Figure 1.1a. In FEM simulations, however, unstructured meshes with spatially varying resolution are often used. An example of an unstructured mesh is shown in Figure 1.1b. The main advantage of this unstructured nature is that one can concentrate the element density in regions of steep gradients or complex geometry, and coarsen it in areas where the solution is smooth. This is also an iterative process, as the mesh can be refined at spatial locations where large errors were observed. Applying a CNN to such a mesh requires interpolating the solution onto a regular grid before processing and mapping it back afterwards, which introduces additional interpolation errors. GNNs avoid this issue altogether, because they operate on a much more flexible topological entity. By treating the mesh as a graph, where nodes sit at the mesh vertices and edges connect neighbouring nodes, GNNs can process the solution directly on the unstructured topology without any interpolation.

Prior to this thesis, a literature review was carried out to survey GNN architectures that learn the time evolution of Eulerian dynamical systems[1] governed by PDEs. The focus was on models that mimic a single time-integration step: given the current state of the system, the GNN predicts the update over one time step Δt. Applying the model repeatedly from an initial condition produces a trajectory, referred to as a rollout. We will briefly discuss the main findings of this review, as they motivate the main research questions of this thesis.

[1] Eulerian systems are formulated on a mesh that is fixed in space, in contrast to Lagrangian formulations in which the mesh moves with the material.



(a) An arbitrary structured mesh with triangular mesh elements.

(b) An arbitrary unstructured mesh with triangular mesh elements.

Figure 1.1: The visual difference between an arbitrary structured mesh (left) and an arbitrary unstructured mesh (right). CNNs can operate directly on the structured mesh, but applying them to the unstructured one requires interpolation onto a regular grid.

1.2 Related work and the scientific gap

The literature review examined several GNN-based surrogate models. The MeshGraphNets architecture [7] and its multiscale extension [10], the SWE-GNN model [6] and its multiscale variant [11], and the ReMUS-GNN [12] and CFD-CGN [13] models were all assessed in terms of training data, architecture, training procedure, and performance reporting.

A number of architectural similarities were found across the reviewed models. Most adopt an encode-process-decode structure, use (P)ReLU activations, and rely on message passing or graph convolution as the core computation step, though with different specific formulations. The main differences were in how error accumulation over long rollouts is dealt with, the underlying model problem and in how the multiscale hierarchies are designed. A notable shared aspect, however, is the evaluation approach: performance on test data is reported only as aggregate statistics, mean and standard deviation, over a set of test simulations. While this is standard in classification and regression applications for ML-based methods, where a single accuracy figure is enough, it leaves important questions unanswered for surrogate modelling. For which physical configurations does the model perform accurately, and for which does it create instabilities? How does the stability of the surrogate change as a function of the relevant dimensionless numbers of the underlying problem? And how do single-scale and multiscale architectures compare in this regard? These questions were not addressed in the surveyed literature and form the scientific gap this thesis aims to fill.

1.3 Research questions

The model problem considered throughout this thesis is the two-dimensional advection-diffusion equation, introduced in Chapter 2. By independently varying the amplitude of the velocity field and the magnitude of the diffusion tensor, the simulations cover a wide range of physical regimes, characterised by three dimensionless numbers: the CFL number, the Fourier number, and the mesh Péclet number, formally defined in Section 2.3. The surrogate models are trained on a fixed velocity and diffusion profile and evaluated on fields that were not seen during training. Three surrogate time step sizes are considered, $\Delta t \in \{0.015, 0.030, 0.060\}$, referred to as strides. The different timesteps offer us an insight into the stability and accuracy of the models across varying temporal resolutions. To assess the performance, we will later introduce the functional metrics $\mathscr{E}_{L^2}$ and $\mathscr{E}_{H^1}$.

The first main question is in line with the general evaluation approach found in the surrogate modelling literature: it asks whether the models can learn the dynamics and generalise. The second main question goes beyond this and addresses the scientific gap identified above, by providing a systematic analysis of the stability limits as a function of the dimensionless parameters. Together, the main findings will be built around the two main questions and their sub-questions.

Main Question 1

Can GNN-based surrogate time integrators learn to accurately reproduce the FEM solutions of the advection-diffusion equation when applied to seen and unseen velocity and diffusion fields?

Sub-questions for Main Question 1

SQ1. Can surrogate models qualitatively reproduce one-step updates comparable to the FEM solutions?

SQ2. For every surrogate time step, which model reports the lowest one-step error on the validation set, measured in terms of $\mathscr{E}_{L^2}(\Delta u)$, $\mathscr{E}_{H^1}(\Delta u)$?

SQ3. When performing simulations over horizons that exceed the training window, do the models maintain the advection-diffusion dynamics beyond the training horizon?

SQ4. For every surrogate time step, which model reports the lowest error on the test set during rollouts when measured in terms of $\mathscr{E}_{L^2}(u)$, $\mathscr{E}_{H^1}(u)$ at the final checkpoint $t = 1.92$?

Main Question 2

What are the empirical stability limits of GNN surrogate time integrators in terms of the CFL and Fourier numbers?

Sub-questions for Main Question 2

SQ5. At the smallest surrogate time step, how do the CFL and Fourier stability limits of single-scale models change as the number of message passing layers increases?

SQ6. At the medium surrogate time step, how does the allocation of message passing blocks between the fine and coarse levels in a two-level multiscale model affect the stability limits relative to a single-scale model with the same total message passing layers budget?

SQ7. At the largest surrogate time step, does a three-level model extend the stability region beyond what the two-level and single-scale models achieve?

SQ8. When single-scale models at different surrogate time steps that should have comparable stability regions are compared, does oversmoothing become more prominent as the number of message passing layers increases?

1.4 Structure of the thesis

The remainder of this thesis is organised as follows. Chapter 2 introduces the model problem and the numerical background that underpins the work, covering the two-dimensional advection-diffusion equation, its finite element discretisation, and the relevant dimensionless numbers, namely the CFL number, the Fourier number, and the mesh Péclet number. It also reviews explicit and implicit time-stepping schemes and their stability, SUPG stabilisation, and multigrid methods including algebraic multigrid. Chapter 3 develops the surrogate modelling framework based off the literature, starting from graphs, message passing, and GNNs, and continuing with layer normalisation and oversmoothing, the encode-process-decode graph neural operator, the use of GNNs as surrogate time integrators, and the learned restriction and prolongation operators that enable the multiscale architecture. Chapter 4 describes the experimental setting and methodology, explaining how the mesh is turned into a graph and which input features are used, presenting the datasets that consist of the training and validation set together with the two structured advection and diffusion test sets, and detailing the training setup, the single-scale and multiscale model variants that are studied, and the error metrics along with the procedure used to extract the empirical stability limits.

Chapter 5 presents the results, comprising a benchmark of conventional time integrators, the one-step accuracy on the training and validation sets, the multi-step rollout (trajectory) performance, and

the determination of the empirical stability limits. Chapter 6 discusses and interprets the main findings. Chapter 7 answers the research questions and states the conclusions. Chapter 8 gathers the limitations of the study and the recommendations for future work. This thesis was written to obtain the degree of Master of Science in Applied Mathematics at the Delft University of Technology.

Chapter 2

Background: model problem and numerical methods

To apply a surrogate model, a model problem is introduced described by a PDE. PDEs are fundamental in modelling a wide range of physical processes, including fluid dynamics [14], heat transfer [15], electrodynamics [16], and quantum mechanics [17]. In this chapter, we introduce the advection-diffusion equation as considered in the present work. This is a PDE that captures the combined effects of advection and diffusion. We discuss the physical interpretation and review numerical methods for solving PDEs, with a focus on finite element methods and time-stepping schemes, which are essential to creating reference simulations on which the surrogate model is trained and evaluated. As we will be creating simulations that are both in the advection-dominated and diffusion-dominated regimes, we also introduce the Streamline-Upwind Petrov-Galerkin (SUPG) stabilisation method that is used to generate stable reference trajectories in the advection-dominated regime. Finally, we introduce the algebraic multigrid method which, whilst not directly utilised in this thesis, is relevant for understanding the multigrid-inspired architecture of the surrogate model and provides the theoretical framework for the graph coarsening procedure.

2.1 The advection-diffusion equation

The advection-diffusion equation is a fundamental PDE that describes the transport of a scalar quantity, such as temperature, concentration, or vorticity, under the combined influence of advection and diffusion. Applications include the reconstruction of pollution sources in rivers [18], air pollution modelling [19], and analytical models for drug delivery from multilayer spherical capsules [20]. More generally, the equation governs how a scalar field u, representing any transported quantity such as heat or concentration, evolves within a fluid flow. The advection term captures the transport of u by the velocity field $\boldsymbol{v}$, whilst the diffusion term accounts for the spreading of u due to diffusive processes [21], which may arise from random molecular motion in a fluid [22]. A simple instance of the separate elements can be examined in Figure 2.1, where the most basic velocity field and diffusion tensor are considered, to illustrate the behaviour of the two terms in isolation on a Gaussian bump as initial condition. The advection term translates the bump without change of shape, whilst the diffusion term spreads the bump isotropically without change of centre of mass.

In this thesis, we restrict ourselves to a two-dimensional spatial domain $\Omega \subset \mathbb{R}^2$ over a time interval $J = (0, T]$, where $T > 0$ denotes the final time. Furthermore, both the diffusion tensor and the velocity field are chosen to be independent of the time t and solution u, where the latter ensures that the resulting PDE is linear. We only work with symmetric positive definite (SPD) diffusion tensors and divergence free velocity fields, making the equation parabolic [23]. SPD tensors ensure non-negative diffusion and therefore positive entropy, whereas the divergence free condition on the velocity field ensures that the advection term does not create or destroy mass, resulting in terms of the equation that are physically consistent [24, 25]. We do not consider any source term, so that $f \equiv 0$. Under this combination of conditions, namely a divergence-free velocity field, the SPD diffusion tensor, the absence of a source term, and the periodic boundary conditions introduced below, the total mass $\int_\Omega u \, \mathrm{d}\boldsymbol{x}$ is conserved over time, as there are no sources and sinks in the problem and/or domain. Moreover, the constant $\int_\Omega u \, \mathrm{d}\boldsymbol{x}$ is also the steady state solution as $t \to \infty$. The resulting PDE is given by

$$\frac{\partial u}{\partial t} = \nabla \cdot (D(\boldsymbol{x})\nabla u) - \nabla \cdot (\boldsymbol{v}(\boldsymbol{x})u) \text{ where } \boldsymbol{x} \in \Omega,\ t \in (0,T] = J, \tag{2.1}$$
$$u(\boldsymbol{x},0) = u_0(\boldsymbol{x}).$$

In the present work, the spatial domain is taken to be the unit square, defined as:

$$\Omega = \{(x,y) \in \mathbb{R}^2 | 0 \le x \le 1, 0 \le y \le 1\}.$$

Periodic boundary conditions are imposed in both the x- and y-directions. This means that a point at the boundary at $x = 0$ is spatially identical to another point at the boundary at $x = 1$, and similarly for the y-direction. We have chosen these boundary conditions such that no insertion of boundary conditions is needed for the surrogate model later. This way, we witness the true functionality of the surrogate model without interference from boundary effects, which may interfere with the results. Formally, these conditions are expressed as:

$$\begin{aligned} u(0,y,t) &= u(1,y,t) \quad \text{for all } y \in [0,1], t \in (0,T], \\ u(x,0,t) &= u(x,1,t) \quad \text{for all } x \in [0,1], t \in (0,T]. \end{aligned} \tag{2.2}$$

As the initial condition, we employ a random sum of Fourier modes, analogous to the approach in [12], given by:

$$u_0(x,y) = \Big(\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c_{m,n} \cos\big(2\pi(mx+ny)\big) \Big) \exp\big(-2(x-x_c)^2 - 2(y-y_c)^2\big), \tag{2.3}$$

where $(x_c, y_c) = (\frac{1}{2}, \frac{1}{2})$ is the centre of the domain. The numbers of modes M and N in the x- and y-directions are sampled as independent random integers between 2 and 7 inclusive, and each coefficient $c_{m,n}$ is drawn independently from the uniform distribution on $[0,1]$. Sampling the mode numbers in this way yields a wide range of spatial frequencies in each realisation, so that every initial condition contains both low-frequency (large-scale) and higher-frequency (small-scale) structure. Moreover, this random initial condition has infinite possible realisations, which allows us to train and evaluate the surrogate model on a wide variety of initial states, promoting generalisation and robustness. An example of this random initial condition on the unit square is shown in Figure 2.2. This initial condition is applied across all test cases considered in this work. Having such a random initial condition ensures that the surrogate model is trained and evaluated on a wide variety of initial states, which promotes generalisation and robustness. Moreover, the random Fourier structure of the initial condition allows us to control the spatial frequency content of the initial state, which is relevant for testing the surrogate's ability to capture both large-scale and small-scale features of the solution.

2.2 The finite element method

The finite element method (FEM) is the numerical scheme used throughout this thesis to generate the reference trajectories against which the surrogate models are trained and tested. This section introduces the FEM in the level of detail required for the remainder of the thesis. We begin with the general continuous weak form of a linear second-order problem and then specialise the derivation to the advection-diffusion equation, obtaining the semi-discrete linear system whose time integration is discussed in Section 2.4. The contents of the sections follow the standard references [2, 23, 26] related to this topic.

Consider a scalar function $u(\boldsymbol{x},t)$ with $\boldsymbol{x} \in \Omega \subset \mathbb{R}^n$ and $t \in (0,T]$, then we define the strong form of a general linear second-order PDE as: find u such that

$$\frac{\partial u}{\partial t} + Lu = f(\boldsymbol{x}) \qquad \text{on } \Omega \times (0,T], \tag{2.4}$$

for every $\boldsymbol{x} \in \Omega$ and $t \in (0,T]$, where L is a second-order spatial differential operator of the general form

$$Lu := -\sum_{i,j=1}^{n} \frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) + \sum_{i=1}^{n} \frac{\partial}{\partial x_i}(b_i u) + \sum_{i=1}^{n} c_i \frac{\partial u}{\partial x_i} + a_0 u, \tag{2.5}$$

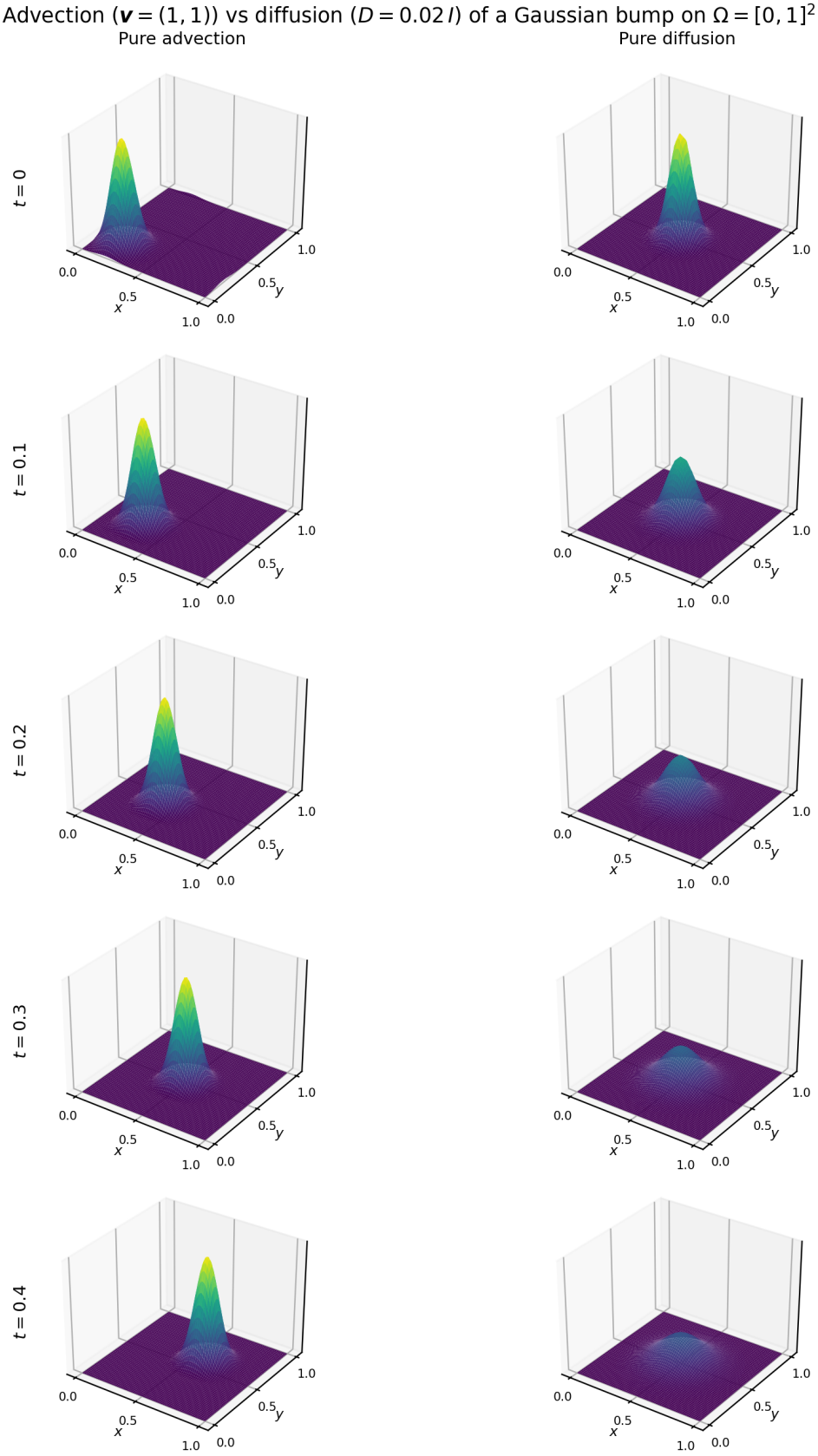


Figure 2.1: Decomposition of the most simple form of the advection-diffusion equation into its two elementary building blocks, applied to a Gaussian bump initial condition on $\Omega = [0,1]^2$ with periodic boundary conditions. The left column shows pure advection with a spatially constant velocity field $\boldsymbol{v} = (1,1)^\top$ and a bump centred at $(0.25, 0.25)$. The bump is translated along the diagonal without change of shape. The right column shows pure isotropic diffusion with $D = 0.02\,I$ and a bump centred at $(0.5, 0.5)$; the bump spreads isotropically whilst preserving its centre of mass. The rows correspond to the time instances $t \in \{0, 0.25, 0.5, 0.75, 1.0\}$.

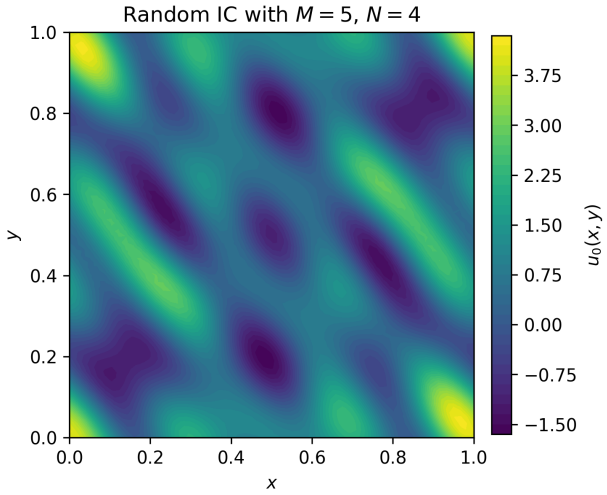


Figure 2.2: A single realisation of the random Fourier initial condition of equation 2.3 on the unit square with $M = 5$ and $N = 4$.

and $f(\boldsymbol{x})$ is a source term. The coefficients are assumed to be bounded, that is $a_{ij}, b_i, c_i, a_0 \in L^\infty(\Omega)$, and the diffusion matrix (a_{ij}) is assumed to be symmetric and positive (semi-)definite, so that the operator L and the bilinear form derived from it below are well-defined. Equation 2.4 is completed by boundary conditions on $\partial\Omega$ (which can be of Dirichlet, Neumann, Robin, mixed, periodic or Cauchy type) and by an initial condition at $t = 0$. Together with f, the boundary data and the initial data these coefficients form the problem data of the PDE. Depending on the signs and relative magnitudes of the a_{ij}, b_i, c_i and a_0 the equation is classified as elliptic, parabolic or hyperbolic [23]. It was already stated in the previous subsection that the model problem in the present work is parabolic. A problem is said to be well-posed in the sense of Hadamard [27] when a solution exists, is unique, and depends continuously on the problem data. For the strong form these three requirements are difficult to uphold directly for problems of practical interest, which is one of the reasons the weak formulation introduced next is preferred, which yields some relaxations with respect to these requirements.

For this reason the FEM does not work with the strong form directly but with an equivalent integral statement. We introduce a solution space $\mathscr{S}$ and a test space $\mathscr{W}$ (both subspaces of $H^1(\Omega)$ in our case, but this depends on the problem at hand) chosen so that any essential (for example, Dirichlet) boundary data are built into $\mathscr{S}$ and the corresponding homogeneous condition into $\mathscr{W}$. Defining the residual of the strong form as

$$R(u) := \frac{\partial u}{\partial t} + Lu - f(\boldsymbol{x}),$$

we require $R(u)$ to be orthogonal to every test function $w \in \mathscr{W}$. Which means we can define the continuous weak form of equation 2.4 as: find $u(\cdot, t) \in \mathscr{S}$ such that

$$\int_\Omega R(u)\, w \, d\Omega = 0 \qquad \forall\, w \in \mathscr{W}$$

for every test function $w \in \mathscr{W}$, $\boldsymbol{x} \in \Omega$ and $t \in (0, T]$. Applying Green's identity [28] to the second-order and first-order terms yields the bilinear form $B(\cdot,\cdot)$ and linear form $F(\cdot)$ associated with equation 2.4, defined for $u \in \mathscr{S}$ and $w \in \mathscr{W}$ by

$$\begin{aligned} B(u, w) := &\int_\Omega \left(\sum_{i,j=1}^{n} a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial w}{\partial x_i} - \sum_{i=1}^{n} b_i\, u \frac{\partial w}{\partial x_i} + \sum_{i=1}^{n} c_i \frac{\partial u}{\partial x_i}\, w + a_0\, u\, w \right) d\Omega \\ &+ \oint_{\partial\Omega} \left(\sum_{k=1}^{n} b_i\, n_k\, u - \sum_{i,j=1}^{n} a_{ij} \frac{\partial u}{\partial x_j} n_k \right) w \, dS, \\ F(w) := &\int_\Omega f\, w \, d\Omega, \end{aligned}$$

where $\boldsymbol{n} = (n_1, \ldots, n_n)^\top$ is the outward unit normal on $\partial\Omega$. The continuous weak form then reads: find $u(\cdot, t) \in \mathscr{S}$ such that

$$\frac{d}{dt}(u, w)_{L^2(\Omega)} + B(u, w) = F(w) \qquad \forall\, w \in \mathscr{W}, \tag{2.6}$$

where $(\cdot,\cdot)_{L^2(\Omega)}$ denotes the standard L^2 inner product on Ω. Under standard coercivity and continuity hypotheses on B,

$$\exists \alpha, \forall v \in \mathscr{S}, \ \sup_{w \in \mathscr{W}\setminus\{\mathbf{0}\}} \frac{B(v,w)}{\|w\|_{\mathscr{W}}} \geq \alpha \|v\|_{\mathscr{S}}, \qquad \text{Coercivity}$$

$$\forall w \in \mathscr{W}: \quad B(v,w) = 0 \text{ for all } v \in \mathscr{S} \implies w = \mathbf{0}, \qquad \text{Uniqueness}$$

existence and uniqueness of a solution to equation 2.6 follow from the Lax-Milgram [23] or the more general Banach-Nečas-Babuška theorem [29].

We now apply this framework to the advection-diffusion equation 2.1. Rewriting the strong form with the time derivative on the left,

$$\frac{\partial u}{\partial t} - \nabla \cdot \big(D(\boldsymbol{x}) \nabla u\big) + \nabla \cdot \big(\boldsymbol{v}(\boldsymbol{x})\, u\big) = 0, \qquad \boldsymbol{x} \in \Omega,\ t \in (0,T], \tag{2.7}$$

one can see that equation 2.7 is a special case of equations 2.4-2.5 with $Lu = -\nabla\cdot(D\nabla u) + \nabla\cdot(\boldsymbol{v}u)$, so that $a_{ij} = D_{ij}$, $b_i = v_i$, and $c_i = a_0 = 0$. On the unit square $\Omega = [0,1]^2$ with doubly-periodic boundary conditions all essential data are encoded in the function space rather than on the geometric boundary, so the trial and test spaces coincide.

$$\mathscr{S} = \mathscr{W} = H^1_{\text{per}}(\Omega) := \big\{ w \in H^1(\Omega) : \gamma_{\{x=0\}} w = \gamma_{\{x=1\}} w,\ \gamma_{\{y=0\}} w = \gamma_{\{y=1\}} w \big\},$$

where γ_Γ denotes the trace operator that restricts an $H^1(\Omega)$ function to the boundary segment Γ, so that the traces on opposite edges of the domain are identified. So given that $f = 0$, multiplying equation 2.7 by a test function $w \in \mathscr{S}$, integrating over Ω, and applying Green's identity for the diffusive term yields

$$\underbrace{\frac{d}{dt}(u,w)_{L^2(\Omega)}}_{\text{evolution}} + \underbrace{\int_\Omega (D\nabla u)\cdot\nabla w \, d\Omega}_{\text{diffusion}} - \underbrace{\int_\Omega (\boldsymbol{v}u)\cdot\nabla w \, d\Omega}_{\text{advection}} = 0, \qquad \forall w \in \mathscr{S}. \tag{2.8}$$

Where the periodic boundary conditions ensure that the boundary terms vanish in the integration by parts. The bilinear form $B(u,w)$ associated with equation 2.7 is thus the sum of a symmetric positive semi-definite diffusive term (whenever D is symmetric positive semi-definite, which it is in this thesis) and a non-symmetric advective term responsible for the hyperbolic transport. Now it shows that for the solution and test spaces, we must at least have $u, w \in H^p(\Omega)$ with $p \geq 1$ to make the diffusive term well-defined.

The trial and test spaces in equation 2.8 are infinite-dimensional and must be approximated by finite-dimensional subspaces before the problem can be solved. We therefore partition Ω into a triangulation $\mathscr{T}_h = \{K_1, K_2, \ldots, K_{N_{\text{el}}}\}$ of N_{el} non-overlapping elements such that $\bigcup_{K\in\mathscr{T}_h} \bar{K} = \bar{\Omega}$ and the intersection of any two elements is either empty, a common vertex or a common edge. In two dimensions these elements are typically triangles or quadrilaterals. In this thesis we use an unstructured triangulation, mainly because this facilitates a broader range of edge attributes for the surrogate model to train on. The parameter h denotes the characteristic edge length of the triangulation, which in practice varies across $\mathscr{T}_h$. We write $h_K = \text{diam}(K)$ for the diameter of a single element.

On top of $\mathscr{T}_h$ we introduce a finite-dimensional conforming subspace $\mathscr{S}_h \subset \mathscr{S}$. Its elements are globally continuous functions whose restriction to each element K is a polynomial of fixed degree p, which is the standard continuous Lagrange space $\mathbb{P}_p(\mathscr{T}_h)$. In the periodic setting the degrees of freedom on opposite edges of Ω are identified, so the dimension $n := \dim \mathscr{S}_h$ equals the number of independent nodal unknowns. Throughout this thesis we take $p = 1$, i.e. piecewise-linear elements on triangles. Higher value of p are possible and can be used to achieve higher-order accuracy, but this also yields more computational cost. Higher order elements improve accuracy at a functional level, but do not have particular emphasis on being nodally exact, and therefore do not necessarily improve the accuracy of the surrogate models. That being said, a stabilisation method such as SUPG (introduced in Section 2.5) is required to obtain accurate solutions in the advection-dominated regime, which additionally is a method that aims at being nodally accurate, which is beneficial for the surrogate models.

Let $\{N_i\}_{i=1}^n$ denote the standard nodal Lagrange basis of $\mathscr{S}_h$, i.e. the piecewise-linear hat functions satisfying $N_i(\boldsymbol{x}_j) = \delta_{ij}$ at the mesh nodes $\{\boldsymbol{x}_j\}_{j=1}^n$. We write the discrete solution as a linear combination of these basis functions with time-dependent coefficients,

$$u_h(\boldsymbol{x}, t) = \sum_{j=1}^{n} u_j(t)\, N_j(\boldsymbol{x}), \tag{2.9}$$

so that the coefficient vector $\boldsymbol{u}(t) = (u_1(t), \dots, u_n(t))^\top$ is exactly the vector of nodal values of u_h at the n mesh nodes. The Galerkin approximation of equation 2.8 consists in replacing $\mathscr{S}$ by $\mathscr{S}_h$ and, since $\mathscr{S}_h$ is finite-dimensional, testing only against the n basis functions: find $\boldsymbol{u}(t) \in \mathbb{R}^n$ such that

$$\frac{d}{dt}(u_h, N_i)_{L^2(\Omega)} + B(u_h, N_i) = 0, \qquad i = 1, \dots, n. \tag{2.10}$$

Substituting the expansion 2.9 into equation 2.10 gives an $n \times n$ linear system of ordinary differential equations in time,

$$M\dot{\boldsymbol{u}}(t) + A\,\boldsymbol{u}(t) = \boldsymbol{0}, \tag{2.11}$$

with the symmetric positive-definite mass matrix M and the (in general non-symmetric) stiffness matrix A given by

$$M_{ij} = \int_\Omega N_i N_j \, d\Omega, \qquad A_{ij} = \int_\Omega (D\nabla N_j)\cdot\nabla N_i \, d\Omega - \int_\Omega (\boldsymbol{v} N_j)\cdot\nabla N_i \, d\Omega.$$

Both matrices are sparse, since the hat functions N_i and N_j have local support on the mesh and their entries vanish whenever nodes i and j do not share an element. The symmetric part of A is dominated by the diffusive contribution and is positive semi-definite because D is symmetric positive definite by construction in the current work, whilst the advective contribution contributes a skew-symmetric component that is responsible for the numerical difficulties encountered when $\|\boldsymbol{v}\|$ is large relative to $\|D\|$.

The entries of M and A cannot in general be computed in closed form and are evaluated by numerical quadrature [30]. Since the restriction of $\boldsymbol{v}$ and the nodal basis to each element K is polynomial and D is smooth, an appropriately chosen Gauss rule integrates each element contribution to machine precision, so that the only discretisation error in equation 2.11 comes from the finite-dimensional approximation $\mathscr{S}_h$ itself. Equation 2.11 is a linear, stiff system of ordinary differential equations; its integration in time completes the fully discrete scheme and is the subject of Section 2.4.

2.3 Dimensionless numbers

Three dimensionless numbers govern the behaviour of a discretised advection-diffusion problem: the cell Péclet number, the CFL number and the Fourier number. Since we will be applying surrogate models to mimic the simulations on graphs, we introduce nodal variants. This allows us to probe the stability of the surrogate model as a function of these dimensionless numbers, which is one of the central questions of this thesis.

Definitions of the numbers are given in standard CFD and FE references such as [14, 31]. The cell Péclet number quantifies the relative strength of advection and diffusion. For an isotropic scalar diffusivity k, [31] defines it as $\mathrm{Pe}_h = \|\boldsymbol{v}\|\, h/(2k)$. The diffusion tensors considered in this thesis are anisotropic, so we adopt instead the streamline-diffusivity generalisation of the one introduced in [32],

$$\mathrm{Pe}_h = \max_{\boldsymbol{x}\in\Omega} \frac{\|\boldsymbol{v}\|_2\, h_{max}}{2\kappa_s}, \qquad \kappa_s := \frac{\boldsymbol{v}^\top D\,\boldsymbol{v}}{\|\boldsymbol{v}\|^2}, \tag{2.12}$$

where h_{max} is the maximum local mesh size and κ_s is the projection of D onto the flow direction, i.e. the diffusivity actually felt along the streamlines. For an isotropic tensor $D = kI$ one has $\kappa_s = k$, so equation 2.12 reduces to the scalar definition of [31], given that the grid is uniform. When $\mathrm{Pe}_h \lesssim 1$ the problem is diffusion-dominated and a standard Galerkin discretisation produces accurate solutions, whilst for $\mathrm{Pe}_h \gg 1$ the discrete problem becomes advection-dominated and spurious oscillations appear unless a stabilisation technique is employed (see Section 2.5).

The Courant-Friedrichs-Lewy (CFL) number quantifies how far the advected information travels within a single time step relative to the mesh size. More intuitively, it is the fraction of physical speed divided by the numerical speed. On a Cartesian grid with grid spacings in the k-th direction of h_k and velocity components v_k, the CFL number is given by the directional sum

$$\mathrm{CFL} = \sum_{k=1}^{d} \frac{|v_k|\,\Delta t}{h_k}. \tag{2.13}$$

On the unstructured triangular meshes used in this thesis, no preferred coordinate directions exist and the directional sum 2.13 cannot be evaluated as such. We therefore use the replacement

$$\mathrm{CFL} = \max_{\boldsymbol{x}\in\Omega} \frac{\|\boldsymbol{v}\|\,\Delta t}{h_{\min}},$$

where $h_{\min}$ is the smallest local edge length. This is a definition rather than a quoted theorem: $|v_k| \le \|\boldsymbol{v}\|$ for every k and $h_k \ge h_{\min}$ on a regular grid.

The Fourier number plays the analogous role for the diffusive part of the equation. The textbook one-dimensional bound for explicit Euler is $\mathrm{Fo} = D\Delta t/h^2 \le 1/2$ [14]. As with the CFL number, we define our adaptation to the case of an anisotropic full tensor D modelled on an unstructured mesh, yielding

$$\mathrm{Fo} = \max_{\boldsymbol{x}\in\Omega} \frac{\|D\|_F \Delta t}{h^2}, \tag{2.14}$$

where $\|D\|_F = \sqrt{\sum_{i,j} D_{ij}^2}$ is the Frobenius norm of the diffusion tensor, has two convenient properties. First, $\|D\|_F$ is an upper bound on the spectral radius $\rho(D) = \max_i|\lambda_i(D)|$ [14], so 2.14 dominates any directional diffusive time scale of the problem and is therefore conservative. Second, it reduces to the textbook scalar $D\Delta t/h^2$ for $D = kI$ in one space dimension. The precise form of the explicit-scheme stability constraints, including the way in which CFL and Fo couple, is derived in Section 2.4.

We subsequently convert our definitions of Pe, CFL and Fo into nodal quantities that can be evaluated at the mesh nodes of the graph and used to evaluate the stability of the surrogate model as a function of these dimensionless numbers. To this end, we introduce some notation: each node i we associate the set $\mathcal{N}(i)$ of neighbouring nodes connected to i by a mesh edge and define two local mesh sizes

$$h_i^{\max} := \max_{j\in\mathcal{N}(i)} \|\boldsymbol{x}_j - \boldsymbol{x}_i\|, \qquad h_i^{\min} := \min_{j\in\mathcal{N}(i)} \|\boldsymbol{x}_j - \boldsymbol{x}_i\|,$$

The former controls the coarsest local resolution (and hence the upper bound on the cell Péclet number) while the latter controls the tightest CFL and Fourier constraints. For the diffusion coefficient we use the streamline diffusivity [32] at node i,

$$\kappa_s(\boldsymbol{x}_i) := \frac{\boldsymbol{v}(\boldsymbol{x}_i)^\top D(\boldsymbol{x}_i)\, \boldsymbol{v}(\boldsymbol{x}_i)}{\|\boldsymbol{v}(\boldsymbol{x}_i)\|^2}, \tag{2.15}$$

which measures the strength of diffusion in the local flow direction. This is the quantity that competes directly with the advective flux and therefore matches the role of D in equation 2.12. The corresponding graph-based dimensionless numbers are

$$\begin{aligned}
\mathrm{Pe} &= \max_{i\in V} \frac{\|\boldsymbol{v}(\boldsymbol{x}_i)\|\, h_i^{\max}}{2\,\kappa_s(\boldsymbol{x}_i)}, \\
\mathrm{CFL} &= \max_{i\in V} \frac{\|\boldsymbol{v}(\boldsymbol{x}_i)\|\,\Delta t}{h_i^{\min}}, \\
\mathrm{Fo} &= \max_{i\in V} \frac{\|D(\boldsymbol{x}_i)\|_F\,\Delta t}{(h_i^{\min})^2},
\end{aligned}$$

where V are the nodes of the graph $G = (V, E)$. We formally introduce the graph later in Section 3.1, but knowing that we are referring to graph nodes is sufficient for the current discussion. Once again, it is worth noting that these graph definitions, which will be more compatible with the surrogate integrator setting, can be seen as upper bounds of the former, more classical definitions.

2.4 Time-stepping schemes

The Galerkin discretisation of Section 2.2 reduces the advection-diffusion equation to the linear stiff system of ordinary differential equations $M\dot{\boldsymbol{u}} + A\,\boldsymbol{u} = \boldsymbol{0}$ of equation 2.11. To obtain a fully discrete scheme this system has to be integrated in time. The four numerical integrators that appear later in this thesis are forward Euler, backward Euler, Crank-Nicolson and the classical four-stage Runge-Kutta method (RK4). The aim of this section is to introduce these schemes, fix the notion of stability that we will use, and derive their stability bounds in the one-dimensional constant-coefficient setting of [14, 31]: the model problem $\partial_t u + v\,\partial_x u = D\,\partial_{xx} u$ with $v, D > 0$ constant, on a uniform Cartesian grid with central finite differences. The actual setting of this thesis is of course more involved (anisotropic diffusion, spatially varying velocity, unstructured mesh, continuous-Galerkin FEM). They serve as an illustrative example of how the stability of a time-stepping scheme can be fully characterised by the dimensionless numbers as introduced in Section 2.3, and as a motivation for the empirical stability analysis that will be performed in Chapter 5.

2.4.1 Definition of stability

We partition $(0, T]$ into N_t uniform sub-intervals of length $\Delta t = T/N_t$ and approximate $\boldsymbol{u}(t_n)$ by $\boldsymbol{u}^n$, with $\boldsymbol{u}(t_0) = \boldsymbol{u}_0$. It is convenient first to write the semi-discrete system $M\dot{\boldsymbol{u}} + A\boldsymbol{u} = \boldsymbol{0}$ in the explicit form $\dot{\boldsymbol{u}} = \mathscr{H}\boldsymbol{u}$, where the system matrix is

$$\mathscr{H} := -M^{-1}A.$$

A linear timestepping scheme applied to this system can always be cast as

$$\boldsymbol{u}^{n+1} = G(\Delta t)\,\boldsymbol{u}^n, \tag{2.16}$$

where the amplification matrix $G(\Delta t)$ depends on Δt and $\mathscr{H}$ but not on $\boldsymbol{u}^n$. A definition on stability, can then be characterised as in [30], and is given by

Definition 2.4.1 (Absolute stability)**.** The scheme 2.16 is absolutely stable for the step size Δt if its stability function $R : \mathbb{C} \to \mathbb{C}$, defined by replacing $\mathscr{H}$ in $G(\Delta t)$ by the scalar λ, satisfies

$$|R(\Delta t\,\lambda)| \le 1 \qquad \forall \lambda \in \sigma(\mathscr{H}), \tag{2.17}$$

where $\sigma(\mathscr{H})$ denotes the spectrum of $\mathscr{H}$. The set $S := \{z \in \mathbb{C} : |R(z)| \le 1\}$ is the stability region of the scheme.

Combined with consistency, condition 2.17 yields convergence of a perturbation at the order of the scheme by the Lax equivalence theorem [31]. That conversely means that when this condition is violated, the scheme is unstable and errors are amplified each iteration and will become unbounded: a so-called blow-up.

Forward Euler, backward Euler and Crank-Nicolson are obtained directly from equation 2.16 by approximating $\dot{\boldsymbol{u}}$ at t_n, at t_{n+1}, and at the midpoint, respectively [30]. The stability functions and stability regions are given by

$$\begin{aligned}
&\text{Forward Euler:} && \boldsymbol{u}^{n+1} = (I + \Delta t\,\mathscr{H})\,\boldsymbol{u}^n, && R_{\mathrm{FE}}(z) = 1 + z,\\
&\text{Backward Euler:} && \boldsymbol{u}^{n+1} = (I - \Delta t\,\mathscr{H})^{-1}\,\boldsymbol{u}^n, && R_{\mathrm{BE}}(z) = \frac{1}{1-z},\\
&\text{Crank-Nicolson:} && \boldsymbol{u}^{n+1} = \big(I - \tfrac{\Delta t}{2}\mathscr{H}\big)^{-1}\big(I + \tfrac{\Delta t}{2}\mathscr{H}\big)\boldsymbol{u}^n, && R_{\mathrm{CN}}(z) = \frac{1+z/2}{1-z/2}.
\end{aligned}$$

The classical four-stage Runge-Kutta method is given by the following scheme. Given $\boldsymbol{u}^n$ and the right-hand side $\boldsymbol{f}(\boldsymbol{u}) = \mathscr{H}\boldsymbol{u}$, the classical RK4 step computes

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{u}^n), & \boldsymbol{k}_2 &= \boldsymbol{f}\big(\boldsymbol{u}^n + \tfrac{\Delta t}{2}\boldsymbol{k}_1\big),\\
\boldsymbol{k}_3 &= \boldsymbol{f}\big(\boldsymbol{u}^n + \tfrac{\Delta t}{2}\boldsymbol{k}_2\big), & \boldsymbol{k}_4 &= \boldsymbol{f}\big(\boldsymbol{u}^n + \Delta t\,\boldsymbol{k}_3\big),\\
\boldsymbol{u}^{n+1} &= \boldsymbol{u}^n + \tfrac{\Delta t}{6}\big(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\big).
\end{aligned}$$

For the linear right-hand side $\boldsymbol{f}(\boldsymbol{u}) = \mathscr{H}\boldsymbol{u}$ successive substitution of the stages collapses the update to $\boldsymbol{u}^{n+1} = R_{\mathrm{RK4}}(\Delta t\,\mathscr{H})\,\boldsymbol{u}^n$ with

$$R_{\mathrm{RK4}}(z) = 1 + z + \tfrac{z^2}{2} + \tfrac{z^3}{6} + \tfrac{z^4}{24},$$

i.e. the truncated Taylor expansion of e^z to fourth order, which is also the order of consistency of the scheme. The corresponding stability regions are summarised in Table 2.1 and visualised in Figure 2.3.

Table 2.1: Stability functions and stability regions of the four schemes considered in this thesis.

Scheme	Stability function $R(z)$	Stability region S
Forward Euler	$1 + z$	disc $\lvert 1 + z\rvert \le 1$ (radius 1, centre -1)
Backward Euler	$1/(1 - z)$	exterior of disc $\lvert 1 - z\rvert \le 1$, A-stable
Crank-Nicolson	$(1 + z/2)/(1 - z/2)$	closed left half-plane $\mathrm{Re}\, z \le 0$, A-stable
RK4	$1 + z + z^2/2 + z^3/6 + z^4/24$	bounded; intersects the negative real axis at ≈ -2.785

Backward Euler and Crank-Nicolson are A-stable: their stability regions contain the entire closed left half-plane, so condition 2.17 is satisfied for every $\Delta t > 0$ regardless of the spectrum of $\mathscr{H}$. Forward Euler and RK4 are conditionally stable: their stability regions are bounded, so Δt must be small enough that $\Delta t\,\sigma(\mathscr{H}) \subset S$.

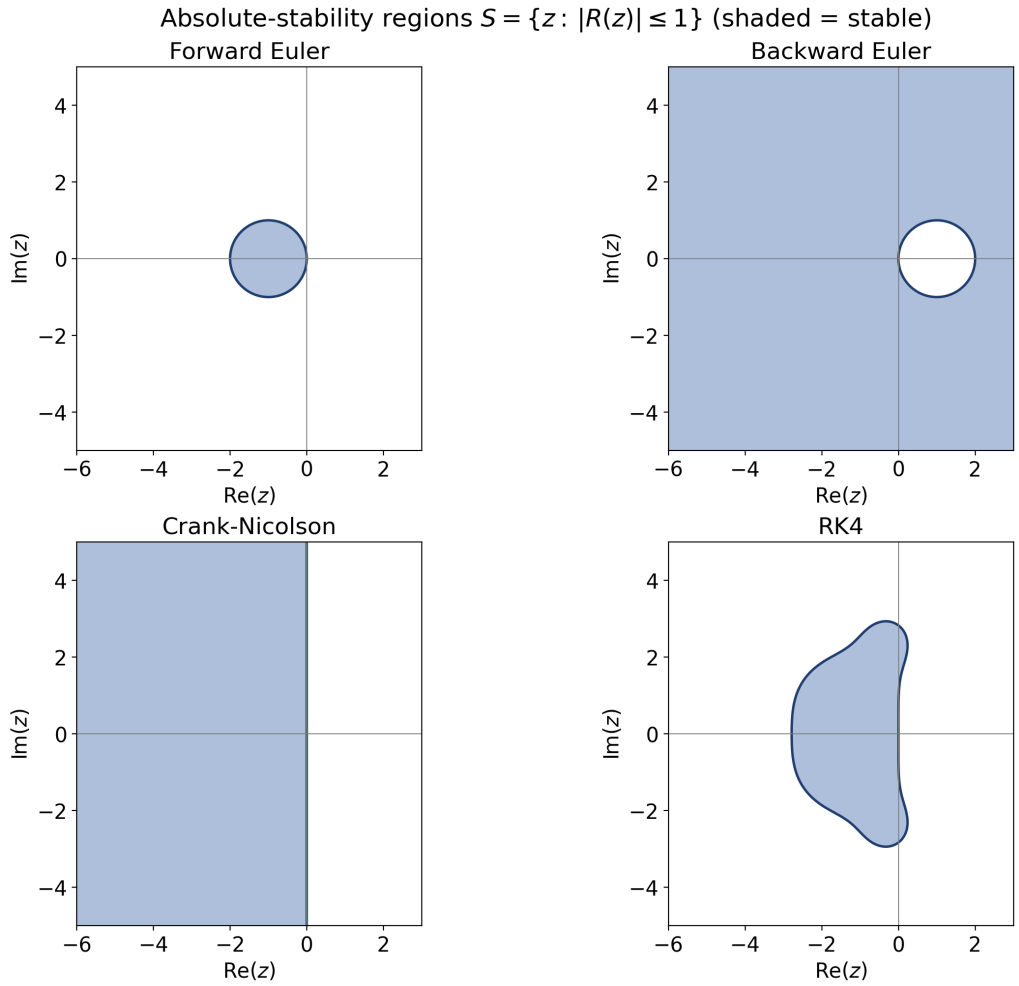


Figure 2.3: Absolute-stability regions $S = \{z \in \mathbb{C} : |R(z)| \le 1\}$ of the four time-stepping schemes in the complex plane $z = \Delta t\,\lambda$. The shaded area marks where $|R(z)| \le 1$, so that a scheme is absolutely stable for a given Δt whenever the scaled spectrum $\Delta t\,\sigma(\mathscr{H})$ lies inside the shaded region. Backward Euler and Crank-Nicolson are A-stable (the whole left half-plane is stable), whereas forward Euler and RK4 have bounded stability regions.

2.4.2 Stability bounds of special case in 1D for forward Euler

To illustrate how the stability relates to the dimensionless numbers, we consider the one-dimensional constant-coefficient advection-diffusion equation

$$\partial_t u + v\,\partial_x u = D\,\partial_{xx} u, \qquad x \in \mathbb{R},\ t > 0, \tag{2.18}$$

with $v, D > 0$ constants, discretised on the uniform grid $x_j = jh$ with central differences in space:

$$\dot{u}_j(t) = -\frac{v}{2h}\big(u_{j+1} - u_{j-1}\big) + \frac{D}{h^2}\big(u_{j+1} - 2u_j + u_{j-1}\big). \tag{2.19}$$

Throughout this subsection we use the classical scalar definitions $\mathrm{CFL} = v\Delta t/h$ and $\mathrm{Fo} = D\Delta t/h^2$ from equations 2.13–2.14. We will extract two complementary stability conditions from forward Euler applied to 2.19: a pointwise positivity condition that guarantees ℓ^∞ stability (no spurious extrema), and the standard von Neumann condition that guarantees ℓ^2 stability of every Fourier mode [14].

Discretising the time derivative in 2.19 by forward Euler, $\dot{u}_j(t) \approx (u_j^{n+1} - u_j^n)/\Delta t$, gives

$$u_j^{n+1} = u_j^n + \Delta t\left[-\frac{v}{2h}\big(u_{j+1}^n - u_{j-1}^n\big) + \frac{D}{h^2}\big(u_{j+1}^n - 2u_j^n + u_{j-1}^n\big)\right].$$

Substituting the scalar definitions $\mathrm{CFL} = v\Delta t/h$ and $\mathrm{Fo} = D\Delta t/h^2$ and collecting separately the terms

that multiply u_{j-1}^n, u_j^n and u_{j+1}^n gives

$$u_j^{n+1} = \underbrace{\left(\mathrm{Fo} + \tfrac{1}{2}\mathrm{CFL}\right)}_{c_{-1}} u_{j-1}^n + \underbrace{\left(1 - 2\,\mathrm{Fo}\right)}_{c_0} u_j^n + \underbrace{\left(\mathrm{Fo} - \tfrac{1}{2}\mathrm{CFL}\right)}_{c_{+1}} u_{j+1}^n.$$

The continuous problem in Equation 2.18 obeys a maximum principle: diffusion smears extrema and advection merely translates the profile, so u never exceeds its initial range. The discrete update inherits this property only if every $c_k \geq 0$, since then u_j^{n+1} is a convex combination of its stencil neighbours and $\min_k u_{j+k}^n \leq u_j^{n+1} \leq \max_k u_{j+k}^n$ [14]. A negative coefficient instead lets a single neighbouring spike drive u_j^{n+1} outside this range, producing the spurious oscillations that the continuous equation forbids. The conditions $c_{-1}, c_0, c_{+1} \geq 0$ read

$$\mathrm{Fo} \leq \tfrac{1}{2}, \qquad \mathrm{CFL} \leq 2\,\mathrm{Fo},$$

Which is equivalent to saying

$$\mathrm{CFL} \leq 1, \qquad \mathrm{Fo} \leq \tfrac{1}{2},$$

($c_{-1} \geq 0$ is automatic for $v, D > 0$). Also note that $\mathrm{CFL} \leq 2\,\mathrm{Fo}$ is equivalent to

$$2\frac{\mathrm{CFL}}{\mathrm{Fo}} = \frac{vh}{D} = \mathrm{Pe}_h \leq 1,$$

meaning that the mesh must be fine enough that diffusion locally dominates advection, otherwise the downwind coefficient turns negative and the scheme produces wiggles. This is the classical motivation for upwinding and SUPG-type stabilisation [31]. On the present central-difference grid it is the binding constraint whenever advection is strong. These limits illustrate how, for this illustrative simple problem, the stability of forward Euler is fully determined by the dimensionless parameters CFL, Fo, and Pe_h.

2.5 SUPG stabilisation

When the mesh Péclet number of Section 2.3 exceeds one, meaning advection is relatively more prominent than diffusion, the standard Galerkin discretisation of the advection-diffusion equation 2.8 becomes unstable: the discrete solution develops globally supported, non-physical oscillations. The origin of these oscillations is well understood, looking at the stability analysis of the previous subsection. A stabilisation term is therefore added to the weak form that introduces upwind-biased dissipation in the direction of the flow without compromising the consistency of the scheme. We use the streamline-upwind Petrov-Galerkin (SUPG) method of Brooks and Hughes [32], which is the residual-based stabilisation employed when creating the simulation data on which the model is trained.

The general residual-based stabilisation framework adds to the Galerkin weak form a term that is proportional, element-wise, to the strong-form residual $R(u_h)$ tested against a chosen perturbation $P(w_h)$ of the test function. With $f \equiv 0$ in our setting, the stabilised problem reads: find $u_h(\cdot, t) \in \mathscr{S}_h$ such that

$$\frac{d}{dt}(u_h, w_h)_{L^2(\Omega)} + B(u_h, w_h) + \sum_{K \in \mathscr{T}_h} \tau_K \int_K R(u_h)\, P(w_h)\, d\Omega = 0 \qquad \forall\, w_h \in \mathscr{S}_h, \tag{2.20}$$

where $\tau_K \geq 0$ is an element-wise stabilisation parameter with the dimension of time. Consistency is immediate: if $u_h = u$ then $R(u_h) \equiv 0$ and the extra term vanishes, so equation 2.20 reduces to the unmodified weak form. Different choices of $P(w_h)$ yield different classical schemes; the SUPG method corresponds to

$$P(w_h) = \boldsymbol{v} \cdot \nabla w_h, \tag{2.21}$$

which biases the test function along the flow direction.

Combining equations 2.20 and 2.21, the SUPG-stabilised semi-discrete problem is: find $u_h(\cdot, t) \in \mathscr{S}_h$ such that for every $w_h \in \mathscr{S}_h$,

$$\frac{d}{dt}(u_h, w_h)_{L^2(\Omega)} + B(u_h, w_h) + \sum_{K \in \mathscr{T}_h} \tau_K \int_K R(u_h)\, (\boldsymbol{v} \cdot \nabla w_h)\, d\Omega = 0.$$

On the space $\mathbb{P}_1(\mathcal{T}_h)$ of continuous piecewise-linear elements adopted in this thesis, the diffusive part of the strong-form residual vanishes element-wise, because a linear function has zero second derivatives on the interior of each triangle. Consequently $R(u_h)|_K = \partial_t u_h + \boldsymbol{v} \cdot \nabla u_h$ on each K, which means no evaluation of second derivatives is needed to compute the stabilisation term.

The stabilisation parameter τ controls the magnitude of the added streamline diffusion and must be chosen carefully. The classical one-dimensional analysis of Brooks and Hughes [32] shows that

$$\tau = \frac{h}{2\|\boldsymbol{v}\|}\,\xi(\mathrm{Pe}_h), \qquad \xi(\mathrm{Pe}_h) := \coth(\mathrm{Pe}_h) - \frac{1}{\mathrm{Pe}_h}, \tag{2.22}$$

with $\mathrm{Pe}_h = \|\boldsymbol{v}\|h/(2D)$ the classical cell Péclet number of equation 2.12, renders the discrete scheme nodally exact: the discrete solution coincides with the exact continuous solution at every mesh node. The function $\xi(\mathrm{Pe}_h)$ is doubly asymptotic, with $\xi \sim \mathrm{Pe}_h/3$ as $\mathrm{Pe}_h \to 0$ and $\xi \to 1$ as $\mathrm{Pe}_h \to \infty$. As a result, the added streamline diffusion automatically vanishes in the diffusion-dominated regime and saturates to a fixed fraction of $h\|\boldsymbol{v}\|/2$ in the advection-dominated regime, which is precisely the behaviour one expects of a well-designed stabilisation term. While nodal exactness is a property for the specific setting of the paper of Brooks and Hughes and therefore does not hold in general, the stabilisation parameter is designed with the aim of doing so. This is useful considering that the surrogate model evaluates the solution at the mesh nodes, so that the surrogate is trained on targets that are as close as possible to the exact solution.

On the unstructured, anisotropic meshes used in this thesis, the scalar D in equation 2.22 is replaced by the streamline diffusivity κ_s of equation 2.15, and h is replaced by a local element size, yielding the element-wise τ_K used in the implementation:

$$\tau_K = \frac{h_K}{2\|\boldsymbol{v}\|}\,\xi\!\left(\frac{\|\boldsymbol{v}\|h_K}{2\kappa_s}\right). \tag{2.23}$$

The effect of this stabilisation is illustrated in Figure 2.4, which contrasts the unstabilised Galerkin discretisation with the SUPG-stabilised discretisation in an advection-dominated regime.

2.6 Multigrid methods

We do not use multigrid as an actual solver in this work, but its underlying philosophy directly motivates the multi-scale architecture of the surrogate simulator developed in the next chapter. In particular, the way coarse representations are constructed and the role of the prolongation and restriction operators reappear, in a learned form, in our surrogate model later. The aim of this section is therefore not to give an exhaustive treatment, but to fix the ideas and notation that we will reuse later. For this section, the main sources are [33–36].

In the most general setting, multigrid methods solve a linear system of the form

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

with $A \in \mathbb{R}^{n\times n}$, $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n$. Each implicit time step of Section 2.4 (backward Euler or Crank–Nicolson) requires the solution of exactly such a linear system, so advancing the numerical integrator in time ultimately comes down to solving $A\boldsymbol{x} = \boldsymbol{b}$ repeatedly. Multigrid is in principle just one more (and rather powerful) candidate for this task, attractive both for its computational cost and convergence properties.

The key observation behind any multigrid method is that simple iterative solvers behave very differently on different error frequencies. A smoother (or relaxation method) is a cheap iterative method that is applied for only a few iterations: its purpose is not to solve the system, but to damp specific components of the error. A classical pointwise smoother such as Jacobi or Gauss-Seidel [37] reduces the high-frequency components of the error quickly, but is essentially blind to the low-frequency, geometrically smooth ones.

The remedy is to represent those smooth components on a coarser grid, where they look oscillatory again, and to correct them there. Concretely, one applies a few smoothing steps on the fine level, restricts the resulting residual to a coarser level by means of a restriction operator R, solves a smaller system for the coarse-grid correction, and prolongates this correction back to the fine level via an operator P. Applying this idea recursively across several levels yields the well-known V- and W-cycles.

In what follows we focus on algebraic multigrid (AMG), since it is the variant whose data-driven analogue we will eventually build. The components of interest are the construction of the coarser graphs and the choice of R and P, of which we will later define learned counterparts.

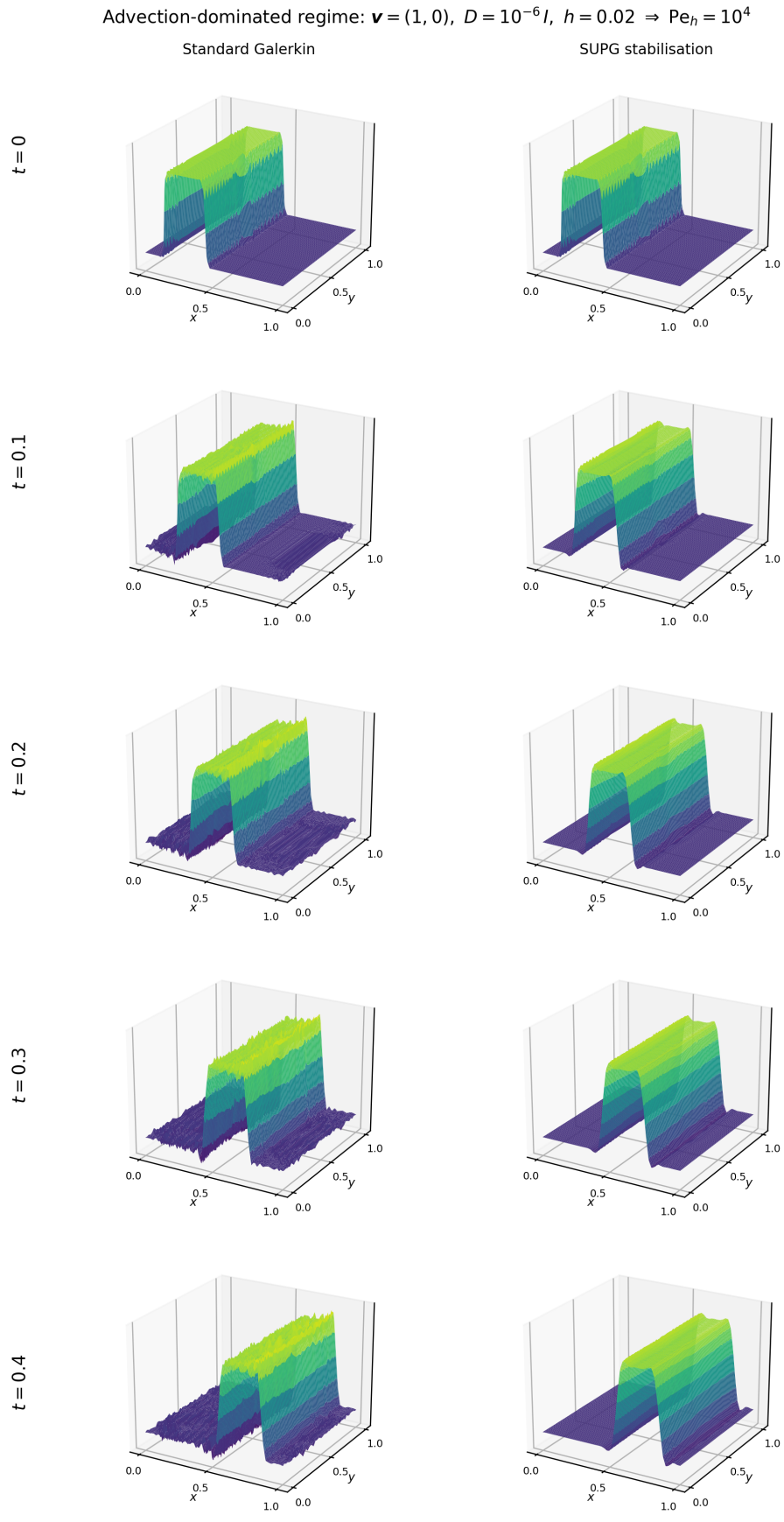


Figure 2.4: Time evolution of the standard Galerkin (left column) and SUPG-stabilised (right column) discretisations of the advection-diffusion equation in an advection-dominated regime. A smoothed top-hat profile is transported by a constant velocity field $\boldsymbol{v} = (1, 0)^\top$ in the presence of very weak isotropic diffusion $D = 10^{-6} I$, so that the mesh Péclet number of Equation 2.12 is of order 10^4. Rows correspond to the time instances $t \in \{0, 0.1, 0.2, 0.3, 0.4\}$. The unstabilised Galerkin discretisation is essentially oscillation-free at $t = 0$ but develops progressively larger, globally supported spurious oscillations as time advances, whilst the SUPG-stabilised solution using the τ of equation 2.23 transports the profile with the same speed and remains free of oscillations throughout the simulation.

2.6.1 Algebraic multigrid method (AMG)

The exposition below follows the review of [38] together with the textbook treatment in [33]. Multigrid methods are usually divided into two families: geometric and algebraic. Geometric multigrid relies on a coarsening hierarchy that is dictated by the underlying mesh: one fixes a coarsening pattern beforehand (for instance, halving the mesh size in each spatial direction) and uses simple geometric interpolation. The price for this simplicity is that the smoother must reduce the error to something that varies geometrically smoothly between the fine grid and its coarse subset. For certain problems, such as solving these systems for unstructured meshes, this is hard to guarantee with a cheap pointwise smoother. Then one has to resort to more complex relaxation schemes such as line or block smoothers, which somewhat undermines the appeal of the multigrid approach.

Algebraic multigrid takes the opposite point of view. Instead of designing a smoother that fits a prescribed coarsening, one fixes a simple smoother (typically Gauss-Seidel) and then constructs the coarsening, the restriction and the prolongation in such a way that they cooperate well with that smoother. Crucially, this construction is based purely on the entries of the fine scale matrix A_h. This means that no geometric information enters. As a consequence, AMG can be applied to systems that have no obvious geometric background at all, which is one of the main reasons it is so widely used as a black-box solver.

In classical AMG the coarse-level variables are taken as a subset of the fine-level ones. We denote the fine-level index set by $\mathscr{I}_h$ and the coarse-level index set by $\mathscr{I}_H$. The fine-level index set is split into two disjoint subsets,

$$\mathscr{I}_h = C \cup F,$$

where C collects the indices that are kept on the coarse level (and so are shared with $\mathscr{I}_H$) and F collects the remaining fine-only indices. We restrict ourselves to a two-level setting with a fine level h and a coarse level H. The multilevel method is obtained by applying the same construction recursively to the coarse-level system.

The fine-level equation reads

$$A_h \boldsymbol{x}_h = \boldsymbol{b}_h,$$

where A_h is the system matrix A of the linear system above, equipped with a subscript to indicate that it lives on the finest level h. Analogously, on the coarse level

$$A_H \boldsymbol{x}_H = \boldsymbol{b}_H.$$

The coarse-level operator A_H is not assembled from a coarse discretisation of the original PDE, but is obtained via restriction and prolongation operators, which possess the Galerkin property:

$$A_H = R\, A_h\, P,$$

with $R \in \mathbb{R}^{n_H \times n_h}$ the restriction and $P \in \mathbb{R}^{n_h \times n_H}$ the prolongation. For a symmetric positive-definite A_h it is standard to take $R = P^T$, which guarantees that A_H inherits symmetry and positive definiteness from A_h.

A smoothing step on the fine level is a relaxation of the form

$$\boldsymbol{x}_h \to \hat{\boldsymbol{x}}_h = \boldsymbol{x}_h + M_h^{-1}\big(\boldsymbol{b}_h - A_h \boldsymbol{x}_h\big),$$

where M_h is a cheaply invertible approximation of A_h (for Gauss–Seidel, $M_h = D_h + L_h$, with D_h and L_h the diagonal and strictly-lower-triangular parts of A_h). The associated error-propagation operator is

$$S_h := I_h - M_h^{-1} A_h,$$

so that, denoting the exact fine-level solution by $\boldsymbol{x}_h^*$, the error $\boldsymbol{e}_h = \boldsymbol{x}_h^* - \boldsymbol{x}_h$ transforms under one smoothing step as

$$\boldsymbol{e}_h \to S_h\, \boldsymbol{e}_h,$$

which makes explicit that the role of the smoother is to damp the components of $\boldsymbol{e}_h$ that lie in the high part of the spectrum.

Given a current iterate $\boldsymbol{x}_h^{\mathrm{old}}$, a two-level correction step then proceeds as follows. The fine-level residual

$$\boldsymbol{r}_h^{\mathrm{old}} = \boldsymbol{b}_h - A_h \boldsymbol{x}_h^{\mathrm{old}}$$

is restricted to the coarse level, $\boldsymbol{r}_H = R\,\boldsymbol{r}_h^{\text{old}}$, and the coarse-grid correction (error) $\boldsymbol{e}_H$ is obtained by solving the coarse-level system

$$A_H\,\boldsymbol{e}_H = \boldsymbol{r}_H.$$

This correction is finally prolongated back and applied on the fine level,

$$\boldsymbol{x}_h^{\text{new}} = \boldsymbol{x}_h^{\text{old}} + P\,\boldsymbol{e}_H.$$

A full two-level cycle consists of a few pre-smoothing sweeps, the coarse-grid correction described above, and a few post-smoothing sweeps. Applying this scheme recursively to the coarse system $A_H\boldsymbol{e}_H = \boldsymbol{r}_H$ yields the multilevel V- or W-cycle, depending on how often one descends to the coarsest level before going back up.

The efficiency of AMG relies on a delicate interplay between the smoother, the C/F-splitting and the restriction operator: the components of the error that the smoother leaves behind must be exactly the ones that the coarse-grid correction can resolve well [35]. So far we have assumed the C/F-splitting to be given. The algebraic construction of this splitting is one of the central building blocks of AMG, and is also the part that will guide the design of our own coarsening algorithm. We therefore describe the standard procedure in the next section.

2.6.2 The Ruge–Stüben coarsening algorithm

For the construction of the C/F-splitting we use the classical Ruge–Stüben algorithm [35, 38]. The algorithm is built on a heuristic characterisation of algebraically smooth error, which we briefly motivate before stating it.

Assume that A_h has been scaled so that its largest eigenvalue equals 1, and let $\boldsymbol{e}$ be a normalised eigenvector with associated eigenvalue λ. Then

$$\boldsymbol{e}^T A_h\,\boldsymbol{e} = \lambda\,\boldsymbol{e}^T\boldsymbol{e} = \lambda.$$

For symmetric matrices with vanishing row sums (which is, up to boundary contributions, typical for discretised second-order elliptic operators) one can rewrite this expression as

$$\boldsymbol{e}^T A_h\,\boldsymbol{e} = \sum_{i<j}(-a_{ij})(e_i - e_j)^2.$$

A small eigenvalue $\lambda \ll 1$ thus corresponds to an eigenvector for which the differences $e_i - e_j$ are small whenever the off-diagonal coefficient $-a_{ij}$ is large and positive. Since smooth error is essentially built up from such small eigenmodes, this leads directly to the central heuristic of classical AMG: algebraically smooth error varies slowly along the directions in which the negative off-diagonal coefficients of A_h are large in magnitude.

This heuristic translates into a notion of strength of connection. We say that variable i strongly depends on variable j if

$$-a_{ij} \;\geq\; \theta \max_{k\neq i}\bigl(-a_{ik}\bigr),$$

where $\theta \in (0,1]$ is a user-defined threshold. The set of variables on which i strongly depends is denoted

$$S_i = \{\, j \in N_i \;:\; -a_{ij} \geq \theta \max_{k\neq i}(-a_{ik})\,\},$$

with $N_i = \{\, j \neq i : a_{ij} \neq 0\}$ the algebraic neighbourhood of i. Note that strength of connection is in general not symmetric, even when A_h itself is. We define the transpose of the set S, which means we now collect j instead of i:

$$S_i^T = \{\, j \in \mathscr{I}_h \;:\; i \in S_j\,\},$$

collects the variables that strongly depend on i (i.e. those j for which i belongs to the dependency set S_j), and is the set that matters when deciding whether i should become a C-point: a good C-point is one that influences many other variables.

The algorithm proceeds in two passes. In the first pass, every variable starts out as undecided ($U = \mathscr{I}_h$, $C = F = \emptyset$) and a score

$$\mu_i = |S_i^T \cap U| + 2\,|S_i^T \cap F|$$

is computed for each $i \in U$. The score counts how many undecided neighbours strongly depend on i, plus twice the number of F-points that already do. Note that the second term reflects the fact that an F-point should preferably be surrounded by C-points from which it can interpolate, so a node that resolves several existing F-points is an attractive candidate. The variable with the largest score is added to C, all variables in $S_i^T \cap U$ are moved to F, and the scores of the remaining undecided variables are updated. The procedure is repeated until U is empty. Schematically:

1. Initialise $C = \emptyset$, $F = \emptyset$, $U = \mathscr{I}_h$, and compute μ_i for all $i \in U$.
2. While $U \neq \emptyset$:
   (a) pick $i \in U$ with maximal score μ_i and set $C \leftarrow C \cup \{i\}$, $U \leftarrow U \setminus \{i\}$;
   (b) for each $j \in S_i^T \cap U$, set $F \leftarrow F \cup \{j\}$, $U \leftarrow U \setminus \{j\}$;
   (c) update the scores μ_k for the remaining $k \in U$.

A second pass is then performed to repair the splitting where it is locally too coarse: whenever two strongly connected F-points do not share a common C-neighbour, one of them is promoted to a C-point so that the interpolation rules used by classical AMG remain well-defined [35]. The output is a C/F-splitting in which the C-points form an approximately maximal independent set with respect to strong connections, and in which every F-point has enough strongly connected C-neighbours to interpolate from. This splitting, together with the corresponding interpolation P, the restriction $R = P^T$ and the Galerkin coarse operator $A_H = RA_hP$, is what defines a single level of the AMG hierarchy.

From this construction, the surrogate model developed in the next chapter borrows two ingredients. The first is the C/F-splitting itself, used to obtain the coarse graph. We do not have a fine-level matrix A_h at our disposal, so the strength of connection cannot be read off from matrix entries. Instead, we replace it by a similarity score computed from the edge features and from differences in node features, which, somewhat ironically, reintroduces a geometric component into the otherwise purely algebraic framework. The Ruge–Stüben procedure is then applied with this score in place of $-a_{ij}$, and the resulting C-points, together with the F-points clustered around them, define both the nodes and the connectivity of the coarse graph.

The second ingredient is the pair of restriction and prolongation operators, which we replace by learned counterparts so that information can be propagated between scales. We also slightly relax their support: rather than restricting interpolation to direct C/F-neighbours as in classical AMG, we define dedicated up- and down-graphs that prescribe which fine and coarse nodes exchange information. The precise construction of these graphs and operators is deferred to the next chapter.

Chapter 3

Surrogate modelling framework

To understand the surrogate modelling problem, the necessary framework is introduced. To understand the framework, a literature review was conducted that informed the development of the surrogate modelling approach. This chapter establishes the fundamental building blocks of a GNN with a specific focus on the surrogate modelling application. It provides a rigorous mathematical definition of a graph, followed by the general architecture of a graph-based surrogate model and the learning paradigm for surrogate modelling. The chapter concludes with the introduction of the learned prolongation and restriction operators used for the multiscale architecture, which are central to the multigrid-inspired architecture of the GNNs used in this thesis.

3.1 Definition of a graph

The goal of surrogate simulators is to model PDEs over a set of nodes and edges that represent the physical domain. The graph may be a one-to-one copy of the mesh, as is done in this thesis, or it may be constructed differently. For instance, [39] selects random points over the domain and connects them based on proximity. Despite differences in graph construction, all such methods operate on a graph entity. It is therefore desirable to have a rigorous definition of a graph. The definition of [40] is adopted here, which formalises the identity of a graph and its associated features on which a GNN operates. This definition is general enough to encompass the various graph constructions used in surrogate modelling and can be described as follows:

Definition 3.1.1 (Attributed Graph)**.** Let $G = (V, E)$ denote a graph, where V is a set of nodes, and E is a set of edges connecting the nodes. Let $v_i \in V$ be a node and $e_{ij} = (v_i, v_j) \in E$ an edge pointing from v_i to v_j. We define the neighbourhood $N(v)$ of a node by the set $N(v) = \{u \in V | (u, v) \in E\}$ (i.e., all nodes that have an edge pointing to v). The adjacency matrix A is a matrix where $A_{ij} = 1$ if $e_{ij} \in E$ and 0 otherwise. A graph may have node features $\boldsymbol{X}^v = \begin{pmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \dots & \boldsymbol{v}_{|V|} \end{pmatrix}^T \in \mathbb{R}^{|V| \times n_v}$, edge features $\boldsymbol{X}^{\boldsymbol{e}} = \begin{pmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \dots & \boldsymbol{e}_{|E|} \end{pmatrix}^T \in \mathbb{R}^{|E| \times n_e}$ and a global graph feature vector $\boldsymbol{g} \in \mathbb{R}^{n_g \times 1}$, where n_v, n_e, n_g are the dimensions of the node, edge and global features respectively.

A symmetric adjacency matrix corresponds to a bidirectional graph, in which the presence of an edge $(i, j) \in E$ implies $(j, i) \in E$. It is worth noting that weighted adjacency matrices are not considered here; these represent the special case of graphs with one-dimensional edge features. The combination of a binary adjacency matrix with higher-dimensional edge features thus constitutes a generalisation of the weighted adjacency matrix. Since the adjacency matrix encodes only edge presence, the edge index matrix is employed in practice: a $2 \times |E|$ matrix, with integer entries, whose first row contains the source-node indices and whose second row contains the target-node indices, so that each of the $|E|$ edges contributes exactly one column. This representation enables efficient message passing in GNNs by providing direct access to neighbouring nodes and their associated edge features without requiring computation of the full adjacency matrix. Furthermore, since the graph represents a mesh, it has a sparse structure, and the use of the edge index matrix results in a smaller matrix than the full adjacency matrix, thereby reducing memory cost. The graph is undirected when the additional condition $\boldsymbol{e}_{ij} = \boldsymbol{e}_{ji}$ holds. An example of an attributed graph constituting a triangular unstructured mesh is shown in Figure 3.1, illustrating the node, edge, and global graph features together with the neighbourhood of a selected node.

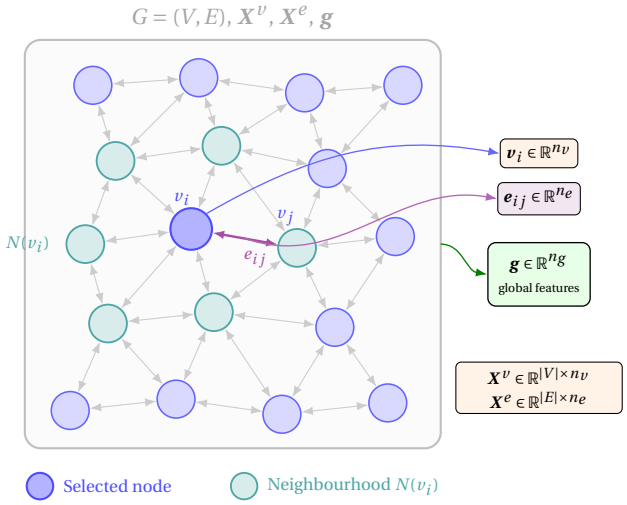


Figure 3.1: An attributed graph with node-, edge-, and global features representing an arbitrary unstructured triangular mesh. The selected node v_i and its neighbourhood $N(v_i)$ are highlighted. Global features concern the entire graph and are therefore denoted with a box encompassing the entire graph. The edge features are shown for one edge, but are present for all edges, same for the node features. Since the graph represents a mesh, it has a sparse structure and is at least bidirectional ($e_{ij} \in E \implies e_{ji} \in E$).

Definition 3.1.1 extends naturally to the spatiotemporal setting, where features evolve over time, yielding $G^{(t)} = (V, E, \boldsymbol{X}^v(t), \boldsymbol{X}^e(t), \boldsymbol{g}(t))$. In the surrogate modelling context, whether the graph is directed or undirected is determined by the choice of edge attributes, which frequently correspond to physical distances between nodes. The graph induced by a mesh is inherently bidirectional. Consequently, the graph is undirected provided $\boldsymbol{e}_{ij} = \boldsymbol{e}_{ji}$.

If the displacement vector between adjacent nodes is adopted as the edge attribute, the feature vector associated with edge (i, j) is no longer equal to that of its converse (j, i), resulting in a directed graph. Encoding only the scalar inter-node distance, by contrast, preserves symmetry and yields an undirected structure. In this work, the vectorised displacement between nodes is chosen as the edge representation, motivated by consistency with other features that inherently require a vectorised form, such as the vorticity field. This homogeneity is desirable from a machine learning perspective, as mixing data types tends to degrade model performance [41]. Moreover, [42] has shown that the representation using the norm of inter-nodal displacement vector is inferior to using the actual vector. As a consequence, the graph is directed. However, the edge attributes of (i, j) and (j, i) are not independent, as $\boldsymbol{e}_{ij} = -\boldsymbol{e}_{ji}$.

Given such a graph structure, learning tasks may be formulated at the node, edge, or graph level, and are often carried out in a semi-supervised regime [43], wherein only a subset of nodes or edges carries labels whilst the remainder remain unlabelled. This contrasts with the fully supervised setting, where samples are assumed to be independent and identically distributed (i.i.d.) [44]. This assumption does not universally hold in the graph setting, as individual nodes and edges may exhibit interdependent distributions due to the underlying graph topology. At the global graph level, however, samples may be treated as i.i.d., provided that all features and topologies are drawn from the same distribution.

In the surrogate modelling setting for Eulerian systems, the learning task is a node-level regression problem in which the model predicts the solution state at the next time step for all nodes, given the current node and edge features. At each time step t, all node features are fully observed as inputs, encompassing the physical fields, auxiliary quantities such as node type, and the current solution state, whilst the graph topology and edge attributes remain fixed throughout, as is characteristic of Eulerian systems. The model is then tasked with inferring the solution at time $t + dt$, constituting a structured prediction problem in which all graph components are fully specified except for a designated subset of node features at the target time step. The training data contains multiple graphs corresponding to different time steps or initial conditions. And since the physical parameters are drawn from the same distribution, the resulting training set consists of i.i.d. graphs with internally interdependent node and edge features. No global graph feature vector is employed, as all relevant information for the surrogate modelling task is already encapsulated in the node and edge attributes together with the graph structure.

3.2 Message passing

3.2.1 Forward pass through a message passing layer

When operating on graphs, there are generally two main streams of methods: spectral and spatial-based methods [40]. For surrogate timestepping, spatial-based methods align more naturally with the principal mechanisms of mathematical modelling, which exhibits spatiotemporal dependencies. Within spatial-based methods, two algorithms are commonly employed for learning local dynamics on graphs, namely graph convolution and message passing. Whilst convolution has proved successful in classification [45] and in finding steady-state solutions of PDEs [13], and there is some overlap between the two approaches, message passing has demonstrated greater efficacy in modelling spatiotemporal processes. Consequently, message passing is a recurring element of surrogate models such as those proposed in [42, 46, 47], owing to its natural alignment with the structure of PDEs: by aggregating information from local neighbourhoods, the mechanism enables the model to capture the spatially local interactions that govern the underlying physics. The message passing neural network (MPNN) was introduced by [48] in the context of quantum chemistry. A more general framework, which incorporates global graph features, was proposed earlier by [49]. Although global features could prove beneficial in future extensions, for instance to allow the model to handle variable time step sizes dt, their inclusion lies outside the scope of the present work and are therefore omitted from the present formulation. The first step of the message passing algorithm is the aggregation of information from the local neighbourhood, accomplished by aggregating the edge features that terminate at a vertex, which can be represented by the function

$$\rho_{e\to v}\left(\boldsymbol{e}'_{*k}\right) : \mathbb{R}^{|N(v_k)|\times n_v} \to \mathbb{R}^{n_v},$$

where $\boldsymbol{e}'_{*k}$ denotes updated edge features that terminate at vertex v_k, each of dimension n'_e. The notation $|N(v_k)|$ indicates that the number of input vectors to this aggregation function is finite but variable. This is one of two conditions for a valid aggregation function, the other being permutation invariance [50]. Common examples of such aggregation functions include summation, mean, and maximum. In this work, summation is adopted as the aggregation function, as it best mimics the combined in- and outflow of the solution state at a node driven by both advection and diffusion. The aggregation itself can carry trainable parameters, but will not so in the present work. The learnable part of a message passing layer mainly resides in the update functions

$$\phi_e(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j) : \mathbb{R}^{n_e} \times \mathbb{R}^{n_v} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n'_e},$$
$$\phi_v(\boldsymbol{v}_i, \bar{\boldsymbol{e}}_i) : \mathbb{R}^{n_v} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n'_v},$$

where n'_e, n'_v denote the number of output features and $\bar{\boldsymbol{e}}_{\boldsymbol{i}}$ denotes the aggregated edge features at vertex i. These update functions contain the trainable parameters of the layer, and are commonly realised as MLPs as described in Section A.1. These MLPs consist of multiple fully connected layers followed by a non-linear activation function. The resulting forward pass through a message passing layer is summarised in Algorithm 1. The computational graph in Figure 3.2 further visualises the message passing process. From the perspective of a single node, applying n message passing layers implies that information is aggregated from the n-hop neighbourhood, i.e., the set of nodes reachable by traversing at most n edges. Consequently, the receptive field of the model is bounded by the number of message passing layers, and interactions beyond n hops cannot be captured. This draws a parallel with explicit time integration schemes, of which these surrogate models are an instance, as both rely solely on the current state to advance to the next. Classical explicit methods are similarly constrained by the ratio of the numerical propagation speed to the physical speed, and are therefore unable to resolve interactions that propagate faster than the numerical speed. To complement the computational graph in Figure 3.2, Figure 3.3 visualises the three stages of message passing as they occur on an actual triangular mesh.

A natural comparison arises between MPNNs and CNNs, the latter of which is described in Section A.2. Both architectures capture local interactions: message passing aggregates neighbourhood information via learnable update functions, whilst CNNs detect local patterns through convolutional kernels. The message passing framework is considerably more flexible, however, as it accommodates variable input sizes and arbitrary graph topologies, whereas CNNs are confined to fixed, regular grid structures. The kernel size in a CNN directly bounds the spatial extent of interactions captured within a single layer, analogously to how the local neighbourhood determines the receptive field in a single message passing layer [51]; in both cases, repeated application of layers progressively extends the range of global interactions. CNNs have nonetheless been successfully employed as surrogate simulators, albeit necessarily

Algorithm 1 Making a forward pass through a message passing layer, given $\mathcal{G}$ and $\boldsymbol{X}^v$

procedure $Message\_Passing(\mathcal{G} = (V, E), \boldsymbol{X}^e, \boldsymbol{X}^v)$
  for $e_{ij} \in E$ **do**
    $\boldsymbol{e}'_{ij} \leftarrow \phi_e(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j)$
  end for
  for $k \in \{1, 2, \ldots, |V|\}$ **do**
    $\bar{\boldsymbol{e}}_k \leftarrow \rho_{e\rightarrow v}(\boldsymbol{e}'_{*k})$
    $\boldsymbol{v}_k \leftarrow \phi_v(\boldsymbol{v}_k, \bar{\boldsymbol{e}}_k)$
  end for
  $\boldsymbol{X}'^v = \{\boldsymbol{v}_k\}_{k=1:|V|}$
  $\boldsymbol{X}'^e = \{\boldsymbol{e}'_{ij}\}_{e_{ij}\in E}$
  return $(\boldsymbol{X}'^e, \boldsymbol{X}'^v)$
end procedure

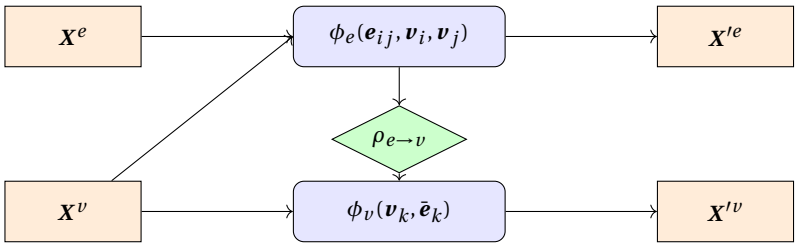


Figure 3.2: The computational graph describing a forward pass through a message passing layer.

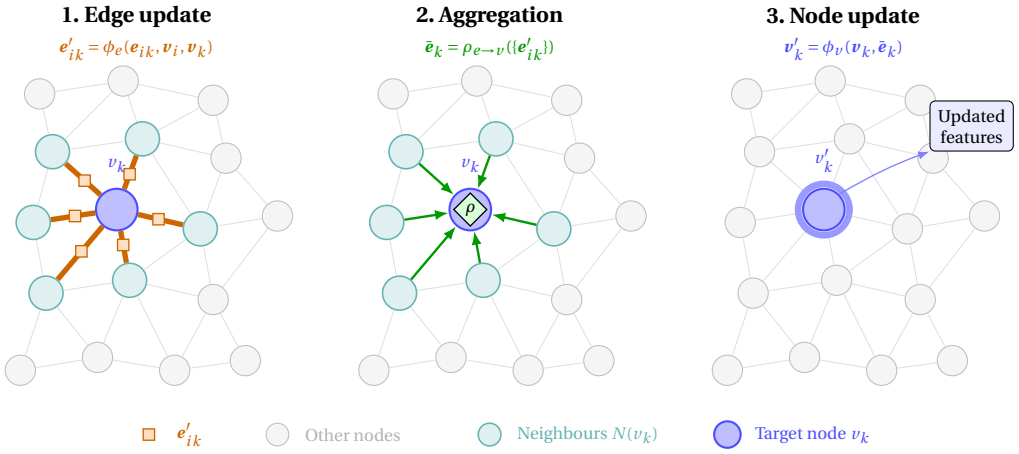


Figure 3.3: All the message passing operation on a graph representing an arbitrary triangular mesh related to a target node v_k (highlighted in blue). The left panel shows the edge update, where the edge features of the incoming edges to v_k are updated using the nodes connected to the edges and its current edge features. The middle panel shows the aggregation, where the updated incoming edge features are aggregated at v_k, which is by means of summation in the present work. The right panel shows the node update, where the aggregated edge features are combined with the current node features of v_k to produce the updated node features.

on regular grids, as demonstrated for CFD applications in [52, 53]. Moreover, to mimic multi-hop interactions, CNNs do not require multiple convolutional operations, but can do so via a larger kernel. This makes it easier for CNNs to model long-range dependencies over the spatial domain.

3.2.2 Gradients of a message passing layer

To update the trainable parameters with an optimisation method, it is necessary to derive the gradients through a single message passing layer by applying the chain rule to the computational graph in Figure 3.2. For reference, the forward pass from Algorithm 1 consists of the following three stages: for each

$(i,j) \in E$ and $k \in \{1,\dots,|V|\}$,

$$\boldsymbol{e}'_{ij} = \phi_e(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j; \boldsymbol{\theta}_e), \qquad \text{(edge update)} \tag{3.1}$$

$$\bar{\boldsymbol{e}}_k = \textstyle\sum_{i:(i,k)\in E} \boldsymbol{e}'_{ik}, \qquad \text{(aggregation)} \tag{3.2}$$

$$\boldsymbol{v}'_k = \phi_v(\boldsymbol{v}_k, \bar{\boldsymbol{e}}_k; \boldsymbol{\theta}_v). \qquad \text{(node update)} \tag{3.3}$$

Where we have already set the aggregation function to a sum, as this is the aggregation function used in the present work. Now let L be a general loss function that depends on the layer outputs $\{\boldsymbol{e}'_{ij}\}$ and $\{\boldsymbol{v}'_k\}$, and suppose the upstream gradients $\frac{\partial L}{\partial \boldsymbol{v}'_k}$ and $\frac{\partial L}{\partial \boldsymbol{e}'_{ij}}$ are known. [1] We propagate these backwards through (3.3), (3.2), and (3.1) in turn to obtain the gradients with respect to the trainable parameters $\boldsymbol{\theta}_v$, $\boldsymbol{\theta}_e$ and the layer inputs $\boldsymbol{v}_k$, $\boldsymbol{e}_{ij}$.

Since $\boldsymbol{\theta}_v$ is shared across all node updates, L depends on $\boldsymbol{\theta}_v$ through every $\boldsymbol{v}'_k$. The chain rule therefore gives

$$\frac{\partial L}{\partial \boldsymbol{\theta}_v} = \sum_{k=1}^{|V|} \frac{\partial L}{\partial \boldsymbol{v}'_k} \frac{\partial \boldsymbol{v}'_k}{\partial \boldsymbol{\theta}_v}.$$

The gradient with respect to $\boldsymbol{\theta}_e$ is more involved, because each $\boldsymbol{e}'_{ij}$ influences L via two paths in the computational graph: directly as a layer output, and indirectly through the aggregation $\bar{\boldsymbol{e}}_j$ into the node update of node j. The total gradient arriving at $\boldsymbol{e}'_{ij}$ is therefore

$$\boldsymbol{\delta}_{ij} := \frac{\partial L}{\partial \boldsymbol{e}'_{ij}} + \frac{\partial L}{\partial \bar{\boldsymbol{e}}_j} = \frac{\partial L}{\partial \boldsymbol{e}'_{ij}} + \frac{\partial L}{\partial \boldsymbol{v}'_j} \frac{\partial \boldsymbol{v}'_j}{\partial \bar{\boldsymbol{e}}_j}, \tag{3.4}$$

where the second equality applies the chain rule to the node update (3.3) for node j.

The parameters $\boldsymbol{\theta}_e$ enter the computation only through the edge updates (3.1). Since $\boldsymbol{\theta}_e$ is shared across all edges and the total gradient at each $\boldsymbol{e}'_{ij}$ is $\boldsymbol{\delta}_{ij}$, the chain rule gives

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\theta}_e} &= \sum_{(i,j)\in E} \boldsymbol{\delta}_{ij} \frac{\partial \boldsymbol{e}'_{ij}}{\partial \boldsymbol{\theta}_e} \\ &= \sum_{(i,j)\in E} \left(\frac{\partial L}{\partial \boldsymbol{e}'_{ij}} + \frac{\partial L}{\partial \boldsymbol{v}'_j} \frac{\partial \boldsymbol{v}'_j}{\partial \bar{\boldsymbol{e}}_j} \right) \frac{\partial \boldsymbol{e}'_{ij}}{\partial \boldsymbol{\theta}_e},\end{aligned}$$

where the second line substitutes (3.4). Note that the aggregation (3.2) does not appear explicitly because $\frac{\partial \bar{\boldsymbol{e}}_j}{\partial \boldsymbol{e}'_{ij}} = \boldsymbol{I}$ for a sum, which is absorbed into $\boldsymbol{\delta}_{ij}$.

The input $\boldsymbol{e}_{ij}$ affects L only through $\boldsymbol{e}'_{ij}$, at which the total gradient is $\boldsymbol{\delta}_{ij}$. By the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{e}_{ij}} = \boldsymbol{\delta}_{ij} \frac{\partial \boldsymbol{e}'_{ij}}{\partial \boldsymbol{e}_{ij}}.$$

The input feature $\boldsymbol{v}_k$ enters the computational graph at three locations: in its own node update (3.3), as the sender in every outgoing edge (k,j) in (3.1), and as the receiver in every incoming edge (i,k) in (3.1). We collect the gradient contribution from each path separately. The node update (3.3) contributes

$$\frac{\partial L}{\partial \boldsymbol{v}_k}\Big|_{\text{node}} = \frac{\partial L}{\partial \boldsymbol{v}'_k} \frac{\partial \boldsymbol{v}'_k}{\partial \boldsymbol{v}_k}. \tag{3.5}$$

Each outgoing edge (k,j) produces $\boldsymbol{e}'_{kj} = \phi_e(\boldsymbol{e}_{kj}, \boldsymbol{v}_k, \boldsymbol{v}_j; \boldsymbol{\theta}_e)$, where $\boldsymbol{v}_k$ appears as the sender. The total gradient at $\boldsymbol{e}'_{kj}$ is $\boldsymbol{\delta}_{kj}$, so each such edge contributes $\boldsymbol{\delta}_{kj} \frac{\partial \boldsymbol{e}'_{kj}}{\partial \boldsymbol{v}_k}$. Summing over all outgoing edges gives

$$\frac{\partial L}{\partial \boldsymbol{v}_k}\Big|_{\text{send}} = \sum_{j:(k,j)\in E} \boldsymbol{\delta}_{kj} \frac{\partial \boldsymbol{e}'_{kj}}{\partial \boldsymbol{v}_k}. \tag{3.6}$$

[1] Throughout, inputs and outputs are assumed to be row vectors, which is more consistent with the batched learning framework. Strictly speaking, the derivations below are Jacobians, but they are referred to as gradients for simplicity.

By the same reasoning, each incoming edge (i,k) contributes $\boldsymbol{\delta}_{ik} \frac{\partial \boldsymbol{e}'_{ik}}{\partial \boldsymbol{v}_k}$, where $\boldsymbol{v}_k$ now appears as the receiver:

$$\frac{\partial L}{\partial \boldsymbol{v}_k}\Big|_{\text{recv}} = \sum_{i:(i,k)\in E} \boldsymbol{\delta}_{ik} \frac{\partial \boldsymbol{e}'_{ik}}{\partial \boldsymbol{v}_k}. \tag{3.7}$$

Summing all three contributions yields

$$\frac{\partial L}{\partial \boldsymbol{v}_k} = \underbrace{\frac{\partial L}{\partial \boldsymbol{v}'_k} \frac{\partial \boldsymbol{v}'_k}{\partial \boldsymbol{v}_k}}_{(3.5)} + \underbrace{\sum_{j:(k,j)\in E} \boldsymbol{\delta}_{kj} \frac{\partial \boldsymbol{e}'_{kj}}{\partial \boldsymbol{v}_k}}_{(3.6)} + \underbrace{\sum_{i:(i,k)\in E} \boldsymbol{\delta}_{ik} \frac{\partial \boldsymbol{e}'_{ik}}{\partial \boldsymbol{v}_k}}_{(3.7)}.$$

This derivation illustrates how the gradients of nodes and edges are mutually dependent, and that the node and edge components of the algorithm complement each other throughout the backpropagation process. It also highlights the inherently topological nature of message passing and why it presents a challenge for parallelisation: the gradient at a node depends on the gradients of its neighbours, which in turn depend on their own neighbours, and so on. This is one of the reasons why training GNNs can be computationally expensive, particularly for large graphs.

3.3 Layer normalisation and oversmoothing

3.3.1 Forward pass through the normalisation layer

In the context of GNNs, but also in other neural network architectures, layer normalisation is a technique used to stabilise and accelerate the training process by normalising the activations of each layer. Moreover, together with the use of residual connections in the update functions, it can help to mitigate oversmoothing. Oversmoothing, as defined in [54], refers to the tendency of node representations to become increasingly similar as the number of message passing layers grows. It can be quantified through the Dirichlet energy of the node features $\boldsymbol{X}^l$ at layer l,

$$E(\boldsymbol{X}^l) = \sum_{(i,j)\in E} \|\boldsymbol{x}_i^l - \boldsymbol{x}_j^l\|^2,$$

where a collapse of $E(\boldsymbol{X}^l)$ towards zero with increasing depth signals that neighbouring nodes have become indistinguishable. The theoretical analysis in [55] shows that residual connections prevent the signal from being too smoothed out, while normalisation layers prevent the signal from collapsing to a one-dimensional subspace. So both are necessary to prevent oversmoothing, and the normalisation layer is therefore an integral part of the surrogate architecture.

We use the layer normalisation as given in [56], which applies normalisation across the feature dimension of each individual node or edge representation, such that there is no dependency on the batch size. This is particularly beneficial for GNNs, as the number of nodes and edges can vary across different graphs, making batch normalisation less effective. For an arbitrary feature vector $\boldsymbol{h} \in \mathbb{R}^d$, we first compute the mean μ and variance σ as follows:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} h_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(h_i-\mu)^2}.$$

Then the normalised output of the layer becomes:

$$\text{LayerNorm}(\boldsymbol{h}) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{h}-\mu}{\sigma} + \boldsymbol{\beta},$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters that allow the model to scale and shift the normalised output.

3.3.2 Gradients of the normalisation layer

Although the backward pass is less involved than the message passing algorithm, it is included here for completeness. Suppose a loss function L is given and the upstream gradients $\frac{\partial L}{\partial \boldsymbol{y}}$ are available, where

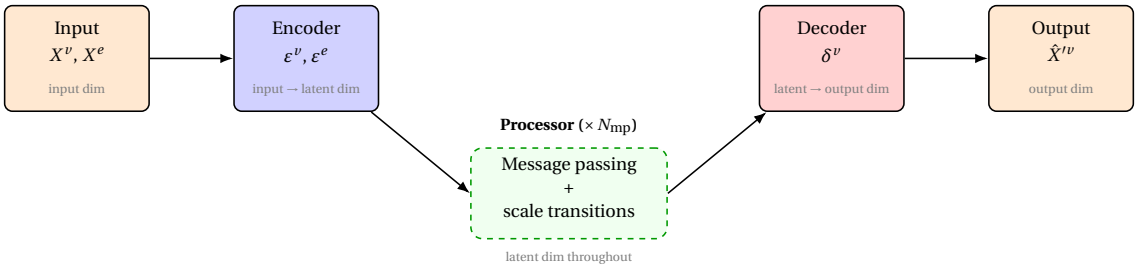


Figure 3.4: Computational graph of the encode-process-decode structure used in GNN surrogate modelling. Each box shows the operation and its data dimensionality. The encoders ε^v and ε^e map raw input features to a shared latent dimension. The dashed Processor box applies N_{mp} message-passing blocks (with interleaved scale transitions) entirely within the latent dimension. The decoder δ^v projects latent node features back to the output dimension to yield the predicted update $\hat{X}'^v$.

$\boldsymbol{y} = \text{LayerNorm}(\boldsymbol{h})$. The gradients of L with respect to the input $\boldsymbol{h}$ and the learnable parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ can then be computed as follows:

$$\frac{\partial L}{\partial \boldsymbol{h}} = \frac{1}{\sigma}\frac{\partial L}{\partial \boldsymbol{y}} \odot \boldsymbol{\gamma},$$
$$\frac{\partial L}{\partial \boldsymbol{\gamma}} = \frac{\partial L}{\partial \boldsymbol{y}} \odot \frac{\boldsymbol{h}-\mu}{\sigma},$$
$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \frac{\partial L}{\partial \boldsymbol{y}}.$$

3.4 Encode-process-decode structure

All works employing a graph neural operator to infer the solution at the next time step adopt a so-called encode-process-decode structure. Such a structure is also adopted here, and is therefore introduced in this section.

1. **Encoding:** after constructing the graph $G = (V, E)$ with its associated features, all node and edge features are first projected into a high-dimensional latent space via encoders ε^v and ε^e, respectively. Applying message passing directly to the raw input features would prevent the model from capturing structure that is not immediately visible in the original representation; by mapping the features into a higher-dimensional space, the model can approximate more complex update functions, which is important for accurately modelling the dynamics of complex physical systems. This is consistent with the universal approximation theorem [57], which motivates the use of sufficiently expressive intermediate representations. Projecting node and edge features to a shared latent dimensionality also ensures that both contribute equally within the message passing algorithm. Moreover, a uniform latent space across all processing layers guarantees that each layer has the same input and output dimensions, giving every layer equal expressive capacity and avoiding the bottleneck that would arise if the first layer were required to operate on the lower-dimensional raw features.

2. **Processing:** once the graph features have been encoded into the high-dimensional latent space, they are passed through a sequence of processing layers. In this context, processing means applying the message passing algorithm to the latent node and edge features in order to model local interactions between neighbouring nodes. In addition, scale transition methods are employed to capture interactions between nodes that are physically further apart, thereby extending the model's ability to represent global dependencies across the computational domain.

3. **Decoding:** this process is the inverse of the encoding operation, denoted by δ^v and δ^e, and aims to transform the feature dimensionality back to that of the desired output dimension.

A visual overview of the encode-process-decode structure is given in Figure 3.4. This provides a high-level overview, as the specific architectural details will be discussed in the next chapter.

3.5 GNNs as surrogate time integrators

In mathematical modelling, the fundamental objective is to advance the solution of a PDE forward in time by means of an appropriate time-stepping scheme. Transitioning from the general PDE modelling framework to the specific context where surrogate models are applicable, we follow the framework of [39] and consider the continuous problem: find $\boldsymbol{u} \in \mathbb{R}^d$, where d denotes the dimension of the solution space, such that

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathscr{L}(\boldsymbol{u}, Z), \quad T \geq t > t_0, \tag{3.8}$$

where $\boldsymbol{u}(t, \boldsymbol{x})$, with $\boldsymbol{u} \in \mathbb{R}^d$ and $\boldsymbol{x} \in \Omega$, denotes the solution to the system of PDEs under consideration, $\mathscr{L}$ is a general, possibly nonlinear, differential operator, and Z is a finite collection of scalar fields encoding the initial and boundary conditions, which may be of Dirichlet, homogeneous Neumann, or periodic type. Time integration of this system reduces to evaluating the right-hand side of equation (3.8) from t_0 to the desired time horizon T.

To apply GNNs to this problem, the domain Ω is first discretised into a finite set of nodes V with coordinates $\boldsymbol{x}_i \in \Omega$, $1 \leq i \leq |V|$, typically obtained via mesh generation heuristics tailored to the geometry and physics of the problem and connected with edges E to form a graph $G = (V, E)$. Let $t_0 < t_1 < \ldots < t_m$ denote a corresponding temporal discretisation. The goal is then to propagate the system from time t_j to t_{j+1}, and the increment in the solution over this interval is given by

$$\mathscr{L}_i^j := \Big(\int_{t_j}^{t_{j+1}} \mathscr{L}(\boldsymbol{u}, Z)\, dt \Big)(\boldsymbol{x_i}), \; \forall\, i \in V, \tag{3.9}$$

which can subsequently be used to time step the solution with

$$\boldsymbol{u}(t_{j+1}, \boldsymbol{x}_i) = \boldsymbol{u}(t_j, \boldsymbol{x}_i) + \mathscr{L}_i^j, \tag{3.10}$$

where $\mathscr{L}_i^j \in \mathbb{R}^d$. Conventionally, numerical methods such as the finite element method described in Section 2.2 would be applied here. But, in the data-driven framework, a surrogate model is sought to approximate the time update described in equation 3.9. Departing from the general formulation of equation 3.8, the objective is to find a model $\mathscr{N}$ such that

$$\mathscr{N}(G, (X_j^v, X_j^e); \boldsymbol{\theta}) \approx L^j, \tag{3.11}$$

where $L^j = \begin{pmatrix} \mathscr{L}_1^j & \ldots & \mathscr{L}_{|V|}^j \end{pmatrix}^T \in \mathbb{R}^{|V| \times d}$ is a matrix filled with time updates for each node at time t_j. Crucially, the surrogate is trained to predict the time update L^j rather than the next state directly. The next solution is then recovered through the explicit correction (3.10), which mirrors the residual structure of explicit time integrators and keeps the learning target centred around zero, which is beneficial for training stability, convergence and generalisation.

3.6 Training strategy and feature construction

Consider a static graph $G = (V, E)$ describing the computational domain, on which a PDE is described by equation (3.8). Each node $i \in V$ is equipped with a feature vector comprising both static and dynamic components. The static features, collected in $S \in \mathbb{R}^{|V| \times d_s}$, include quantities that remain fixed throughout the simulation, such as one-hot vectors $\boldsymbol{d}^i$ encoding node type (e.g., interior or boundary) and/or other static physical fields. The dynamic features are at least the solution of interest $U_j \in \mathbb{R}^{|V| \times d}$ at time step j, but may include other time-dependent features. Optionally, history points of the solution may be appended to enrich the temporal context. The solution tensor is concatenated with the remaining dynamic features to form $D_j \in \mathbb{R}^{|V| \times d_d}$. The total node feature matrix, which serves as the node input to the surrogate model at time step j, is then given by the concatenation

$$X_j^v = \Big[D_j \;\Big|\; S \Big] \in \mathbb{R}^{|V| \times (d_d + d_s)}, \tag{3.12}$$

where $[\cdot \,|\, \cdot]$ denotes per-node (row-wise) concatenation of the dynamic and static feature columns. Positional coordinates $\boldsymbol{x}_i$ are deliberately excluded, as their inclusion would break equivariance to spatial

translations and rotations [58], a property that is desirable in physical applications governed by (3.8). Moreover, it would open the door for overfitting phenomena, in which the model memorises specific spatial configurations rather than generalising the underlying physical laws. For the edge features, the displacement vector $\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$ and/or its norm is commonly encoded, alongside an optional one-hot vector $\boldsymbol{k}_{ij}$ indicating the edge type. Whilst these design choices are not strict requirements, they are widely adopted in the literature and have demonstrated effectiveness for surrogate modelling of physical systems [7, 11, 46, 47]. The flexibility of the message passing algorithm does allow for extensions such as dynamic edge features, for instance to incorporate higher-order functional approximation of the true underlying function $\boldsymbol{u}(\boldsymbol{x}, t)$ analogous to quadratic basis functions defined on the boundaries of elements, though this is not standard practice in the current literature. The complete one-step surrogate time integration pipeline is summarised in Figure 3.5.

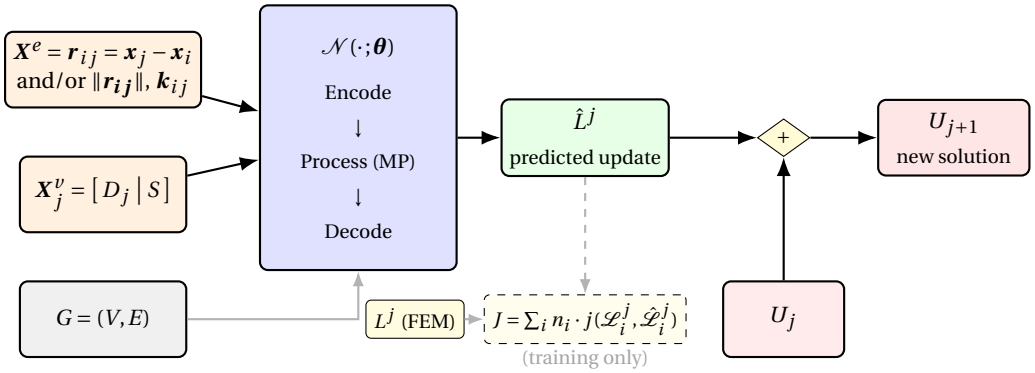


Figure 3.5: General overview surrogate time integrator. Node features $\boldsymbol{X}_j^v$ (dynamic features D_j concatenated with static features S), edge features $\boldsymbol{X}^e$ and graph topology G are passed through the GNN to predict the time update $\hat{L}^j$. The new solution is obtained as $U_{j+1} = U_j + \hat{L}^j$. During training, the predicted update is compared against the ground-truth label L^j via the cost function J.

To train the surrogate model, a cost function is minimised following the general machine learning framework of [9]. The labels L^j are typically generated using numerical methods such as the finite element method described in Section 2.2, though experimental data may in principle also be used. Since boundary-constrained nodes are not free degrees of freedom, their contributions are excluded via a binary mask $\boldsymbol{n} \in \{0,1\}^{|V|}$, yielding the one-step model cost

$$J(\boldsymbol{\theta}, L^j, \hat{L}^j) := \sum_{i=1}^{|V|} n_i \cdot j\left(\mathscr{L}_i^j, \hat{\mathscr{L}}_i^j\right) + \lambda\, \Omega(\boldsymbol{\theta}), \tag{3.13}$$

where $\boldsymbol{\theta}$ are the trainable network parameters, $j(\cdot,\cdot)$ is a generic per-node loss such as the mean squared error, and $\Omega(\cdot)$ is a regularisation term weighted by the hyperparameter λ. Once training is terminated, in practice once the validation loss stops improving (see Section 3.6 and the experimental setup), the parameters are frozen and the model is ready for inference.

During inference, two evaluation regimes are distinguished. In the one-step regime, ground-truth inputs are provided at every time step, isolating the single-step prediction error. In the rollout regime, the model is initialised from U_0 and autoregressively propagates the solution forward for the full simulation duration, as visualised in Figure 3.6. The latter is the more practically relevant setting, but is susceptible to accumulation of errors over time. As formalised in [59], the initial solution input of the model belongs to p_k and thereafter the solver maps $p_k \to \mathscr{N}_\# p_k$ at iteration $k+1$ where $\mathscr{N}_\# : P(X) \to P(X)$. The operator will output samples that have distribution $\mathscr{N}_\# p_k$ instead of p_{k+1}. This distribution shift leads to error accumulation, which can cause the model to diverge from the true solution over time. Mitigation strategies include the injection of training noise [7] and multi-step training objectives [11, 46].

3.7 Interscale transition methods

As discussed in the previous sections, stacking message passing layers extends the receptive field of the model. However, for problems governed by global interactions, the number of layers required can become prohibitively large. Classical multigrid methods address an analogous challenge: smoothing eliminates high-frequency errors efficiently, while coarse-grid correction resolves the remaining low-frequency components at reduced cost [35, 36]. This philosophy of leveraging a hierarchy of progres-

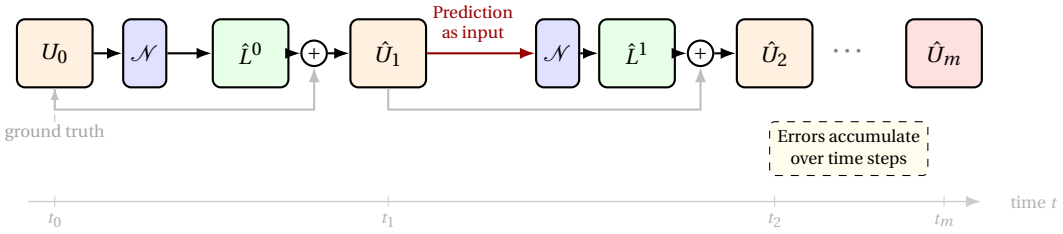


Figure 3.6: Autoregressive application of the surrogate model. The model is initialised with the ground-truth state U_0 and produces a predicted update $\hat{L}^0$, yielding $\hat{U}_1 = U_0 + \hat{L}^0$. When rolling out further, the distribution of the output will include the trained distribution with the addition of the model its own noise, which can lead to error accumulation.

sively coarser representations is adopted here to extend the effective receptive field of the surrogate time integrator.

Consider a hierarchy of graphs

$$G^0 = (V^0, E^0), \quad G^1 = (V^1, E^1), \quad \ldots, \quad G^L = (V^L, E^L), \tag{3.14}$$

with $|V^0| > |V^1| > \cdots > |V^L|$, so that G^0 is the original fine mesh. The construction of the coarser graphs is discussed in Section 2.6 and will be revisited in the experimental setup, where it is defined for the specific implementation of this thesis. Each graph G^ℓ in the hierarchy carries its own features: the static node features at a coarse level are obtained by per-cluster averaging of the fine-level static features, while the raw edge attributes at coarser levels are obtained using the fine edges, detailed later in Section 3.7.3. Regardless of the coarsening strategy, transitioning between levels requires two operators: a restriction operator that transfers information from a fine level to a coarser one, and a prolongation operator for the reverse direction. In this work both operators are realised as learnable message passing steps on dedicated transfer graphs, which are trained end-to-end with the rest of the model, comparable to the implementation in [10].

Figure 3.7 provides a top-level overview of the resulting V-cycle architecture. The remainder of this section formalises the transfer graphs, the restriction and prolongation operators that act on them, and the coarse-grid correction that combines fine and coarse information. As a navigational aid, Table 3.1 summarises the key notation used throughout the section.

Table 3.1: Summary of notation used in the interscale transition section.

Symbol	Type	Meaning
$G^\ell = (V^\ell, E^\ell)$	graph	level-ℓ graph in the hierarchy
$V^{\ell+1} = C^\ell \subseteq V^\ell$	set	coarse nodes are a subset of fine nodes
$F^\ell = V^\ell \setminus V^{\ell+1}$	set	fine-only (F-) nodes not present at level $\ell+1$
$\boldsymbol{h}^\ell \in \mathbb{R}^{\lvert V^\ell\rvert \times d}$	features	latent node features at level ℓ
$\boldsymbol{a}^\ell \in \mathbb{R}^{\lvert E^\ell\rvert \times d}$	features	latent edge attributes at level ℓ
$\boldsymbol{h}^\ell_{\text{mp}}$	features	node features after down-path message passing on G^ℓ
$G^{\ell\to\ell+1}_{\downarrow}$	graph	down transfer graph (fine→coarse)
$G^{\ell+1\to\ell}_{\uparrow}$	graph	up transfer graph (coarse→fine)
$R^{\ell\to\ell+1}$	operator	learned restriction (see Section 3.7)
$P^{\ell+1\to\ell}$	operator	learned prolongation (see Section 3.7)
$\text{MP}(G, \boldsymbol{h}, \boldsymbol{a})$	function	message passing on G (Algorithm 1)

As considerable new notation is employed in what follows, it is introduced beforehand. At each level ℓ, let $\boldsymbol{h}^\ell \in \mathbb{R}^{|V^\ell|\times d}$ denote the latent node features in the d-dimensional hidden space, $\boldsymbol{a}^\ell \in \mathbb{R}^{|E^\ell|\times d}$ denote the latent edge attributes, and let $\boldsymbol{X}^{e,\ell}_{\text{raw}} \in \mathbb{R}^{|E^\ell|\times n_e}$ denote the raw (unencoded) edge attributes, which are the inter-nodal displacement vectors $\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$ between the endpoints of each edge. Each level has its own edge encoder ε^ℓ that maps raw edge attributes to the latent space. The notation $\text{MP}(G, \boldsymbol{h}, \boldsymbol{a})$ refers to applying one or more message passing layers on graph G with node features $\boldsymbol{h}$ and edge attributes $\boldsymbol{a}$, following Algorithm 1.

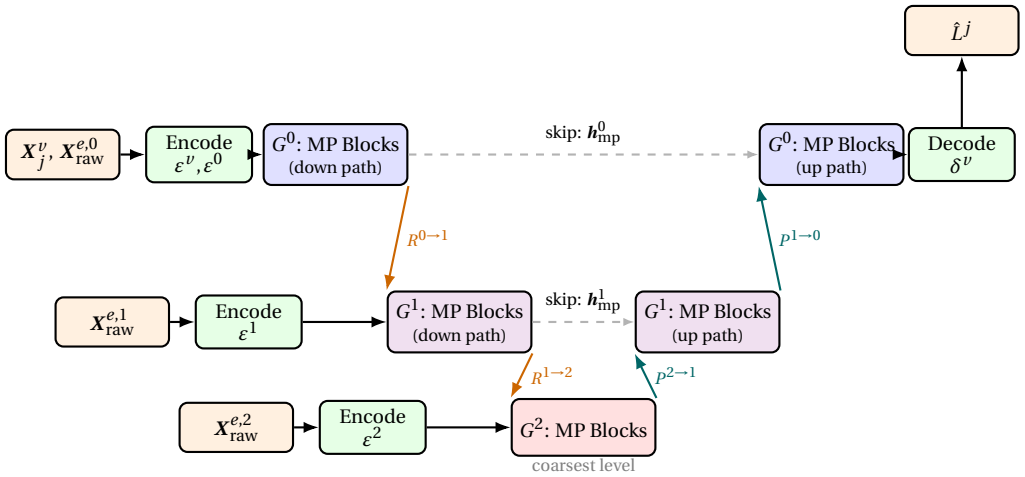


Figure 3.7: Overview of the hierarchical V-cycle architecture for $L = 2$ levels. Each level ℓ has its own graph G^ℓ with raw edge attributes encoded by a per-level encoder ε^ℓ. On the down path (left-to-right), each level applies message-passing (MP) blocks before passing information to the next coarser level via the learned restriction operator $R^{\ell\to\ell+1}$ (orange arrows; see Figure 3.8 and equations (3.17)–(3.18)). On the up path (right-to-left), the learned prolongation operator $P^{\ell+1\to\ell}$ (teal arrows; see Figure 3.10 and equations (3.21)–(3.22)) injects coarse-level corrections back to the fine level. The dashed skip connections carry the fine-level features $\boldsymbol{h}_{\text{mp}}^\ell$ from the down path directly to the coarse-grid correction step: because coarse-level processing cannot capture fine-scale detail, the fine features are added back (rather than overwritten) to ensure that no information is lost, mirroring the additive correction $u^\ell \leftarrow u^\ell + P\,e^{\ell+1}$ of classical multigrid.

3.7.1 Down transfer graph and learned restriction

As discussed previously, the C/F-splitting strategy from the AMG method [38] partitions the fine-level node set V^ℓ into coarse nodes C^ℓ and fine-only nodes F^ℓ:

$$V^\ell = C^\ell \cup F^\ell, \qquad C^\ell \cap F^\ell = \emptyset.$$

The coarse level inherits exactly the C-nodes: $V^{\ell+1} = C^\ell \subseteq V^\ell$. This subset relation is the key structural property of the hierarchy: because every coarse-level index is also a fine-level index, no interpolation of node identity is required when selecting coarse features from a fine-level tensor. The F-nodes $F^\ell = V^\ell \setminus V^{\ell+1}$ have no direct counterpart at the coarser level and their information must be aggregated into their neighbouring C-nodes by restriction.

Consider two consecutive levels $G^\ell = (V^\ell, E^\ell)$ and $G^{\ell+1} = (V^{\ell+1}, E^{\ell+1})$. The down transfer graph $G_\downarrow^{\ell\to\ell+1}$ retains the full fine-level node set V^ℓ (so that F-nodes can send messages) and contains exactly those fine-level edges whose receiver is a C-node. Or in other words, the edges that funnel fine-scale information into each coarse representative:

$$V_\downarrow^{\ell\to\ell+1} = V^\ell, \tag{3.15}$$

$$E_\downarrow^{\ell\to\ell+1} = \{(i,j) \in E^\ell \mid j \in C^\ell\}. \tag{3.16}$$

Equation (3.16) retains only inbound edges to C-nodes, so each C-node aggregates messages from its entire fine-level neighbourhood. F-node-to-F-node edges (whose receiver is not a C-node) are discarded. This construction is illustrated in Figure 3.9 (a).

Suppose the fine-level features $\boldsymbol{h}^\ell$ have already been processed by the down-path message passing blocks on G^ℓ, yielding $\boldsymbol{h}_{\text{mp}}^\ell$ and $\boldsymbol{a}_{\text{mp}}^\ell$. Restriction is then performed in two steps. First, a message passing step is applied on the down transfer graph, using the subset of the processed fine-level latent edge attributes indexed by $E_\downarrow^{\ell\to\ell+1}$:

$$\widetilde{\boldsymbol{h}}^\ell = \text{MP}\big(G_\downarrow^{\ell\to\ell+1},\, \boldsymbol{h}_{\text{mp}}^\ell,\, \boldsymbol{a}_{\text{mp}}^\ell\big|_{E_\downarrow^{\ell\to\ell+1}}\big). \tag{3.17}$$

Second, the result is restricted to the coarse node set to obtain the initial coarse-level features:

$$\boldsymbol{h}^{\ell+1} = \widetilde{\boldsymbol{h}}^\ell\big|_{V^{\ell+1}}. \tag{3.18}$$

These two operators constitute the restriction operator $R^{\ell\to\ell+1}$, which is summarised in Figure 3.8.

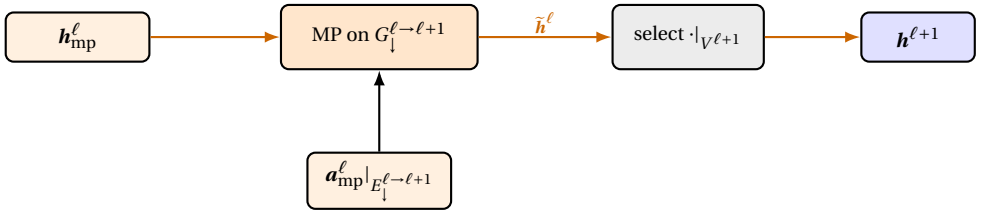
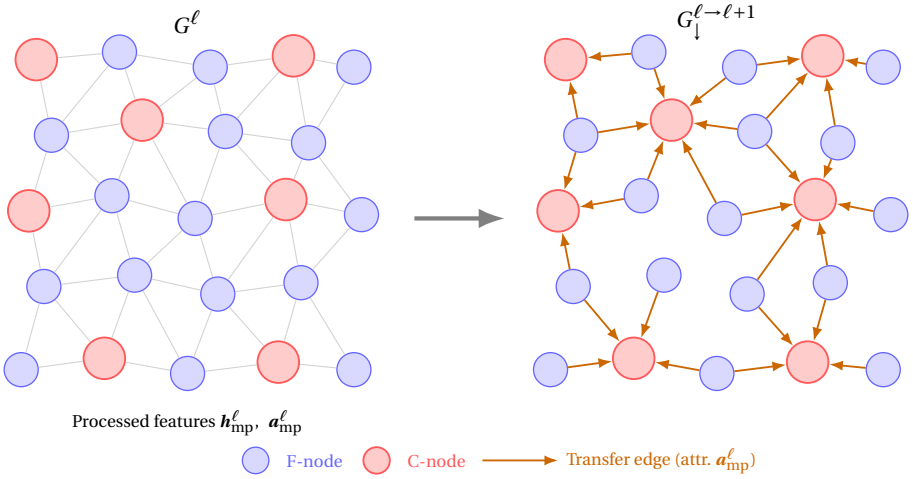
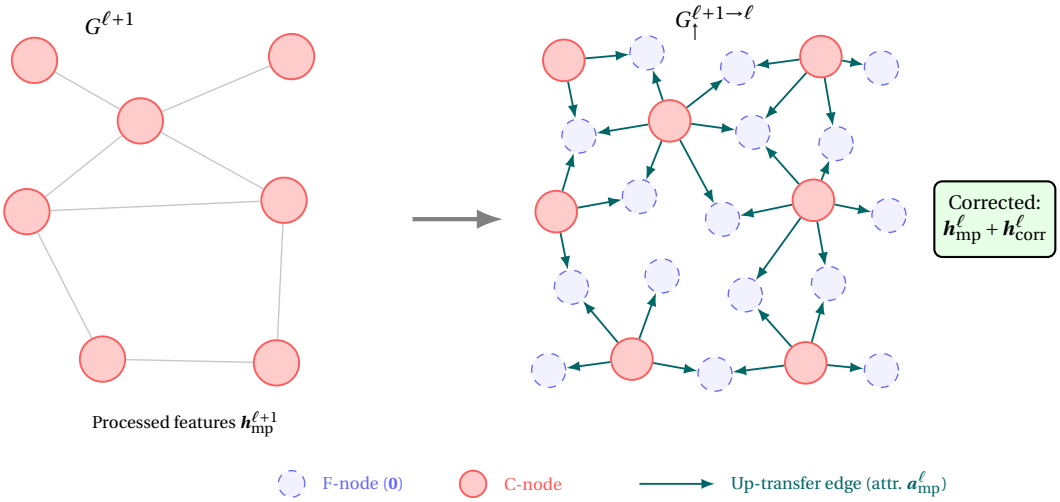


Figure 3.8: Internal flow of the learned restriction operator $R^{\ell\to\ell+1}$ (orange arrows in Figure 3.7). The processed fine-level features $\boldsymbol{h}^{\ell}_{\text{mp}}$ enter a message passing step on the down transfer graph $G^{\ell\to\ell+1}_{\downarrow}$ (equation (3.16)), whose edge attributes are the processed latent edge features $\boldsymbol{a}^{\ell}_{\text{mp}}$ restricted to $E^{\ell\to\ell+1}_{\downarrow}$. Selection onto the C-node set $V^{\ell+1} = C^{\ell} \subseteq V^{\ell}$ then yields the initial coarse-level features $\boldsymbol{h}^{\ell+1}$ (equation (3.18)).



(a) Down transfer graph $G^{\ell\to\ell+1}_{\downarrow}$: all fine-level edges whose target is a C-node are retained (orange). F-nodes (blue) send messages to their C-node neighbours; the result is selected onto $V^{\ell+1} = C^{\ell}$ to obtain $\boldsymbol{h}^{\ell+1}$.



(b) Up transfer graph $G^{\ell+1\to\ell}_{\uparrow}$: the same edges are reversed (teal) so that C-nodes broadcast their processed features to all fine-level neighbours. F-nodes are zero-padded before the MP step, and the result is added to the skip-connected fine features $\boldsymbol{h}^{\ell}_{\text{mp}}$.

Figure 3.9: Construction of the transfer graphs used for restriction (a) and prolongation (b). Both graphs share the same node set V^{ℓ} and the same underlying edges; they differ only in direction. The C/F-splitting ensures that every F-node has at least one C-node neighbour, so no fine-scale information is stranded.

3.7.2 Up transfer graph and learned prolongation

The up transfer graph $G_{\uparrow}^{\ell+1\to\ell}$ is the directed reverse of the down transfer graph: every edge direction is flipped so that information now flows from C-nodes to all their fine-level neighbours.

$$V_{\uparrow}^{\ell+1\to\ell} = V^{\ell}, \tag{3.19}$$

$$E_{\uparrow}^{\ell+1\to\ell} = \{(j,i) \mid (i,j) \in E_{\downarrow}^{\ell\to\ell+1}\}. \tag{3.20}$$

Since the fine graph is bidirectional, for every down-transfer edge $(i,j) \in E_{\downarrow}^{\ell\to\ell+1}$ the reversed edge (j,i) belongs to E^{ℓ} as well. Consequently, $E_{\uparrow}^{\ell+1\to\ell} \subseteq E^{\ell}$, and the processed latent edge attributes $\boldsymbol{a}_{\text{mp}}^{\ell}$ are well-defined on all edges of the up transfer graph. The up transfer graph is shown alongside the down transfer graph in Figure 3.9 (b).

After the coarse-level features $\boldsymbol{h}^{\ell+1}$ have been processed by message passing blocks on $G^{\ell+1}$, yielding $\boldsymbol{h}_{\text{mp}}^{\ell+1}$, prolongation proceeds as follows. First, a zero-padded feature vector is constructed on the fine-level node set:

$$\boldsymbol{p}_i^{\ell} = \begin{cases} \boldsymbol{h}_{\text{mp},i}^{\ell+1} & \text{if } i \in V^{\ell+1}, \\ \boldsymbol{0} & \text{if } i \in V^{\ell} \setminus V^{\ell+1}. \end{cases} \tag{3.21}$$

A message passing step on the up transfer graph then distributes the coarse information to all fine nodes, reusing the processed fine-level latent edge attributes before restriction, which are confined to $E_{\uparrow}^{\ell+1\to\ell}$:

$$\boldsymbol{h}_{\text{corr}}^{\ell} = \text{MP}\big(G_{\uparrow}^{\ell+1\to\ell},\, \boldsymbol{p}^{\ell},\, \boldsymbol{a}_{\text{mp}}^{\ell}\big|_{E_{\uparrow}^{\ell+1\to\ell}}\big). \tag{3.22}$$

The corrected fine-level features are obtained by adding the prolongated coarse correction to the fine features carried over from the down path via the skip connection:

$$\boldsymbol{h}_{\text{out}}^{\ell} = \boldsymbol{h}_{\text{mp}}^{\ell} + \boldsymbol{h}_{\text{corr}}^{\ell}. \tag{3.23}$$

The additive (rather than replacing) form is essential: $\boldsymbol{h}_{\text{mp}}^{\ell}$ retains fine-scale detail that the coarser level cannot fully represent. Overwriting those features with $\boldsymbol{h}_{\text{corr}}^{\ell}$ alone would discard the fine-scale information processed on the down path. By adding the two contributions, each provides what the other lacks: the coarse correction supplies long-range, global information while the skip-connected fine features preserve local detail. This is the direct analogue of the classical coarse-grid correction $u^{\ell} \leftarrow u^{\ell} + P\, e^{\ell+1}$, where P is the prolongation operator and $e^{\ell+1}$ the coarse-level error estimate; in both cases the original fine-level quantity is updated rather than replaced. The corrected features $\boldsymbol{h}_{\text{out}}^{\ell}$ are subsequently smoothed by a further set of message passing blocks on G^{ℓ} (the up-path blocks in Figure 3.7). The full internal flow of the prolongation operator $P^{\ell+1\to\ell}$, is summarised in Figure 3.10.

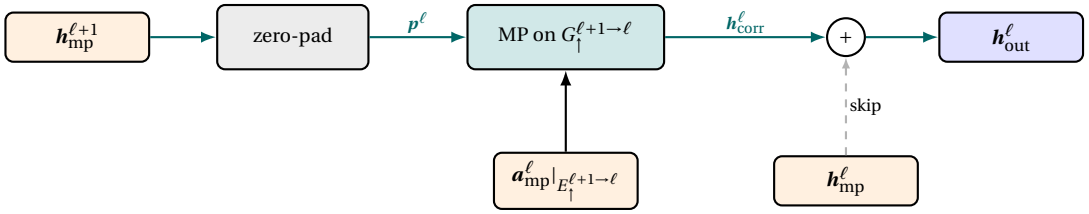


Figure 3.10: Internal flow of the learned prolongation operator $P^{\ell+1\to\ell}$ (teal arrows in Figure 3.7) and the additive coarse-grid correction. The processed coarse-level features $\boldsymbol{h}_{\text{mp}}^{\ell+1}$ are zero-padded onto V^{ℓ} to form $\boldsymbol{p}^{\ell}$ (Eq. (3.21)) and then fed through a message passing step on the up transfer graph $G_{\uparrow}^{\ell+1\to\ell}$ using the processed fine-level latent edge features $\boldsymbol{a}_{\text{mp}}^{\ell}$ restricted to $E_{\uparrow}^{\ell+1\to\ell}$, producing the coarse correction $\boldsymbol{h}_{\text{corr}}^{\ell}$ (Eq. (3.22)). The skip-connected fine features $\boldsymbol{h}_{\text{mp}}^{\ell}$ are added at the summation junction to realise $\boldsymbol{h}_{\text{out}}^{\ell}$ (Eq. (3.23)).

Because the transfer message passing blocks are trainable and the transfer edge attributes are drawn from the latent fine-level representations that are themselves learned, the restriction and prolongation operators are learned end-to-end, allowing the model to adapt the interscale transitions to the specific dynamics of the problem at hand.

3.7.3 Coarse-level edge features

The existence of raw attributes on a coarser graph $G^{\ell+1}$ was mentioned earlier but has not yet been specified. Because the coarse nodes correspond to a subset of the fine-level nodes, a natural choice is to derive the coarse edge attributes from the fine-level graph using shortest-path information. Recall that each fine-level edge $(i, j) \in E^{\ell}$ carries a displacement vector $\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$, i.e. the signed vector pointing from node i to node j in physical space. For a coarse edge $(C_A, C_B) \in E^{\ell+1}$, each endpoint is the representative C-node of its cluster. Note that two cluster nodes are connected if their clusters are on the fine graph. The coarse edge attribute is set to the sum of the fine-level displacement vectors along the shortest path connecting C_A to C_B through the fine graph:

$$\boldsymbol{X}^{e,\ell+1}_{(C_A,C_B)} = \sum_{k=1}^{K} \boldsymbol{r}_k, \tag{3.24}$$

where $\boldsymbol{r}_k = \boldsymbol{x}_{v_k} - \boldsymbol{x}_{v_{k-1}}$ is the inter-nodal displacement vector along the k-th hop of the path $C_A = v_0 \to v_1 \to \cdots \to v_{K-1} \to v_K = C_B$ in the fine graph, with "shortest" measured by Euclidean edge length. The summing of displacement vectors is the only aggregation that correctly recovers the net displacement from C_A to C_B: because each $\boldsymbol{r}_k$ is a signed offset in physical space, their vector sum can be seen as the overall spatial offset traversed along the path, regardless of the number of intermediate hops. The shortest path is found by Dijkstra's algorithm [60] restricted to the fine-level topology. This procedure is illustrated in Figure 3.11. By summing displacement vectors rather than scalar distances, the resulting coarse edge attribute retains directional information, which is important since the edge features encode inter-nodal displacements. One might wonder why the coarse edge attributes are not simply taken as the difference of coarse node positions. The key reason is that the problem domain is periodic: on a periodic domain, the Euclidean offset between two node positions does not correctly reflect the direction of traversal through the graph, as coarse edge whose endpoints lie on opposite sides of the periodic boundary would receive the wrong displacement. The path-based sum avoids this, since it inherits the topologically consistent, boundary-aware displacements already encoded in the fine-level edge attributes.

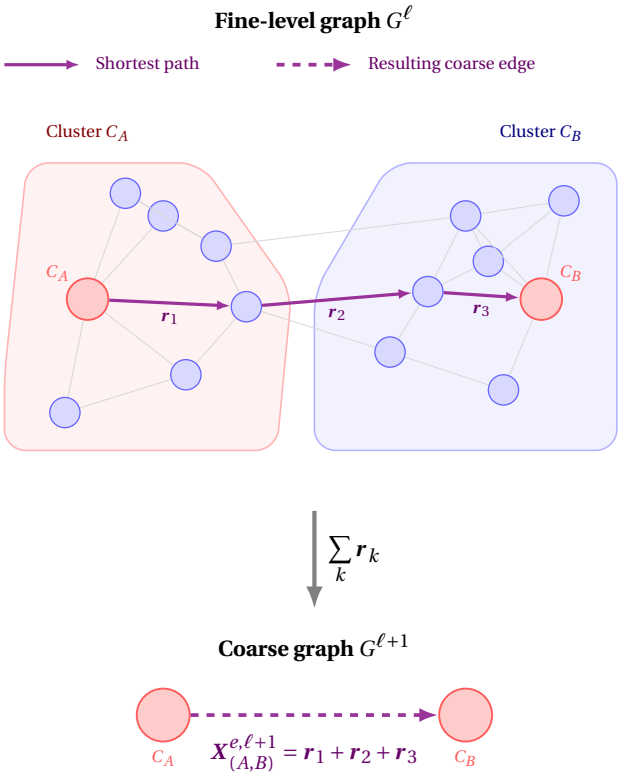


Figure 3.11: Construction of coarse-level edge attributes. The shortest path between two C-nodes through the fine graph (violet arrows) is found via Dijkstra's algorithm. The coarse edge attribute is the sum of the fine-level displacement vectors $\boldsymbol{r}_k$ along this path, preserving directional information.

Chapter 4

Experimental setting and methodology

The theory of Section 2.4 showed that the stability of explicit time-stepping schemes depends on dimensionless numbers. The central experimental question of this thesis is whether an analogous relation can be observed empirically for the surrogate model on an unstructured mesh, with spatially varying diffusion tensors and velocity vectors as in equation 2.1. Moreover, it is investigated whether the multiscale architecture, which is facilitated by the learned prolongation and restriction operators of Section 3.7, can extend this limit, relative to its single-scale counterpart. The remainder of this chapter describes the experimental ingredients that are needed to address those questions: the graph construction and input features in the first section, the dataset generation in Section 4.2, the training setup in Section 4.3, the model variants in Section 4.4, and the evaluation metrics in Section 4.5.

4.1 Graph construction and input features

The graph $G = (V, E)$ defined in Section 3.1, is obtained in a direct manner from the underlying triangular mesh $\mathcal{T}_h$. The mesh vertices are taken as the graph nodes and every mesh edge is duplicated into a pair of directed graph edges (i, j) and (j, i), which yields a bidirectional graph in the sense of Definition 3.1.1. No additional nodes or edges are introduced, so the graph topology and the mesh topology coincide.

4.1.1 Periodic boundary condition handling

When such a graph is retrieved from the mesh, the geometry of the unit square is respected, but not the periodic boundary conditions imposed in equations 2.2. Two opposing boundary nodes located at $(0, y)$ and $(1, y)$, or at $(x, 0)$ and $(x, 1)$, are spatially identical from a perspective of Ω. Moreover, the four corner nodes $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$ correspond to a single physical point. To realise these identifications at the level of message passing without removing any of the nodes from V, we pair each duplicate boundary node j with its twin i and redirect all edges connected to j towards i, both incoming and outgoing. Concretely, for every periodic pair (i, j) each edge $(k, j) \in E$ is replaced by (k, i) and each edge $(j, k) \in E$ by (i, k). The four corner nodes are merged into a single representative in the same way. After the rewiring, the twin nodes remain in V but no longer carry incoming edges, which means the output is evaluated only at the representative node of each periodic pair, while the loss masks the redundant copies as is detailed in Section 4.3, as this prevents double-counting that would result in giving the boundary nodes twice as much weight. The result of this procedure, applied to an illustrative mesh, is shown in Figure 4.1.

4.1.2 Input features

The node and edge features follow the general construction of Section 3.6. At every simulation node $i \in V$, the static node features collect the components of the local diffusion tensor and velocity entries of the velocity vector. As $D(\boldsymbol{x})$ is symmetric positive definite throughout, only its three independent entries concerning the diffusion tensor are included. With two velocity components, the static node features are therefore

$$\boldsymbol{s}_i = \big(D_{xx}(\boldsymbol{x}_i),\, D_{yy}(\boldsymbol{x}_i),\, D_{xy}(\boldsymbol{x}_i),\, v_x(\boldsymbol{x}_i),\, v_y(\boldsymbol{x}_i)\big)^\top \in \mathbb{R}^5. \tag{4.1}$$

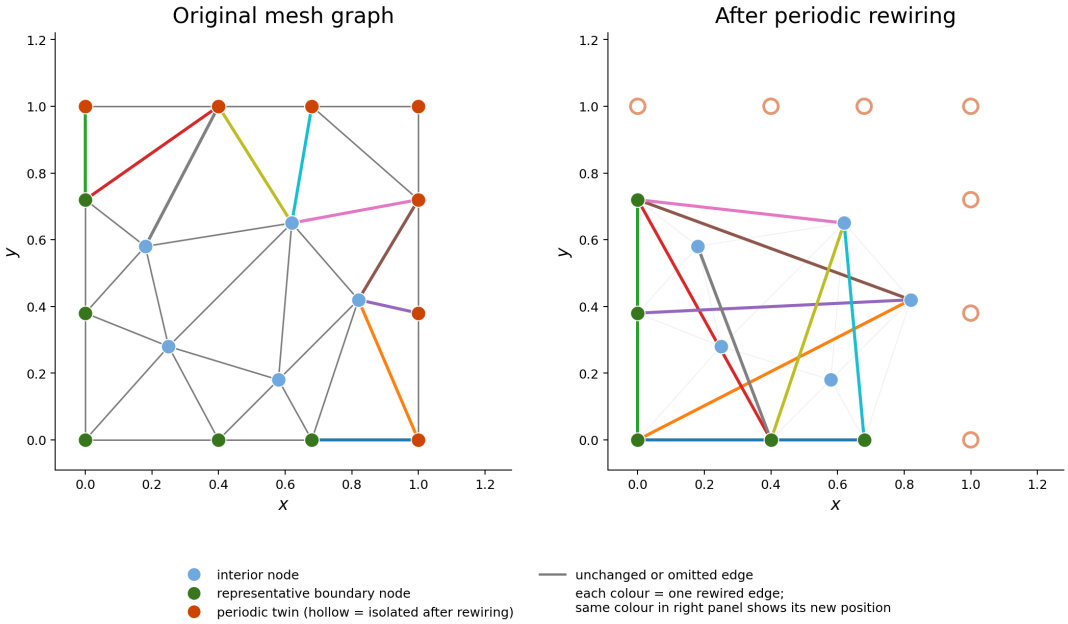


Figure 4.1: Effect of the periodic rewiring on a small illustrative unstructured mesh. Left: the original mesh graph. Right: the rewired graph, in which the rewired edges are shown with the same colors as in the original graph.

The dynamic node feature is the scalar solution at the current discrete time t_j, i.e. $\boldsymbol{d}_i^j = u_h(\boldsymbol{x}_i, t_j) \in \mathbb{R}$. The complete node feature matrix at time step j is the row-wise concatenation

$$\boldsymbol{X}_j^v = \left[D_j \,\middle|\, S \right] \in \mathbb{R}^{|V|\times 6}, \tag{4.2}$$

in line with equation 3.12. As argued in Section 3.6, we do not encode node coordinates $\boldsymbol{x}_i$ as input features.

The edge features encode the inter-nodal displacement vector

$$\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i = (r_x, r_y)^\top \in \mathbb{R}^2, \tag{4.3}$$

which is purely geometric and therefore static, as our mesh is. The associated edge feature matrix is $\boldsymbol{X}^e = (\boldsymbol{r}_{ij})_{(i,j)\in E} \in \mathbb{R}^{|E|\times 2}$ and inherits the antisymmetry $\boldsymbol{r}_{ij} = -\boldsymbol{r}_{ji}$ already discussed in Section 3.1, so the graph is bidirectional, but not undirected. The boundary rewiring of the previous subsection ensures that the displacement $\boldsymbol{r}_{ij}$ associated with an edge crossing a periodic identification is the geometrically correct one. That is, the short displacement through the periodic boundary rather than the long one across the interior of the unit square.

Table 4.1 summarises all input features used by the model.

Table 4.1: Input features for the surrogate model. Node features are concatenated into the node feature matrix $\boldsymbol{X}_j^v$ (equation (4.2)); edge features form the edge feature matrix $\boldsymbol{X}^e$.

Feature type	**Symbol**	**Dim**	**Description**
Static node	$D_{xx}(\boldsymbol{x}_i)$	1	$(1,1)$-entry of diffusion tensor at node i
	$D_{yy}(\boldsymbol{x}_i)$	1	$(2,2)$-entry of diffusion tensor at node i
	$D_{xy}(\boldsymbol{x}_i)$	1	$(1,2)$-entry of diffusion tensor at node i
	$v_x(\boldsymbol{x}_i)$	1	x-component of velocity at node i
	$v_y(\boldsymbol{x}_i)$	1	y-component of velocity at node i
Dynamic node	$u_h(\boldsymbol{x}_i, t_j)$	1	Scalar FEM solution at node i, time t_j
Edge	$r_x = x_j - x_i$	1	x-component of inter-nodal displacement
	$r_y = y_j - y_i$	1	y-component of inter-nodal displacement
Total		**8**	6 node features (5 static + 1 dynamic), 2 edge features

4.1.3 Coarsening of the fine graph

The multiscale models of Section 4.4 require a hierarchy of graphs $G^0, G^1, \dots, G^L$ in the sense of equation 3.14, with G^0 the fine graph constructed above. The hierarchy is generated once for every rollout in the dataset and stored alongside the simulation data. The construction follows an adaptation of the classical Ruge-Stüben procedure described at the end of Section 2.6: the notions of strength of connection, of a C/F-splitting and of a cluster-based coarse graph are used from the algebraic theory, while the matrix-based ingredients are replaced by purely feature-based surrogates and the score-update rule is simplified into a single-pass greedy variant. The construction is applied recursively, level by level, with the procedure below described between a fine level ℓ and its coarse counterpart $\ell+1$.

Since no fine-level matrix A_h is present in the surrogate setting, the negative off-diagonal coefficient $-a_{ij}$ that drives the classical strength rule is replaced by a similarity score computed from the edge feature $\boldsymbol{r}_{ij}$ and from the static node features $\boldsymbol{s}_i$. For every edge $(i,j) \in E^\ell$ we set

$$s_{ij} = \exp\Big(-\alpha \tfrac{\|\boldsymbol{r}_{ij}\|}{\tilde{r}}\Big) \cdot \exp\Big(-\beta \tfrac{\|D(\boldsymbol{x}_i) - D(\boldsymbol{x}_j)\|_F}{\tilde{D}}\Big) \cdot \exp\Big(-\gamma \tfrac{\|\boldsymbol{v}(\boldsymbol{x}_i) - \boldsymbol{v}(\boldsymbol{x}_j)\|}{\tilde{v}}\Big), \tag{4.4}$$

where $\tilde{r}$, $\tilde{D}$ and $\tilde{v}$ denote the medians of the corresponding quantities over E^ℓ, S^ℓ and serve to make the score scale-invariant across levels. The fixed coefficients $\alpha = 1.0$, $\beta = 0.5$ and $\gamma = 0.5$ are persisted with the dataset and cannot be modified at training time. This choice gives the inter-nodal distance a slightly dominant contribution, so that spatially nearby nodes are preferentially grouped into the same cluster and the resulting coarse graphs cover, on average, a larger spatial extent per message-passing step than the underlying fine graph. An edge is considered strong if $s_{ij} \geq \theta$ with the absolute threshold $\theta = 0.1$, and the strong-neighbour set $S_i = \{j : s_{ij} \geq \theta\}$ plays the role of the algebraic strong-dependence set of Section 2.6. Note that the coefficient values above are heuristics, chosen on the basis of trial and error rather than a systematic investigation.

The C/F-splitting is computed by a single-pass greedy variant of the Ruge–Stüben rule, in which the iteratively-updated lexicographic score $\mu_i = |S_i^\top \cap U| + 2\,|S_i^\top \cap F|$ of the classical algorithm is replaced by the static, weighted incoming-strength degree

$$\mu_i = \sum_{j \in S_i^\top} s_{ji} + \varepsilon_i,$$

where ε_i is a deterministic, seeded uniform tie-breaker of order 10^{-2} and $S_i^\top = \{j : i \in S_j\}$ collects the nodes that strongly depend on i. Nodes are then visited in descending order of μ_i: an undecided node becomes a C-node, and every undecided node strongly connected to it (i.e. in $S_i^\top$) is immediately marked as an F-node and is added to the cluster of the C-node. Periodic-twin nodes, which are isolated in the fine graph after the rewiring of Section 4.1.2, are excluded from the splitting and never become C-nodes. If there are still undecided nodes after this first greedy sweep, these nodes will become F-nodes and appended to the cluster of a C-node to which it has the strongest connection. F-nodes that, at the end of this sweep, have no direct C-neighbour are promoted to singleton C-clusters, so that every non-isolated node is eventually associated to exactly one coarse parent. The result is a partition $V^\ell = C^\ell \cup F^\ell \cup I^\ell$, with I^ℓ the isolated periodic-twin nodes that do not participate in the message passing algorithm.

Mirroring the algebraic construction in which the C-nodes C^ℓ are taken as the coarse index set $\mathcal{I}_H$, the coarse-level node set is set to $V^{\ell+1} = C^\ell$. Each F-node is assigned to the cluster of its strongest C-neighbour, yielding a cluster map $\mathrm{cl} : V^\ell \setminus I^\ell \to V^{\ell+1}$ that, on C^ℓ, reduces to the identity. Two coarse nodes $C_A, C_B \in V^{\ell+1}$ are connected by an edge in $E^{\ell+1}$ whenever there exists a fine-level edge $(i,j) \in E^\ell$ with $\mathrm{cl}(i) = C_A$ and $\mathrm{cl}(j) = C_B$, i.e. whenever their clusters are adjacent in the fine graph. The coarse edge attribute is the sum of the fine-level displacement vectors along a shortest path between C_A and C_B through the fine graph, as motivated in Section 3.7.3 and made precise by equation 3.24. This approach preserves the directional information of the original mesh on a periodic domain. The static node feature vector at coarse level is the per-cluster average of the fine-level static features, which is only used to create even coarser graphs. Only coarser raw features for the edges are newly introduced into the multiscale model, which can be seen by the general V-cycle described in 3.7. Down and up transfer graphs are generated with the coarse graph through equations 3.15–3.16 and 3.19–3.20, so that all interscale connectivity needed by the prolongation and restriction operators of Section 4.4 is fixed once and for all by the dataset. Figure 4.2 illustrates the resulting fine graph, the transfer graph and the coarse graph of an actual mesh from the dataset, zoomed in for clarity, as using the complete graph would be too dense to visualise.

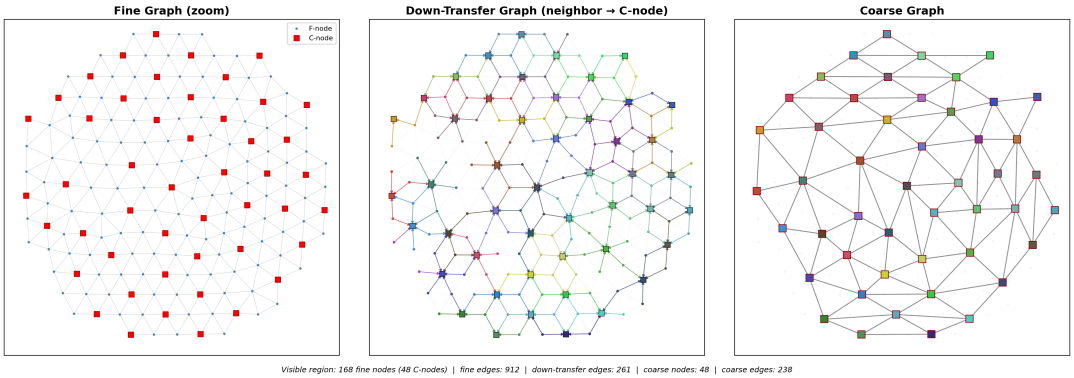


Figure 4.2: Visualisation of one level of the AMG-style coarsening on a representative fine graph (zoomed in for clarity). The left panel shows the fine graph with the C/F-splitting produced by the single-pass greedy Ruge–Stüben variant of Section 4.1.3. C-nodes (red squares) form an approximately maximal independent set with respect to the strength-of-connection graph defined in equation 4.4, while F-nodes (blue dots) are clustered around them via the cluster map cl. The middle panel shows the down transfer graph $G_{\downarrow}^{0\to 1}$ of equations 3.15–3.16, which contains every fine-level edge whose destination is a C-node and thus collects the full fine-graph neighbourhood of each C-node. Edges are coloured according to the cluster of their source node. The right panel shows the resulting coarse graph G^1, in which two C-nodes are connected whenever their clusters are adjacent in the fine graph and the corresponding raw edge attribute is given by the path-based sum of equation 3.24.

4.2 Datasets

The training, validation and test datasets are all generated from the same finite element pipeline, but they differ in the velocity and diffusion fields and in the length of the simulation horizon. The common ingredients are described first, followed by the dataset-specific choices.

4.2.1 Simulation setup

For every rollout, an unstructured triangular mesh of the unit square Ω is generated using the open-source NGSolve finite element package [61], with a target element size h drawn uniformly at random from $[0.01, 0.02]$, thus resulting in graphs that have nodes ranging from approximately 2,500 to 10,000. The mesh is conforming and respects the periodic identifications of Ω. The discretisation of the finite element space employs continuous piecewise-linear Lagrange elements (hat functions), that is $p = 1$, together with the SUPG stabilisation parameter given by equation 2.23.

The initial condition follows the construction of equation 2.3: the numbers of Fourier modes M and N are drawn independently from $\{2, \dots, 7\}$ and the coefficients $c_{m,n}$ are drawn independently from the uniform distribution on $[0, 1]$, with the centre of the Gaussian envelope fixed at $(x_c, y_c) = (1/2, 1/2)$. The advection scale is sampled from

$$b \sim \mathcal{U}(-1, 1)$$

and multiplies the velocity field, whereas the diffusion scale is sampled from

$$\log_{10} a \sim \mathcal{U}(-5, -1)$$

and multiplies the diffusion tensor, so that the diffusion entries span four orders of magnitude. Time integration uses a backward Euler scheme with the small time step

$$\Delta t_{\text{sim}} = 10^{-4},$$

which was chosen because of its unconditional stability. In hindsight this was a conservative choice, whose implications for the comparison with the surrogate are revisited in the discussion. A convergence study, with the training and validation profile introduced later, is included in Appendix A.5 to investigate how the FEM solutions converge for different levels of diffusion.

To limit storage, only every tenth finite element solution is saved, subsampling the simulation in time and yielding an effective base timestep of

$$\Delta t_{\text{base}} = 10^{-3}$$

between every column of the solution vector U. So the full saved trajectory is then given by

$$U \in \mathbb{R}^{|V| \times N_T}, \qquad U_{i,n} = u_h(\boldsymbol{x}_i, n\,\Delta t_{\text{base}}), \tag{4.5}$$

with $N_T = T/\Delta t_{\text{base}}$ the number of saved snapshots and T the simulation horizon, which depends on the dataset. Note that the surrogate models are trained on even larger timesteps, which is why saving the full trajectory at the simulation time step would have been redundant.

The supervised one-step regression target of equation 3.9 is constructed from U by selecting an input snapshot index n and a stride $k \in \mathbb{N}$. The model input at the corresponding sample is the snapshot $U_{:,n}$ together with the static node features and edge features of equations 4.1–4.3, and the regression label is the solution k steps ahead, which therefore reads

$$L_i^j = u_h(\boldsymbol{x}_i, t_{j+1}) - u_h(\boldsymbol{x}_i, t_j), \qquad t_j = n\,\Delta t_{\text{base}}, \quad t_{j+1} = (n+k)\,\Delta t_{\text{base}}, \tag{4.6}$$

in line with equation 3.11. The effective surrogate time step is therefore $\Delta t = k\,\Delta t_{\text{base}}$. This decoupling allows the same saved trajectory U to be reused for surrogate models trained at different temporal resolutions, which is exploited in Section 4.3. This way, we retrieve surrogates that are trained at different timesteps, yet have been trained on the same underlying trajectories, which allows for a more controlled comparison of the surrogate performance at different Δt.

4.2.2 Training and validation dataset

The training and validation rollouts share the same realisation of the velocity field and diffusion tensor, sampled from the distribution introduced above. The velocity is the divergence-free double-gyre field

$$\boldsymbol{v}(\boldsymbol{x}) = b \begin{pmatrix} 2\pi \sin(\pi x)\cos(2\pi y) \\ -\pi \cos(\pi x)\sin(2\pi y) \end{pmatrix}, \tag{4.7}$$

and the diffusion tensor is the trigonometric anisotropic field

$$D(\boldsymbol{x}) = a \begin{pmatrix} 1+\sin^2(2\pi x) & \sin(2\pi x)\cos(2\pi y) \\ \sin(2\pi x)\cos(2\pi y) & 1+\cos^2(2\pi y) \end{pmatrix}, \tag{4.8}$$

which is symmetric positive definite for every $\boldsymbol{x} \in \Omega$ and $a > 0$. The simulation horizon is $T = 0.24$. A total of 5000 rollouts are generated, of which 4500 are used for training and the remaining 500 for validation. The training and validation splits therefore share the same parameter distribution but use disjoint realisations of the random generated simulation settings. A single representative rollout from this dataset is shown in Figure 4.3. The empirical distributions of the node and edge features across the dataset are reported in Figure 4.4.

4.2.3 Testing datasets

The two testing datasets are designed to assess two complementary properties of the surrogate model: generalisation to velocity and diffusion fields that were not seen during training, and the empirical determination of stability limits in terms of the dimensionless numbers. Both test sets use the same out-of-distribution velocity and diffusion fields but differ in which physical parameter is swept and which is held fixed, thereby isolating the advective and diffusive stability limits respectively.

The velocity field for both test sets is the divergence-free stacked tri-gyre

$$\boldsymbol{v}(\boldsymbol{x}) = b \begin{pmatrix} 2\pi \sin(\pi x)\cos(3\pi y) \\ -\dfrac{2\pi}{3} \cos(\pi x)\sin(3\pi y) \end{pmatrix}, \tag{4.9}$$

which produces a one-by-three stacked arrangement of rotating cells rather than the one-by-two layout of the double-gyre of equation 4.7. The component-wise ranges $v_x \in [-2\pi b, 2\pi b]$ and $v_y \in [-\frac{2\pi}{3} b, \frac{2\pi}{3} b]$ lie within those of the training velocity field, keeping the nodal velocity features inside the support of the training distribution. The diffusion tensor is

$$D(\boldsymbol{x}) = a \begin{pmatrix} 1+\sin^2(\pi x) & \cos(\pi x)\sin(2\pi y) \\ \cos(\pi x)\sin(2\pi y) & 1+\cos^2(\pi y) \end{pmatrix}, \tag{4.10}$$

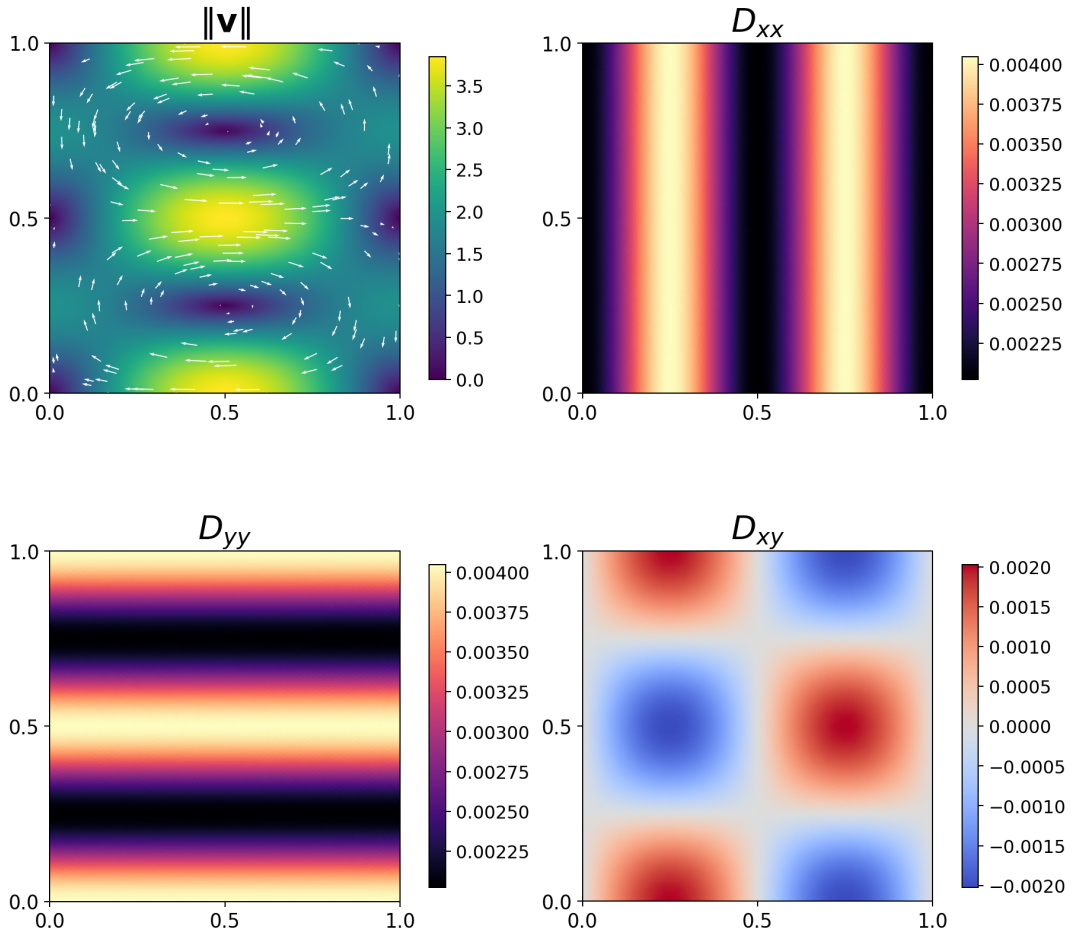


Figure 4.3: One representative realisation of the velocity and diffusion profile. The four panels show the magnitude $\|\boldsymbol{v}(\boldsymbol{x})\|$ with a quiver overlay of the double-gyre velocity field of equation 4.7 (top left), and the three independent components D_{xx} (top right), D_{yy} (bottom left) and D_{xy} (bottom right) of the anisotropic diffusion tensor of equation 4.8, evaluated on the underlying triangular mesh.

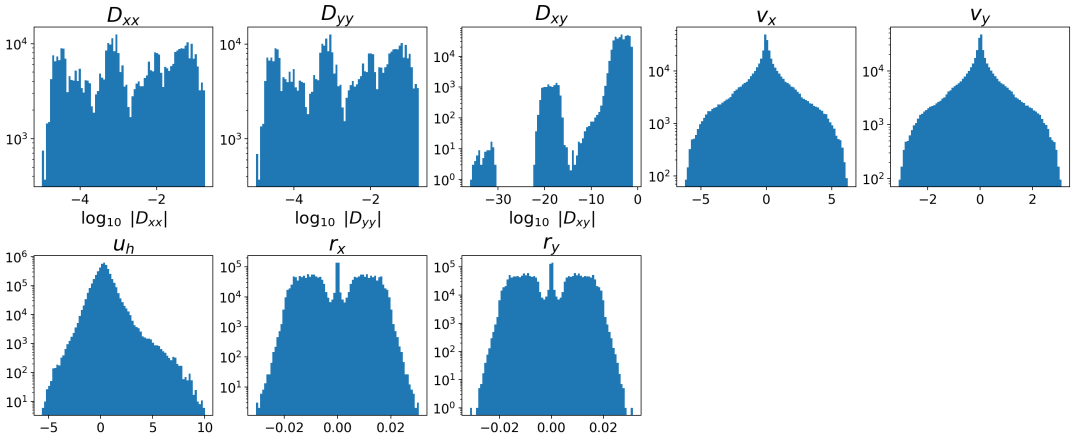


Figure 4.4: Empirical distributions of all node and edge features over the training/validation dataset. The top row shows the static node features D_{xx}, D_{yy}, D_{xy}, v_x and v_y of equation 4.1, the bottom row the dynamic node feature u_h of equation 4.5 and the two edge feature components r_x, r_y of equation 4.3.

which is symmetric positive definite and has component-wise marginal distributions that match those of equation 4.8 under uniform sampling of (x, y), while the spatial profile differs structurally. Both fields therefore differ from the training fields of equations 4.7–4.8.

In the advection test set, the diffusion magnitude is fixed at $a = a_{\min} \approx 1.49 \times 10^{-4}$, chosen so that the Fourier number satisfies $\mathrm{Fo} \leq 0.1$ at the smallest surrogate time step for every mesh size in $[h_{\min}, h_{\max}]$.

This ensures that any observed instability is attributable to advection rather than diffusion, since even the smallest model is stable at this diffusion magnitude, given that the advection is not causing instabilities. The velocity magnitude b is swept over a uniform grid of 300 values,

$$b \in \left\{ \frac{j}{299} \mid j = 0, 1, \ldots, 299 \right\},$$

covering the interval $[0, 1]$.

In the diffusion test set, the velocity magnitude is fixed at $b = b_{\min} \approx 8.70 \times 10^{-2}$, chosen so that the CFL number satisfies CFL ≤ 1.0 for every mesh size in $[h_{\min}, h_{\max}]$ at the smallest surrogate time step, ensuring that any instability is attributable to diffusion rather than advection. The diffusion magnitude a is swept over a log-uniform grid of 300 values,

$$\log_{10} a \in \left\{ -5 + j \, \frac{4}{299} \mid j = 0, 1, \ldots, 299 \right\},$$

covering the interval $[10^{-5}, 10^{-1}]$.

Each test set therefore consists of 300 rollouts (600 in total), providing controlled one-dimensional coverage of the advective and diffusive parameter axes. The mesh size and initial condition are sampled from the same distributions as in the training dataset, and the simulation horizon is $T = 1.92$, to test the models' performance beyond the training simulation window. No rollout from either test set enters the training or validation pipeline. A representative rollout is shown in Figure 4.5, and the empirical feature distributions for the advection and diffusion test sets are given in Figures 4.6 and 4.7, respectively.

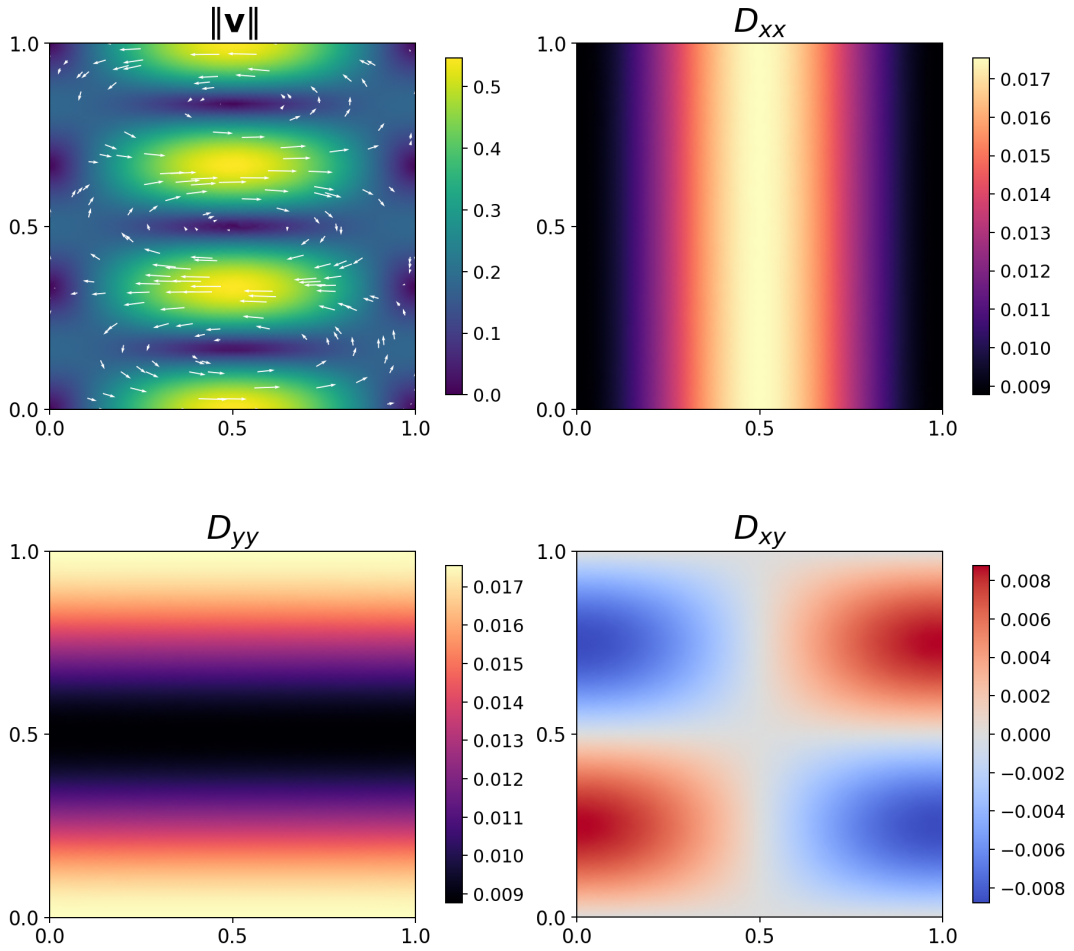


Figure 4.5: Representative realisation of the testing profiles. The panels show the magnitude and quiver representation of the stacked tri-gyre velocity field of equation 4.9 and the three independent components D_{xx}, D_{yy}, D_{xy} of the trigonometric anisotropic diffusion tensor of equation 4.10.

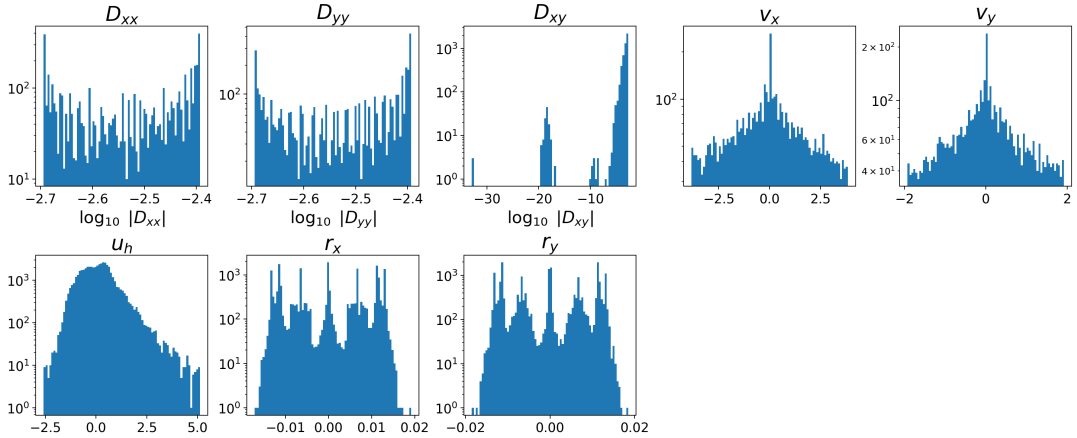


Figure 4.6: Empirical distributions of the node and edge features over the advection test set. The diffusion features D_{xx}, D_{yy}, D_{xy} are sharply concentrated because a is held fixed at $a_{\min}$. The velocity features v_x, v_y are broadly distributed as b sweeps from 0 to 1. All feature values lie within the support of the training distribution.

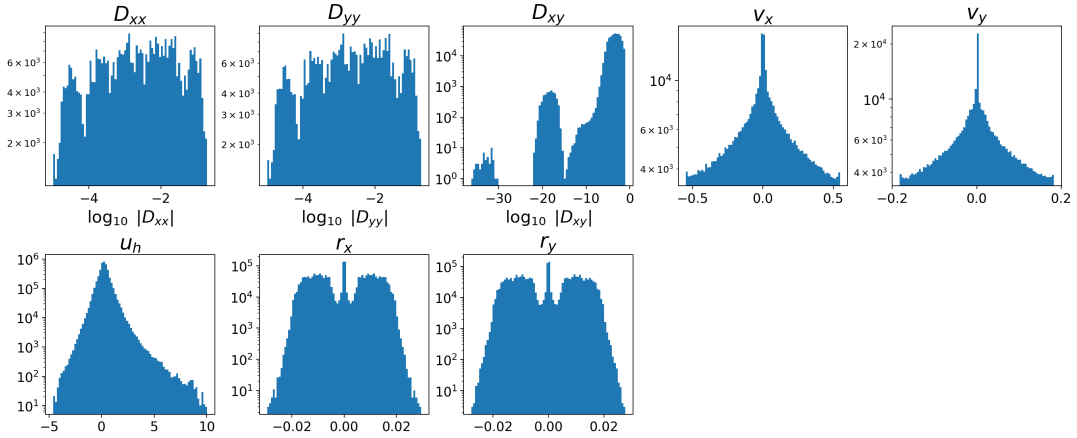


Figure 4.7: Empirical distributions of the node and edge features over the diffusion test set. The velocity features v_x, v_y are sharply concentrated because b is held fixed at $b_{\min}$. The diffusion features D_{xx}, D_{yy}, D_{xy} span the full four-decade range as $\log_{10} a$ sweeps from -5 to -1. All feature values lie within the support of the training distribution.

4.3 Training setup

The surrogate model is trained as a one-step regressor in the sense of equation 3.11. Its parameters $\boldsymbol{\theta}$ are optimised so that the predicted increment $\hat{L}^j$ matches the finite element label L^j of equation 4.6 on the training rollouts of Section 4.2.2.

Within each rollout the saved trajectory U of equation 4.5 contains $N_T = T/\Delta t_{\text{base}}$ snapshots, of which only a small subset is used during an epoch. During training, the following strategy is employed: M admissible input indices n (those satisfying $n+k \le N_T$, with k the surrogate stride) are drawn, without replacement, from each rollout at the start of every epoch and are resampled independently for the next epoch. Over the course of a complete training session, the model will therefore have been exposed to the full simulation window of all the individual rollouts.

The per-node cost is the masked mean-squared error between the predicted increment and the label of equation 4.6, normalised by the surrogate time step Δt so that the loss yields

$$J(\boldsymbol{\theta}; L^j, \hat{L}^j) = \frac{1}{\Delta t^2} \frac{1}{\sum_{i \in V} n_i} \sum_{i \in V} n_i \left(\hat{L}_i^j - L_i^j\right)^2, \tag{4.11}$$

where $n_i \in \{0,1\}$ is the binary mask of equation 3.13 that excludes the duplicate periodic nodes identified in Section 4.1.2. Dividing by Δt^2 ensures that the loss scale is independent of the surrogate stride k, which allows us to use the same hyperparameters, with regard to optimisation, for different strides. No explicit regularisation is added to this loss function.

The cost is minimised with the Adam optimiser from [62], using a base learning rate of $\eta_0 = 10^{-4}$, a

mini-batch size of five graphs per time and an exponential learning-rate schedule

$$\eta_e = \eta_0 \gamma^e, \qquad e = 0, 1, 2, \ldots,$$

where e indexes the epoch and $\gamma \in (0,1)$ is a fixed decay factor chosen such that the learning rate decays with 10^{-1} over 60 epochs. Learning rate scheduling is employed to ensure that the optimisation process remains stable and converges efficiently, by making larger steps in the beginning of the optimisation and smaller steps as training progresses. The latter is almost always employed in the literature and is therefore considered standard practice. Training is run for at most 200 epochs and is halted by an early-stopping rule that monitors the validation cost, computed on the held-out 500 rollouts of the validation set at the same surrogate stride: training stops as soon as the validation cost has not improved by at least $\delta = 0.05$, where the patience for this improvement is set to $P = 10$ epochs. The surrogate model with the lowest validation cost is retained as the final model for assessment on the test dataset.

Three surrogate strides are considered, corresponding to $k = 15, 30, 60$, which through $\Delta t = k \Delta t_{\text{base}}$ yield the surrogate time steps

$$\Delta t_{\text{small}} = 0.015, \qquad \Delta t_{\text{medium}} = 0.030, \qquad \Delta t_{\text{large}} = 0.060.$$

The three values double the temporal stride and therefore double both CFL and Fo at fixed mesh and field parameters, while leaving Pe unaffected. Comparing the surrogate accuracy at the three values therefore isolates the role of the time step from that of the spatial dimensionless number, and provides a direct empirical handle on the question of how the surrogate stability is governed by CFL and Fo at different strides, but also allows for cross-stride comparison.

4.4 Models

Two families of surrogate models are compared. The single-scale family is the encode-process-decode architecture of [42], applied directly to the fine graph G^0 as described in Section 4.1.2. The multiscale family augments the single-scale architecture with the learned restriction and prolongation operators of Section 3.7, organised in a V-cycle over the fine graph and coarsened graph(s) as introduced in Section 4.1.3.

The common components that both the single- and multiscale models possess are introduced now. Firstly, the architectures of the encoders $\varepsilon^v, \varepsilon^e$, decoder δ^v, message passing block of Algorithm 1 all share the same architecture. All encoders map to an encoding dimension of 128. The MLPs present in the architecture use ReLU activations and are followed by a LayerNorm in the latent space (with the exception of the output layer, which is not normalised). All the update functions, edge and node, have residual connections. Together with the Layernorm, they should help mitigate oversmoothing, as discussed in Section 3.3. They also share the optimiser, loss, batch size, learning rate, learning-rate scheduler and early-stopping rule of Section 4.3. The sensitivity with regard to some of these hyperparameters is investigated in the hyperparameter study of Appendix A.4, which resulted in the aforementioned choices.

The variants therefore differ only in two architectural quantities: the total number of message passing blocks and, for the multiscale models, the way in which those blocks are distributed across the levels of the V-cycle. The fixed AMG hyperparameters $\alpha = 1.0$, $\beta = 0.5$, $\gamma = 0.5$ and the strength threshold $\theta = 0.1$ of equation 4.4 are persisted in all datasets. Because all variants share the same encoder, decoder and message passing architecture, their trainable parameter counts are nearly identical: the multiscale models differ from a single-scale model only by the few additional encoders required for the coarse levels and learned restriction and prolongation operators. The comparisons that follow are therefore made at a comparable parameter budget, so that any performance difference is attributable to the message passing budget and its distribution across scales rather than to model size.

Throughout this thesis a self-describing identifier is used for each trained model, summarised in Table 4.2. Each identifier has the form <family>-dt<step>-<blocks/allocation>, encoding the architecture family, the surrogate time step and either the number of message passing blocks or the specific depth allocation.

Table 4.2: Naming convention for the model identifiers used throughout this thesis.

Token	Meaning
Single, Multi	architecture family: single-scale (encode-process-decode on the fine graph only) or multiscale (V-cycle over the coarsened graph(s))
dt15, dt30, dt60	surrogate time step $\Delta t = 0.015, 0.030, 0.060$
<n>mp (single-scale)	total number of message passing blocks N_{mp}
a x b x c (multiscale)	depth allocation across the V-cycle levels, summing to N_{mp} (see Tables 4.3–4.4)

4.4.1 Single-scale variants

For the single-scale family, a sweep over the number of message passing blocks N_{mp} is performed at each surrogate time step. At the smaller stride $\Delta t_{\text{small}} = 0.015$ the values

$$N_{\text{mp}} \in \{2, 4, 6, 8, 10\}$$

are considered, with identifiers Single-dt15-2mp, Single-dt15-4mp, Single-dt15-6mp, Single-dt15-8mp and Single-dt15-10mp. We will use this stride to investigate the stability limits of the single-scale architecture as a function of the number of message passing blocks.

At the larger strides $\Delta t_{\text{medium}}, \Delta t_{\text{large}} = 0.030, 0.060$, but one single scale model is considered, which serves as a baseline to which the multiscale variants are compared. We have $N_{\text{mp}} = 8$ at Δt_{medium} and $N_{\text{mp}} = 16$ at Δt_{large}, with identifiers Single-dt30-8mp and Single-dt60-16mp. Besides serving as a single-scale reference, this also allows us to investigate how the stability limits of the single-scale models varies between strides, given that they should be stable on the same subset of the simulations. Because if the timestep increases with a factor 2, the model can capture the same local dynamics, given that the number of message passing blocks is multiplied by 2 as well. But for the last statement, one should remark that since the mesh is unstructured, this won't scale as linear, but over the complete dataset it should. So if a decrease in performance is witnessed at larger strides between these single-scale models, then this would suggest oversmoothing, especially if this hurts one-step performance.

4.4.2 Multiscale variants

The multiscale models are sized such that the total number of message passing blocks is equal to that of a single-scale baseline of 8 blocks at Δt_{medium} and 16 blocks at Δt_{large}. The $L = 1$ (two-level) V-cycle has three message passing groups: a down-path block on the fine graph, a coarse block on G^1, and an up-path block on the fine graph after the additive correction of equation 3.23. The depth allocation is denoted $[n_{\downarrow}^{(0)}, n^{(1)}, n_{\uparrow}^{(0)}]$ with $\sum n^{(\ell)} = 8$ for Δt_{medium}. The variants considered are listed in Table 4.3. We use symmetric allocations which are more in line with the V-cycle interpretation and therefore have more balanced pre-smoothing and post-smoothing. One could consider more skewed allocations as well, where the coarse scale correction is either earlier or later in the network. An argument for early coarse correction could be that we want to correct the fine representation as early as possible, so that the subsequent fine blocks can work with a better representation. An argument for later coarse correction could be that that it is closer to the output and therefore more directly optimised by the loss. A systematic exploration of skewed (asymmetric) allocations is beyond the scope of this thesis and is left for future work. At the largest stride $\Delta t_{\text{large}} = 0.060$ two complementary studies are performed. First,

Table 4.3: Two-level V-cycle multiscale variants at the surrogate time step $\Delta t_{\text{medium}} = 0.030$. The triplet $[n_{\downarrow}^{(0)}, n^{(1)}, n_{\uparrow}^{(0)}]$ denotes the number of message passing blocks on the fine down-path, on the coarse graph G^1 and on the fine up-path respectively. All variants share the same total budget of eight blocks and the same baseline single-scale reference of eight blocks.

Identifier	Allocation $[n_{\downarrow}^{(0)}, n^{(1)}, n_{\uparrow}^{(0)}]$	Purpose
Multi-dt30-3x2x3	[3, 2, 3]	Heavy fine smoothing, light coarse correction
Multi-dt30-2x4x2	[2, 4, 2]	Balanced fine smoothing and coarse correction
Multi-dt30-1x6x1	[1, 6, 1]	Light fine smoothing, heavy coarse correction

the best-performing two-level allocation at Δt_{medium}, identified by the criteria of Section 4.5, is doubled in every position to obtain a 16-block two-level reference. Second, a three-level double V-cycle ($L = 2$) model is introduced. The depth allocation is now denoted $[n_{\downarrow}^{(0)}, n_{\downarrow}^{(1)}, n^{(2)}, n_{\uparrow}^{(1)}, n_{\uparrow}^{(0)}]$ with $\sum n^{(\ell)} = 16$. The multiscale variants trained on Δt_{large} are listed in Table 4.4.

Table 4.4: Multiscale variants at the surrogate time step $\Delta t_{\text{large}} = 0.060$. The two-level row uses the best two-level allocation at Δt_{medium} (identifier Multi-dt60-4x8x4) with each entry doubled. The three-level rows use $[n_{\downarrow}^{(0)}, n_{\downarrow}^{(1)}, n^{(2)}, n_{\uparrow}^{(1)}, n_{\uparrow}^{(0)}]$ with sixteen total blocks, distributed across the fine graph G^0, the intermediate coarse graph G^1 and the coarsest graph G^2.

Identifier	Allocation	Purpose
Multi-dt60-4x8x4	$2[n_{\downarrow}^{(0)}, n^{(1)}, n_{\uparrow}^{(0)}]_{\text{best}}$	Doubled best two-level
Multi-dt60-3x3x4x3x3	[3, 3, 4, 3, 3]	Symmetric three-level

The multiscale variants of Tables 4.3–4.4 are systematically compared against their respective single-scale baselines (eight and sixteen blocks). The comparison serves two purposes. It quantifies the gain that a coarse correction can provide at fixed parameter and message passing budget, but also how the allocation of blocks across different scales affects the performance. To answer the main questions posed in the introduction, we will therefore compare the limits and overall performance between the following sets of models:

- **Determine single scale limits at Δt_{small}:** Single-dt15-2mp, Single-dt15-4mp, Single-dt15-6mp, Single-dt15-8mp, Single-dt15-10mp.
- **Single-scale vs multiscale limits at Δt_{medium}:** Single-dt30-8mp vs Multi-dt30-3x2x3, Multi-dt30-2x4x2, Multi-dt30-1x6x1.
- **Single-scale vs multiscale limits at Δt_{large}:** Single-dt60-16mp vs Multi-dt60-4x8x4, Multi-dt60-3x3x4x3x3.
- **Determine temporal influence to stability limit between surrogate timesteps:** Single-dt15-4mp, Single-dt30-8mp and Single-dt60-16mp.

4.5 Model assessment and limit determination

To evaluate the performance of the surrogate models and to determine their stability limits, a set of error metrics and procedures is defined in this section. We distinguish one-step error metrics, which measure how well the surrogate predicts the next increment given the current state, and rollout error metrics, which measure how well the surrogate trajectory matches the reference trajectory over time when applied autoregressively. For the one-step metrics, we will use the surrogate- and label updates $\hat{\boldsymbol{L}}^j$, $\boldsymbol{L}^j$ as reference. To evaluate the surrogate performance in rollout, we will compare the model solution to the reference solution at a time somewhere in the simulation window, so u_h versus u_{GNN}. Moreover, the rollout metrics will show us for which cases the surrogate trajectory blows up, which is crucial to determine the stability limits of the surrogate. For the latter, we will use the two test sets with structured coverage of the dimensionless numbers CFL and Fo, which will allow us to extract empirical stability limits as a function of these numbers.

4.5.1 General error metrics

The mean squared error (MSE) of equation 4.11 is used as the training cost on which the model is optimised. This value is also reported on the validation set and it is used for the early stopping criterion. For those two functionalities, the MSE is ideal as it is fast to compute and gives an idea of the generalisation on unseen samples relative to the training performance. This MSE is already divided by Δt^2 to ensure that the scale of the loss is independent of the surrogate time step, which allows us to use the same hyperparameters for different surrogate time steps. However, this does not necessarily mean that the MSE is a good metric to compare performance between different time steps, as the MSE is not normalised by the scale of the underlying true updates. The dividing of Δt^2 does not completely resolve this issue, as the scale of the true updates may not scale linearly with the time step.

To resolve this issue, we introduce the R2 score, which computes the proportion of variance explained between the surrogate- and the label update per graph. The R2 score is defined as

$$R^2 = 1 - \frac{\sum_{i \in V} n_i \, (\hat{L}_i^j - L_i^j)^2}{\sum_{i \in V} n_i \, (L_i^j - \bar{L}^j)^2},$$

where $\bar{L}^j = \sum_{i \in V} n_i L_i^j / \sum_{i \in V} n_i$ is the mean of the label updates for that specific time step. The R2 score is a normalised metric that gives an idea of how well the surrogate prediction matches the label relative to a naive baseline that always predicts the mean of the label. An R2 score of 1 means perfect prediction, while an R2 score of 0 means that the surrogate is no better than the naive baseline. Negative R2 scores indicate that the surrogate is worse than the naive baseline. The R2 score can be used to compare surrogate performance between surrogate time steps, as it is normalised by the scale of the label and therefore easy to interpret [63]. It is also still a relatively fast metric to compute, as it only requires summations over the nodes and does not require integration over the domain. However, it is still a pointwise metric that does not capture the functional nature of the surrogate. For example, shouldn't a node that is connected to large mesh elements have a larger influence on the surrogate performance than a node that is connected to small mesh elements?

To address the latter, a metric is defined that is able to measure performance in a more functional manner. That is, we want to infer solutions on an unstructured mesh, so it should only be logical that we evaluate the surrogate performance in a way that is independent of the mesh. Besides error between the functions on the mesh, we also want to see if the gradients match. To this end, we define the relative L^2 and H^1-seminorm errors as in Definition 4.5.1 below. These metrics are evaluated by integrating over the domain Ω and therefore do not depend on the particular discretisation of Ω into a mesh. This way, we can compare surrogate predictions to the reference solution in a way that is independent of the mesh, which is more in line with the functional nature of the surrogate. The definition yields:

Definition 4.5.1 (Relative L^2 and H^1-seminorm errors)**.** Let $u_h \in \mathbb{P}_1(\mathcal{T}_h)$ be the linear interpolant of the finite element reference solution at the mesh nodes, and let $\boldsymbol{u}_{\text{GNN}} \in \mathbb{R}^{|V|}$ denote the surrogate prediction at the same nodes. Let $u_{\text{GNN}}(\boldsymbol{x}, t) \in \mathbb{P}_1(\mathcal{T}_h)$ be the linear interpolant of $\boldsymbol{u}_{\text{GNN}}$, and let $e := u_h - u_{\text{GNN}}$. The relative L^2 and H^1-seminorm errors are

$$\mathscr{E}_{L^2}(u_h, u_{\text{GNN}}) := \sqrt{\frac{\int_\Omega e^2(\boldsymbol{x}, t)\, d\boldsymbol{x}}{\delta + \int_\Omega u_h^2(\boldsymbol{x}, t)\, d\boldsymbol{x}}}, \tag{4.12}$$

$$\mathscr{E}_{H^1}(u_h, u_{\text{GNN}}) := \sqrt{\frac{\int_\Omega \|\nabla e(\boldsymbol{x}, t)\|^2\, d\boldsymbol{x}}{\delta + \int_\Omega \|\nabla u_h(\boldsymbol{x}, t)\|^2\, d\boldsymbol{x}}}, \tag{4.13}$$

with $\delta = 10^{-12}$ a small regularisation term that prevents division by zero on rollouts whose reference solution converge to a steady-state near zero.

The domain integrals in equations (4.12)–(4.13) are evaluated with the Gaussian quadrature rules provided by NGSolve [61], applied to the piecewise-linear interpolants on the reference mesh. The L^2 error captures the global magnitude of the discrepancy, while the H^1-seminorm error captures whether the gradients of the surrogate solution match those of the reference. The metrics of Definition 4.5.1 are evaluated in two regimes. In the one-step regime, the squared error is divided by the label-update, rather than the function at $t + \Delta t$. Meaning instead of $u_h(\boldsymbol{x}, t + \Delta t)$, the denominator is $u_h(\boldsymbol{x}, t + \Delta t) - u_h(\boldsymbol{x}, t)$. Naturally, the error function e between the increments is equal to the error between the functions at $t + \Delta t$, as they have the same starting point. We will indicate the one-step error metrics by $\mathscr{E}_{L^2}(\Delta u)$, $\mathscr{E}_{H^1}(\Delta u)$, and the rollout error metrics by $\mathscr{E}_{L^2}(u)$, $\mathscr{E}_{H^1}(u)$. The one-step errors are evaluated on the validation set, while the rollout errors are evaluated on the test sets.

The one-step error is reported on the validation rollouts by averaging the relative norms over 100 randomly drawn input snapshots per rollout, which gives a low-variance estimate of the performance over the complete dataset. In the rollout regime, the surrogate is initialised from the reference state $U_{:,0}$ and applied autoregressively as depicted in Figure 3.6, so that the relative norms measure the cumulative deviation of the surrogate trajectory from the reference. To track the temporal growth of this deviation, the rollout error is reported at the checkpoints

$$t \in \{0.24, 0.48, 0.72, 0.96, 1.20, 1.44, 1.68, 1.92\}, \tag{4.14}$$

which are the eight integer multiples of the training horizon $T = 0.24$, and therefore span the training horizon together with an extrapolation horizon up to $8\,T$. These checkpoints were chosen as all different surrogate timesteps have a value at these checkpoints and they allow us to track the error growth over time, especially beyond the training horizon.

4.5.2 Determination of stability limits

The structured grid sampling of Section 4.2.3, that is, the one-dimensional sweeps of the diffusion magnitude parameter a and advection magnitude parameter b over an equispaced grid, yields two test sets:

one from which an empirical stability limit can be extracted as a function of CFL, and one as a function of Fo. Since the procedure is identical for both limits, we describe it for the CFL case only as the procedure is the same for both test sets.

Let $\mathscr{D}_{\text{test}}$ denote the test dataset of Section 4.2.3 and let $\text{CFL}^{(s)}$ denote the CFL number of trajectory $s \in \mathscr{D}_{\text{test}}$. For each trajectory $s \in \mathscr{D}_{\text{test}}$, the rollout error $\mathscr{E}^{(s)}_{\text{rollout}}$ is evaluated at the final checkpoint $t = 1.92$. A continuous piecewise-linear function with $K = 3$ segments is then fitted to the datapoints $\{(\text{CFL}^{(s)}, \mathscr{E}^{(s)}_{\text{rollout}})\}_{s\in\mathscr{D}_{\text{test}}}$, similar to [64]. Do note that three line segments make sense for the problem considered, as the following three regimes are to be expected: the stable domain, which is before the limit, some interfacing behaviour between stable and unstable and the unstable regime. So making these three line segments make sense with the hypothetical expectation of how the function will look like. If there is no instability observed, the fitted curve is a flat line, which can also be constructed by three linear line segments. Two free internal knots $\xi_1 < \xi_2$ partition the x-range into three intervals; the fit is parametrised via the truncated-power basis, also known as linear B-spline basis, as

$$f(x; \boldsymbol{\xi}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x + \beta_2 (x - \xi_1)_+ + \beta_3 (x - \xi_2)_+, \tag{4.15}$$

where $(z)_+ = \max(0, z)$ are the linear B-splines. It can basically be seen as performing three separate linear regression with a continuity constraint at the knots. The stability limit is defined as the smallest CFL at which f exceeds a prescribed error threshold $\varepsilon = 1$, beyond which the surrogate trajectory is considered to have diverged significantly from the reference: a so-called blow-up.

To fit equation (4.15), the $|\mathscr{D}_{\text{test}}|$ datapoints (x_s, y_s), with labels $y_s = \mathscr{E}^{(s)}_{\text{rollout}}$, are stacked into the label vector $\boldsymbol{y}$ and the design matrix $\boldsymbol{A}(\boldsymbol{\xi}) \in \mathbb{R}^{|\mathscr{D}_{\text{test}}|\times 4}$, whose s-th row evaluates the four basis functions of equation (4.15) at x_s, namely $[1,\ x_s,\ (x_s - \xi_1)_+,\ (x_s - \xi_2)_+]$. Evaluating the model at every datapoint is then the single matrix product $\boldsymbol{A}(\boldsymbol{\xi})\,\boldsymbol{\beta}$, and the fit proceeds in two stages: for a fixed knot pair $\boldsymbol{\xi}$ the coefficients $\boldsymbol{\beta}$ follow in closed form, after which the knots are selected by a search over $\boldsymbol{\xi}$.

Consider the first stage. For any fixed $\boldsymbol{\xi} = (\xi_1, \xi_2)$, equation (4.15) is linear in $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \beta_3)^\top$, and the coefficients are determined by minimising the ridge-penalised least-squares objective

$$J(\boldsymbol{\beta}; \boldsymbol{\xi}) = \left\| \boldsymbol{y} - \boldsymbol{A}(\boldsymbol{\xi})\,\boldsymbol{\beta} \right\|_2^2 + \lambda_r \left\| \boldsymbol{\beta} \right\|_2^2, \tag{4.16}$$

where $\lambda_r = 10^{-6}$ is a small ridge penalty added for numerical stability. Since J is convex and quadratic in $\boldsymbol{\beta}$, its unique minimiser satisfies the stationarity condition

$$\nabla_{\boldsymbol{\beta}} J = -2\,\boldsymbol{A}(\boldsymbol{\xi})^\top \big(\boldsymbol{y} - \boldsymbol{A}(\boldsymbol{\xi})\,\boldsymbol{\beta}\big) + 2\lambda_r\,\boldsymbol{\beta} = \boldsymbol{0},$$

which rearranges into the regularised normal equations $\big(\boldsymbol{A}^\top \boldsymbol{A} + \lambda_r\,\boldsymbol{I}\big)\boldsymbol{\beta} = \boldsymbol{A}^\top \boldsymbol{y}$ and hence the closed-form solution

$$\hat{\boldsymbol{\beta}}(\boldsymbol{\xi}) = \big(\boldsymbol{A}(\boldsymbol{\xi})^\top \boldsymbol{A}(\boldsymbol{\xi}) + \lambda_r\,\boldsymbol{I}\big)^{-1} \boldsymbol{A}(\boldsymbol{\xi})^\top \boldsymbol{y}. \tag{4.17}$$

This is the standard generalised (kernel) regularised least-squares solution [65]. Here λ_r denotes the ridge penalty and should not be confused with the loss-regularisation weight λ of equation 3.13.

In the second stage the knots are selected. Because the penalty $\lambda_r \left\| \boldsymbol{\beta} \right\|_2^2$ in equation (4.16) serves only to keep the 4×4 system well conditioned, it is excluded from the model-selection criterion: the knots minimise the unpenalised residual sum of squares of the resulting fit,

$$\hat{\boldsymbol{\xi}} = \operatorname*{arg\,min}_{\xi_1 < \xi_2} \left\| \boldsymbol{y} - \boldsymbol{A}(\boldsymbol{\xi})\,\hat{\boldsymbol{\beta}}(\boldsymbol{\xi}) \right\|_2^2, \tag{4.18}$$

so that candidate knot configurations are compared purely on goodness-of-fit. Since $\hat{\boldsymbol{\beta}}(\boldsymbol{\xi})$ is available analytically for every $\boldsymbol{\xi}$, equation (4.18) is solved by a grid search: $n_g = 50$ candidate knot positions are placed equispaced in the interior of the observed CFL-range, and all $\binom{n_g}{2} = 1225$ knot pairs with $\xi_1 < \xi_2$ are evaluated. Each evaluation solves a single 4×4 system, making the total computational cost negligible. The pair $\hat{\boldsymbol{\xi}}$ achieving the smallest residual is retained as the final knot configuration.

Since $\mathscr{E}^{(s)}_{\text{rollout}}$ varies over several orders of magnitude, ranging from well below 10^{-2} in stable regimes to values exceeding the blow-up threshold of $\varepsilon = 1$ in unstable ones, fitting in the linear scale would allow the large blow-up errors to dominate and distort the regression in the stable regime where precision matters most. The regression is therefore performed in $\log_{10}$ space: the label vector is transformed to $\tilde{y}_s = \log_{10}(\mathscr{E}^{(s)}_{\text{rollout}})$ before solving (4.17), so that each order of magnitude receives equal weight. This is the only part where the procedure for the Fourier-number axis differs, as the x-values are additionally mapped to $\log_{10}(\text{Fo})$ before fitting; the resulting fit is therefore a piecewise-linear curve in $(\log_{10} \text{Fo}, \log_{10} \mathscr{E}_{L^2})$ space.

Chapter 5

Results

In this chapter, the aim is to provide results, such that we can later answer the main questions posed in the introduction. The question whether the surrogate time integrators can learn to accurately reproduce the FEM solutions of the advection-diffusion equation in terms of qualitative behaviour, one-step performance and multi-step performance is answered in Sections 5.2 and 5.3. The qualitative behaviour of the surrogate time integrators can furthermore be confirmed by the snapshot visualisations that allows for comparison with the FEM reference solutions.

For the second main question, we will zoom in on the performance of individual trajectories more and evaluate the errors at the final checkpoint $t = 1.92$ of the trajectories. This way, we can establish a general trend in the performance of the surrogates as a function of the CFL and Fo numbers. The trend is made concrete by applying the piecewise linear regression fit introduced in the prior chapter, which allows us to determine the empirical limits of stability for the surrogate time integrators.

5.1 Performance of conventional numerical time integrators

Before delving into the surrogate simulators, we establish a reference benchmark for conventional time integrators on the advection-diffusion problem. We evaluate the explicit methods forward Euler and RK4, alongside the implicit Crank–Nicolson method. As an implicit method, Crank–Nicolson is expected to be more stable at larger time steps, but for the same reason it does not compare directly to the explicit surrogate models. At the fine timestep size $\Delta t = 0.0001$ (Figure A.6), all methods perform acceptably, and the performance even increases as the amount of diffusion increases. However, the picture changes dramatically at larger timestep sizes. In Figure A.7, the explicit time integrators forward Euler and RK4 both diverge rapidly at $\Delta t = 0.015$ and $\Delta t = 0.030$, remaining stable only for CFL numbers below 1. Only Crank–Nicolson maintains stability across a significant portion of the test set, as expected for an implicit method, although its performance does decrease. As these results are therefore not very informative for the surrogate models, we will not include them in the main text but instead place them in the appendix (Section A.6). It is interesting to see, however, that some of the models discussed later do remain stable in the same physical regions where the explicit methods diverge.

5.2 One-step performance on the training and validation sets

This section reports the one-step predictive accuracy of all surrogate variants on the training and validation sets of Section 4.2.2. The models are grouped by surrogate time step and assessed via the metrics of Section 4.5: the scaled mean squared error $\mathrm{MSE}/\Delta t^2$ (equation 4.11), the R^2 score relative to the true one-step increments, and the functional relative norms $\mathscr{E}_{L^2}(\Delta u)$ and $\mathscr{E}_{H^1}(\Delta u)$ (Definition 4.5.1). Reporting both the training and validation $\mathrm{MSE}/\Delta t^2$ reveals any systematic generalisation gap that grows as the model capacity decreases.

5.2.1 Single-scale models at $\Delta t_{\mathrm{small}} = 0.015$

Table 5.1 collects the one-step performance of the five single-scale variants at $\Delta t_{\mathrm{small}}$. Increasing N_{mp} only isolates the effect of model depth at a fixed surrogate time step and allows an assessment of whether additional message-passing layers provide a consistent improvement in one-step accuracy or not. An

example of a successful inferred one-step update on the unseen validation rollouts can be seen in Figure 5.1. This is just one of many examples, where this surrogate is able to qualitatively capture the true increment function over the entire domain.

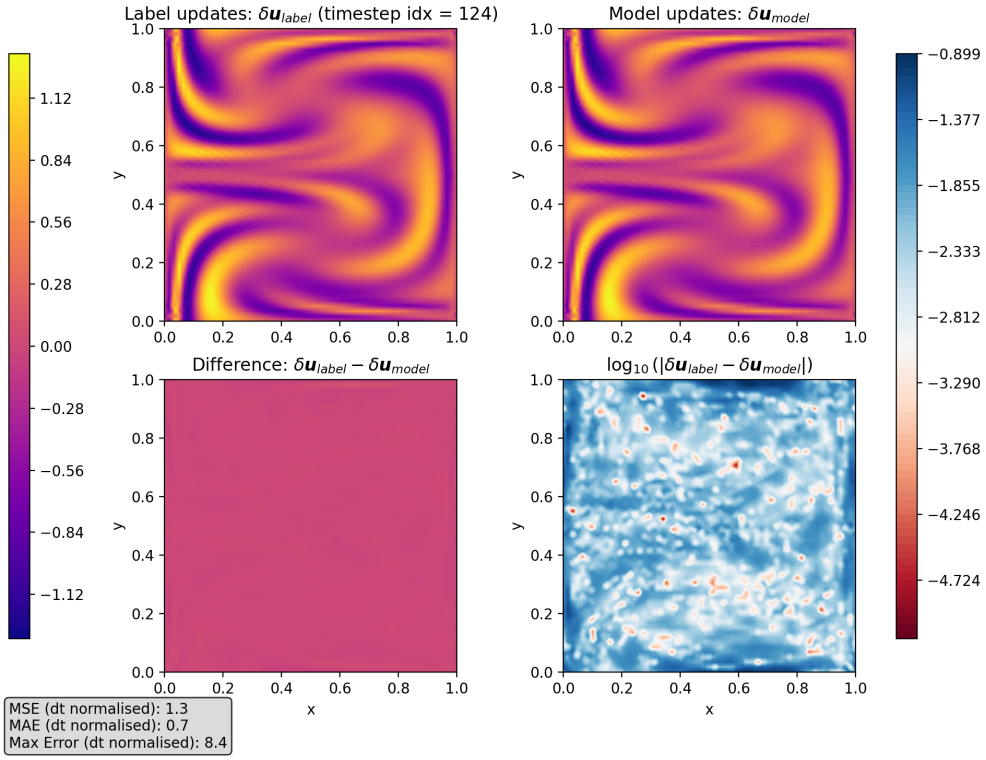


Figure 5.1: Example of a successful one-step prediction by the surrogate model. The left upper panel shows the true increment Δu_{label} of the solution at a random time instance of a random simulation from the training set, and the right upper panel shows the surrogate's prediction of this increment. The lower panels show the pointwise error between the true and predicted increments in normal and log scale. The surrogate model in this example is the single-scale 10-layer model at Δt_{small}.

Table 5.1: One-step performance of the single-scale models at $\Delta t_{\text{small}} = 0.015$ on the training and validation sets, where 100 random time instances are taken from each rollout. All values are reported as mean ± standard deviation across samples.

Model	MSE/Δt^2 (train)	MSE/Δt^2 (val)	ΔMSE/Δt^2	R^2 (val)	$\mathscr{E}_{L^2}(\Delta u)$ (val)	$\mathscr{E}_{H^1}(\Delta u)$ (val)
`Single-dt15-2mp`	0.966 ± 4.49	1.01 ± 5.41	0.0423	0.985 ± 0.128	0.0595 ± 0.091	0.304 ± 0.64
`Single-dt15-4mp`	0.212 ± 1.14	0.243 ± 1.44	0.0315	0.994 ± 0.0445	0.0372 ± 0.0584	0.18 ± 0.381
`Single-dt15-6mp`	0.121 ± 0.488	0.147 ± 0.645	0.0253	0.995 ± 0.065	0.0328 ± 0.0596	0.13 ± 0.262
`Single-dt15-8mp`	0.0633 ± 0.226	0.0822 ± 0.331	**0.0189**	0.997 ± 0.0224	0.0259 ± 0.0404	0.0979 ± 0.184
`Single-dt15-10mp`	$\mathbf{0.047 \pm 0.137}$	$\mathbf{0.0664 \pm 0.221}$	0.0195	$\mathbf{0.998 \pm 0.019}$	$\mathbf{0.0236 \pm 0.038}$	$\mathbf{0.0846 \pm 0.156}$

Across every column of Table 5.1 the ranking is monotone in N_{mp}: Single-dt15-10mp attains the lowest scaled mean squared error on both the training and validation sets, the highest R^2, and the smallest $\mathscr{E}_{L^2}(\Delta u)$ and $\mathscr{E}_{H^1}(\Delta u)$ relative norms. The train-validation gap ΔMSE/Δt^2 shrinks monotonically as model capacity grows, from 0.042 for Single-dt15-2mp down to 0.020 for Single-dt15-10mp. The deeper architectures therefore overfit relatively less on the training data than the shallower ones, and there seems to be no sign of oversmoothing, as the performance just improves. Comparing the two functional norms shows that $\mathscr{E}_{H^1}(\Delta u)$ is consistently larger than $\mathscr{E}_{L^2}(\Delta u)$, indicating that the surrogate increments reproduce the function Δu_h more accurately than its gradient.

Figure 5.2 visualises the training and validation MSE/Δt^2 as a function of the number of message-passing blocks, showing both training and validation curves.

The curve in Figure 5.2 shows that the bulk of the improvement is realised between $N_{\text{mp}} = 2$ and $N_{\text{mp}} = 4$. Beyond four blocks the gains become progressively smaller, and the step from Single-dt15-8mp to Single-dt15-10mp amounts to only about 0.03 in MSE/Δt^2. On the linear vertical axis the train-validation gap narrows visibly with depth, consistent with the decreasing ΔMSE/Δt^2 column of Table 5.1.

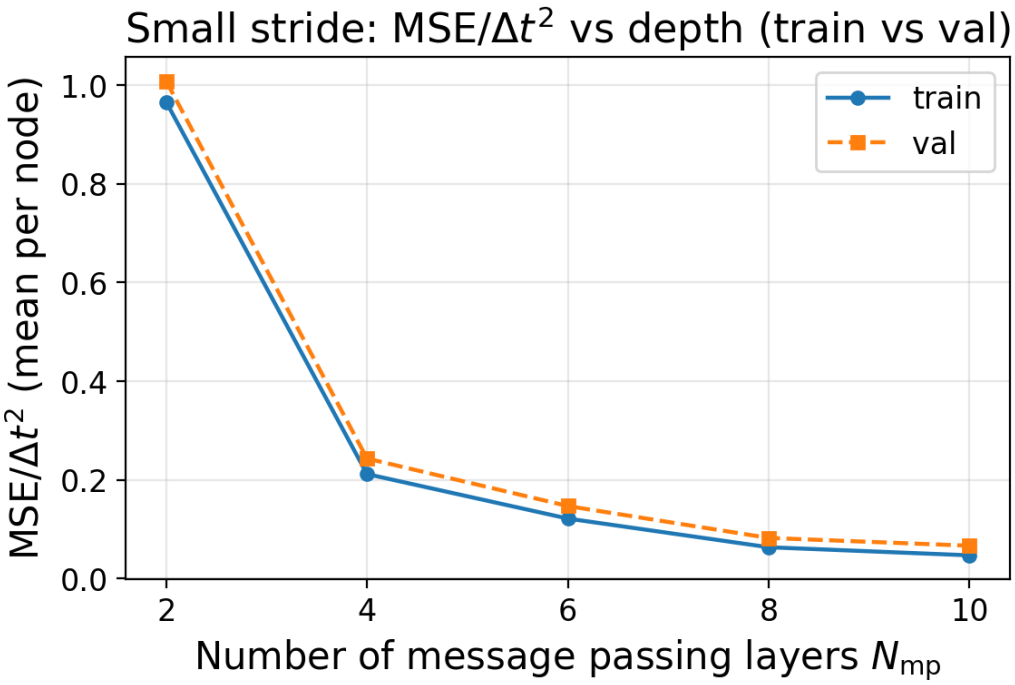


Figure 5.2: MSE/Δt^2 as a function of the number of message-passing blocks N_{mp} for the single-scale models at $\Delta t_{\text{small}} = 0.015$. Solid and dashed lines correspond to the training and validation sets respectively.

5.2.2 Single-scale and multiscale models at $\Delta t_{\text{medium}} = 0.030$

Table 5.2 reports the one-step performance for the baseline single-scale model and the three two-level multiscale variants at Δt_{medium}, all sharing a total budget of eight message-passing blocks. The comparison reveals whether distributing blocks across fine and coarse levels improves one-step accuracy relative to concentrating all blocks on the fine graph.

Table 5.2: One-step performance of the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$. The allocation column indicates the distribution of message-passing blocks across the V-cycle levels $[n_{\downarrow}^{(0)}, n^{(1)}, n_{\uparrow}^{(0)}]$. All values are mean ± std.

Model	Allocation	MSE/Δt^2 (train)	MSE/Δt^2 (val)	ΔMSE/Δt^2	R^2 (val)	$\mathcal{E}_{L^2}(\Delta u)$ (val)	$\mathcal{E}_{H^1}(\Delta u)$ (val)
`Single-dt30-8mp`	[8]	0.0975 ± 0.421	0.167 ± 1.47	0.0697	0.997 ± 0.0171	0.0289 ± 0.041	0.1 ± 0.179
`Multi-dt30-3x2x3`	[3, 2, 3]	0.0614 ± 0.169	0.0943 ± 0.461	0.0329	**0.998 ± 0.0217**	0.0229 ± 0.0358	0.0789 ± 0.135
`Multi-dt30-2x4x2`	[2, 4, 2]	**0.0464 ± 0.12**	**0.0664 ± 0.272**	**0.0201**	0.998 ± 0.033	**0.02 ± 0.0349**	**0.0758 ± 0.204**
`Multi-dt30-1x6x1`	[1, 6, 1]	0.0886 ± 0.257	0.11 ± 0.369	0.0217	0.997 ± 0.0344	0.0237 ± 0.0414	0.0861 ± 0.167

In contrast with the small-stride sweep, no variant dominates every metric in Table 5.2 as they all fall within their respective standard deviations for every considered metric. However, on average the multi-dt30-2x4x2 attains all the best values, with exception to the validation R^2, where it is tied with multi-dt30-3x2x3. It should be stressed that, on this one-step level, the differences between the multiscale variants are minor on all metrics, and particularly so on the relative L^2 and H^1 norms. Little can be concluded from them beyond the consistent gap to the single-scale baseline. But since Multi-dt30-2x4x2 is one of the best contenders on average and has the best balance between coarse and fine blocks, its allocation $[2,4,2]$ is the one carried over (doubled to $[4,8,4]$) to the large-stride model Multi-dt60-4x8x4 in the next section, when we extend the comparison to the large stride. But it can be seen that for one-step performance, the multiscale models are not very sensitive to the exact allocation of blocks.

Figure 5.3 shows the same data graphically: on both the training and validation sets all three multiscale variants attain a lower MSE/Δt^2 than the single-scale baseline Single-dt30-8mp. Within the multiscale group, Multi-dt30-3x2x3 and Multi-dt30-2x4x2 sit very close to each other, with Multi-dt30-2x4x2 slightly below Multi-dt30-3x2x3 on the validation MSE bar.

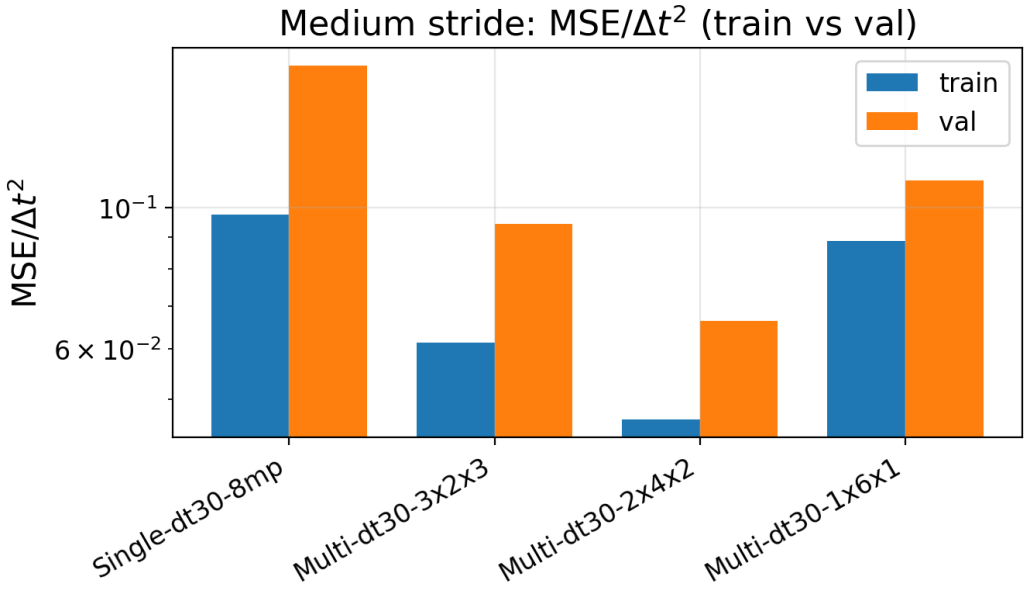


Figure 5.3: Training and validation MSE/Δt^2 for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$.

5.2.3 Single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$

Table 5.3 extends the comparison to the largest surrogate time step, where the single-scale baseline uses sixteen message-passing blocks. The two-level model Multi-dt60-4x8x4 is the best-performing two-level allocation from Δt_{medium} with all block counts doubled, and Multi-dt60-3x3x4x3x3 introduces a three-level V-cycle.

Table 5.3: One-step performance of the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$. The allocation column for Multi-dt60-3x3x4x3x3 lists the five-entry V-cycle distribution $[n_{\downarrow}^{(0)}, n_{\downarrow}^{(1)}, n^{(2)}, n_{\uparrow}^{(1)}, n_{\uparrow}^{(0)}]$. All values are mean ± std.

Model	Allocation	MSE/Δt^2 (train)	MSE/Δt^2 (val)	ΔMSE/Δt^2	R^2 (val)	$\mathscr{E}_{L^2}(\Delta u)$ (val)	$\mathscr{E}_{H^1}(\Delta u)$ (val)
`Single-dt60-16mp`	[16]	0.071 ± 0.33	0.226 ± 1.93	0.155	0.996 ± 0.0359	0.0308 ± 0.0507	0.0792 ± 0.134
`Multi-dt60-4x8x4`	[4, 8, 4]	0.0792 ± 0.2	0.15 ± 0.673	0.0707	**0.997 ± 0.0215**	0.0303 ± 0.0363	0.08 ± 0.119
`Multi-dt60-3x3x4x3x3`	[3, 3, 4, 3, 3]	**0.0305 ± 0.0742**	**0.0607 ± 0.255**	**0.0302**	0.997 ± 0.0501	**0.0202 ± 0.0457**	**0.0624 ± 0.131**

At Δt_{large} the three-level model Multi-dt60-3x3x4x3x3 attains the lowest score on every metric of Table 5.3 except for the validation R^2, on which it is tied with the two-level Multi-dt60-4x8x4 to the reported precision. The single-scale Single-dt60-16mp is outperformed by both multiscale variants on the training and validation MSE, as well as on the two relative norms. As at the medium stride, the differences between the two multiscale variants on the relative norms are small, so the table is read mainly for the consistent advantage of the multiscale models over the single-scale baseline.

Figure 5.4 confirms the table-level ranking: both multiscale variants achieve a lower training and validation MSE/Δt^2 than the single-scale Single-dt60-16mp, and the three-level Multi-dt60-3x3x4x3x3 sits clearly below the two-level Multi-dt60-4x8x4 on both bars.

5.2.4 Comparison across surrogate time steps

To investigate whether the surrogate accuracy, measured on a scale that is independent of Δt, is consistent across the three training strides, Table 5.4 compares Single-dt15-4mp, Single-dt30-8mp and Single-dt60-16mp. These models are chosen such that when the surrogate time step is doubled, the number of message-passing blocks is also doubled, so that the surrogate should cover the same portion of the CFL and Fourier domain on which they are stable. If one-step R^2 is stable across strides, the models learn the same underlying mapping regardless of Δt. Any systematic trend would indicate an intrinsic dependence of learning difficulty on the surrogate time step.

The ranking in Table 5.4 is not consistent across metrics. The large-stride Single-dt60-16mp attains the lowest training MSE/Δt^2 and the smallest $\mathscr{E}_{H^1}(\Delta u)$ on the validation set, whereas the medium-stride

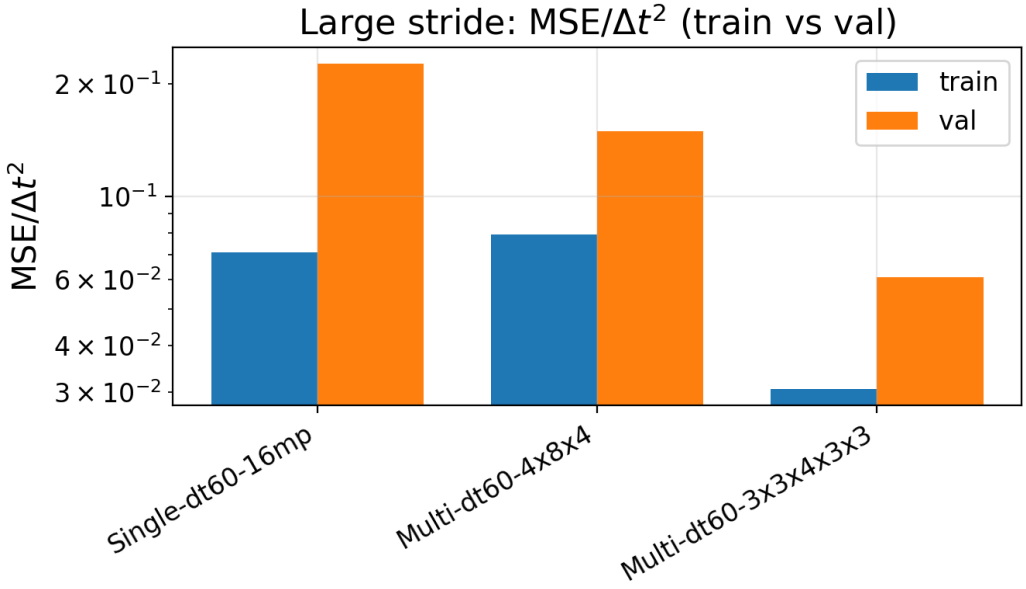


Figure 5.4: Training and validation MSE/Δt^2 for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$.

Table 5.4: One-step performance of the single-scale models compared across surrogate time steps, Single-dt15-4mp, Single-dt30-8mp and Single-dt60-16mp, evaluated on their respective validation sets, where 100 random time instances are taken from each rollout. All values are mean ± std.

Model	Δt	MSE/Δt^2 (train)	MSE/Δt^2 (val)	ΔMSE/Δt^2	R^2 (val)	$\mathscr{E}_{L^2}(\Delta u)$ (val)	$\mathscr{E}_{H^1}(\Delta u)$ (val)
`Single-dt15-4mp`	0.015	0.212 ± 1.14	0.243 ± 1.44	**0.0315**	0.994 ± 0.0445	0.0372 ± 0.0584	0.18 ± 0.381
`Single-dt30-8mp`	0.030	0.0975 ± 0.421	$\mathbf{0.167 \pm 1.47}$	0.0697	$\mathbf{0.997 \pm 0.0171}$	$\mathbf{0.0289 \pm 0.041}$	0.1 ± 0.179
`Single-dt60-16mp`	0.060	$\mathbf{0.071 \pm 0.33}$	0.226 ± 1.93	0.155	0.996 ± 0.0359	0.0308 ± 0.0507	$\mathbf{0.0792 \pm 0.134}$

Single-dt30-8mp takes the lead on the validation MSE/Δt^2, the validation R^2 and the validation $\mathscr{E}_{L^2}(\Delta u)$. Single-dt15-4mp has the best generalisation gap.

Figure 5.5 shows the R^2 score for the three single-scale models side by side, making any trend across Δt immediately visible.

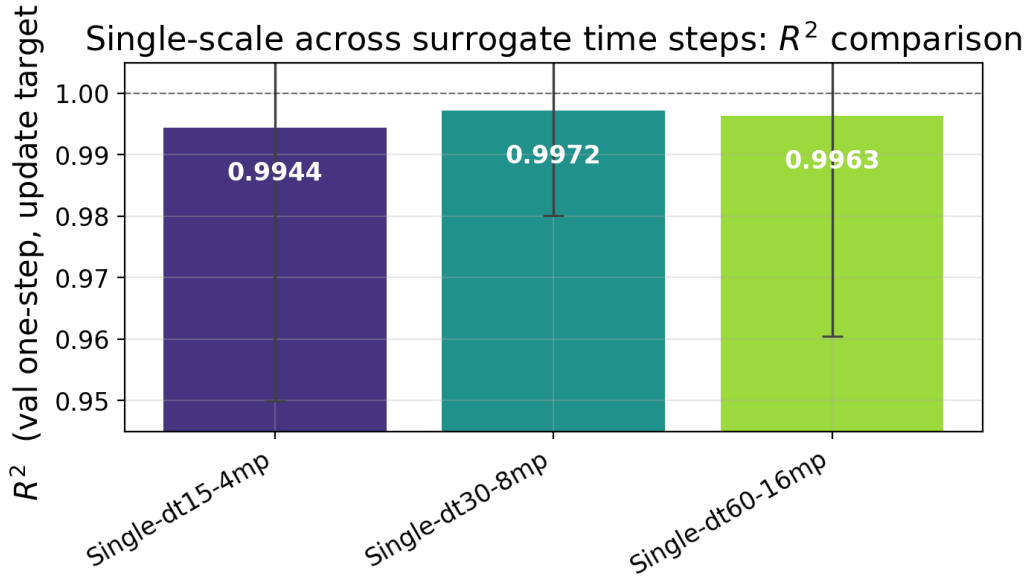


Figure 5.5: Validation R^2 score for the single-scale models across surrogate time steps: Single-dt15-4mp ($\Delta t = 0.015$), Single-dt30-8mp ($\Delta t = 0.030$) and Single-dt60-16mp ($\Delta t = 0.060$). The three scores are deliberately close: once normalised by the scale of the true one-step increments, the models that cover the same proportion of the training CFL and Fourier domain attain comparable one-step accuracy regardless of the surrogate time step. Error bars show the standard deviation across the validation cases.

The R^2 values in Figure 5.5 are very close across the three strides. Once the predictions are normalised by the scale of the true one-step increments, the three models that cover the same proportion of the training CFL and Fourier domain achieve comparable one-step accuracy. The table shows the same story when comparing the $\mathscr{E}_{L^2}(\Delta u)$. However, the $\mathscr{E}_{H^1}(\Delta u)$ does seem to improve more drastically at larger strides, which may indicate that the surrogate is better at learning the gradient of the increment function if the model has a larger receptive field, which is the case for the larger strides.

5.3 Surrogate timestepping performance

This section evaluates the mean autoregressive performance of all surrogate variants on the test sets of Section 4.2.3. The surrogate is initialised from the reference state and stepped forward to the final horizon $t = 1.92$, which is eight times the training horizon $T = 0.24$. For each model group the relative $\mathscr{E}_{L^2}(u)$ and $\mathscr{E}_{H^1}(u)$ errors (Definition 4.5.1) are tabulated at the eight checkpoints of equation 4.14 and visualised as time series. A selection of field snapshots is additionally shown to illustrate qualitative behaviour of the surrogates on selected test rollouts from the advection- and diffusion test sets. The two structured test sets of Section 4.2.3 are referred to throughout as the advection test set (varying CFL) and the diffusion test set (varying Fo). Only the $\mathscr{E}_{L^2}(u)$ error is shown as a time series in the figures. The $\mathscr{E}_{H^1}(u)$ seminorm is still tabulated, but does not provide additional insight into the surrogate performance as the trend is essentially the same as for the $\mathscr{E}_{L^2}(u)$ error, but with a much higher magnitude. The $\mathscr{E}_{H^1}(u)$ seminorm is therefore not visualised as a time series, but only tabulated in the Appendix A.3 for the interested reader.

Throughout this section, all $\mathscr{E}_{L^2}(u)$ and $\mathscr{E}_{H^1}(u)$ errors are evaluated relative to the FEM reference solution $u_h(\cdot, t)$ at each checkpoint time t (see Definition 4.5.1). This contrasts with Section 5.2, where both norms are taken relative to the one-step increment Δu_h.

Exclusion of anomalous test rollout. Rollout 287 of the diffusion test set corresponds to an extreme physical regime with $\mathrm{Fo}_{\max} = 30.15$ and a minimum element size $h_{\min} = 0.009$. This value of $\mathrm{Fo}_{\max}$ is more than twice the next largest value in the diffusion test set ($\mathrm{Fo}_{\max} = 13.73$ for rollout 293), which means that this rollout is governed by a far stronger diffusion than any other rollout in the set, on a much finer mesh, and therefore lies well outside the parameter range the surrogates were trained and tested on. Every surrogate variant, including the strongest models (Multi-dt60-4x8x4 and Multi-dt60-3x3x4x3x3), diverges catastrophically on this rollout. Here, and throughout this chapter, a rollout is said to diverge or blow up when its relative $\mathscr{E}_{L^2}(u)$ error crosses the threshold $\epsilon = 1$ defined in Section 4.5.2: the relative $\mathscr{E}_{L^2}(u)$ error exceeds 10 already at the first checkpoint $t = 0.24$ for the weaker models, and grows to over 10^2 for all models by $t = 1.92$. Because every model fails uniformly on this single rollout, it provides no discriminative information about relative model performance. Including it in the mean would inflate the reported diffusion-set errors by two to three orders of magnitude and obscure genuine differences between models. Rollout 287 from the diffusion test set is therefore excluded from all mean and standard-deviation calculations in this section.

5.3.1 Single-scale models at $\Delta t_{\text{small}} = 0.015$

Tables A.1–A.4 in Appendix A.3 report the rollout $\mathscr{E}_{L^2}(u)$ and $\mathscr{E}_{H^1}(u)$ errors at each checkpoint for the five single-scale variants at Δt_{small}. Because all five models share the same surrogate time step, any difference in rollout accuracy is attributable solely to the number of message-passing blocks. In general, it can be seen that for both test sets, adding more message passing layers increases the surrogate modelling performance when you consider the mean error per simulation, both in terms of $\mathscr{E}_{L^2}(u)$ and $\mathscr{E}_{H^1}(u)$. Moreover, it can be seen that for all the models, the advection test set is easier to model than the diffusion test set. Especially the $\mathscr{E}_{H^1}(u)$ semi-norm explodes on the diffusion test set, with no model as exception.

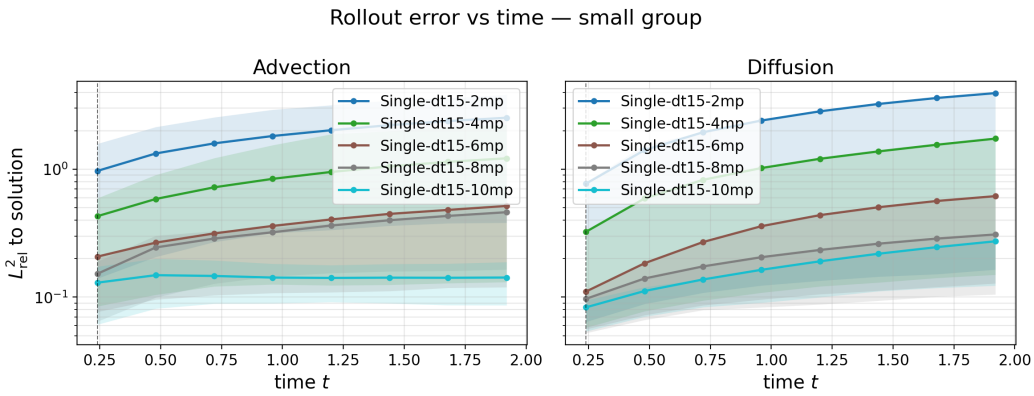


Figure 5.6: Rollout $\mathscr{E}_{L^2}(u)$ error over time for the single-scale models at $\Delta t_{\text{small}} = 0.015$. Left: advection test set. Right: diffusion test set. Lines show the mean over test rollouts and the shaded band the interquartile range across rollouts. $t = 0.24$.

Figure 5.6 visualises the table entries of $\mathscr{E}_{L^2}(u)$. On the advection test set (left panel), the average rollout error decreases as the number of message-passing blocks grows. Single-dt15-2mp sits highest and Single-dt15-10mp lowest, with the latter remaining essentially flat over the entire horizon. This flatness reflects that Single-dt15-10mp never blows up on any rollout of the advection test set: as it has enough message passing blocks to cover the whole training domain of CFL and Fo, it keeps the error bounded on every test rollout, meaning the slight slope can be attributed to accumulation error only. This contrasts with the shallower models, which share a common slope due to blow-ups. The curves for Single-dt15-6mp and Single-dt15-8mp track each other closely throughout the rollouts on the advection set and end within roughly 0.04 of one another at $t = 1.92$, in contrast with the larger separations seen between the other adjacent depths. On the diffusion test set (right panel), the expected ordering is recovered cleanly: the average rollout error decreases monotonically as the number of message-passing blocks grows. This time, the curves for Single-dt15-10mp and Single-dt15-8mp track each other closely throughout the rollouts and are separated by ± 0.03 at $t = 1.92$, in contrast with the larger separations seen between the other adjacent depths. The standard deviations of the errors are quite large for all the models, which will also be the case for the upcoming strides. But this is not surprising given the large variation in the physical regimes and the performance of the models across the test rollouts.

5.3.2 Single-scale and multiscale models at $\Delta t_{\text{medium}} = 0.030$

Tables A.5 toA.8 in Appendix A.3 report the rollout L^2 and H^1 errors at each checkpoint for the single-scale baseline Single-dt30-8mp and the three two-level multiscale variants Multi-dt30-3x2x3, Multi-dt30-2x4x2 and Multi-dt30-1x6x1, all sharing the same surrogate stride. Differences in rollout accuracy across rows therefore reflect the effect of the multiscale block allocation. The corresponding visualisation of these results is shown in Figure 5.7.

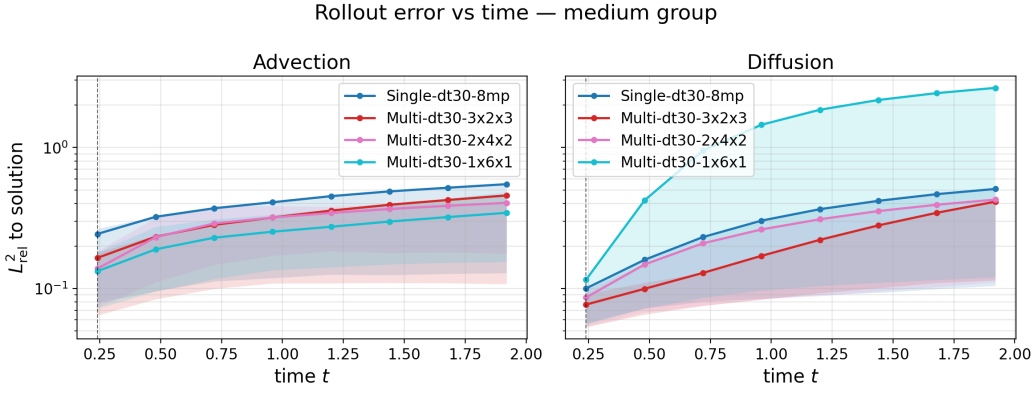


Figure 5.7: Rollout $\mathscr{E}_{L^2}(u)$ error over time for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$. Left: advection test set. Right: diffusion test set. Lines show the mean over test rollouts and the shaded band the interquartile range across rollouts.

The two panels of Figure 5.7 present essentially mirrored orderings, where two things stand out from these plots. Firstly, while the Multi-dt30-1x6x1 model was consistently performing worse than the other

two multiscale models in terms of one-step performance, it is now superior on the advection test set. On the other hand, on the diffusion test set, the multi-scale models that have relatively less coarse scale correction are the better performers. So this seems like a trade-off: the allocation of more coarse scale correction appears to enhance performance in the advection-dominated regime, while in the more diffusion-dominated regime, one should have more fine scale processing. But pure fine processing is not the best either in the diffusion-dominated regime, indicating that there is some sweet spot in the allocation of coarse and fine scale correction for the diffusion-dominated regime, which in this case is hit by the Multi-dt30-3x2x3 model. But in general, all the multiscale models are performing similar to each other on average, the only exception being the Multi-dt30-1x6x1 model on the diffusion test set, which is performing much worse than the other two multiscale models, and even worse than the single scale model.

5.3.3 Single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$

Tables A.9–A.12 in Appendix A.3 report the rollout L^2 and H^1 errors at each checkpoint for the single-scale baseline Single-dt60-16mp, the two-level multiscale Multi-dt60-4x8x4 and the three-level multiscale Multi-dt60-3x3x4x3x3, all at the largest surrogate stride. The corresponding time-resolved visualisation is shown in Figure 5.8.

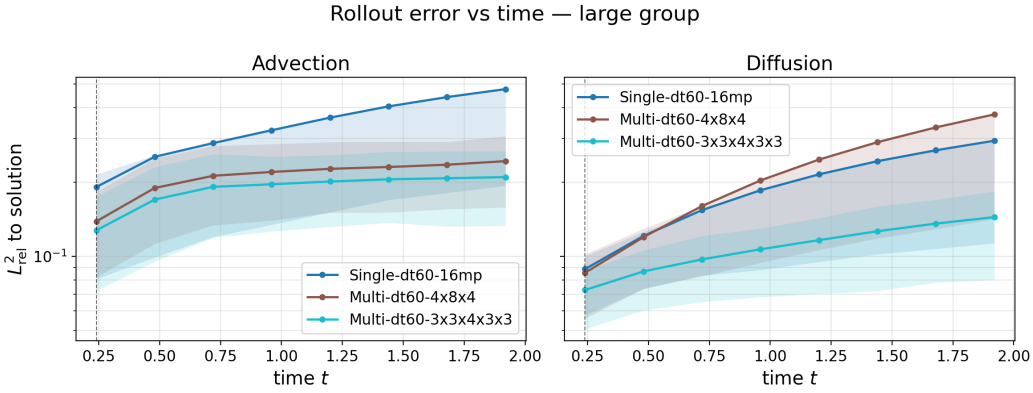


Figure 5.8: Rollout $\mathscr{E}_{L^2}(u)$ error over time for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$. Left: advection test set. Right: diffusion test set. Lines show the mean over test rollouts and the shaded band the interquartile range across rollouts.

On the advection test set of Figure 5.8, both multiscale variants attain a lower mean error than the single-scale Single-dt60-16mp at every checkpoint. The Multi-dt60-4x8x4 and Multi-dt60-3x3x4x3x3 curves visibly flatten beyond the end of the training horizon, whereas Single-dt60-16mp continues to accumulate error over the full simulation window. Where the latter can be attributed to blow-ups, similar to the observation we made for the small-stride models. On the diffusion test set the error accumulation is more pronounced for all three models, with the exception of Multi-dt60-3x3x4x3x3. In addition, Multi-dt60-4x8x4 now sits above Single-dt60-16mp for the latter part of the rollout, so the two-level model performs worse than the single-scale baseline when averaged over the diffusion test set. We already observed this phenomenon for the medium-stride models, and it is now even more pronounced for the large-stride models. It then seems contradictory that the three-level model Multi-dt60-3x3x4x3x3 is still outperforming the single-scale and 2-scale model on the diffusion test set, as it has even more coarse scale correction.But apparently adding the third level of correction seems to overcome this issue of the two-level model, which is an interesting observation.

5.3.4 Comparison across surrogate time steps

Tables A.13–A.16 in Appendix A.3 report the rollout L^2 and H^1 errors at each checkpoint for the three reference single-scale models Single-dt15-4mp ($\Delta t = 0.015$), Single-dt30-8mp ($\Delta t = 0.030$) and Single-dt60-16mp ($\Delta t = 0.060$). The corresponding time-resolved visualisation is shown in Figure 5.9.

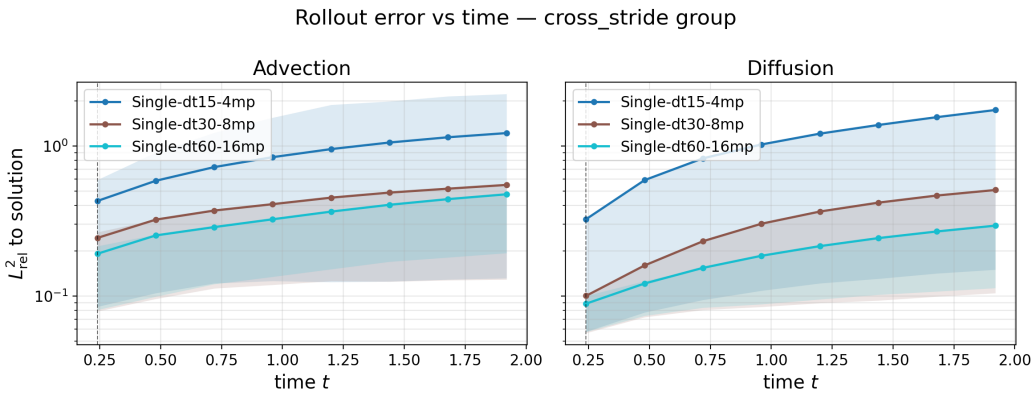


Figure 5.9: Rollout $\mathscr{E}_{L^2}(u)$ error over time for the single-scale models Single-dt15-4mp ($\Delta t = 0.015$), Single-dt30-8mp ($\Delta t = 0.030$) and Single-dt60-16mp ($\Delta t = 0.060$). Left: advection test set. Right: diffusion test set. Lines show the mean over test rollouts and the shaded band the interquartile range across rollouts.

In both panels of Figure 5.9 the mean rollout error decreases as the number of message-passing blocks grows: Single-dt60-16mp sits lowest, Single-dt30-8mp in the middle, and Single-dt15-4mp highest. Whereas the one-step R^2 in Figure 5.5 placed the three strides very close to one another after normalisation by the increment scale, the rollout setting produces a clearly visible separation between the three strides, while, by theory, they should be stable on the same fraction of the training domain. But the results suggest that the larger receptive field of the deeper models is beneficial for the autoregressive setting, even when the surrogate time step is larger.

5.3.5 Solution field snapshots

To complement the aggregate statistics, representative rollouts are chosen per model group to qualitatively illustrate how different models behave. For each group, an illustrative rollout is selected from either test set. Rows correspond to time checkpoints $t \in \{0.48, 0.96, 1.44, 1.92\}$. Each column shows, for one model in the group, the predicted field $\hat{u}$ alongside the pointwise log-error $\log_{10}|\hat{u} - u_h|$. The ground-truth field is exhibited in the first column of each figure for reference.

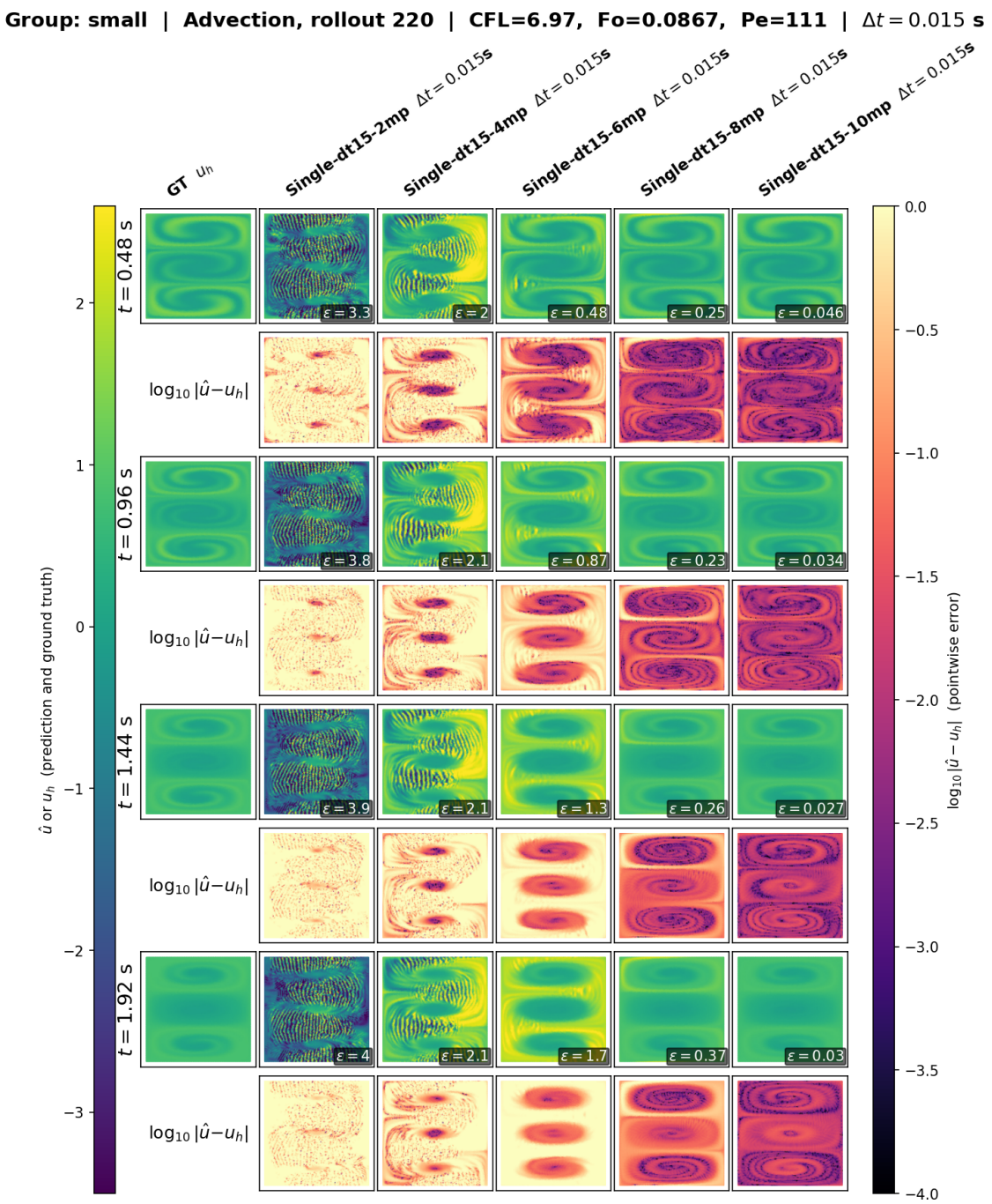


Figure 5.10: Solution field snapshots for the small-stride single-scale models (Single-dt15-2mp–Single-dt15-10mp) on an test rollout from the advection test set. Each row corresponds to a checkpoint time $t \in \{0.48, 0.96, 1.44, 1.92\}$.

Figure 5.10 corresponds to a rollout that is unstable for the shallowest three variants (Single-dt15-2mp, Single-dt15-4mp and Single-dt15-6mp) and stable for the two deepest ones (Single-dt15-8mp and Single-dt15-10mp). When the surrogate diverges, the instability emerges first in the region of highest velocity magnitude and then spreads through the domain until it becomes uncontrollable. The end-of-rollout snapshot of Single-dt15-2mp shows an essentially unbounded chaotic state. In contrast, Single-dt15-8mp and Single-dt15-10mp converge towards the expected steady state with the pointwise log-error remaining bounded throughout.

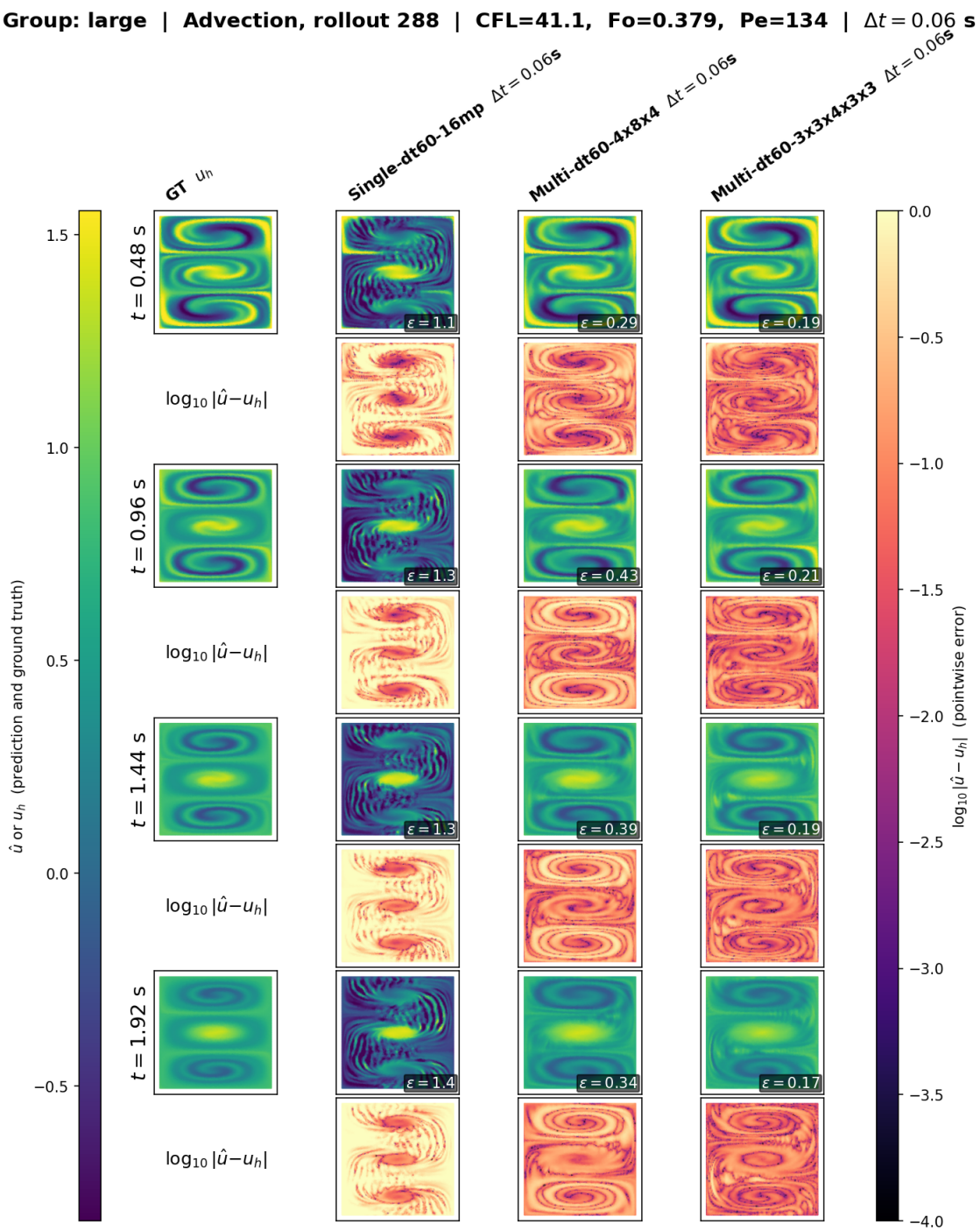


Figure 5.11: Solution field snapshots for the large-stride models (Single-dt60-16mp, Multi-dt60-4x8x4, Multi-dt60-3x3x4x3x3) on an advection test rollout.

Figure 5.11 shows the single-scale baseline Single-dt60-16mp diverging early in the rollout, with a rapidly growing log-error throughout the domain. Both multiscale variants converges towards the expected solution; at the final checkpoint $t = 1.92$, the three-level Multi-dt60-3x3x4x3x3 attains a relative $\mathscr{E}_{L^2}(u)$ that is approximately a factor of two smaller than that of the two-level Multi-dt60-4x8x4. Though, some minor instabilities can be seen in both multiscale models near the boundary layers of the solution.

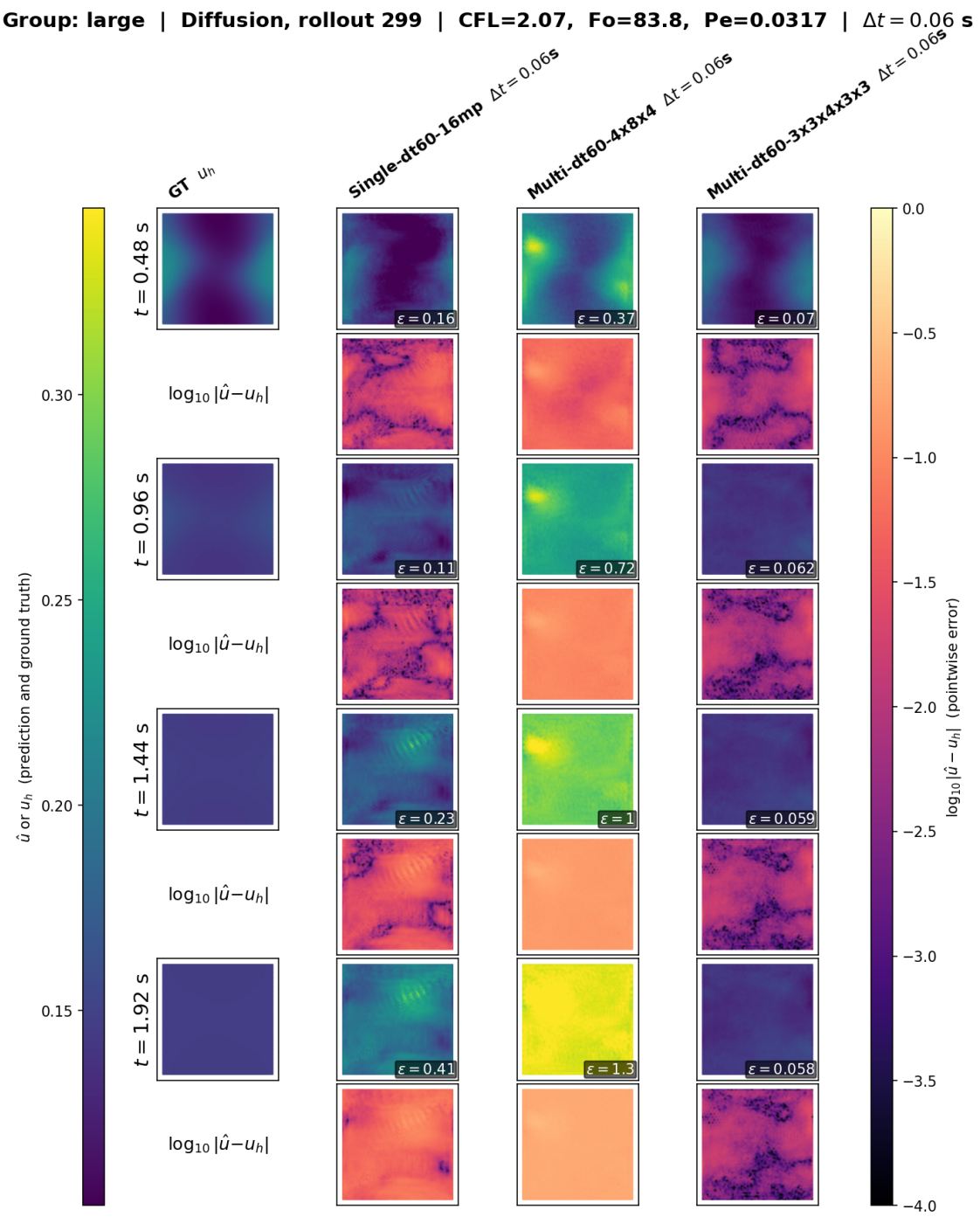


Figure 5.12: Solution field snapshots for the large-stride models (Single-dt60-16mp, Multi-dt60-4x8x4, Multi-dt60-3x3x4x3x3) on a diffusion test rollout.

Figure 5.12 presents a contrasting picture for the diffusion-dominated regime. The single-scale Single-dt60-16mp converges to a bounded state, but not to the correct steady state. The two-level Multi-dt60-4x8x4 shows visible oscillations and a modest blow-up of the prediction. The three-level Multi-dt60-3x3x4x3x3 converges cleanly to the correct steady state.

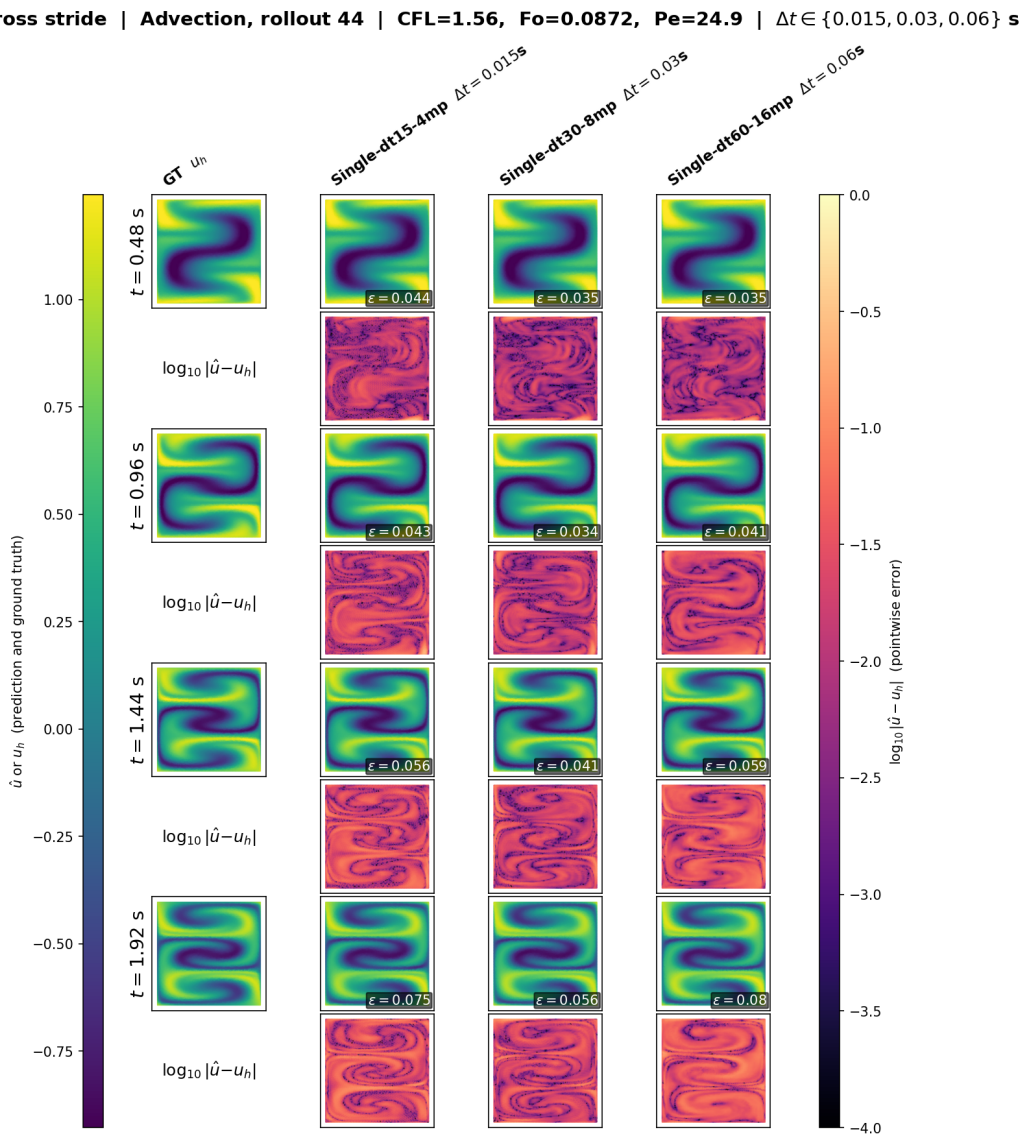


Figure 5.13: Solution field snapshots for the single-scale models across surrogate time steps (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp) on an advection test rollout.

In Figure 5.13 the three single-scale models produce qualitatively similar checkpoint sequences despite being trained at different strides. Although Figure 5.9 shows that Single-dt60-16mp attains the lowest mean error when averaged over the complete advection test set, on this particular rollout the medium-stride Single-dt30-8mp reaches the lowest relative $\mathscr{E}_{L^2}(u)$ at the final checkpoint.

More snapshots are shown in Appendix A.3.5 to illustrate the behaviour of the surrogates on additional test rollouts.

5.4 Limits of the models

This section characterises the empirical stability limits of each surrogate variant in terms of the dimensionless numbers CFL and Fo introduced in Section 2.4. Using the two structured test sets of Section 4.2.3, the rollout error at the final checkpoint $t = 1.92$ is mapped against $\mathrm{CFL}_{\max}$ and $\mathrm{Fo}_{\max}$ for every test trajectory. A piecewise-linear fit to this scatter (Section 4.5.2) yields a quantitative stability limit for each model, defined as the smallest dimensionless number at which the fitted error curve exceeds the blow-up threshold $\epsilon = 1$. The positions where the three line segments of this fit meet (the knots) are not chosen by hand but are determined by the residual-minimising grid search of Section 4.5.2. The vertical dashed lines in each scatter plots is the point where either $\mathrm{CFL}_{\max}$ or $\mathrm{Fo}_{\max}$ is equal to the number of message passing blocks in the model. This way we can see if the number of message passing layers is a

good predictor for the stability limit of the single-scale models. For the multiscale model, the same vertical dashed lines are plotted, but we expect that the stability limit will not follow the number of message passing blocks as closely this time, as we expect that the coarse layers will increase the receptive field of the model, which should increase the stability limit.

5.4.1 Single-scale models at $\Delta t_{\text{small}} = 0.015$

Figure 5.14 shows the scatter plots for the five single-scale variants at Δt_{small}. Comparing the plots across the rows of the figure reveals how increasing the number of message-passing blocks shifts the empirical stability limit. These five single-scale models are all trained and evaluated at the smallest surrogate stride Δt_{small} (Section 4.4).

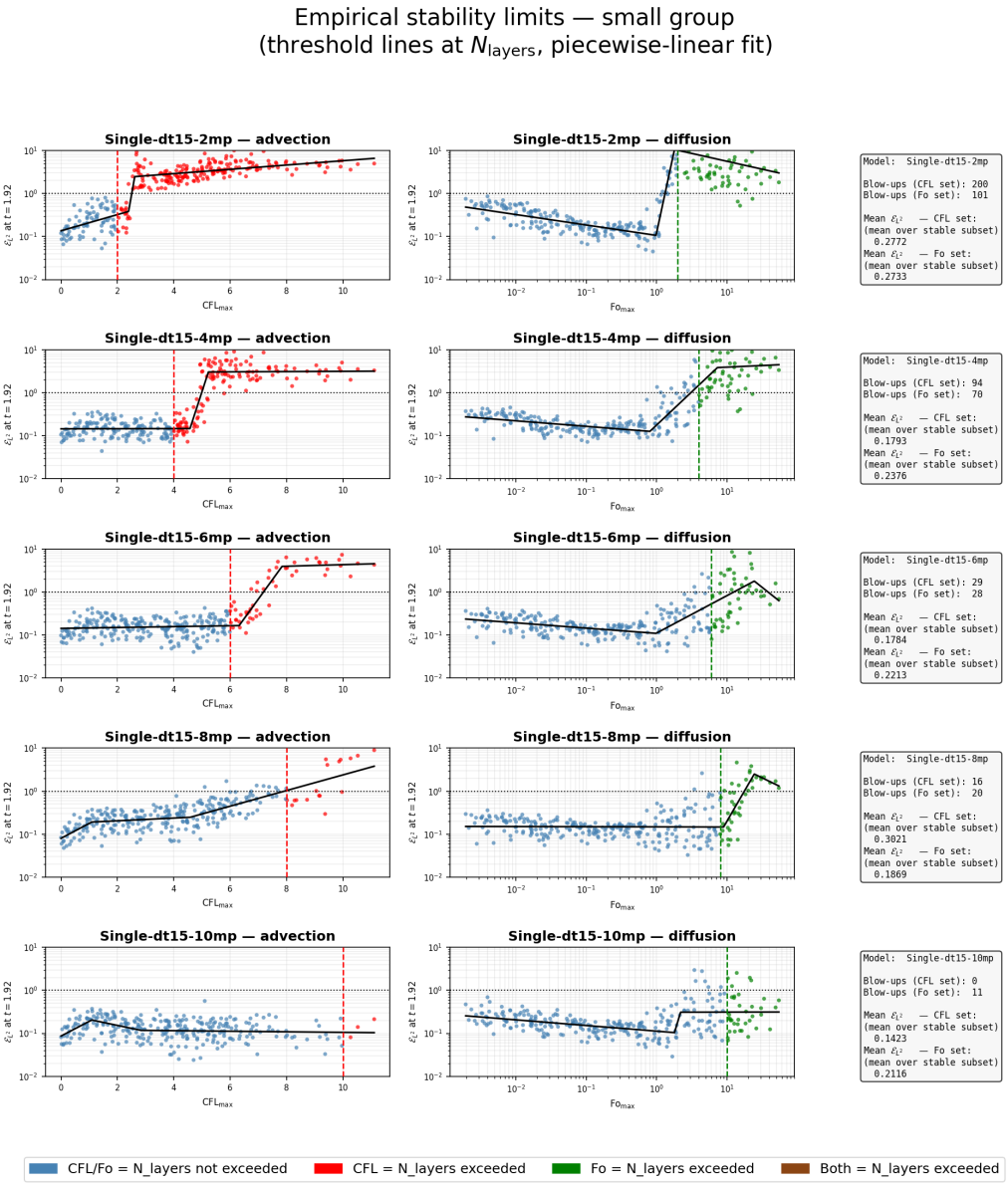


Figure 5.14: Empirical stability analysis for the single-scale models at $\Delta t_{\text{small}} = 0.015$. Each row corresponds to one model. Columns show the rollout $\mathcal{E}_{L^2}(u)$ at $t = 1.92$ as a function of CFL_{max} (left) and Fo_{max} (centre), and a summary of the architecture and error statistics (right).

Reading Figure 5.14 from top to bottom, the empirical CFL* in Table 5.5 grows monotonically with the number of message-passing blocks, from CFL* = 2.51 for Single-dt15-2mp to an effectively unbounded value for Single-dt15-10mp. The diffusion side shows a similar but tighter pattern: Fo* increases from 1.32 for Single-dt15-2mp through 12.5 and 19.3 for Single-dt15-6mp and Single-dt15-8mp, to an effectively unbounded value for Single-dt15-10mp. However, the fit does not look as convincing

for the diffusion test set as it does for the advection test set, as there is a lot of variance in the error of trajectories for the diffusion samples with Fo $> 10^0$. The high variance between trajectories for Fo $> 10^0$ is not an exception for the smallest stride, as it is also observed for all the models in Δt_{medium} and Δt_{large}.

5.4.2 Single-scale and multiscale models at $\Delta t_{\text{medium}} = 0.030$

Figure 5.15 compares the single-scale baseline Single-dt30-8mp against the three two-level multiscale variants Multi-dt30-3x2x3, Multi-dt30-2x4x2 and Multi-dt30-1x6x1 that share the same surrogate stride.

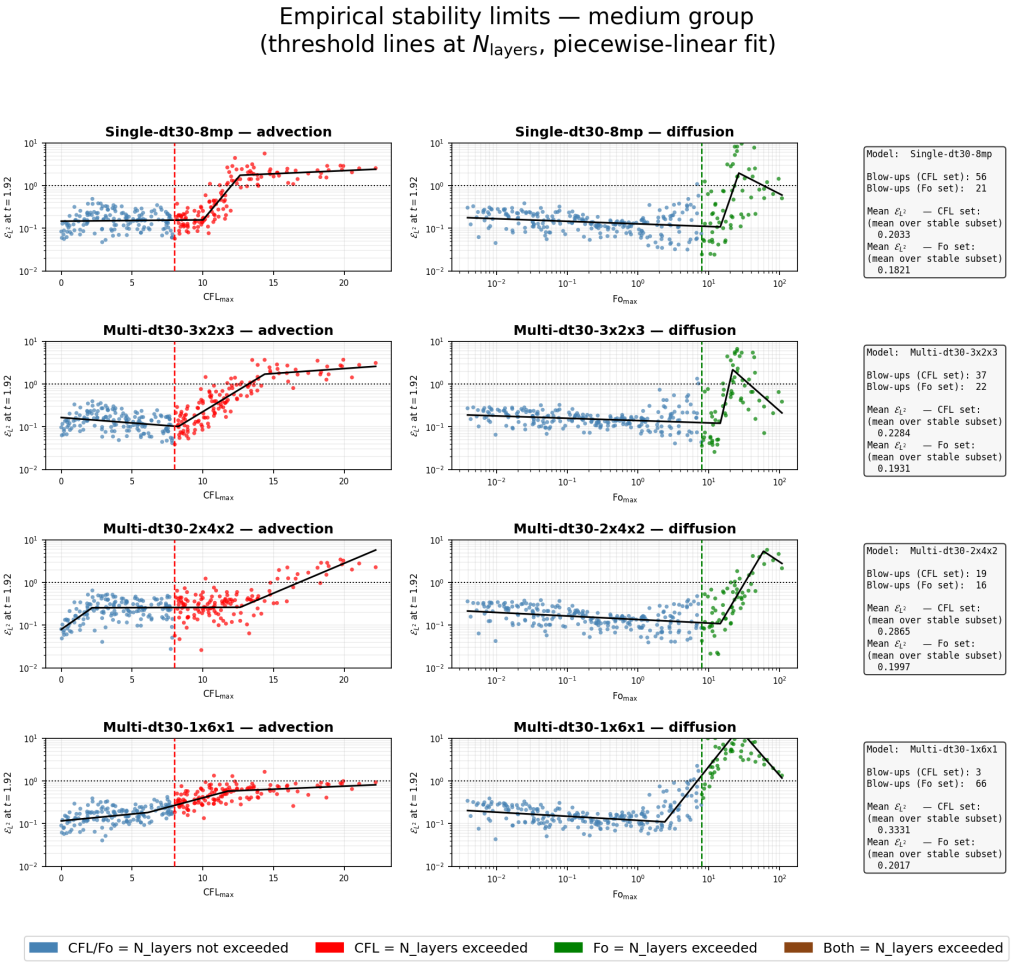


Figure 5.15: Empirical stability analysis for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$ (Single-dt30-8mp, Multi-dt30-3x2x3, Multi-dt30-2x4x2, Multi-dt30-1x6x1). Each row corresponds to one model. Columns show the rollout $\mathscr{E}_{L^2}(u)$ at $t = 1.92$ as a function of CFL_{max} (left) and Fo_{max} (centre), and a summary of the architecture and error statistics (right).

The single-scale baseline Single-dt30-8mp diverges at $\text{CFL}^* = 12$. Among the multiscale variants the fitted CFL threshold shifts further to the right as the coarse correction grows: $\text{CFL}^* = 13.2$ for Multi-dt30-3x2x3, $\text{CFL}^* = 16.8$ for Multi-dt30-2x4x2, and an effectively unbounded fitted threshold for Multi-dt30-1x6x1. It must be noted, however, that even though Multi-dt30-1x6x1 is stable for almost all CFL values, its trajectories sit fairly close to the blow-up threshold and therefore retain a relatively high error on the stable subset. On the diffusion side the ordering between the multiscale variants is different: Multi-dt30-2x4x2 attains the largest $\text{Fo}^* = 32.4$, followed by the single-scale Single-dt30-8mp with $\text{Fo}^* = 23.2$, then Multi-dt30-3x2x3 with 19.6 and finally Multi-dt30-1x6x1 with 6.9. So, adding more coarse scale correction blocks does not improve stability for trajectories with high diffusion.

5.4.3 Single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$

Figure 5.16 repeats the same analysis for the large-stride single-scale baseline Single-dt60-16mp, the two-level multiscale Multi-dt60-4x8x4 and the three-level multiscale Multi-dt60-3x3x4x3x3.

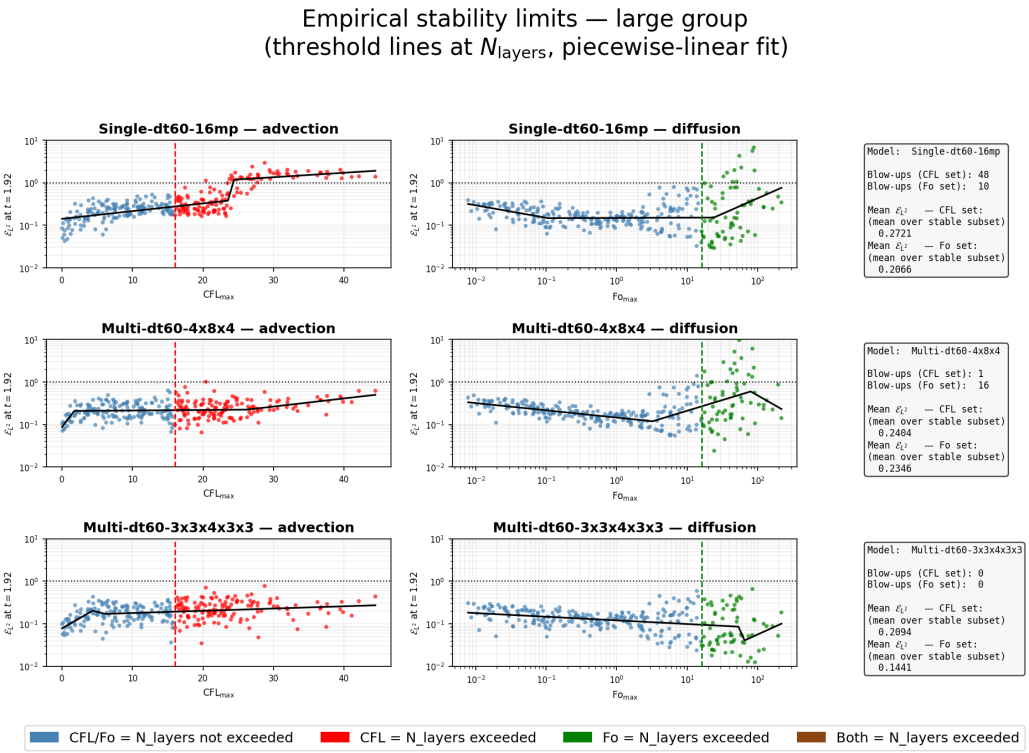


Figure 5.16: Empirical stability analysis for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$ (Single-dt60-16mp, Multi-dt60-4x8x4, Multi-dt60-3x3x4x3x3). Each row corresponds to one model; columns show the rollout $\mathcal{E}_{L^2}(u)$ at $t = 1.92$ as a function of $\mathrm{CFL}_{\max}$ (left) and $\mathrm{Fo}_{\max}$ (centre), and a summary of the architecture and error statistics (right). Colour coding and line conventions as in Figure 5.14.

The single-scale Single-dt60-16mp diverges at $\mathrm{CFL}^* = 24.3$, while the multiscale variants remain below the blow-up threshold across the entire tested CFL range: the fitted CFL^* is effectively unbounded for Multi-dt60-4x8x4 and for Multi-dt60-3x3x4x3x3. On the diffusion side the fitted thresholds are all effectively unbounded. This disputes the observations from the medium stride, where the multiscale models had worse stability on the diffusion set. And also on this stride, the 2-scale model does not show improved stability relative to the single scale on the diffusion set, while the 3-scale model does show improved stability. The only differentiating factor between the multiscale models is the number of coarse levels used, which in this case seems to enhance the stability region for both advection and diffusion test trajectories. And in general it can be seen that the multiscale models have a much more differentiating factor relative to the single scale model, something we did not see as convincingly in the medium stride. So it seems that the multiscale architecture is more beneficial for stability when the surrogate stride is larger.

5.4.4 Comparison across surrogate time steps

Figure 5.17 compares the three reference single-scale models Single-dt15-4mp, Single-dt30-8mp and Single-dt60-16mp. As $\mathrm{CFL}_{\max}$ and $\mathrm{Fo}_{\max}$ scale linearly with the surrogate stride, the horizontal axes are not directly comparable between rows, which is why we normalise the horizontal axes by their respective timestep size Δt. This places all three stride scales on the same horizontal axis and allows for a direct comparison of the stability limits across different surrogate time step sizes. On this normalised axis, we can see that the larger the model, the higher the normalised stability limit. So even though these models should cover the same trajectories, the larger models are able to maintain stability for larger normalised CFL and Fo numbers. This wider stability region does not come for free, however. On the advection test set, the mean $\mathcal{E}_{L^2}(u)$ at $t = 1.92$ over the stable subset, reported in the summary column of Figure 5.17, grows with the number of message-passing blocks, from 0.18 for Single-dt15-4mp to 0.20 for Single-dt30-8mp and 0.27 for Single-dt60-16mp. So on the trajectories where they stay stable, the smaller models are the more accurate ones, and extending the stability region with more blocks trades off against accuracy inside it. On the diffusion test set this trade-off does not appear: there the stable-subset error is non-monotone (0.24, 0.18 and 0.21 for the three models), with the medium Single-dt30-8mp the most accurate and the smallest model the least. This difference between the two test sets is interesting, be-

cause it might suggest that the stability with regard to the diffusion is more related to the receptive field of the model, i.e. the number of neighbours it interacts with, rather than the dimensionless numbers.

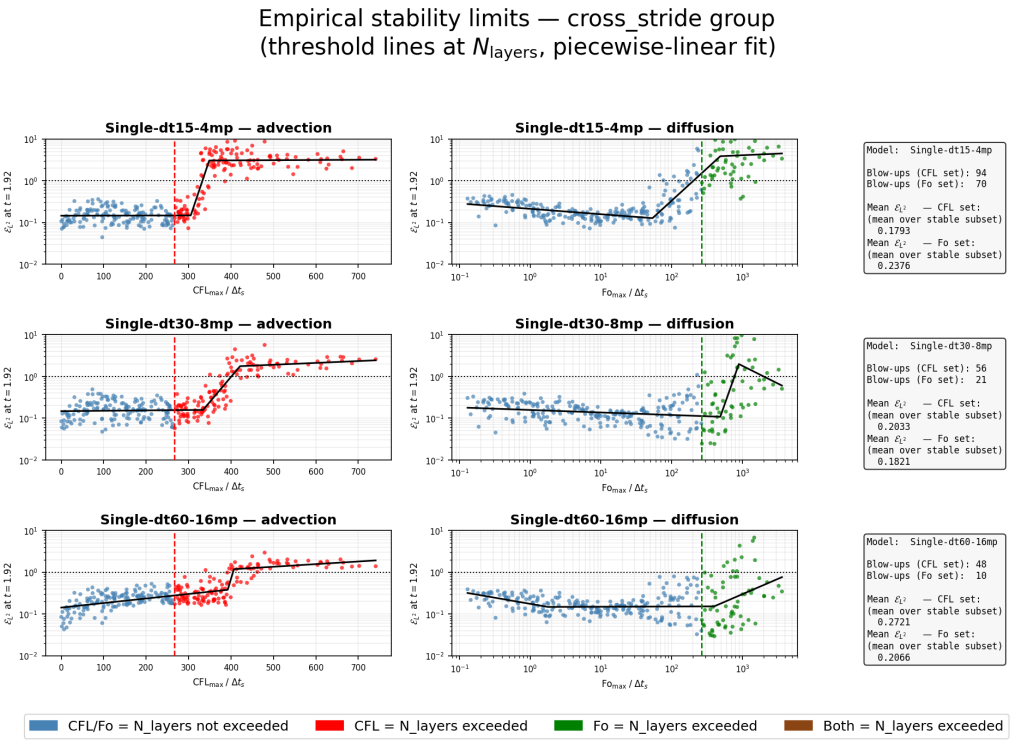


Figure 5.17: Stability comparison across surrogate time steps for the single-scale models Single-dt15-4mp ($\Delta t = 0.015$), Single-dt30-8mp ($\Delta t = 0.030$) and Single-dt60-16mp ($\Delta t = 0.060$). Each row corresponds to one model; columns show the rollout $\mathcal{E}_{L^2}(u)$ at $t = 1.92$ as a function of CFL_{max} (left) and Fo_{max} (centre), and a summary of the architecture and error statistics (right). Note that CFL_{max} scales linearly with Δt, so the horizontal axes are not directly comparable across rows. Colour coding and line conventions as in Figure 5.14.

5.4.5 Summary of empirical stability limits

Table 5.5 collects the empirical CFL and Fo stability limits extracted from the piecewise-linear fits of the preceding subsections. The limits are defined as the smallest dimensionless number at which the fitted mean error curve exceeds the blow-up threshold $\epsilon = 1$ (see Section 4.5.2).

Table 5.5: Empirical stability limits for all surrogate variants. CFL^* and Fo^* are the smallest dimensionless numbers at which the fitted rollout error exceeds the blow-up threshold $\epsilon = 1$ on the respective test set.

Model	Δt	CFL^*	Fo^*
Single-scale, $\Delta t_{\mathrm{small}} = 0.015$			
`Single-dt15-2mp`	0.015	2.51	1.32
`Single-dt15-4mp`	0.015	4.99	3.06
`Single-dt15-6mp`	0.015	7.19	12.5
`Single-dt15-8mp`	0.015	7.93	17.6
`Single-dt15-10mp`	0.015	∞	∞
Single-scale and multiscale, $\Delta t_{\mathrm{medium}} = 0.030$			
`Single-dt30-8mp`	0.030	12	23.2
`Multi-dt30-3x2x3`	0.030	13.2	19.6
`Multi-dt30-2x4x2`	0.030	16.8	32.4
`Multi-dt30-1x6x1`	0.030	∞	6.9
Single-scale and multiscale, $\Delta t_{\mathrm{large}} = 0.060$			
`Single-dt60-16mp`	0.060	24.3	∞
`Multi-dt60-4x8x4`	0.060	∞	∞
`Multi-dt60-3x3x4x3x3`	0.060	∞	∞

The piecewise-linear fits behind these limits are revisited group by group in Chapter 7, where each panel is placed alongside the research sub-question it answers.

Chapter 6

Discussion

This chapter interprets the stability and accuracy results of the preceding chapter and sets out the main caveats and open questions behind them. We begin with the reliability of the reference data that the surrogates were trained and scored against, turn to the conflicting architectural mechanism behind the coarse-versus-fine trade-off observed in the multiscale models, and close with the resulting uncertainty in the empirical Fourier stability limit. Further limitations of the study, together with the recommendations that follow from them, are gathered later in Chapter 8.

6.1 The reference integrator and the diffusion-side variance

The reference trajectories are produced by backward Euler with $\Delta t_{\text{sim}} = 10^{-4}$ (Section 4.2). The scheme is unconditionally stable but dissipative. In the diffusion-dominated regime its numerical dissipation is masked by the physical operator, so the integrator produces plausible trajectories. In the advection-dominated regime, by contrast, the same dissipation smooths gradients that the underlying PDE would preserve, so the resulting reference is not in line with the true dynamics. A surrogate trained on this data therefore sees advection-dominated samples whose effective diffusion has been artificially inflated, and learns a relation in which the role of the diffusion term is partly distorted.

The spatial convergence study of Appendix A.5 points in the same direction. The reference error against a finely resolved solution is small at high diffusion, where the field is smooth, but grows for low-diffusion samples, whose sharp boundary layers are not resolved at the working mesh resolution. Under-resolving these layers smears them, which is once more equivalent to the simulator overestimating the effective diffusion of exactly those samples. The reference is therefore least trustworthy in precisely the regime where the surrogate is asked to preserve the sharpest features.

A plausible footprint of this poorly constrained diffusion response is the large trajectory-to-trajectory variance observed for $\text{Fo}_{\text{max}} > 10^1$ in the right columns of Figures 5.14–5.17, which is markedly wider than the corresponding scatter on the CFL axis at every stride. Because the surrogate is scored against a target whose diffusion content is itself imperfect and regime-dependent, the absence of a clear consensus between the surrogates on the strongly diffusive samples is not surprising. A temporal convergence or stability study of the integrator itself was not carried out, so the magnitude of the backward-Euler dissipation relative to a fully resolved reference remains unquantified, and we treat it as a limitation of this work.

6.2 Coarse correction versus fine-scale message passing

The medium-stride ordering of Table 5.5, in which CFL^* rises monotonically with the coarse allocation while Fo^* does not, follows directly from what each path contributes. The fine-level blocks perform message passing at the original mesh resolution and resolve the fine-scale structure of the solution locally, whereas the coarse-level blocks operate on the coarsened graph and propagate information across a larger physical extent per cycle, resolving the coarse-scale structure. The latter is what the regime demands as Δt grows, the velocity increases, or the resolution decreases. Shifting blocks from the fine path to the coarse path therefore widens the effective receptive field, and with it the advective stability range, at the direct cost of fine-scale message-passing capacity. This is exactly the trade-off the medium-stride results expose: added coarse processing extends the advective limit but does nothing systematic for the

diffusive one, and the most coarse-heavy allocation, Multi-dt30-1x6x1, also carries the highest mean error inside its own stable region (Figure 5.7), which points to a lack of fine-scale resolution, apparently needed for this regime at this stride.

At the largest stride, this trend is contradicted. Both the two-level Multi-dt60-4x8x4 and the three-level Multi-dt60-3x3x4x3x3 reach effectively unbounded limits, and the hierarchy with even more coarse levels does not inherit the loss of fine-scale accuracy that more coarse processing would be expected to bring. The latter was observed at the medium stride, but instead the three-level V-cycle attains the lowest mean error of its group on both test sets (Figure 5.8). So these two different conclusions from the medium and large stride yield conflicting evidence on the coarse-versus-fine trade-off, and the mechanism behind it is not completely clear.

6.3 Validity of the Fourier stability limit

On every stride the scatter on the Fourier axis is much wider than on the CFL axis. The right columns of Figures 5.14, 5.15, 5.16 and 5.17 show that, for $\mathrm{Fo}_{\max}$ above roughly 10^1, the rollout error spreads over more than an order of magnitude at any given Fo value, with bounded and near-blow-up trajectories sitting next to one another.

Because the spread is this large, the piecewise-linear fit that defines Fo^* rests on a noisy relation between $\mathrm{Fo}_{\max}$ and the rollout error, and the diffusion-side limits in Table 5.5 should be read as indicative rather than sharp. Part of this variance need not originate with the surrogate at all. As argued in Section 6.1, the backward-Euler reference is a regime-dependently imperfect target whose diffusion content is least trustworthy on the sharpest samples, so as the reference itself is inconsistent the surrogate cannot be expected to track the relation cleanly. The diffusion-axis scatter therefore combines genuine surrogate error with noise inherited from the reference, which is a further reason to treat Fo^* with caution.

Chapter 7

Conclusion

This thesis investigated whether graph neural network (GNN) surrogate time integrators can reproduce finite element (FEM) solutions of the two-dimensional advection–diffusion equation on unstructured meshes with periodic boundary conditions, and what empirical stability limits govern them in terms of the CFL and Fourier numbers. Two model families were compared at a fixed message-passing budget: single-scale models, and multiscale models organised as V-cycles over coarsened graphs with close parameter counts (Section 4.4). The two main research questions of Section 1.3 and their sub-questions are restated and answered in turn below.

7.1 Main question 1: learning and generalisation

> Can GNN-based surrogate time integrators learn to accurately reproduce the FEM solutions of the advection–diffusion equation when applied to seen and unseen velocity and diffusion fields?

SQ1 (qualitative one-step reproduction). Yes. Figure 5.1 shows an example prediction where the predicted increment matches the FEM target everywhere. Moreover, one-step validation errors $\mathscr{E}_{L^2}(\Delta u)$ are as low as 0.0236 for the Single-dt15-10mp model, which taken over the entire validation set is a strong indication that the surrogates have learned to reproduce the one-step dynamics of the reference solution.

SQ2 (lowest one-step validation error). One-step accuracy improves monotonically with the number of message-passing blocks N_{mp} at the small stride, and at the medium and large strides the multiscale models outperform the single-scale baseline at equal budget. The best variant per stride, in the relative increment norms of Definition 4.5.1, is collected in Table 7.1.

Table 7.1: Answer to SQ2. Lowest one-step validation error per surrogate stride (best variant), in the relative increment norms of Definition 4.5.1. Extracted from Tables 5.1–5.3.

Stride	Best variant	$\mathscr{E}_{L^2}(\Delta u)$	$\mathscr{E}_{H^1}(\Delta u)$
$\Delta t_{\text{small}} = 0.015$	`Single-dt15-10mp`	0.0236	0.0846
$\Delta t_{\text{medium}} = 0.030$	`Multi-dt30-2x4x2`	0.0200	0.0758
$\Delta t_{\text{large}} = 0.060$	`Multi-dt60-3x3x4x3x3`	0.0202	0.0624

SQ3 (behaviour beyond the training horizon). Yes. Rolled out to $t = 1.92 = 8\,T$, eight times the training horizon $T = 0.24$, the stronger models in each group keep the advection–diffusion dynamics. The rollout error does grow beyond $t = T$, but in most cases steadily and in several cases flattens, with no abrupt regime change at the training horizon (Figures 5.6–5.9). The advection-set $\mathscr{E}_{L^2}(u)$ of Single-dt15-10mp drifts only from 0.130 at $t = T$ to 0.142 at $t = 1.92$, and that of Multi-dt60-3x3x4x3x3 from 0.127 to 0.209. The field snapshots (Figures 5.10–5.13) confirm that the gross spatial structure of the reference is reproduced at every checkpoint (given that the model is stable), including those well beyond the training horizon, with the differences limited to fine-scale features in the right places rather than a complete loss of structure.

SQ4 (lowest rollout error at $t = 1.92$). The best variant per stride and test set is collected in Table 7.2. On the small stride the deepest single-scale model is best on both test sets, on the medium stride the ranking depends on the regime, and on the large stride the three-level multiscale model is best on both.

Table 7.2: Answer to SQ4. Lowest rollout $\mathscr{E}_{L^2}(u)$ error at the final checkpoint $t = 1.92$ per surrogate stride, on each structured test set (best variant). Gathered from Tables A.1–A.10.

	Advection test set		Diffusion test set	
Stride	Best variant	$\mathscr{E}_{L^2}(u)$	Best variant	$\mathscr{E}_{L^2}(u)$
Δt_{small}	`Single-dt15-10mp`	0.142	`Single-dt15-10mp`	0.273
Δt_{medium}	`Multi-dt30-1x6x1`	0.344	`Multi-dt30-3x2x3`	0.413
Δt_{large}	`Multi-dt60-3x3x4x3x3`	0.209	`Multi-dt60-3x3x4x3x3`	0.144

Answer to Main Question 1. Within their stability limits, the surrogates learn the advection–diffusion dynamics and reproduce the FEM solution accurately on both seen and unseen velocity and diffusion fields, and over horizons eight times longer than the training window. The accuracy is not the same in every physical regime: the advection-dominated test set is consistently easier than the diffusion-dominated one.

7.2 Main question 2: empirical stability limits

What are the empirical stability limits of GNN surrogate time integrators in terms of the CFL and Fourier numbers?

The piecewise-linear fits of Section 4.5.2 define, for every model, the smallest CFL or Fourier number at which the fitted rollout error at $t = 1.92$ crosses the blow-up threshold $\epsilon = 1$. We call these limits CFL* and Fo* and report the full set in Table 5.5. The four sub-questions are answered below, each next to the fitted-error panel and the part of that table that supports it.

SQ5 (single-scale limits versus depth, small stride). Both limits rise monotonically with N_{mp} (Figure 7.1, Table 7.3): CFL* grows from 2.51 at two blocks to effectively unbounded at ten, and Fo* from 1.32 to unbounded over the same range. The empirical CFL* therefore tracks N_{mp} and sits slightly above it, which fits the idea that each block extends the receptive field by one hop.

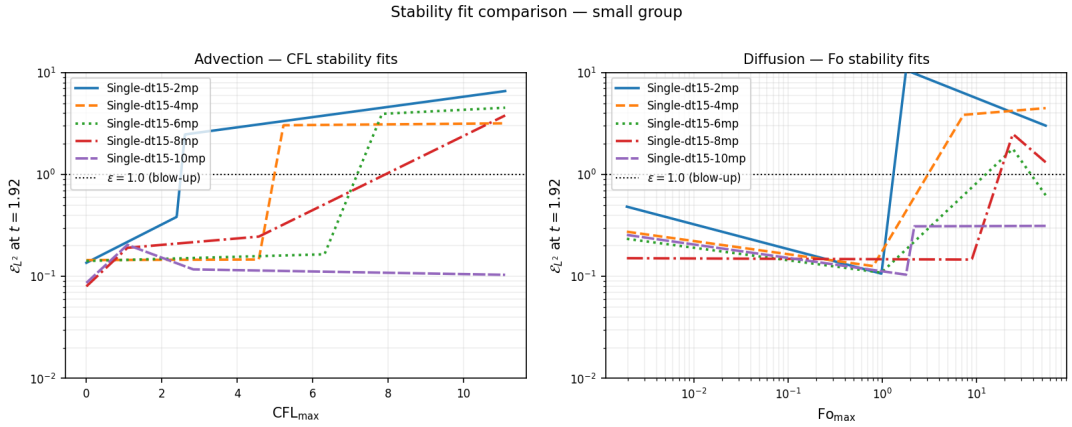


Figure 7.1: Answer to SQ5. Fit of error $\mathscr{E}_{L^2}(u)$ at $t = 1.92$ against CFL_{max} (left) and Fo_{max} (right) for the small-stride single-scale models, with the piecewise-linear fits. The inflection of the error curve shifts to larger dimensionless numbers as N_{mp} grows; the dotted line marks the blow-up threshold $\epsilon = 1$.

Table 7.3: Answer to SQ5. Empirical stability limits at $\Delta t_{\text{small}} = 0.015$ (extracted from Table 5.5).

Model	CFL*	Fo*
`Single-dt15-2mp`	2.51	1.32
`Single-dt15-4mp`	4.99	3.06
`Single-dt15-6mp`	7.19	12.5
`Single-dt15-8mp`	7.93	17.6
`Single-dt15-10mp`	∞	∞

SQ6 (fine/coarse allocation, medium stride). At a fixed eight-block budget, shifting blocks onto the coarse level widens the advective range monotonically but does not systematically help the diffusive

one (Figure 7.2, Table 7.4). CFL* rises from 12 for the single-scale baseline through 13.2 The diffusive limit reverses this ordering: Fo* peaks at the balanced [2,4,2] allocation (32.4) and collapses for the most coarse-heavy [1,6,1] (6.9). Adding coarse blocks therefore trades diffusive stability, and accuracy inside the stable region, for a wider advective range.

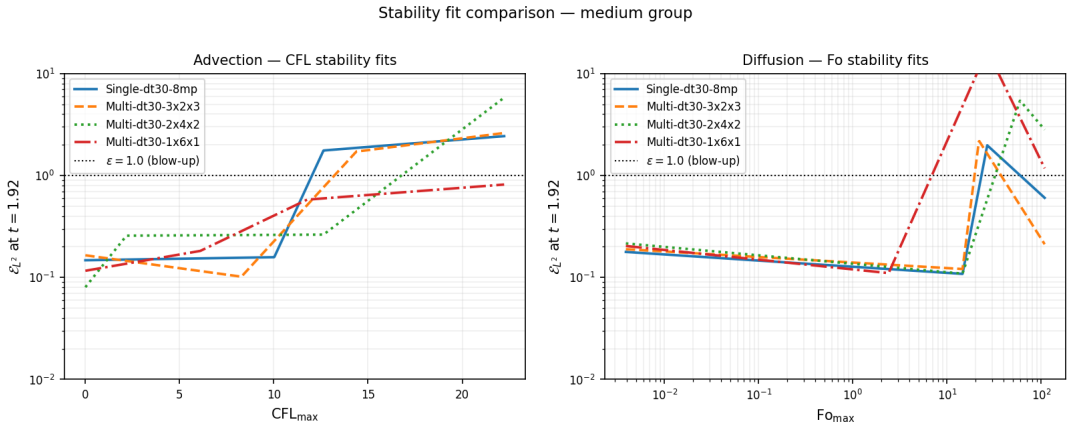


Figure 7.2: Answer to SQ6. Fitted rollout error for the single-scale baseline and the three two-level multiscale allocations at $\Delta t_{\text{medium}} = 0.030$. Adding coarse blocks pushes the CFL inflection to the right but leaves the Fo behaviour non-monotone across the four variants.

Table 7.4: Answer to SQ6. Empirical stability limits at $\Delta t_{\text{medium}} = 0.030$ (extracted from Table 5.5).

Model	CFL*	Fo*
`Single-dt30-8mp`	12	23.2
`Multi-dt30-3x2x3`	13.2	19.6
`Multi-dt30-2x4x2`	16.8	32.4
`Multi-dt30-1x6x1`	∞	6.9

SQ7 (three levels versus two, large stride). Yes. Both multiscale variants reach effectively unbounded CFL* and Fo* (Table 7.5). The third level therefore extends the stable region, beyond both the two-level and the single-scale model.

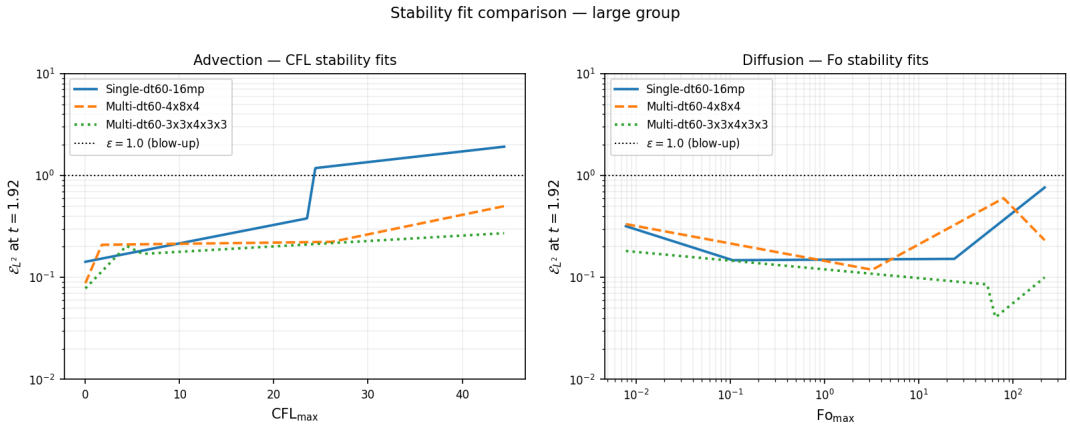


Figure 7.3: Answer to SQ7. Fitted rollout error for the single-scale baseline, the two-level [4,8,4] and the three-level [3,3,4,3,3] multiscale models at $\Delta t_{\text{large}} = 0.060$. Both multiscale variants stay below the threshold across the CFL range, and only the three-level variant stays below it across the whole Fo range.

Table 7.5: Answer to SQ7. Empirical stability limits at $\Delta t_{\text{large}} = 0.060$ (extracted from Table 5.5).

Model	CFL*	Fo*
`Single-dt60-16mp`	24.3	∞
`Multi-dt60-4x8x4`	∞	∞
`Multi-dt60-3x3x4x3x3`	∞	∞

SQ8 (oversmoothing with depth, cross-stride). It is a trade-off. The three single-scale models cover the same fraction of the CFL and Fourier domain (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp; Figure 7.4). The deeper, larger-stride model has the widest stable region and the lowest mean rollout error over the full test set, but this is mostly because it blows up the least. On the trajectories where all three stay stable, the advection-set error instead grows with the number of message-passing blocks, from a mean $\mathscr{E}_{L^2}(u)$ at $t = 1.92$ of about 0.18 at four blocks to 0.20 at eight and 0.27 at sixteen, so the wider stable region is paid for with a higher error where the shallower models are also stable. This is consistent with mild oversmoothing in the deeper model. The effect is limited to the advection-dominated regime: on the diffusion set the stable-subset error is non-monotone, lowest for the medium Single-dt30-8mp. Depth therefore does not cause a breakdown, the one-step $\mathscr{E}_{H^1}(\Delta u)$ still improves with it (Section 5.2.4) and the snapshots stay faithful (Figure 5.13), but widening the stable region with more blocks comes at a mild cost in accuracy on the commonly stable advection trajectories.

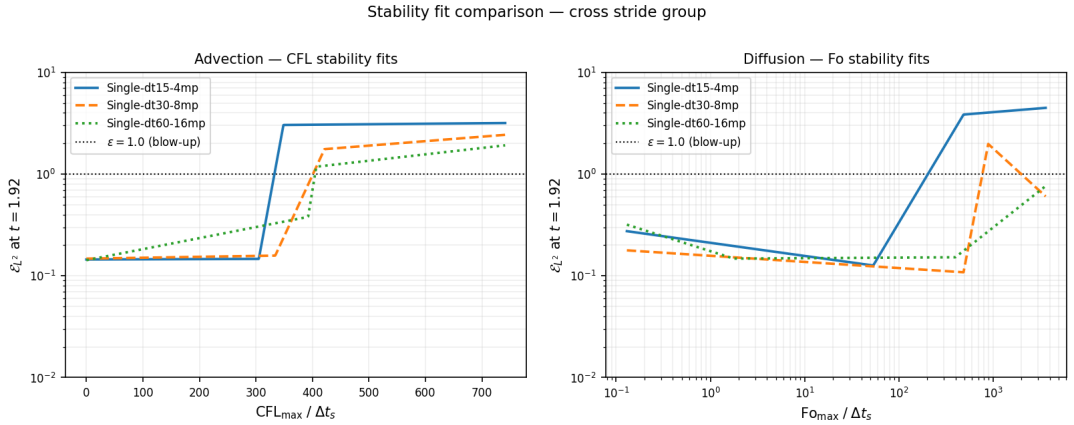


Figure 7.4: Answer to SQ8. Fitted rollout error for the three reference single-scale models, with both axes normalised by Δt so that the models cover the same span. The deeper, larger-stride models reach larger normalised stability limits, but on the advection trajectories where the shallower models are also stable they carry a slightly higher error.

Answer to Main Question 2. The single-scale surrogate has well-defined empirical stability limits that track the number of message-passing blocks and lie slightly above it. Multiscale V-cycles broaden the CFL range at a fixed budget, partly at the medium stride and clearly at the large stride, where CFL* rises from 24.3 for Single-dt60-16mp to unbounded for both multiscale variants. This benefit is not uniform. At the medium stride,the wider advective range comes at the cost of a smaller diffusive range and a higher mean error inside the stable region, and only the three-level configuration removes this penalty, reaching zero blow-ups on both axes. Multiscale therefore broadens the range of applicability of the surrogate, and the allocation of blocks across the hierarchy controls the trade-off between stability and accuracy.

Chapter 8

Limitations and recommendations

This study isolates the behaviour of graph neural network surrogates on a deliberately controlled problem, and several of its choices bound the conclusions that can be drawn. Each limitation is stated below together with the recommendation that follows most directly from it.

All reference trajectories were produced by implicit backward Euler with $\Delta t_{\text{sim}} = 10^{-4}$ (Section 4.2), and this reference is least trustworthy in the strongly diffusive regime for two related reasons. First, backward Euler is unconditionally stable but dissipative, and this numerical dissipation acts differently on the advection- and diffusion-dominated regimes. As argued in Section 6.1, it is a plausible contributor to the high trajectory-to-trajectory variance on the diffusion test set. Second, the spatial convergence study of Appendix A.5 shows that the reference is least accurate where the boundary layers are sharpest, which is the high-diffusion regime, because the mesh used for data generation does not resolve those layers well. Each cause points to its own fix. The dissipation can be removed by replacing backward Euler with a scheme that does not add artificial dissipation to the advection-dominated regime, such as Crank–Nicolson or an IMEX integrator that treats diffusion implicitly and advection explicitly. Regenerating the datasets with such a reference would directly test whether the diffusion-side variance shrinks. The under-resolution can also be reduced by shrinking the range of the diffusion magnitude a, so that the boundary layers are less sharp to begin with, or by refining the mesh.

The surrogates were trained as pure one-step regressors, without injected noise, rollout-aware losses, or architectural stabilisation, so the empirical limits of Table 5.5 reflect the one-step-trained baseline rather than the best achievable rollout results. A direct consequence is that the blow-up criterion $\varepsilon = 1$ of Section 4.5.2 cannot always separate genuine numerical instability from the slow accumulation of rollout error: the mean rollout error never fully settles, and even the strongest models retain a slight upward trend, so several variants reported as stable on the CFL axis in fact sit just below the threshold. Introducing training noise on the input snapshot, or a push-forward loss that exposes the surrogate to a few of its own predictions during training, would reduce this drift and make the reported limits interpretable as genuine instability thresholds rather than drifts. Re-measuring the limits after such stabilisation would reveal which of the borderline trajectories truly blow up.

A further mismatch lies between the training objective and the evaluation criterion. Training minimises the masked nodal MSE/Δt^2 of equation 4.11, in which every retained node carries equal weight, whereas the reported metrics $\mathscr{E}_{L^2}(u)$ and $\mathscr{E}_{H^1}(u)$ of Definition 4.5.1 are mesh-aware functional norms that weight each node by the size of the elements it belongs to. The discrepancy can be closed at negligible cost by multiplying the per-node squared error by a weight proportional to the area of mesh elements it is connected to. A more ambitious plan would be to reformulate the surrogate to learn second-order, piecewise-quadratic finite element coefficients on both nodes and edges rather than nodal values alone, which matches the true solution space more closely and could improve both accuracy and generalisation. This is already possible given the current message passing framework, but it would require a redesign of the input and output features and a more complex loss function.

The dimensionless numbers CFL_{max} and Fo_{max} used throughout are nodal extrema, recording the worst-case ratio of velocity or diffusion magnitude to mesh spacing at a single vertex, and therefore do not encode the geometry of the N_{mp}-hop neighbourhood that the message-passing architecture actually exercises. Two trajectories with the same nodal CFL_{max} may consequently stress the model in quantitatively different ways, which is also the most plausible reason why the single-scale CFL limits sit slightly above N_{mp} rather than at it. Replacing these extrema with an indicator that integrates over the N_{mp}-hop neighbourhood, would be expected to predict the empirical blow-up boundary more sharply

and to transfer better to families of meshes with differing local resolution.

On the architectural side, only a few symmetric block allocations were trained: three two-level allocations at the medium stride, and one two-level and one three-level allocation at the large stride (Tables 4.3–4.4), with asymmetric allocations and deeper V-cycles left unexplored. A denser sweep of asymmetric two-level allocations at fixed budget, together with a systematic variation of the V-cycle depth, would isolate the mechanism by which the three-level variant at the large stride overcomes the diffusion deficit of its two-level relative, an effect that the present results expose but cannot explain. In the same spirit, the feature-based coarsening of equation 4.4 relies on heuristic exponents and a fixed threshold that were set once and never varied. Studying these coefficients, or replacing the strength score with a small learnable function of the geometric and physical features, would clarify how strongly the coarsening choice shapes the resulting stability limits.

The physical setting is likewise deliberately restricted. The PDE solved is linear, on a fixed unit-square domain with periodic boundary conditions and smooth analytic velocity and diffusion fields (Section 4.2). This isolates the behaviour of the surrogate from boundary handling and geometric complexity, but it also avoids the regimes in which classical solvers become genuinely expensive and in which a surrogate would be most valuable. Repeating the stability experiments on nonlinear problems such as the viscous Burgers or Navier–Stokes equations at moderate Reynolds number, on geometrically complex domains, and with Dirichlet and Neumann boundary conditions, would test whether the operating ranges identified here persist in the settings where the multiscale architecture would face its most meaningful test.

Appendix A

Appendix

A.1 Feedforward neural networks

Following [9], the feedforward neural network, or multilayer perceptron (MLP), is defined as a model that approximates some function f. Specifically, an MLP defines a mapping $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$, learning the parameter values $\boldsymbol{\theta}$ that best approximate the target function. The term feedforward reflects the direction of information flow: from the input $\boldsymbol{x}$, through the intermediate layers, to the output $\boldsymbol{y}$, with no feedback connections. Networks with feedback connections are known as recurrent neural networks (RNNs), which lie outside the scope of the present work. MLPs are of fundamental importance in machine learning, as they form the foundation for numerous other architectures, such as the CNN described in Section A.2, which is a specialised variant of the feedforward network.

A.1.1 General architecture

Let the MLP have a certain input $\boldsymbol{x}$ and l hidden layers (also called the depth of a network), then we can define it as:

$$\boldsymbol{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)T}\boldsymbol{x} + \boldsymbol{b}^{(1)}) = g^{(1)}(\boldsymbol{z}^{(1)}) \tag{A.1}$$

$$\boldsymbol{h}^{(i)} = g^{(i)}(\boldsymbol{W}^{(i)T}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}) = g^{(i)}(\boldsymbol{z}^{(i)}) \text{ for } i \in \{2,\dots,l-1\}$$

$$\hat{\boldsymbol{y}} = g^{(l)}(\boldsymbol{W}^{(l)T}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) = g^{(l)}(\boldsymbol{z}^{(l)}) \tag{A.2}$$

Where the matrices $\boldsymbol{W}^{(i)} \in \mathbb{R}^{dim(h^{(i-1)})\times dim(h^{(i)})}$ are trainable weights and $\boldsymbol{b}^{(i)} \in \mathbb{R}^{dim(h^{(i)})}$ is a trainable bias vector, where the dimension of the hidden neurons is called the width of the hidden layer. Note that this definition assumes a fully connected network, although some weights could be so small that some neurons are virtually not connected. The functions $g^{(i)}$ are nonlinear element-wise activation functions. Then the equation A.1 to A.2 completely describe a forward pass through this MLP. We can also give them in terms of the design matrix, which is a more convenient representation with respect to training. So assume we have n training samples (or the size of the batch), then the design matrix $\boldsymbol{X}$ can be represented as:

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{pmatrix}.$$

You can then describe the forward passes as:

$$\boldsymbol{H}^{(1)} = g^{(1)}(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{B}^{(1)}) = g^{(1)}(\boldsymbol{Z}^{(1)})$$

$$\boldsymbol{H}^{(i)} = g^{(i)}(\boldsymbol{H}^{(i-1)}\boldsymbol{W}^{(i)} + \boldsymbol{B}^{(i)}) = g^{(i)}(\boldsymbol{Z}^{(i)}) \text{ for } i \in \{2,\dots,l-1\}$$

$$\hat{\boldsymbol{Y}} = g^{(l)}(\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)} + \boldsymbol{B}^{(l)}) = g^{(l)}(\boldsymbol{Z}^{(l)}),$$

Where, $\boldsymbol{H}^{(i)}$, $\boldsymbol{Y}$ are defined similarly as the design matrix for the input samples and $\boldsymbol{B}^{(i)}$ is a matrix where each row consists of $\boldsymbol{b}^{(i)T}$.

These networks are particularly powerful: the universal approximation theorem [57] states that an MLP with at least one hidden layer and a squashing activation function can approximate any Borel-measurable function between finite-dimensional spaces to within an arbitrary error $\epsilon > 0$, given sufficiently many hidden units. In other words, for any given target function, an MLP can in principle represent it, provided the network has sufficient width or depth. This does not, however, guarantee that the network will learn the function in practice, as the optimisation procedure may fail to converge, or the model may overfit. Practical limitations may also arise when the required hidden dimensions become prohibitively large, though increasing network depth can often alleviate this.

A.1.2 Gradients of the feedforward neural network

Given the forward pass defined in the preceding subsection, the network produces an output $\hat{\boldsymbol{y}}$, from which a scalar cost $J(\theta; y, \hat{y}) = \mathscr{L}(y, \hat{y}) + \lambda\Omega(\theta)$ is computed during training. Here, the first term measures the discrepancy between the target output and the MLP prediction, whilst the second term is a regularisation penalty. Backpropagation is then applied to compute Jacobians [1] with respect to the trainable parameters, which are subsequently used to update the model via a gradient-based optimisation algorithm. For this, we can start off by computing the Jacobian with respect to the output $\hat{\boldsymbol{y}}$ of the network:

$$\frac{\partial J}{\partial \hat{\boldsymbol{y}}} = \frac{\partial \mathscr{L}}{\partial \hat{\boldsymbol{y}}} = \boldsymbol{\delta} \in \mathbb{R}^{1\times dim(out)}.$$

Now denote $vec(\boldsymbol{W^{(i)^T}}) \in \mathbb{R}^{dim(h^{(i-1)})dim(h^{(i)})\times 1}$ as the vector of the weight matrix in lexicographical order (i.e. the rows transposed and stacked under each other). Then, using the chain rule, we get for the last layer:

$$\begin{aligned}
\frac{\partial J}{\partial vec(\boldsymbol{W}^{(l)T})} &= \frac{\partial \mathscr{L}}{\partial vec(\boldsymbol{W}^{(l)T})} + \lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{W}^{(l)T})} = \frac{\partial \mathscr{L}}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial vec(\boldsymbol{W}^{(l)T})} + \lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{W}^{(l)T})} \\
&= \frac{\partial \mathscr{L}}{\hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(l)}}\frac{\partial \boldsymbol{z}^{(l)}}{\partial vec(\boldsymbol{W}^{(l)T})} + \lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{W}^{(l)T})} \\
&= \boldsymbol{\delta} diag(g^{(l)'}(\boldsymbol{W}^{(l)T}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}))(\boldsymbol{h}^{(l-1)T} \otimes I) + \lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{W}^{(l)T})},
\end{aligned}$$

where $\otimes$ denotes the Kronecker product) Where in the last step we used that:

$$\begin{aligned}
\frac{\partial}{\partial vec(\boldsymbol{W}^{(l)T})}(\boldsymbol{W}^{(l)T}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) &= \frac{\partial}{\partial vec(\boldsymbol{W}^{(l)T})} vec(\boldsymbol{W}^{(l)T}\boldsymbol{h}^{(l-1)}) \\
&= \frac{\partial}{\partial vec(\boldsymbol{W}^{(l)T})}(\boldsymbol{h}^{(l-1)T} \otimes I)\, vec(\boldsymbol{W^{(l)T}}) = (\boldsymbol{h}^{(l-1)T} \otimes I)
\end{aligned}$$

Where we used an identity for the vectorization of a triple matrix product from [66] (IV.3). One could reshape the first term back into a $dim(h^{(i-1)}) \times dim(h^{(i)})$ matrix to see that:

$$\begin{aligned}
&\begin{pmatrix} \frac{\partial \mathscr{L}}{\partial w_{1,1}} & \cdots & \frac{\partial \mathscr{L}}{\partial w_{1,dim(h^{(i)})}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathscr{L}}{\partial w_{dim(h^{(i-1)}),1}} & \cdots & \frac{\partial \mathscr{L}}{\partial w_{dim(h^{(i-1)}),dim(h^{(i)})}} \end{pmatrix} = \\
&\begin{pmatrix} g^{(l)'}(z_1^{(l)})\delta_1 h_1^{(l-1)} & \cdots & g^{(l)'}(z_{dim(h^{(i)})}^{(l)})\delta_{dim(h^{(i)})} h_1^{(l-1)} \\ \vdots & \ddots & \vdots \\ g^{(l)'}(z_1^{(l)})\delta_1 h_{dim(h^{(i-1)})}^{(l-1)} & \cdots & g^{(l)'}(z_{dim(h^{(i)})}^{(l)})\delta_{dim(h^{(i)})} h_{dim(h^{(i-1)})}^{(l-1)} \end{pmatrix} \\
&= \boldsymbol{h^{(l-1)}}(\boldsymbol{\delta}^T \odot g'(\boldsymbol{z^{(l)}}))^T.
\end{aligned}$$

Here, $\odot$ denotes the element wise product. Similarly, we get for the bias term:

$$\frac{\partial J}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta} diag(g^{(l)'}(\boldsymbol{W}^{(l)T}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)})) = (\boldsymbol{\delta}^T \odot g'(\boldsymbol{z}^{(l)}))$$

[1] Jacobians are preferred over gradients here, as they admit compact expressions via matrix algebra. The gradient of $J(\theta; y, \hat{y})$ with respect to any parameter can be recovered from the corresponding Jacobian.

Note that the final expressions make it easier to use the stacked versions of X, Y and H from before, where you can express it as an inner product of a matrix filled with vectors. Subsequently, one can use the sum/mean of the summed Jacobians to update the parameters. Then, finally, we can propagate back through the network using the chain rule repeatedly, yielding:

$$\begin{aligned}\frac{\partial J}{\partial vec(\boldsymbol{W^{(i)T}})} &= \boldsymbol{\delta}\Big(\prod_{j=l}^{i+1}\frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{h}^{(j-1)}}\Big) diag(g^{(i)'}(\boldsymbol{W}^{(i)T}\boldsymbol{h}^{(i-1)}+\boldsymbol{b}^{(i)}))(\boldsymbol{h}^{(i-1)^T}\otimes I)+\lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{W}^{(l)T})}\\ \frac{\partial J}{\partial vec(\boldsymbol{b^{(i)}})} &= \boldsymbol{\delta}\Big(\prod_{j=l}^{i+1}\frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{h}^{(j-1)}}\Big) diag(g^{(i)'}(\boldsymbol{W}^{(i)T}\boldsymbol{h}^{(i-1)}+\boldsymbol{b}^{(i)}))+\lambda\frac{\Omega(\theta)}{\partial vec(\boldsymbol{b}^{(l)})},\end{aligned} \tag{A.3}$$

for the layers ranging from $l-1$ to the first layer, where

$$\frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{i-1}} = diag(g^{i'}(\boldsymbol{W}^{(i)T}\boldsymbol{h}^{(i-1)}+\boldsymbol{b}^{(i)})\boldsymbol{W}^{(i)T}$$

From a computational standpoint, backpropagation proceeds by starting at the last layer, computing the Jacobians and updating the corresponding weight matrix and bias vector. The gradient with respect to the previous layer's output is then computed and stored, after which the Jacobians for that layer are evaluated and the parameters updated. This process is repeated back to the first layer, avoiding redundant computations: as Equation A.3 shows, advancing to a previous layer requires only multiplying the stored Jacobian by the Jacobian of the current layer with respect to the outputs of the preceding layer.

A.2 Convolutional neural networks

Following [9], convolutional neural networks (CNNs) are a specialised class of feedforward neural network designed for structured, grid-like data such as images. More specifically, CNNs replace standard matrix multiplication with a convolution operation in at least one layer. The application of convolution to image data exploits three key properties:

1. **Sparse interactions:** Instead of having interactions between every input- and output neuron (like with a fully connected layer), we make use of a kernel that is smaller than the input image. This means that we need to store less parameters and do fewer computations. Sparse interactions therefore reduce memory requirements and improve statistical efficiency. Note that if we use a deep CNN, the hidden units in the deeper layers of the net may still interact with a larger portion of the input.

2. **Parameter sharing:** Before, we saw that for every input neuron to another output neuron, we used different weights. This is not the case for a CNN, where the same weights are used at different positions of the input [2]. This leads to less storage requirements.

3. **Equivariant representations:** The parameter sharing has as effect that it is equivariant to translation. So if we translate an object a certain amount in the input, it will translate the same amount in the output.

In order to fully characterise CNNs, the convolution operation is introduced first.

A.2.1 The convolution operation

Convolution is a mathematical operation that arises across many fields of science. In the machine learning context, the input is typically a tensor that is modified by a learnable kernel tensor. For a two-dimensional image $I \in \mathbb{R}^{n_1\times n_2}$, a two-dimensional kernel $K \in \mathbb{R}^{s_1\times s_2}$ yields the (commutative) convolution,

$$\begin{aligned}S(i,j) = (I*K)(i,j) &= \sum_m\sum_n I(m,n)K(i-m,j-n)\\ &= \sum_m\sum_n I(i-m,j-n)K(m,n) = (K*I)(i,j).\end{aligned} \tag{A.4}$$

[2] However, this also depends on the treatment of boundary inputs. Sometimes we apply 'valid convolution', where we only look at inputs where the kernel can be completely placed over the input.

Here, m and n sum over the valid indices of the image and the kernel. Equation A.4 is commonly adopted in machine learning, owing to less variation in the valid index range. The commutativity of convolution follows from the kernel being flipped before being placed over the image. In network applications this distinction is often immaterial, and cross-correlation, which is identical to convolution but without the kernel flip, is frequently used instead:

$$\hat{S}(i,j) = (I \hat{*} K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

Convolution and cross-correlation are often used interchangeably in machine learning. Since the kernel parameters are learnable, flipping the kernel prior to application has no effect on the network's expressive capacity. Convolution without kernel flipping is therefore adopted throughout, unless stated otherwise.

A.2.2 General architecture

The forward pass of a convolutional layer for a 2D image is now described, following [67] with minor adaptations. Let $X \in \mathbb{R}^{H \times W \times C_{in}}$ denote the input tensor with H rows, W columns, and C_{in} input channels; for instance, $C_{in} = 3$ for an RGB image. During training, a batch dimension N is appended, giving $X \in \mathbb{R}^{H \times W \times C_{in} \times N}$. The convolutional layer maps this to an output tensor $Y \in \mathbb{R}^{H' \times W' \times C_{out}}$, again extendable with a batch dimension. The kernel tensor is $K \in \mathbb{R}^{s_1 \times s_2 \times C_{in} \times C_{out}}$, so that for each output channel there is a three-dimensional kernel that is applied to the inputs. This amounts to C_{in} independent two-dimensional convolutions per output channel, each with its own two-dimensional kernel, whose results are summed over the input channel dimension. This process is repeated for each of the C_{out} output channels. Tensor elements are indexed by i, j, k, l starting at 0, so that $X_{0,0,0,0}$ refers to the first row, column, channel, and sample of the input tensor.
The kernel slides over the image with a stride S, meaning it is shifted S positions in either the height or width direction at each step. Applying convolution reduces the spatial dimensions of the output relative to the input; to counteract this, padding P can be added to the input borders. These additional entries are typically set to zero, though other values may be used. Assuming that we apply the same amount of padding to each border, the output dimensions will be:

$$H' = 1 + \lfloor (H + 2P - s_1)/S \rfloor,$$
$$W' = 1 + \lfloor (W + 2P - s_2)/S \rfloor,$$

where $\lfloor \cdot \rfloor$ denote a floor division. Note that if the division in the floor division operator is not integer, we are not using all the inputs of the image and one might want to change their stride. Denote X' as the padded kernel, then with everything defined, we can denote the output of convolution without flipping kernel as

$$Y_{i',j',k',l'} = \sum_{c=0}^{C_{in}-1} \sum_{i=0}^{s_1-1} \sum_{j=0}^{s_2-1} K_{i,j,c,k'} \cdot X'_{i' \cdot S + i, j' \cdot S + j, k', l'} + b_{k'},$$

where $0 \leq i' < H'$, $0 \leq j' < W'$, $0 \leq k' < C_{out}$, $0 \leq l' < N$ and $b \in \mathbb{R}^{C_{out}}$ is a bias vector. This then defines the forward pass through a convolutional layer. But since this might be a bit vague, it is also depicted in pseudocode in algorithm 2.

A.2.3 Gradients of the convolutional neural network

Analogously to the MLP, analytical expressions for the partial derivatives with respect to the weights and biases are derived here. Since the backpropagation procedure through the layers was already presented in the MLP section, it is not repeated. Instead, the Jacobian of the cost function with respect to the layer output, $\frac{\partial J}{\partial \text{vec}(Y)} \in \mathbb{R}^{1 \times C_{out} \cdot H' \cdot W'}$, is assumed to be known. The focus is on the partial derivatives with respect to the kernel weights and the bias vector. For clarity, $N = 1$ is assumed, so that $Y \in \mathbb{R}^{H' \times W' \times C_{out}}$ and $X \in \mathbb{R}^{H \times W \times C_{in}}$. For this, note that:

$$\frac{\partial J}{\partial vec(K)} = \frac{\partial J}{\partial vec(Y)} \frac{\partial vec(Y)}{\partial vec(K)} := \boldsymbol{\delta} \frac{\partial vec(Y)}{\partial vec(K)}.$$

Algorithm 2 Making a forward pass through a convolutional layer, given the tensors X, K, $\mathbf{b}$ and integers P, S, N

procedure $Forward\_Pass\_CNN(X, \mathbf{b}, K, P, S, N)$
  $X' = \text{pad}(X, P)$ ▷ Add a patch of P inputs at all the boundaries
  Retrieve s_1, s_2 from Kernel size
  $H' = 1 + \lfloor (H + 2P - s_1)/S \rfloor$ ▷ Output Height
  $W' = 1 + \lfloor (W + 2P - s_2)/S \rfloor$ ▷ Output width
  Initialize $Y = 0$ ▷ $H' \times W' \times C_{out} \times N$ array
  for $i = 0, 1 \ldots, H' - 1$ **do**
    for $j = 0, 1 \ldots, W' - 1$ **do**
      $h_{\text{offset}} = i * S$ ▷ Find starting row of receptive field
      $w_{\text{offset}} = j * S$ ▷ Find starting column of receptive field
      $X'_{\text{Window}} = X'[h_{\text{offset}} : h_{\text{offset}} + s_1, w_{\text{offset}} : w_{\text{offset}} + s_2, :, :]$
      for $k = 0, 1, \ldots, N - 1$ **do**
        $X_{\text{temp}} \leftarrow$ Stack $X'_{window}[:, :, :, k]$ C_{out} times in 4th dimension ▷ Match shape of kernel
        $Y[i, j, :, k] = \sum_{\text{In dimensions of rows, columns and input channels}} (X_{temp} \odot K) + \mathbf{b}$
      end for
    end for
  end for
end procedure

Since the first term of the product is known, the key to finding the Jacobian lies in finding an expression for $\frac{\partial vec(Y)}{\partial vec(K)}$. We define the vectorization of the tensor Y to be as follows (Same as [67]): let Y_0, Y_1, …, $Y_{C_{out}-1}$ be the matrices corresponding to each channel. Then we define the vectorization of this tensor to be:

$$vec(Y) = \begin{pmatrix} vec(Y_0) \\ vec(Y_1) \\ \vdots \\ vec(Y_{C_{out}-1}) \end{pmatrix}.$$

We define the vectorization of X, K similarly. As done in [67], we can write the vectorization of the convolution Y_{c_2} ($c_2 \in \{0, \ldots, C_{out} - 1\}$) without kernel flipping kernel as

$$vec(Y_{c_2}) = vec(X \hat{*} K[:, :, :, c_2]) := vec(X \hat{*} K_{:,c_2}) = \begin{pmatrix} vec(\hat{X}^{0T} K_{:,c_2}[:, :, 0]) \\ vec(\hat{X}^{1T} K_{:,c_2}[:, :, 1]) \\ \vdots \\ vec(\hat{X}^{C_{in}-1T} K_{:,c_2}[:, :, C_{in} - 1]) \end{pmatrix}$$

$$:= \begin{pmatrix} vec(\hat{X}^{0T} K_{0,c_2}) \\ vec(\hat{X}^{1T} K_{1,c_2}) \\ \vdots \\ vec(\hat{X}^{C_{in}-1T} K_{C_{in}-1,c_2}) \end{pmatrix} \text{ for } c_2 \in \{0, 1, \ldots, C_{out}\}.$$

Where $\hat{X}^{c_1} := \begin{pmatrix} \hat{X}^{c_1}_{0,0} & \hat{X}^{c_1}_{1,0} & \cdots & \hat{X}^{c_1}_{H'-1,0} & \hat{X}^{c_1}_{0,1} & \cdots & \hat{X}^{c_1}_{H'-1,W'-1} \end{pmatrix} \in \mathbb{R}^{s_1 s_2 \times H'W'}$ is the im2col transformation defined in [68], which transforms the image X^c into a so-called input-patch-matrix, which is created by copying patches out of the input and unrolling these into columns of this matrix (similar to the X_{Window} we defined before in algorithm 2). Such a patch is in our case defined as

$$\hat{X}^{c_1}_{i,j} = \begin{pmatrix} X_{i \cdot S, j \cdot S, c_1} \\ X_{i \cdot S + 1, j \cdot S, c_1} \\ \vdots \\ X_{i \cdot S + s_1, j \cdot S, c_1} \\ X_{i \cdot S, j \cdot S + 1, c_1} \\ \vdots \\ X_{i \cdot S + s_1, j \cdot S + s_2, c_1} \end{pmatrix} \in \mathbb{R}^{s_1 s_2 \times 1} \text{ for } i \in \{0, 1, \ldots, H' - 1\}, j \in \{0, 1, \ldots, W' - 1\},$$

$$c_1 \in \{0, 1, \ldots, C_{in} - 1\},$$

which allows us to write the vectorization of a convolution without kernel flipping as

$$vec(X^{c_1} \hat{*} K_{c_1,c_2}) = \hat{X}^{c_1 T} vec(K_{c_1,c_2}). \tag{A.5}$$

Taking the partial derivative of A.5 yields

$$\frac{\partial \hat{X}^{c_1 T} vec(K_{c_1,c_2})}{\partial vec(K_{c_1,c_2})} = \hat{X}^{c_1 T}$$

Now first let's define the block matrix which consists of all the input-patch matrices

$$\mathscr{X} = \begin{pmatrix} \hat{X}^{0T} & \hat{X}^{1T} & \dots & \hat{X}^{C_{in-1}T} \end{pmatrix} \in \mathbb{R}^{H' \cdot W' \times C_{in} \cdot s_1 \cdot s_2}.$$

It is then not hard to see that $\frac{\partial vec(Y_{c_2})}{\partial vec(K_{:,c_2})} = \mathscr{X}$ and therefore

$$\frac{\partial vec(Y)}{\partial vec(K)} = \begin{pmatrix} \mathscr{X} & \mathscr{O} & \dots & \mathscr{O} \\ \mathscr{O} & \mathscr{X} & \mathscr{O} & \vdots \\ \vdots & \ddots & \ddots & \ddots \\ \mathscr{O} & \dots & \dots & \mathscr{X} \end{pmatrix} \in \mathbb{R}^{C_{out} \cdot H' \cdot W' \times C_{out} \cdot C_{in} \cdot s_1 \cdot s_2},$$

where $\mathscr{O}$ denotes the zero matrix with appropriate sizes. This then yields the following Jacobian for the cost function with respect to the vectorization of the kernel weights:

$$\frac{\partial J}{\partial vec(K)} = \begin{pmatrix} \frac{\partial J}{\partial vec(Y_0)} \mathscr{X} & \frac{\partial J}{\partial vec(Y_1)} \mathscr{X} & \dots & \frac{\partial J}{\partial vec(Y_{C_{out}-1})} \mathscr{X} \end{pmatrix}$$

A.3 Additional results tables and figures

This appendix collects the full per-checkpoint rollout error tables and a selection of field-snapshot figures from Chapter 5.3. The tables are omitted from the main body to avoid interrupting the narrative flow; cross- references in the main text point back to the corresponding table or figure here.

A.3.1 Single-scale models at $\Delta t_{\text{small}} = 0.015$

Table A.1: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale models at $\Delta t_{\text{small}} = 0.015$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
`Single-dt15-2mp`	0.97 ± 0.847	1.33 ± 1.09	1.59 ± 1.3	1.82 ± 1.48	2.02 ± 1.66	2.21 ± 1.83	2.38 ± 1.98	2.53 ± 2.15
`Single-dt15-4mp`	0.429 ± 0.543	0.585 ± 0.746	0.722 ± 0.938	0.841 ± 1.12	0.952 ± 1.31	1.05 ± 1.49	1.14 ± 1.64	1.22 ± 1.78
`Single-dt15-6mp`	0.207 ± 0.291	0.267 ± 0.43	0.315 ± 0.581	0.36 ± 0.732	0.405 ± 0.869	0.448 ± 0.987	0.481 ± 1.07	0.516 ± 1.17
`Single-dt15-8mp`	0.152 ± 0.153	0.245 ± 0.28	0.287 ± 0.385	0.322 ± 0.519	0.363 ± 0.646	0.399 ± 0.747	0.432 ± 0.836	0.461 ± 0.908
`Single-dt15-10mp`	**0.13 ± 0.092**	**0.149 ± 0.0871**	**0.146 ± 0.0721**	**0.142 ± 0.0682**	**0.141 ± 0.0694**	**0.142 ± 0.0714**	**0.142 ± 0.0722**	**0.142 ± 0.0733**

Table A.2: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale models at $\Delta t_{\text{small}} = 0.015$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
`Single-dt15-2mp`	0.772 ± 1.5	1.41 ± 2.97	1.95 ± 4.28	2.4 ± 5.27	2.84 ± 6.26	3.24 ± 7.25	3.61 ± 8.31	3.94 ± 9.4
`Single-dt15-4mp`	0.325 ± 0.908	0.591 ± 1.69	0.826 ± 2.46	1.02 ± 3	1.21 ± 3.44	1.38 ± 3.79	1.55 ± 4.19	1.73 ± 4.63
`Single-dt15-6mp`	0.111 ± 0.156	0.184 ± 0.348	0.27 ± 0.689	0.359 ± 1.11	0.438 ± 1.47	0.504 ± 1.74	0.565 ± 1.97	0.616 ± 2.14
`Single-dt15-8mp`	0.0971 ± 0.128	0.14 ± 0.211	0.174 ± 0.274	0.205 ± 0.337	0.234 ± 0.393	0.262 ± 0.446	0.287 ± 0.493	0.309 ± 0.536
`Single-dt15-10mp`	**0.0833 ± 0.0509**	**0.112 ± 0.0861**	**0.137 ± 0.127**	**0.164 ± 0.168**	**0.191 ± 0.212**	**0.218 ± 0.26**	**0.246 ± 0.313**	**0.273 ± 0.368**

Table A.3: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale models at $\Delta t_{\text{small}} = 0.015$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
`Single-dt15-2mp`	3.08 ± 3.46	3.91 ± 3.91	4.9 ± 4.91	6.18 ± 6.46	7.7 ± 8.35	9.77 ± 10.8	12 ± 13.4	14.4 ± 15.9
`Single-dt15-4mp`	1.01 ± 1.4	1.28 ± 1.79	1.57 ± 2.3	1.95 ± 3.08	2.43 ± 4.02	3.09 ± 5.24	3.84 ± 6.56	4.54 ± 7.73
`Single-dt15-6mp`	0.497 ± 0.599	0.538 ± 0.818	0.582 ± 1.07	0.654 ± 1.43	0.777 ± 1.93	0.952 ± 2.57	1.12 ± 3.12	1.29 ± 3.67
`Single-dt15-8mp`	**0.325 ± 0.25**	0.418 ± 0.357	0.457 ± 0.459	0.507 ± 0.587	0.588 ± 0.814	0.692 ± 1.13	0.804 ± 1.5	0.915 ± 1.88
`Single-dt15-10mp`	0.374 ± 0.282	**0.375 ± 0.241**	**0.32 ± 0.151**	**0.287 ± 0.112**	**0.282 ± 0.101**	**0.283 ± 0.111**	**0.275 ± 0.0934**	**0.277 ± 0.0893**

Table A.4: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale models at $\Delta t_{\text{small}} = 0.015$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt15-2mp	8.73 ± 30.5	28.3 ± 134	85.6 ± 552	234 ± 1.77e+03	569 ± 4.51e+03	1.31e+03 ± 1.07e+04	2.84e+03 ± 2.38e+04	6.07e+03 ± 5.26e+04
Single-dt15-4mp	4.78 ± 21.2	17.6 ± 103	58.7 ± 444	167 ± 1.44e+03	416 ± 3.68e+03	959 ± 8.64e+03	2.12e+03 ± 1.95e+04	4.59e+03 ± 4.38e+04
Single-dt15-6mp	1.32 ± 5.07	4.58 ± 24.5	14.8 ± 106	41.5 ± 335	104 ± 868	239 ± 2.03e+03	533 ± 4.63e+03	1.13e+03 ± 1.02e+04
Single-dt15-8mp	0.556 ± 1.87	1.74 ± 9.41	5.88 ± 43.1	17.3 ± 143	44.4 ± 381	105 ± 914	234 ± 2.08e+03	506 ± 4.6e+03
Single-dt15-10mp	**0.395 ± 0.799**	**0.96 ± 4.24**	**2.8 ± 18.8**	**7.75 ± 61.5**	**19.2 ± 158**	**45.3 ± 380**	**98.8 ± 840**	**214 ± 1.88e+03**

A.3.2 Single-scale and multiscale models at $\Delta t_{\text{medium}} = 0.030$

Table A.5: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt30-8mp	0.243 ± 0.283	0.323 ± 0.397	0.371 ± 0.462	0.409 ± 0.522	0.452 ± 0.607	0.488 ± 0.68	0.519 ± 0.749	0.549 ± 0.817
Multi-dt30-3x2x3	0.165 ± 0.186	0.233 ± 0.289	0.283 ± 0.359	0.319 ± 0.433	0.357 ± 0.506	0.393 ± 0.571	0.424 ± 0.633	0.457 ± 0.7
Multi-dt30-2x4x2	0.138 ± 0.0934	0.231 ± 0.162	0.289 ± 0.213	0.319 ± 0.272	0.343 ± 0.338	0.366 ± 0.401	0.386 ± 0.46	0.405 ± 0.513
Multi-dt30-1x6x1	**0.132 ± 0.0807**	**0.19 ± 0.116**	**0.229 ± 0.142**	**0.253 ± 0.157**	**0.274 ± 0.176**	**0.298 ± 0.203**	**0.321 ± 0.227**	**0.344 ± 0.254**

Table A.6: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt30-8mp	0.1 ± 0.101	0.16 ± 0.291	0.232 ± 0.613	0.303 ± 0.932	0.365 ± 1.19	0.419 ± 1.36	0.467 ± 1.52	0.508 ± 1.65
Multi-dt30-3x2x3	**0.0768 ± 0.0399**	**0.0994 ± 0.065**	**0.129 ± 0.132**	**0.17 ± 0.241**	**0.221 ± 0.383**	**0.28 ± 0.551**	**0.344 ± 0.741**	**0.413 ± 0.949**
Multi-dt30-2x4x2	0.0863 ± 0.0841	0.148 ± 0.341	0.209 ± 0.587	0.262 ± 0.779	0.31 ± 0.98	0.354 ± 1.17	0.393 ± 1.32	0.427 ± 1.45
Multi-dt30-1x6x1	0.115 ± 0.172	0.422 ± 1.07	0.951 ± 2.78	1.45 ± 4.5	1.85 ± 5.76	2.17 ± 6.65	2.43 ± 7.29	2.65 ± 7.8

Table A.7: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt30-8mp	0.548 ± 0.553	0.659 ± 0.784	0.758 ± 1	0.876 ± 1.32	1.06 ± 1.77	1.3 ± 2.33	1.55 ± 2.87	1.82 ± 3.42
Multi-dt30-3x2x3	0.379 ± 0.315	0.467 ± 0.48	0.552 ± 0.628	0.629 ± 0.847	0.74 ± 1.13	0.905 ± 1.48	1.06 ± 1.79	1.24 ± 2.18
Multi-dt30-2x4x2	**0.279 ± 0.138**	**0.389 ± 0.238**	**0.47 ± 0.316**	**0.513 ± 0.41**	**0.555 ± 0.53**	**0.619 ± 0.705**	**0.672 ± 0.874**	**0.732 ± 1.03**
Multi-dt30-1x6x1	0.354 ± 0.169	0.463 ± 0.25	0.538 ± 0.308	0.605 ± 0.403	0.686 ± 0.538	0.811 ± 0.738	0.946 ± 0.947	1.08 ± 1.13

Table A.8: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale and two-level multiscale models at $\Delta t_{\text{medium}} = 0.030$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt30-8mp	0.832 ± 2.94	2.76 ± 14.2	8.86 ± 61	25 ± 195	62.7 ± 508	148 ± 1.23e+03	323 ± 2.73e+03	686 ± 5.95e+03
Multi-dt30-3x2x3	**0.366 ± 0.716**	**1.01 ± 4.61**	**3.29 ± 21**	**10.1 ± 76.1**	**29.1 ± 238**	**74.8 ± 635**	**180 ± 1.58e+03**	**413 ± 3.73e+03**
Multi-dt30-2x4x2	0.517 ± 2.31	2.21 ± 15.7	8.23 ± 71.2	24.9 ± 237	62.5 ± 615	145 ± 1.44e+03	333 ± 3.33e+03	735 ± 7.49e+03
Multi-dt30-1x6x1	1.4 ± 6.23	8.79 ± 40.4	30 ± 179	84.9 ± 590	216 ± 1.58e+03	513 ± 3.91e+03	1.13e+03 ± 8.77e+03	2.41e+03 ± 1.89e+04

A.3.3 Single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$

Table A.9: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt60-16mp	0.191 ± 0.185	0.253 ± 0.28	0.288 ± 0.322	0.324 ± 0.362	0.365 ± 0.408	0.405 ± 0.45	0.442 ± 0.48	0.476 ± 0.506
Multi-dt60-4x8x4	0.139 ± 0.0771	0.189 ± 0.104	0.212 ± 0.108	0.22 ± 0.106	0.226 ± 0.107	0.23 ± 0.11	0.235 ± 0.115	0.243 ± 0.122
Multi-dt60-3x3x4x3x3	**0.127 ± 0.0717**	**0.17 ± 0.0909**	**0.191 ± 0.0923**	**0.196 ± 0.0885**	**0.201 ± 0.0907**	**0.205 ± 0.0956**	**0.207 ± 0.1**	**0.209 ± 0.106**

Table A.10: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt60-16mp	0.0888 ± 0.053	0.121 ± 0.118	0.154 ± 0.222	0.185 ± 0.32	0.215 ± 0.407	0.243 ± 0.481	0.269 ± 0.544	0.294 ± 0.608
Multi-dt60-4x8x4	0.0858 ± 0.0469	0.12 ± 0.117	0.16 ± 0.251	0.203 ± 0.381	0.247 ± 0.5	0.29 ± 0.608	0.333 ± 0.709	0.376 ± 0.807
Multi-dt60-3x3x4x3x3	**0.073 ± 0.0312**	**0.0869 ± 0.0366**	**0.0971 ± 0.0447**	**0.107 ± 0.0533**	**0.116 ± 0.0632**	**0.126 ± 0.0744**	**0.136 ± 0.0853**	**0.144 ± 0.0952**

Table A.11: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$, on the advection test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt60-16mp	0.439 ± 0.268	0.483 ± 0.373	0.495 ± 0.443	0.53 ± 0.548	0.595 ± 0.718	0.701 ± 0.939	0.811 ± 1.17	0.91 ± 1.36
Multi-dt60-4x8x4	**0.276 ± 0.0918**	0.327 ± 0.131	0.371 ± 0.15	0.388 ± 0.164	0.415 ± 0.185	0.434 ± 0.198	0.44 ± 0.203	0.442 ± 0.203
Multi-dt60-3x3x4x3x3	0.283 ± 0.124	**0.322 ± 0.147**	**0.357 ± 0.139**	**0.359 ± 0.124**	**0.374 ± 0.128**	**0.391 ± 0.14**	**0.395 ± 0.14**	**0.39 ± 0.137**

Table A.12: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale and multiscale models at $\Delta t_{\text{large}} = 0.060$, on the diffusion test set.

Model	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt60-16mp	0.385 ± 0.683	0.925 ± 4.26	2.78 ± 19.8	7.67 ± 64	18.8 ± 164	43.3 ± 388	97.2 ± 902	216 ± 2.06e+03
Multi-dt60-4x8x4	0.299 ± 0.236	0.501 ± 0.976	1.12 ± 4.49	2.77 ± 15.9	6.62 ± 43.7	15.3 ± 109	33.5 ± 249	71.4 ± 549
Multi-dt60-3x3x4x3x3	**0.255 ± 0.115**	**0.341 ± 0.41**	**0.594 ± 1.79**	**1.24 ± 6.02**	**2.77 ± 16.7**	**6.23 ± 42.5**	**13.4 ± 95.2**	**28 ± 201**

A.3.4 Comparison across surrogate time steps

Table A.13: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale models (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp), on the advection test set.

Model	Δt	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt15-4mp	0.015	0.429 ± 0.543	0.585 ± 0.746	0.722 ± 0.938	0.841 ± 1.12	0.952 ± 1.31	1.05 ± 1.49	1.14 ± 1.64	1.22 ± 1.78
Single-dt30-8mp	0.030	0.243 ± 0.283	0.323 ± 0.397	0.371 ± 0.462	0.409 ± 0.522	0.452 ± 0.607	0.488 ± 0.68	0.519 ± 0.749	0.549 ± 0.817
Single-dt60-16mp	0.060	**0.191 ± 0.185**	**0.253 ± 0.28**	**0.288 ± 0.322**	**0.324 ± 0.362**	**0.365 ± 0.408**	**0.405 ± 0.45**	**0.442 ± 0.48**	**0.476 ± 0.506**

Table A.14: Rollout $\mathscr{E}_{L^2}(u)$ error (mean ± std) for the single-scale models (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp), on the diffusion test set.

Model	Δt	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt15-4mp	0.015	0.325 ± 0.908	0.591 ± 1.69	0.826 ± 2.46	1.02 ± 3	1.21 ± 3.44	1.38 ± 3.79	1.55 ± 4.19	1.73 ± 4.63
Single-dt30-8mp	0.030	0.1 ± 0.101	0.16 ± 0.291	0.232 ± 0.613	0.303 ± 0.932	0.365 ± 1.19	0.419 ± 1.36	0.467 ± 1.52	0.508 ± 1.65
Single-dt60-16mp	0.060	**0.0888 ± 0.053**	**0.121 ± 0.118**	**0.154 ± 0.222**	**0.185 ± 0.32**	**0.215 ± 0.407**	**0.243 ± 0.481**	**0.269 ± 0.544**	**0.294 ± 0.608**

Table A.15: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale models (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp), on the advection test set.

Model	Δt	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt15-4mp	0.015	1.01 ± 1.4	1.28 ± 1.79	1.57 ± 2.3	1.95 ± 3.08	2.43 ± 4.02	3.09 ± 5.24	3.84 ± 6.56	4.54 ± 7.73
Single-dt30-8mp	0.030	0.548 ± 0.553	0.659 ± 0.784	0.758 ± 1	0.876 ± 1.32	1.06 ± 1.77	1.3 ± 2.33	1.55 ± 2.87	1.82 ± 3.42
Single-dt60-16mp	0.060	**0.439 ± 0.268**	**0.483 ± 0.373**	**0.495 ± 0.443**	**0.53 ± 0.548**	**0.595 ± 0.718**	**0.701 ± 0.939**	**0.811 ± 1.17**	**0.91 ± 1.36**

Table A.16: Rollout $\mathscr{E}_{H^1}(u)$-seminorm error (mean ± std) for the single-scale models (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp), on the diffusion test set.

Model	Δt	$t = 0.24$	$t = 0.48$	$t = 0.72$	$t = 0.96$	$t = 1.20$	$t = 1.44$	$t = 1.68$	$t = 1.92$
Single-dt15-4mp	0.015	4.78 ± 21.2	17.6 ± 103	58.7 ± 444	167 ± 1.44e+03	416 ± 3.68e+03	959 ± 8.64e+03	2.12e+03 ± 1.95e+04	4.59e+03 ± 4.38e+04
Single-dt30-8mp	0.030	0.832 ± 2.94	2.76 ± 14.2	8.86 ± 61	25 ± 195	62.7 ± 508	148 ± 1.23e+03	323 ± 2.73e+03	686 ± 5.95e+03
Single-dt60-16mp	0.060	**0.385 ± 0.683**	**0.925 ± 4.26**	**2.78 ± 19.8**	**7.67 ± 64**	**18.8 ± 164**	**43.3 ± 388**	**97.2 ± 902**	**216 ± 2.06e+03**

A.3.5 Additional field snapshots

The diffusion-dominated rollout of Figure A.1 shows all five single-scale variants behaving similarly: each model converges to the expected steady state, with comparable log-error patterns across the eight checkpoints. This example illustrates that there are regimes in which every member of the small-stride group produces a qualitatively well-behaved rollout.

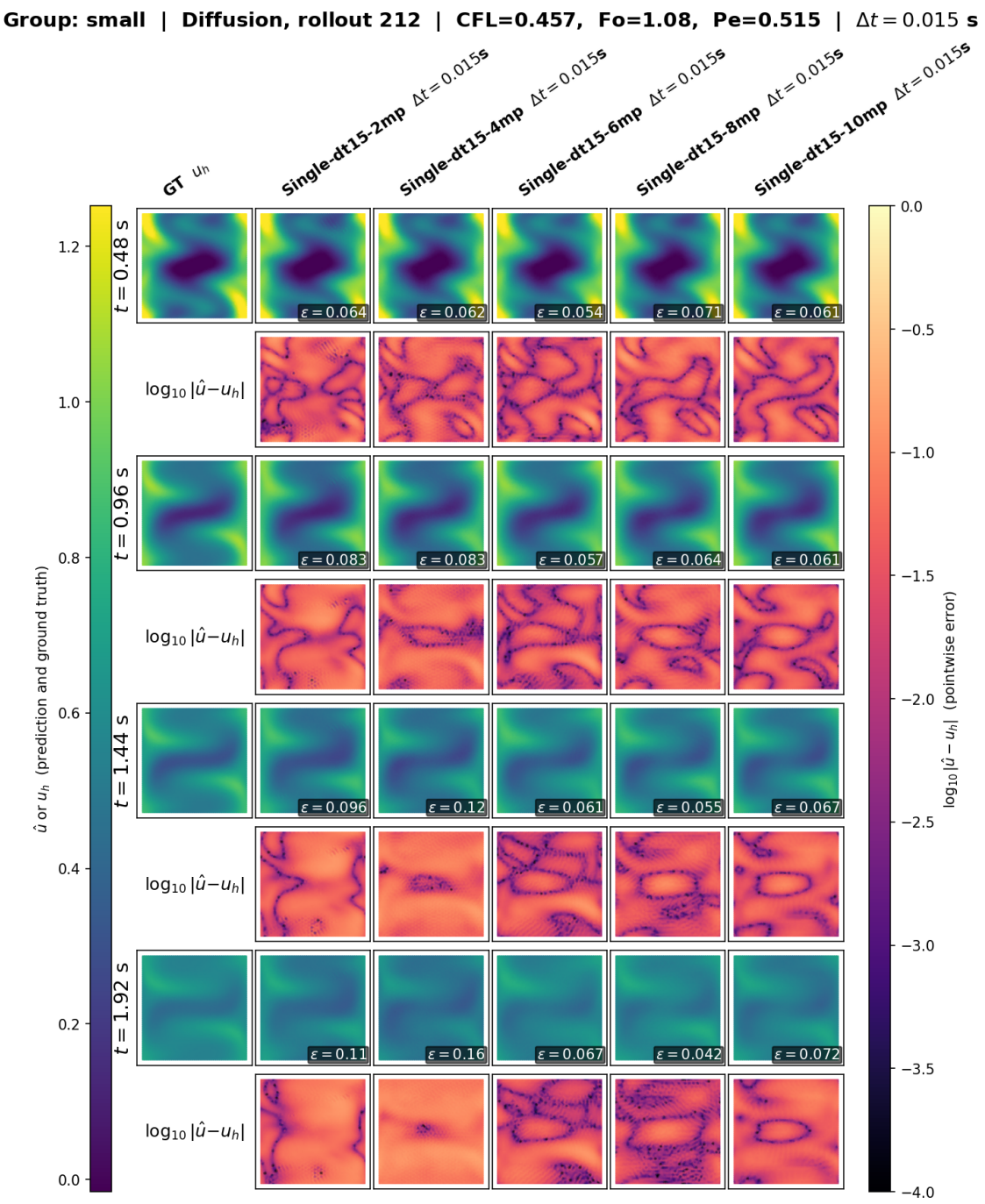


Figure A.1: Solution field snapshots for the small-stride single-scale models (Single-dt15-2mp–Single-dt15-10mp) on a diffusion-dominated test rollout ($\mathrm{Fo}_{\max}$ near the stability limits of this group). Layout as in Figure 5.10.

Figure A.2 shows the single-scale baseline Single-dt30-8mp and Multi-dt30-2x4x2 converging cleanly to the expected steady state. The variants Multi-dt30-3x2x3 and Multi-dt30-1x6x1 display visible oscillations in their prediction panels, although the relative $\mathscr{E}_{L^2}(u)$ at each checkpoint remains below unity, so neither is classified as diverged. In contrast with the group-averaged trend of Figure 5.7, on this particular rollout the single-scale baseline outperforms each of the multiscale variants.

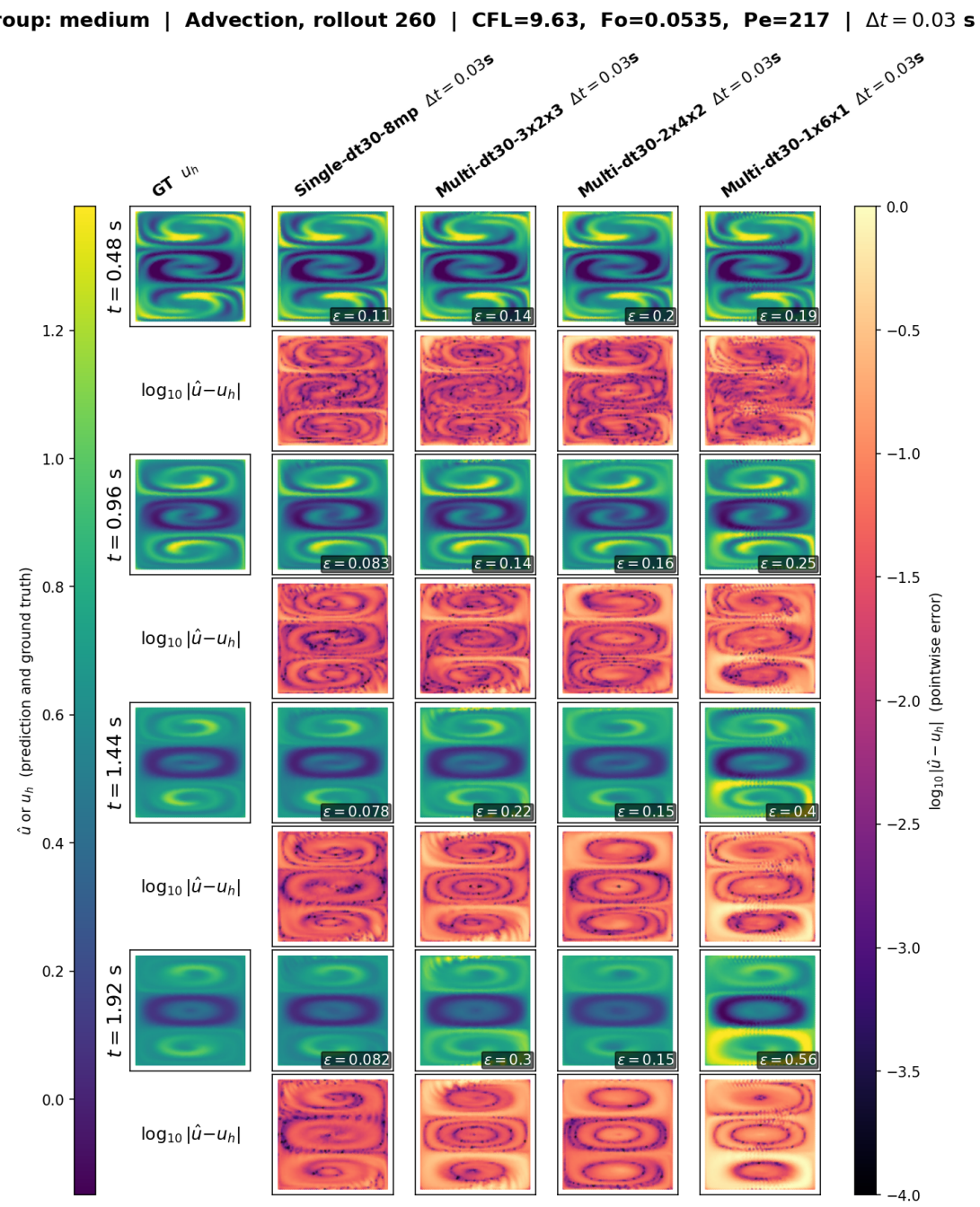


Figure A.2: Solution field snapshots for the medium-stride models (Single-dt30-8mp, Multi-dt30-3x2x3, Multi-dt30-2x4x2, Multi-dt30-1x6x1) on an advection-dominated test rollout (CFL_{max} near the stability limits of this group). Layout as in Figure 5.10.

The diffusion-dominated rollout of Figure A.3 shows all four medium-stride models converging to the expected steady state, with comparable pointwise log-error patterns at the final checkpoint.

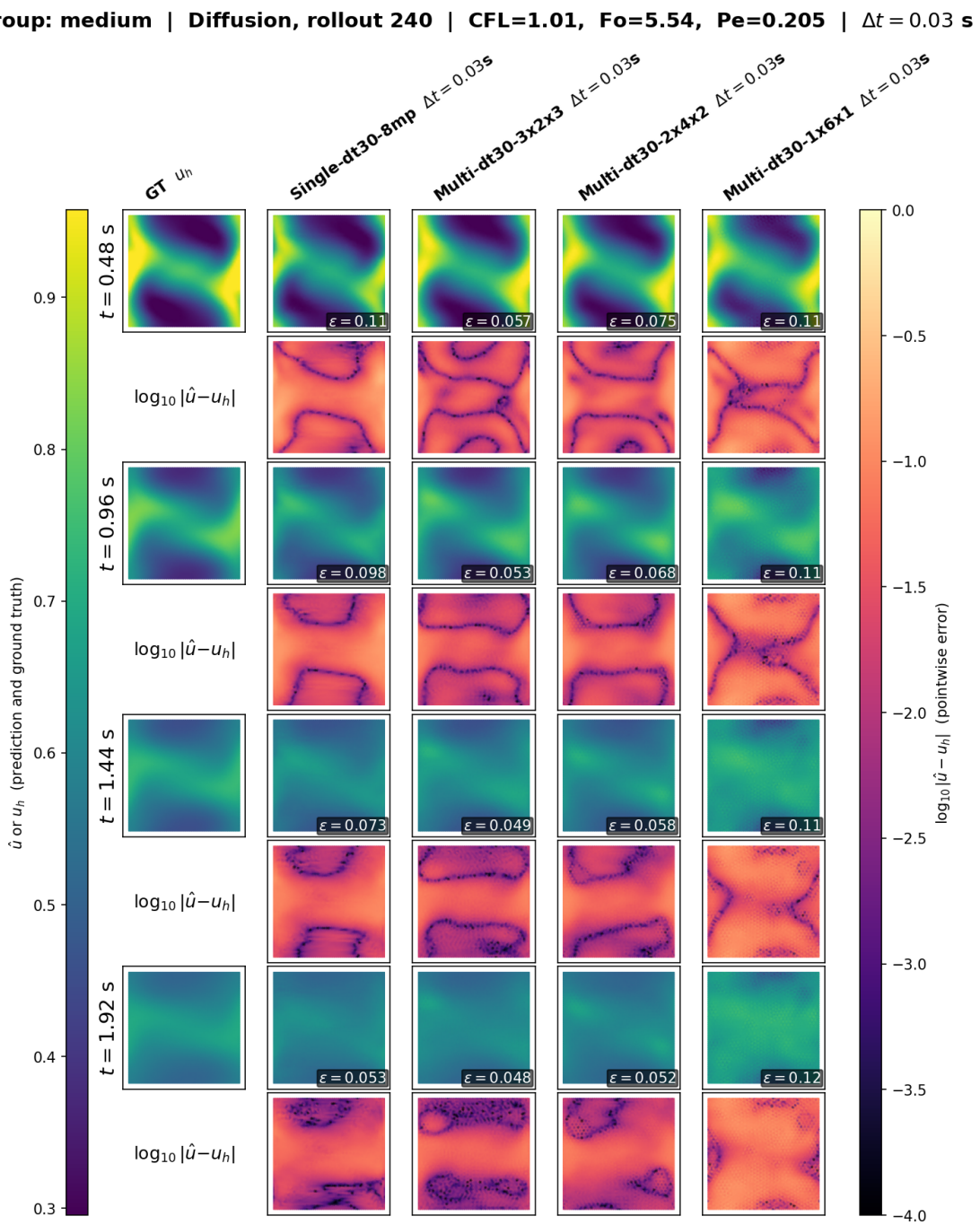


Figure A.3: Solution field snapshots for the medium-stride models (Single-dt30-8mp, Multi-dt30-3x2x3, Multi-dt30-2x4x2, Multi-dt30-1x6x1) on a diffusion-dominated test rollout ($\mathrm{Fo}_{\max}$ near the stability limits of this group). Layout as in Figure 5.10.

A similar conclusion holds in the diffusion-dominated regime of Figure A.4 models produce qualitatively comparable checkpoint sequences, and their relative $\mathscr{E}_{L^2}(u)$ values at the final checkpoint are close to one another.

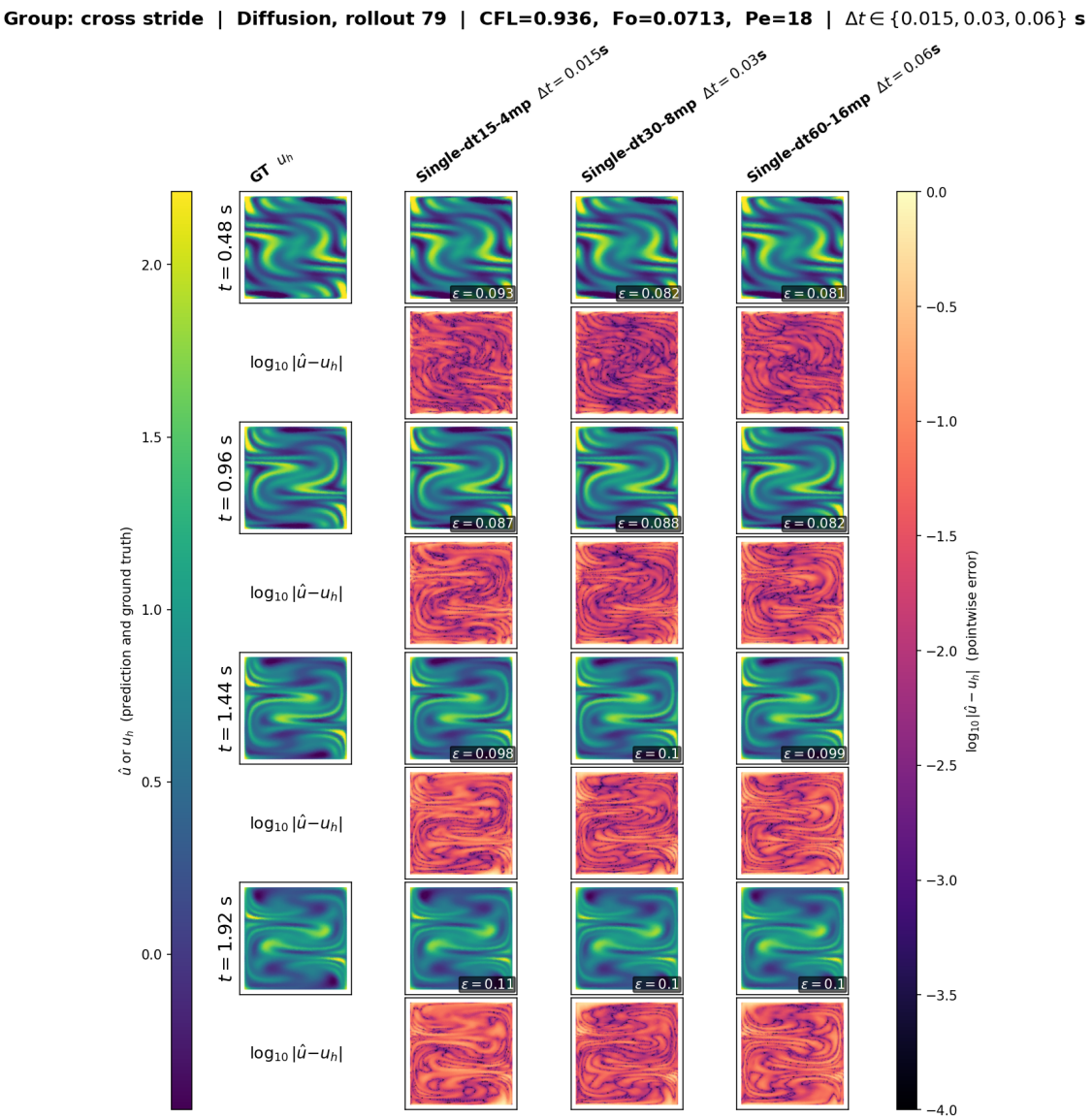


Figure A.4: Solution field snapshots for the single-scale models across surrogate time steps (Single-dt15-4mp, Single-dt30-8mp, Single-dt60-16mp) on a diffusion-dominated test rollout. Layout as in Figure 5.10.

A.4 Hyperparameter study

A hyperparameter study was conducted to investigate the sensitivity to changes in certain hyperparameters besides the number of message passing layers. In the main parts of the thesis, the number of message passing layers is investigated thoroughly enough and is therefore omitted in the hyperparameter study. For the hyperparameter study, we investigate the relation between performance and the following hyperparameters: encoding dimension, batch size, and learning rate. The study was not conducted on the full dataset, but rather 10% of the samples were taken per epoch to reduce training time, where we evaluate the performance after training for 10 epochs. The results of the hyperparameter study are shown in Tables A.17, A.18 and A.19 for encoding dimensions of 64, 128 and 256 respectively. It can be seen that the learning rate of 10^{-4} offers a good sweet spot, which is consistent across all encoding dimensions, and that the batch size does not have a significant influence on the performance. The best performance is achieved for an encoding dimension of 128, but the performance is not significantly different from the encoding dimension of 256. Therefore, the encoding dimension of 128 is chosen for the main experiments, as it offers a good balance between performance and computational cost. The accompanying learning rate and batch size are then chosen to be 10^{-4} and 5 respectively, as they offer the best performance for the encoding dimension of 128.

Table A.17: The L2-error $\mathscr{E}_{L^2}(u)$ and H1-semi norm $\mathscr{E}_{H^1}(u)$, relative to the true update, of the surrogate models after training on a subset of the training set for 10 epochs with the encoding dimension set to 64 and the number of message passing steps set to 4.

configuration index	learning rate	batch size	$\mathscr{E}_{L^2}(u)$	$\mathscr{E}_{H^1}(u)$
1	10^{-5}	4	0.813	1.778
2	10^{-5}	5	0.968	2.549
3	10^{-5}	6	0.898	2.293
4	10^{-4}	4	**0.151**	**0.483**
5	10^{-4}	5	0.203	0.487
6	10^{-4}	6	0.222	0.508
7	10^{-3}	4	1.034	0.995
8	10^{-3}	5	1.041	0.992
9	10^{-3}	6	1.062	0.997

Table A.18: The L2-error $\mathscr{E}_{L^2}(u)$ and H1-semi norm $\mathscr{E}_{H^1}(u)$, relative to the true update, of the surrogate models after training on a subset of the training set for 10 epochs with the encoding dimension set to 128 and the number of message passing steps set to 4.

configuration index	learning rate	batch size	$\mathscr{E}_{L^2}(u)$	$\mathscr{E}_{H^1}(u)$
10	10^{-5}	4	0.417	0.877
11	10^{-5}	5	0.437	0.954
12	10^{-5}	6	0.484	1.01
13	10^{-4}	4	0.125	**0.381**
14	10^{-4}	5	**0.116**	0.382
15	10^{-4}	6	0.128	0.438
16	10^{-3}	4	1.287	0.978
17	10^{-3}	5	1.060	1.00
18	10^{-3}	6	1.009	0.977

Table A.19: The L2-error $\mathscr{E}_{L^2}(u)$ and H1-semi norm $\mathscr{E}_{H^1}(u)$, relative to the true update, of the surrogate models after training on a subset of the training set for 10 epochs with the encoding dimension set to 256 and the number of message passing steps set to 4.

configuration index	learning rate	batch size	$\mathscr{E}_{L^2}(u)$	$\mathscr{E}_{H^1}(u)$
19	10^{-5}	4	0.275	0.721
20	10^{-5}	5	0.239	0.629
21	10^{-5}	6	0.275	0.721
22	10^{-4}	4	0.130	**0.284**
23	10^{-4}	5	**0.127**	0.292
24	10^{-4}	6	0.144	0.395
25	10^{-3}	4	1.090	1.014
26	10^{-3}	5	1.184	1.040
27	10^{-3}	6	1.116	1.032

A.5 Convergence study

The convergence of the ground truth finite element simulations is verified by performing 100 random simulations at different mesh sizes. We copied the same simulation settings from the experimental setting and made one reference simulation, which was run at a really fine structured mesh size of $h_{ref} = 0.001$. The random simulations were compared to this reference solution, as we computed the error at $t = 1.92$ against this reference simulation. The integrals for the L2-errors were performed on the reference mesh, meaning that the solutions of the random simulations were interpolated to the reference mesh before the error was computed. The results can be seen in Figure A.5, where it is shown that the error decreases as the mesh size decreases, indicating that the simulations are converging to the reference solution. For higher diffusion values the convergence is faster, which is expected as the solution becomes smoother. It may seem like the low-diffusion simulations are performing poorly, but this is because the solution has a lot of boundary layers which are not resolved at the coarser mesh sizes. And since the error is computed on the reference mesh, the error is amplified as the coarser simulations cannot 'see' these boundary layers at all.

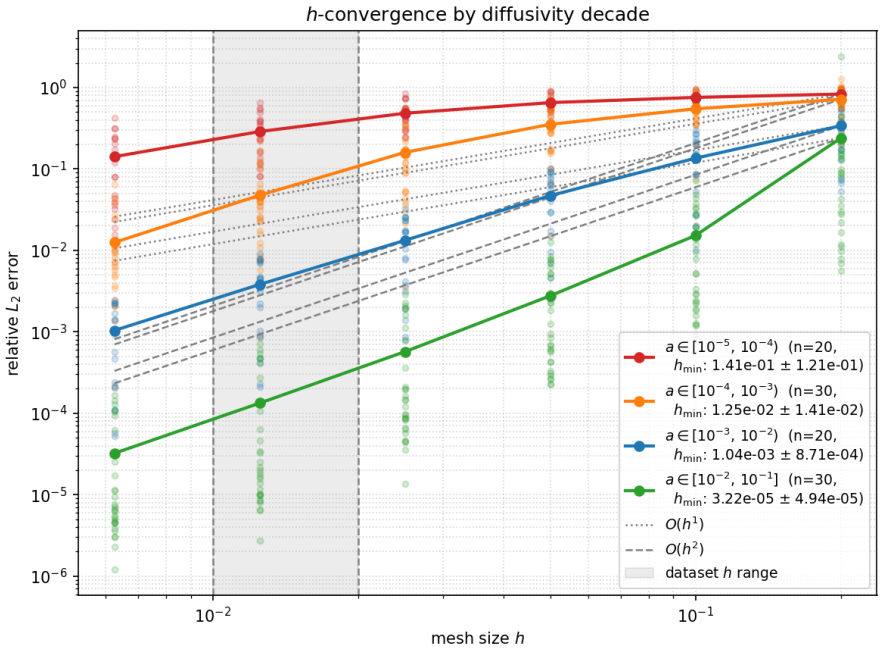


Figure A.5: The convergence of the finite element simulations is verified by performing 100 random simulations at different mesh sizes and comparing them to a reference simulation at a very fine mesh size. The error is computed at $t = 1.92$ against the reference solution, and the integrals for the L2-errors are performed on the reference mesh. The plot shows that the error decreases as the mesh size decreases, indicating that the simulations are converging to the reference solution.

A.6 Conventional numerical time integrators performance plots

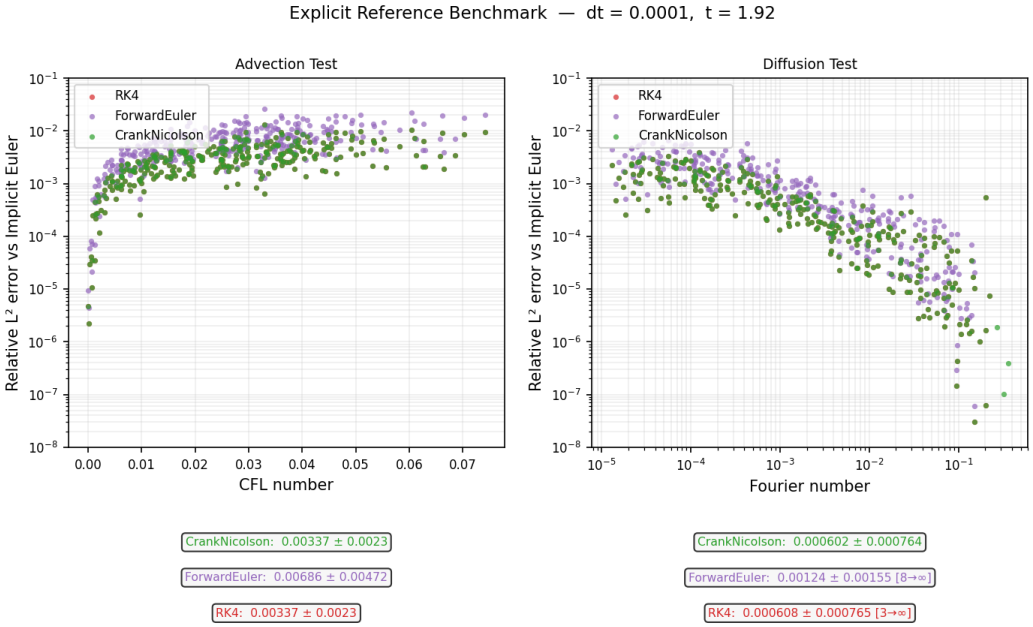
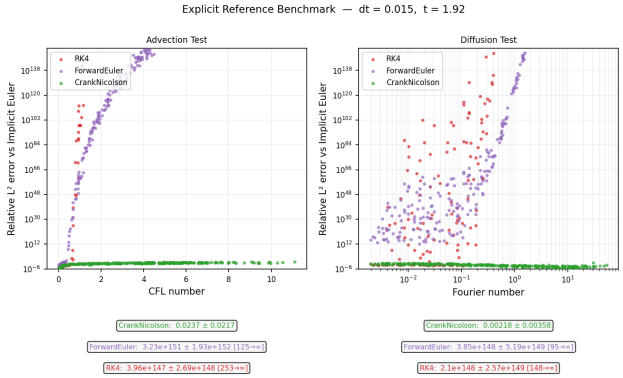
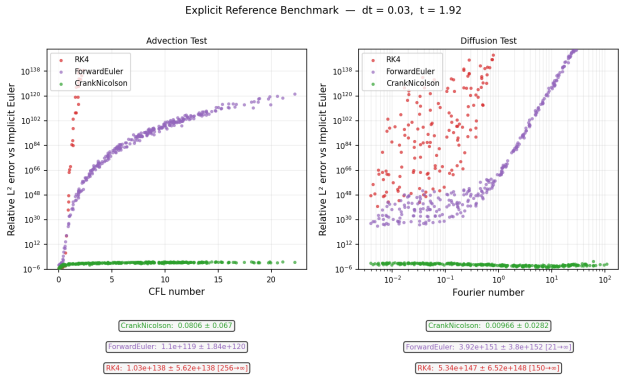


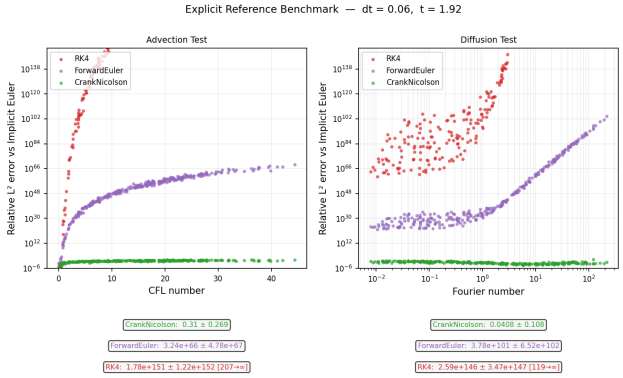
Figure A.6: The performance of the numerical time integrators forward Euler, RK4 and Crank Nicolson is evaluated on the test set with a timestep size $\Delta t = 0.0001$ and a rollout horizon of $T = 1.92$.



(a) $\Delta t = 0.015$



(b) $\Delta t = 0.030$



(c) $\Delta t = 0.060$

Figure A.7: Performance of the numerical time integrators forward Euler, RK4 and Crank Nicolson is evaluated on the test set with different timestep sizes and a rollout horizon of $T = 1.92$.

Bibliography

[1] O. Bouhali, I.G. Economou, and F. El-Mellouhi. “The International Computational Science and Engineering Conference”. In: Journal of Computational Science 15 (July 2016), pp. 1–2. issn: 18777503. doi: 10.1016/j.jocs.2016.02.004. url: https://linkinghub.elsevier.com/retrieve/pii/S1877750316000089 (visited on 06/03/2026).

[2] Mats G. Larson and Fredrik Bengzon. The Finite Element Method: Theory, Implementation, and Applications. Vol. 10. Texts in Computational Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. isbn: 978-3-642-33286-9 978-3-642-33287-6. doi: 10.1007/978-3-642-33287-6. url: https://link.springer.com/10.1007/978-3-642-33287-6 (visited on 03/11/2025).

[3] Olgierd C. Zienkiewicz, Robert L. Taylor, and Jianzhong Zhu. The Finite Element Method: Its Basis and Fundamentals. 6. ed., reprint., transferred to digital print. Amsterdam Heidelberg: Elsevier, 2010. 733 pp. isbn: 978-0-7506-6320-5.

[4] Michael T Heath. Scientific Computing: An Introductory Survey, Revised Second Edition. SIAM, 2018. isbn: 1-61197-557-3.

[5] Thomas Sterling, Maciej Brodowicz, and Matthew Anderson. High Performance Computing: Modern Systems and Practices. Morgan Kaufmann, 2017. isbn: 0-12-420215-2.

[6] Roberto Bentivoglio et al. “Rapid Spatio-Temporal Flood Modelling via Hydraulics-Based Graph Neural Networks”. In: Hydrology and Earth System Sciences 27.23 (Nov. 30, 2023), pp. 4227–4246. issn: 1607-7938. doi: 10.5194/hess-27-4227-2023. url: https://hess.copernicus.org/articles/27/4227/2023/ (visited on 02/28/2025).

[7] Tobias Pfaff et al. Learning Mesh-Based Simulation with Graph Networks. June 18, 2021. doi: 10.48550/arXiv.2010.03409. arXiv: 2010.03409 [cs]. url: http://arxiv.org/abs/2010.03409 (visited on 02/24/2025). Pre-published.

[8] Jonathan Tompson et al. Accelerating Eulerian Fluid Simulation With Convolutional Networks. Version 7. 2016. doi: 10.48550/ARXIV.1607.03597. url: https://arxiv.org/abs/1607.03597 (visited on 06/04/2026). Pre-published.

[9] Ian Goodfellow. Deep Learning. MIT press, 2016.

[10] Meire Fortunato et al. MultiScale MeshGraphNets. Oct. 2, 2022. doi: 10.48550/arXiv.2210.00612. arXiv: 2210.00612 [cs]. url: http://arxiv.org/abs/2210.00612 (visited on 02/28/2025). Pre-published.

[11] Roberto Bentivoglio et al. Multi-Scale Hydraulic Graph Neural Networks for Flood Modelling. Sept. 20, 2024. doi: 10.5194/egusphere-2024-2621. url: https://egusphere.copernicus.org/preprints/2024/egusphere-2024-2621/ (visited on 02/28/2025). Pre-published.

[12] (PDF) Multi-scale Rotation-Equivariant Graph Neural Networks for Unsteady Eulerian Fluid Dynamics. url: https://www.researchgate.net/publication/362150802_Multi-scale_rotation-equivariant_graph_neural_networks_for_unsteady_Eulerian_fluid_dynamics (visited on 02/28/2025).

[13] Filipe de Avila Belbute-Peres, Thomas D Economon, and J Zico Kolter. “Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction”. In: (). url: https://arxiv.org/abs/2007.04439.

[14] Joel H Ferziger, Milovan Perić, and Robert L Street. Computational Methods for Fluid Dynamics. Vol. 3. Springer, 2002. isbn: 3-540-42074-6.

[15] Nakhlé H Asmar. Partial Differential Equations with Fourier Series and Boundary Value Problems. Courier Dover Publications, 2016. isbn: 0-486-80737-1.

[16] David J Griffiths. Introduction to Electrodynamics. Cambridge University Press, 2023. isbn: 1-009-39775-3.

[17] David J Griffiths and Darrell F Schroeter. Introduction to Quantum Mechanics. Cambridge university press, 2018. isbn: 1-108-10034-1.

[18] An Analytical Solution for the Advection-Dispersion Equation Inversely in Time for Pollution Source Identification - ScienceDirect. url: https://www.sciencedirect.com/science/article/abs/pii/S1474706522001486 (visited on 04/23/2026).

[19] Jeevan Kafle, Krishna Adhikari, and Eeshwar Poudel. "Air Pollutant Dispersion Using Advection-Diffusion Equation". In: Nepal Journal of Environmental Science 12 (July 2024), pp. 1–6. doi: 10.3126/njes.v12i1.47531.

[20] Ankur Jain et al. "Theoretical Model for Diffusion-Reaction Based Drug Delivery from a Multilayer Spherical Capsule". In: International Journal of Heat and Mass Transfer 183 (Feb. 1, 2022), p. 122072. issn: 0017-9310. doi: 10.1016/j.ijheatmasstransfer.2021.122072. url: https://www.sciencedirect.com/science/article/pii/S0017931021011789 (visited on 04/23/2026).

[21] Harry Van den Akker and R.F. Mudde. Fysische Transportverschijnselen - Denken in Balansen. Aug. 1, 2014. isbn: 978-90-6562-357-7. doi: 10.5074/T.2023.002.

[22] "Computational Modeling of Diffusion in the Cerebellum". In: Progress in Molecular Biology and Translational Science. Vol. 123. Academic Press, Jan. 1, 2014, pp. 169–189. doi: 10.1016/B978-0-12-397897-4.00007-3. url: https://www.sciencedirect.com/science/chapter/bookseries/abs/pii/B9780123978974000073 (visited on 04/23/2026).

[23] Pavel Ŝolín. Partial Differential Equations and the Finite Element Method. John Wiley & Sons, Dec. 13, 2005. 505 pp. isbn: 978-0-471-76409-0. Google Books: AgWFqIKbeBUC.

[24] Dmitri Kuzmin. "A Guide to Numerical Methods for Transport Equations". In: (2010).

[25] Sybren Ruurds De Groot and Peter Mazur. Non-Equilibrium Thermodynamics. Courier Corporation, 2013. isbn: 0-486-15350-9.

[26] Thomas J. R. Hughes. The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Courier Corporation, Jan. 1, 2003. 706 pp. isbn: 978-0-486-41181-1. Google Books: E9IoAwAAQBAJ.

[27] Jacques Hadamard. "Sur Les Problèmes Aux Dérivées Partielles et Leur Signification Physique". In: Princeton university bulletin (1902), pp. 49–52.

[28] Mathematical Physics with Partial Differential Equations. Elsevier, 2018. isbn: 978-0-12-814759-7. doi: 10.1016/C2016-0-03724-5. url: https://linkinghub.elsevier.com/retrieve/pii/C20160037245 (visited on 04/24/2026).

[29] Raphaël Lecoq. "Banach-Nečas-Babuška Theorem and Proof". In: (2024).

[30] C. Vuik et al. Numerical Methods for Ordinary Differential Equations. Second edition. Delft: DAP, Delft Academic Press, 2015. isbn: 978-90-6562-373-7.

[31] Jean Donea and Antonio Huerta. Finite Element Methods for Flow Problems. 1st ed. Wiley, Apr. 11, 2003. isbn: 978-0-471-49666-3 978-0-470-01382-3. doi: 10.1002/0470013826. url: https://onlinelibrary.wiley.com/doi/book/10.1002/0470013826 (visited on 04/25/2026).

[32] 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 (Sept. 1, 1982), pp. 199–259. issn: 0045-7825. doi: 10.1016/0045-7825(82)90071-8. url: https://www.sciencedirect.com/science/article/pii/0045782582900718 (visited on 07/15/2025).

[33] "Multigrid by U. Trottenberg; C. Oosterlee; A. Schüller | Request PDF". In: ResearchGate (Oct. 22, 2024). doi: 10.2307/4148428. url: https://www.researchgate.net/publication/261977719_Multigrid_by_U_Trottenberg_C_Oosterlee_A_Schuller (visited on 03/26/2025).

[34] J. W. Ruge and K. Stüben. "4. Algebraic Multigrid". In: Multigrid Methods. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 1987, pp. 73–130. isbn: 978-1-61197-188-0. doi: 10.1137/1.9781611971057.ch4. url: https://epubs.siam.org/doi/abs/10.1137/1.9781611971057.ch4 (visited on 04/22/2026).

[35] Robert D Falgout. An Introduction to Algebraic Multigrid. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2006.

[36] Pieter Wesseling. Introduction to Multigrid Methods. 1995.

[37] Yousef Saad. Iterative Methods for Linear Systems of Equations: A Brief Historical Journey. arXiv.org. Aug. 2, 2019. url: https://arxiv.org/abs/1908.01083v1 (visited on 06/06/2026).

[38] K. Stüben. "A Review of Algebraic Multigrid". In: Journal of Computational and Applied Mathematics. Numerical Analysis 2000. Vol. VII: Partial Differential Equations 128.1 (Mar. 1, 2001), pp. 281–309. issn: 0377-0427. doi: 10.1016/S0377-0427(00)00516-1. url: https://www.sciencedirect.com/science/article/pii/S0377042700005161 (visited on 04/22/2026).

[39] Multi-Scale Rotation-Equivariant Graph Neural Networks for Unsteady Eulerian Fluid Dynamics | Physics of Fluids | AIP Publishing. url: https://pubs.aip.org/aip/pof/article/34/8/087110/2847850/Multi-scale-rotation-equivariant-graph-neural (visited on 02/28/2025).

[40] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: IEEE Transactions on Neural Networks and Learning Systems 32.1 (Jan. 2021), pp. 4–24. issn: 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2978386. arXiv: 1901.00596 [cs]. url: http://arxiv.org/abs/1901.00596 (visited on 03/14/2025).

[41] Boris Kovalerchuk and Elijah McCoy. Explainable Machine Learning for Categorical and Mixed Data with Lossless Visualization. Nov. 23, 2023. doi: 10.48550/arXiv.2305.18437. arXiv: 2305.18437 [cs]. url: http://arxiv.org/abs/2305.18437 (visited on 04/14/2026). Pre-published.

[42] Tobias Pfaff et al. Learning Mesh-Based Simulation with Graph Networks. June 18, 2021. doi: 10.48550/arXiv.2010.03409. arXiv: 2010.03409 [cs]. url: http://arxiv.org/abs/2010.03409 (visited on 11/29/2024). Pre-published.

[43] William L Hamilton. "Graph Representation Learning". In: ().

[44] Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition. url: https://hastie.su.domains/ElemStatLearn/ (visited on 04/14/2026).

[45] 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 [cs]. url: http://arxiv.org/abs/1609.02907 (visited on 02/24/2025). Pre-published.

[46] HESS - Rapid Spatio-Temporal Flood Modelling via Hydraulics-Based Graph Neural Networks. url: https://hess.copernicus.org/articles/27/4227/2023/ (visited on 04/14/2026).

[47] 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 [cs]. url: http://arxiv.org/abs/2002.09405 (visited on 02/24/2025). Pre-published.

[48] Justin Gilmer et al. Neural Message Passing for Quantum Chemistry. June 12, 2017. doi: 10.48550/arXiv.1704.01212. arXiv: 1704.01212 [cs]. url: http://arxiv.org/abs/1704.01212 (visited on 02/27/2025). Pre-published.

[49] Peter W. Battaglia et al. Interaction Networks for Learning about Objects, Relations and Physics. Dec. 1, 2016. doi: 10.48550/arXiv.1612.00222. arXiv: 1612.00222 [cs]. url: http://arxiv.org/abs/1612.00222 (visited on 02/25/2025). Pre-published.

[50] Nicholas S. Moore et al. Graph Neural Networks and Applied Linear Algebra. Oct. 21, 2023. doi: 10.48550/arXiv.2310.14084. arXiv: 2310.14084 [math]. url: http://arxiv.org/abs/2310.14084 (visited on 03/15/2025). Pre-published.

[51] Shahaf E. Finder et al. Improving the Effective Receptive Field of Message-Passing Neural Networks. May 29, 2025. doi: 10.48550/arXiv.2505.23185. arXiv: 2505.23185 [cs]. url: http://arxiv.org/abs/2505.23185 (visited on 04/15/2026). Pre-published.

[52] Wenkai Zhang et al. "Convolutional Neural Networks-Based Surrogate Model for Fast Computational Fluid Dynamics Simulations of Indoor Airflow Distribution". In: Energy and Buildings 326 (Jan. 2025), p. 115020. issn: 03787788. doi: 10.1016/j.enbuild.2024.115020. url: https://linkinghub.elsevier.com/retrieve/pii/S0378778824011368 (visited on 04/15/2026).

[53] Matthias Eichinger, Alexander Heinlein, and Axel Klawonn. "Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations". In: ETNA - Electronic Transactions on Numerical Analysis 56 (2022), pp. 235–255. issn: 1068-9613, 1068-9613. doi: 10.1553/etna_vol56s235. url: https://hw.oeaw.ac.at?arp=0x003d4c21 (visited on 04/15/2026).

[54] Keqin Wang et al. Resolving Oversmoothing with Opinion Dissensus. May 16, 2025. doi: 10.48550/arXiv.2501.19089. arXiv: 2501.19089 [cs.LG]. url: http://arxiv.org/abs/2501.19089 (visited on 06/02/2026). Pre-published.

[55] Michael Scholkemper et al. "Residual Connections and Normalization Can Provably Prevent Oversmoothing in GNNs". In: The Thirteenth International Conference on Learning Representations. Oct. 4, 2024. url: https://openreview.net/forum?id=i8vPRlsrYu (visited on 04/14/2026).

[56] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. July 21, 2016. doi: 10.48550/arXiv.1607.06450. arXiv: 1607.06450 [stat]. url: http://arxiv.org/abs/1607.06450 (visited on 04/16/2026). Pre-published.

[57] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: Mathematics of Control, Signals and Systems 2.4 (Dec. 1, 1989), pp. 303–314. issn: 1435-568X. doi: 10.1007/BF02551274. url: https://doi.org/10.1007/BF02551274 (visited on 04/18/2026).

[58] Michael M. Bronstein et al. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. May 2, 2021. doi: 10.48550/arXiv.2104.13478. arXiv: 2104.13478 [cs]. url: http://arxiv.org/abs/2104.13478 (visited on 05/13/2025). Pre-published.

[59] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message Passing Neural PDE Solvers. Mar. 20, 2023. doi: 10.48550/arXiv.2202.03376. arXiv: 2202.03376 [cs]. url: http://arxiv.org/abs/2202.03376 (visited on 02/03/2026). Pre-published.

[60] Edsger W Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: Edsger Wybe Dijkstra: His Life, Work, and Legacy. 2022, pp. 287–290.

[61] Joachim Schöberl. "C++11 Implementation of Finite Elements in NGSolve". In: (2014), pp. 1–23. url: https://repositum.tuwien.at/handle/20.500.12708/28346 (visited on 06/02/2026).

[62] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. Jan. 30, 2017. doi: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs]. url: http://arxiv.org/abs/1412.6980 (visited on 04/28/2026). Pre-published.

[63] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. "The Coefficient of Determination R-squared Is More Informative than SMAPE, MAE, MAPE, MSE and RMSE in Regression Analysis Evaluation". In: PeerJ Computer Science 7 (July 5, 2021), e623. issn: 2376-5992. doi: 10.7717/peerj-cs.623. url: https://peerj.com/articles/cs-623 (visited on 05/22/2026).

[64] Santiago Moreno-Carbonell and Eugenio F. Sánchez-Úbeda. "A Piecewise Linear Regression Model Ensemble for Large-Scale Curve Fitting". In: Algorithms 17.4 (Apr. 2024), p. 147. issn: 1999-4893. doi: 10.3390/a17040147. url: https://www.mdpi.com/1999-4893/17/4/147 (visited on 05/22/2026).

[65] Qing Chang and Max Goplerud. "Generalized Kernel Regularized Least Squares". In: Political Analysis 32.2 (Apr. 2024), pp. 157–171. issn: 1047-1987, 1476-4989. doi: 10.1017/pan.2023.27. arXiv: 2209.14355 [stat.ML]. url: http://arxiv.org/abs/2209.14355 (visited on 06/02/2026).

[66] Gilbert Strang. Linear Algebra and Learning from Data. Vol. 4. Wellesley-Cambridge Press Cambridge, 2019.

[67] Jianxin Wu. "Introduction to Convolutional Neural Networks". In: National Key Lab for Novel Software Technology. Nanjing University. China 5.23 (2017), p. 495.

[68] Aravind Vasudevan, Andrew Anderson, and David Gregg. Parallel Multi Channel Convolution Using General Matrix Multiplication. Version 2. 2017. doi: 10.48550/ARXIV.1704.04428. url: https://arxiv.org/abs/1704.04428 (visited on 12/05/2024). Pre-published.