Eberhard Karls Universität Tübingen

Mathematisch-Naturwissenschaftliche Fakultät
Bachelor Kognitionswissenschaft

Parallel-in-Time with the Fourier Neural Operator

A Study of Parareal Speed-Up as a Function of Training Quality

Bachelorarbeit

von

Clemens Seethaler

Matrikelnummer: 6388785

Gutachter: Prof. Dr. Philipp Hennig

Bearbeitungszeitraum: 26.05.2025 – 26.09.2025

Abgabedatum: **26 September 2025**

Ort: Tübingen

Erklärung

Laut Beschlüssen der Prüfungsausschüsse Bioinformatik, Informatik, Informatik Lehramt, Kognitionswissenschaft, Machine Learning, Medieninformatik und Medizininformatik der Universität Tübingen vom 05.02.2025. Gültig für Abschlussarbeiten (B.Sc./M.Sc./B.Ed./M.Ed.) in den zugehörigen Fächern. Bei Studienarbeiten und Hausarbeiten bitte nach Maßgabe des/der jeweiligen Prüfers/Prüferin.

1. Allgemeine Erklärungen

Hiermit erkläre ich:

- Ich habe die vorgelegte Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
- Ich habe alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet.
- Die Arbeit war weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens.
- Falls ich ein elektronisches Exemplar und eines oder mehrere gedruckte und gebundene Exemplare eingereicht habe (z.B., weil der/die Prüfer/in(nen) dies wünschen): Das elektronisch eingereichte Exemplar stimmt exakt mit dem bzw. den von mir eingereichten gedruckten und gebundenen Exemplar(en) überein.

2. Erklärung bezüglich Veröffentlichungen

Eine Veröffentlichung ist häufig ein Qualitätsmerkmal (z.B. bei Veröffentlichung in Fachzeitschrift, Konferenz, Preprint, etc.). Sie muss aber korrekt angegeben werden. Bitte kreuzen Sie die für Ihre Arbeit zutreffende Variante an:

☒ Die Arbeit wurde bisher weder vollständig noch in Teilen veröffentlicht.

☐ Die Arbeit wurde in Teilen oder vollständig schon veröffentlicht. Hierfür findet sich im Anhang eine vollständige Tabelle mit bibliographischen Angaben.

3. Nutzung von Methoden der künstlichen Intelligenz (KI, z.B. chatGPT, DeepL, etc.)

Die Nutzung von KI kann sinnvoll sein. Sie muss aber korrekt angegeben werden und kann die Schwerpunkte bei der Bewertung der Arbeit beeinflussen. Bitte kreuzen Sie alle für Ihre Arbeit zutreffenden Varianten an und beachten Sie, dass die Varianten 3.4 - 3.6 eine vorherige Absprache mit dem/der Betreuer/in voraussetzen:

☐ 3.1. Keine Nutzung: Ich habe zur Erstellung meiner Arbeit keine KI benutzt.

☒ 3.2. Korrektur Rechtschreibung & Grammatik: Ich habe KI für Korrekturen der Rechtschreibung und Grammatik genutzt, ohne dass es dabei zu inhaltlich relevanter Textgeneration oder Übersetzungen kam. Das heißt, ich habe von mir verfasste Texte in derselben Sprache korrigieren lassen. Es handelt sich um rein sprachliche Korrekturen, sodass die von mir ursprünglich intendierte Bedeutung nicht wesentlich verändert oder erweitert wurde. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☒ 3.3. Unterstützung bei der Softwareentwicklung: Ich habe KI als Unterstützung beim Schreiben von Code in der Softwareentwicklung genutzt. Es handelt sich hierbei lediglich um Unterstützung und nicht um die automatische Generierung von größeren Programm-Teilen. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☐ 3.4. Übersetzung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Übersetzung von mir in einer anderen Sprache geschriebenen Texte genutzt. Jede derartige Übersetzung ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller übersetzten Textstellen und der verwendeten Programme mit Versionsnummer.

☐ 3.5. Code-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Code in der Softwareentwicklung genutzt. Der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

☐ 3.6. Text-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Text in meiner Arbeit genutzt. Jede derartige Verwendung von KI ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

Falls ich in irgendeiner Form KI genutzt haben (siehe oben), dann erkläre ich:

Mir ist bewusst, dass ich die Verantwortung trage, falls es durch die Verwendung von KI zu fehlerhaften Inhalten, zu Verstößen gegen das Datenschutzrecht, Urheberrecht oder zu wissenschaftlichem Fehlverhalten (z.B. Plagiaten) kommt.

4. Abschluss und Unterschrift(en)

Mir ist bekannt, dass ein Verstoß gegen diese Erklärung prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass die Prüfungsleistung mit „nicht ausreichend“ bzw. die Studienleistung mit „nicht bestanden“ bewertet wird und bei mehrfachem oder schwerwiegendem Täuschungsversuch eine Exmatrikulation erfolgen bzw. ein Verfahren zur Entziehung eines eventuell verliehenen akademischen Titels eingeleitet werden kann.

Clemens Seethaler	Tübingen, 26.09.2025	[illegible]
Vorname, Nachname Student/in	Ort, Datum	Unterschrift

Die Punkte 3.4 - 3.6 erfordern eine Zustimmung des/r Betreuer/in. Sollten Sie einen dieser Punkte angekreuzt haben, dann sollte der/die Betreuer/in bitte hier unterschreiben:

Ich habe der oben genannten Nutzung von KI zur Erstellung der Arbeit zugestimmt.

Vorname, Nachname Betreuer/in	Ort, Datum	Unterschrift

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Context

Consider a student who is trying to solve mathematical problems by guessing the solution. Each time they check for the correct solution of a mathematical operation, they learn from their mistakes. With each round of practice, they get better, until every step is performed correctly, and future problems take less time.

This practice of breaking down a mathematical problem into smaller ones can be described algorithmically and a common practice in engineering and scientific research. It facilitates computationally intensive numerical calculations, as it allows for an efficient problem-solving in parallel. Since many scientific and engineering problems can be expressed in the form of *partial differential equations*, research has come up with concepts for efficiently solving them (M. Gander & Halpern, 2007; Lions et al., 2001; Saltz & Naik, 1988).

A prominent one is *Parareal* proposed by Lions et al. (2001). While some approaches perform parallelization across the spatial domains, Parareal is an in-time discretization scheme that computes numerical solutions for differential equations by parallelization of multiple time steps. This is achieved by iteratively refining a coarse initial guess until converged (Lions et al., 2001). Its parallelization potential is utilized in physical simulations that require High-Performance Computing calculations such as molecular dynamics (Legoll et al., 2022).

In order to provide an advantage through parallelization with Parareal, the quality of the coarse initial guess to the solution is crucial to reduce the number of iterative refinements (M. J. Gander & Vandewalle, 2007; Minion, 2010). In the original publication of Parareal, the initial guess is obtained by a less-accurate numerical solver. However, recent research has shown potential in using machine-learning methods instead, where the coarse solver is replaced with a surrogate model trained on the specific problem (Gorynina et al., 2022). In particular, the Fourier Neural Operator proposed by Li et al. (2021) has shown potential for time-in-parallel application for the simulation of specific problems (Ibrahim et al., 2024; Pamela et al., 2024).

1.2 Problem Statement and Research Questions

In the context of physical simulations, Ibrahim et al. (2024) and Pamela et al. (2024) have successfully conducted several numerical experiments, showing the acceleration of simulations using this hybrid approach. However, fundamental literature on this novel approach is quite sparse and the conditions under which it is beneficial remain unclear. This leaves unanswered whether the combination of Parareal and the latest machine-learning methods provides advantages in general or may be limited to specific contexts.

This thesis follows the approach of Ibrahim et al. (2024) and Pamela et al. (2024), training a Fourier Neural Operator to serve as a coarse solver within the Parareal framework. The objective is to determine how quality of the model influences the overall performance. Therefore, the quality of the FNO is systematically varied by adjusting the two key training hyperparameters, namely the size of the training dataset and the number of training epochs. Our investigation addresses convergence efficiency and computational speed-up, which allows for studying the following questions in a small-scale experiment:

- How does the quality of the FNO coarse solver, specifically as a function of training data size and number of epochs, influence the convergence rate and accuracy of the Parareal algorithm?
- To what extent does a high-quality FNO coarse solver provide a computational speed-up for Parareal simulations, and how does it scale with the number of Parareal intervals?

First, we establish a common background for the problems under study. Sections 2.1 and 2.2 therefore explain the concept of differential equations, followed by methods for solving them. Section 2.3 introduces the Parareal algorithm as proposed by Lions et al. (2001). In section 2.4, the necessary fundamentals of Neural Networks are outlined and the concept of neural operators, including the Fourier Neural Operator, is explained. The chapter concludes by discussing the related work.

Chapter 3 describes the methodology of our approach. Based on our research questions, we formulate hypotheses for our experiments in section 3.1. We then outline the foundation of our setup and the problems we examined in section 3.2. Section 3.3 is dedicated to the training pipeline of the Fourier Neural Operator Model and its integration into the Parareal algorithm. In section 3.4, we provide a detailed view on how metrics were recorded and experiments are evaluated. Finally, we outline sanity checks taken prior to the conduction of our experiment in section 3.5.

The next chapter highlights our first experiment, where the impact of the models quality on Parareal convergence was investigated. Section 4.1 provides results of the trained models. Section 4.2 provides the results of our simulations, including statistical analyses. In section 4.3, we examined the occurrence of outliers and assessed how they emerged. Section 4.4 discusses the experimental results, after which we state limitations of our approach.

Chapter 5 presents the findings of our second, smaller-scale experiment. The chapter first establishes a theoretical framework, beginning with a derivation of how boundaries are handled in section 5.1, before presenting the empirical results in section 5.2. Following sections 5.3 and 5.4 discuss the findings and limitations of our approach.

Lastly, we provide an overall conclusion drawn from our findings in chapter 6. We refer back to our research questions, then point to future work that strengthens our conclusions.

In the appendix, we provide additional figures and tables, along with implementation conditions under which our experiments were performed.

2 Background

2.1 Differential Equations

Rather than focusing on absolute states, the more intriguing question often is how we can describe in which way those states naturally change over space and time. One way to do so is offered by the mathematical concept of differential equations, which provides a way to express changes within multidimensional systems (Evans, 2010; LeVeque, 2007). Unless otherwise noted, the definitions in this chapter follow LeVeque (2007) and Evans (2010).

Domains of the system. Let $d \in \{1, 2\}$. The spatial domain is the periodic box $\Omega = [0, L_{spatial}]^d$ with periodic identification of the opposite faces. The time interval is $[0, T]$. The unknowns are functions $u = u(x, t)$, where $x \in \Omega, t \in [0, T]$.

Differential operators. $\frac{\partial u}{\partial t}$ equals the derivative of u with respect to time t, ∇u is spatial gradient, and $\Delta u = \nabla \cdot \nabla u$ is the Laplacian. For $d = 1$, $\nabla \cdot \nabla u = \frac{\partial^2 u}{\partial x^2}$.

This system describes the state of every individual point in space, and how this state will change over time. It requires an initial condition. For ordinary differential equations (ODEs), this is called an initial value problem.

Ordinary Differential Equations, Initial Value Problem. For $f : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$,

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y(t)), \quad y(t_0) = y_0. \tag{2.1}$$

Picard-Lindelöf Theorem. If f is continuous in time t and locally Lipschitz in y (i.e., the function's rate of change is bounded by its inputs rate of change), a unique solution exists on some interval about t_0 (Hairer et al., 1993),

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\mathrm{d}s. \tag{2.2}$$

With more than one independent variable, derivatives are taken with respect to each input separately, leading to partial differential equations (PDEs). For PDEs, the initial value is no longer a scalar but a continuous function over the spatial domain.

Partial Differential Equations. A PDE is of the form

$$F(x, t, u, \nabla u, \Delta u, ...) = 0 \tag{2.3}$$

with $x \in \Omega$, $t \in [0, T]$, and $u = u(x, t)$ being an unknown function.

Linearity. The PDE is linear if it is linear in u and all its derivatives. Otherwise it is non-linear.

A classical example of a linear PDE is the heat equation (also called the diffusion equation). It has been derived from Newton's law of energy conservation and expresses the process of how a particle passes energy to the neighbouring ones.

1D Diffusion Equation.

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}, \quad d = 1, x \in \Omega, t \in [0, T], u(x, 0) = u_0(x), \text{ periodic.} \tag{2.4}$$

PDEs can be extended over spatial domains and with additional terms. A two-dimensional linear scenario which introduces advection speed is the 2D Advection-Diffusion equation.

2D Advection-Diffusion Equation.

$$\frac{\partial u}{\partial t} + \mathbf{c} \cdot \nabla u = \nu \Delta u, \quad d = 2, x \in \Omega, t \in [0, T], u(x, 0) = u_0(x), \text{ periodic.} \tag{2.5}$$

In contrast to these linear equations, the Burgers' equation is considered as a non-linear PDE. Here, the unknown function u is multiplied with one of its derivatives, $\frac{\partial u}{\partial x}$. This non-linearity is capable of encoding important physical phenomena such as the steepening of waves and the formation of shock waves.

1D Burgers' Equation.

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad d = 1, x \in \Omega, t \in [0, T], u(x, 0) = u_0(x), \text{ periodic.} \tag{2.6}$$

In practice, the initial condition of partial differential equations is defined over a finite domain, requiring for explicit handling of its behaviour at its boundaries. Periodic boundary conditions prevent boundary fluxes for initial conditions of a periodic nature. Periodicity removes the artificial boundary handling and supports the Fourier/spectral methods that are introduced later.

Periodic boundary conditions. u is periodic over $L_{spatial}$ if $u(0, t) = u(L_{spatial}, t)$ for a $t \in [0, T]$.

Periodic initial value problem. Given a periodic initial condition u_0, choose the periodic u that solves the PDE with $u(x, 0) = u_0(x)$.

PDEs most often cannot be solved analytically, but with a numerical approximation instead. This assumes the well-posedness of the initial condition.

Well-posedness. An initial value problem is well-posed, if solutions (i) exist, (ii) are unique, and (iii) depend continuously on data. Solving a PDE in the classical sense means showing that a solution exists that satisfies these conditions.

2.2 Numerical Methods for PDEs

There exist different approaches for solving partial differential equations. While finite element methods divide the domain into smaller sub-domains to be solved, finite difference methods apply spatial discretization, thereby yielding a large system of ODEs. This approach is often referred to as method of lines. With infinite degrees of precision possible, the size of the discretization grid is directly related to the numerical accuracy of the solution and the computational effort needed for solving (Hairer et al., 1993; LeVeque, 2007). Unless stated otherwise, definitions within this section follow LeVeque (2007).

Method of lines. Let $\Delta x = L/N_x$, $x_i = i\Delta x$ with $i \in \mathbb{Z}_{N_x} := \{0, ..., N_x - 1\}$ being indexed periodically over space $U_{i+N_x} = U_i$, and unknowns $U_i \approx u(x_i, t)$. The spatial discretization yields a system of ODEs of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}U(t) = F(U(t)), \quad [F(U)]_i := D_x[U]_i, \tag{2.7}$$

where the discretization operator D_x denotes the semi-discrete right-hand side of the PDE.

For the one dimensional diffusion equation, a method of lines discretization would yield

$$\frac{\mathrm{d}}{\mathrm{d}t}U_i(t) = \nu\frac{U_{i+1} - 2U_i + U_{i-1}}{\Delta x^2}, \quad i \in \mathbb{Z}_{N_x}.$$

Using the Picard-Lindelöf theorem shown in Equation 2.2, the initial value problem becomes valid for time stepping (Hairer et al., 1993). The solution at a later time point is computed successively based on the value at the previous time point, thereby moving forward along the temporal domain. Time stepping methods are classified by their order of accuracy, which describes the numerical error introduced with each step.

Euler's Method. The solution at $t^{n+1} = t^n + \Delta t$ is derived from the previous value by replacing $\frac{\mathrm{d}u}{\mathrm{d}t}$, such that

$$\frac{U^{n+1} - U^n}{\Delta t} = F(U^n), \quad U^{n+1} = U^n + \Delta t F(U^n). \tag{2.8}$$

Multi-stage methods, Runge-Kutta. The solution at t^{n+1} is computed using multiple intermediate solutions U^*. The explicit Runge-Kutta method is given by

$$\tilde{U} = U^n + \frac{1}{2}\Delta t F(U^n), \quad U^{n+1} = U^n + \Delta t F(\tilde{U}), \tag{2.9}$$

while the intermediate value is computed using Euler's method.

Truncation errors. For a one-step method $U^{n+1} = \Psi_{\Delta t}(U^n)$, the local truncation error introduced with every step is given by $\tau^{n+1} = (u(t^{n+1}) - \Psi_{\Delta t}(u(t^n)))/\Delta t$.

The local truncation error assumes that the solution at the beginning of the step was exact. Inaccuracy introduces an accumulated global error that increases with every step until the last time point. The local truncation error may be reduced using higher order accuracy methods or implicit methods.

Explicit and implicit methods. Explicit methods compute U^{n+1} directly from known values. Implicit methods define U^{n+1} by an equation to solve, allowing for a larger Δt while increasing computational effort (Hairer et al., 1993; LeVeque, 2007).

One property of differential equations is stiffness. Stiffness arises when the linear part $\mathcal{L}$ has large negative eigenvalues, leading to severe step-size restrictions for numerical stability.

Stiffness. After spatial discretization, the problem of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}U = \mathcal{L}U + \mathcal{N}(U) \tag{2.10}$$

shows large negative eigenvalues for $\mathcal{L}$. Exponential methods integrate the stiff linear part and handle $\mathcal{N}$ explicitly.

For the numerical experiments of this work, an exponential time differencing Runge-Kutta method (ETDRK) is utilized solving semi-linear problems of the form given in Equation 2.10. The particular solver uses a Fourier-spectral method for spatial discretization, where the solution is expanded in Fourier space,

$$u(x,t) = \sum_{\kappa \in (2\pi/L)\mathbb{Z}^d} \hat{u}_\kappa(t) e^{\mathrm{i}\kappa \cdot x}. \tag{2.11}$$

The linear part $\mathcal{L}$ is diagonal in Fourier space and solved by applying scalar factors mode-wise in spectral space, while the non-linear part $\mathcal{N}$ is approximated using a multi-stage method. A more precise definition exceeds the scope of this section and is found in the work of Kassam and Trefethen (2005) and the publication of Montanelli and Bootland

(2020).

Partial differential equations with various system dynamics suffer from great stiffness and the effort of solving rises with non-linearity, the order of the PDE, the size of the equation system, the number of independent variables, and implicitness (Evans, 2010). Physical simulations often rely on specialized solvers and computationally intensive numerical calculations of such can take up to several hours, which motivates algorithmic optimization (Franck et al., 2015; Legoll et al., 2022; Rudi et al., 2015). One way is provided by parallelization techniques, accelerating numerical computations (M. Gander & Halpern, 2007; Legoll et al., 2022).

2.3 The Parareal Algorithm

Traditional time stepping methods are inherently sequential, with every computation being dependent on the result of the previous step and are not able to exploit today's multi-core computer architectures (Minion, 2010). The parallel-in-time application of iterative methods was considered already very early for ordinary differential equations (Saltz & Naik, 1988). Since a method of lines discretization of a partial differential equation yields a semi-discrete system of ODEs, in-time parallelization is also made possible for such: The Parareal algorithm proposed by Lions et al. (2001) creates a uniform time grid for an initial value problem, where time slices are solved in parallel using a coarse and a fine solver. The following definitions are derived from the work of Lions et al. (2001). Figure 1 illustrates the Parareal approximation of an ODE term of the form

$$\frac{\mathrm{d}y}{\mathrm{d}t} = 2(t - 0.5), \quad y_0 = 1.0.$$

Semi-discrete initial value problem. Let $U : [t_0, T] \to \mathbb{R}^m$ solve

$$\frac{\mathrm{d}U}{\mathrm{d}t} = F(U(t)), \quad U(t_0) = U^0, \tag{2.12}$$

and $N \in \mathbb{N}, t_{j+1} = t_j + \Delta t$ define the uniform time grid $\{t_j\}_{j=0}^{N}$ with $\Delta t = (T - t_0)/N$. Time slices are $[t_j, t_{j+1}]$ for $j = 0, ..., N - 1$.

Coarse and fine solver. Let

$$\mathcal{G}(t_j, t_{j+1}, \cdot), \mathcal{F}(t_j, t_{j+1}, \cdot) : \mathbb{R}^m \to \mathbb{R}^m \tag{2.13}$$

be maps, where $\mathcal{G}$ is a less-accurate solver with cost τ_{coarse} per slice, and $\mathcal{F}$ is a more-accurate solver with cost τ_{fine} per slice.

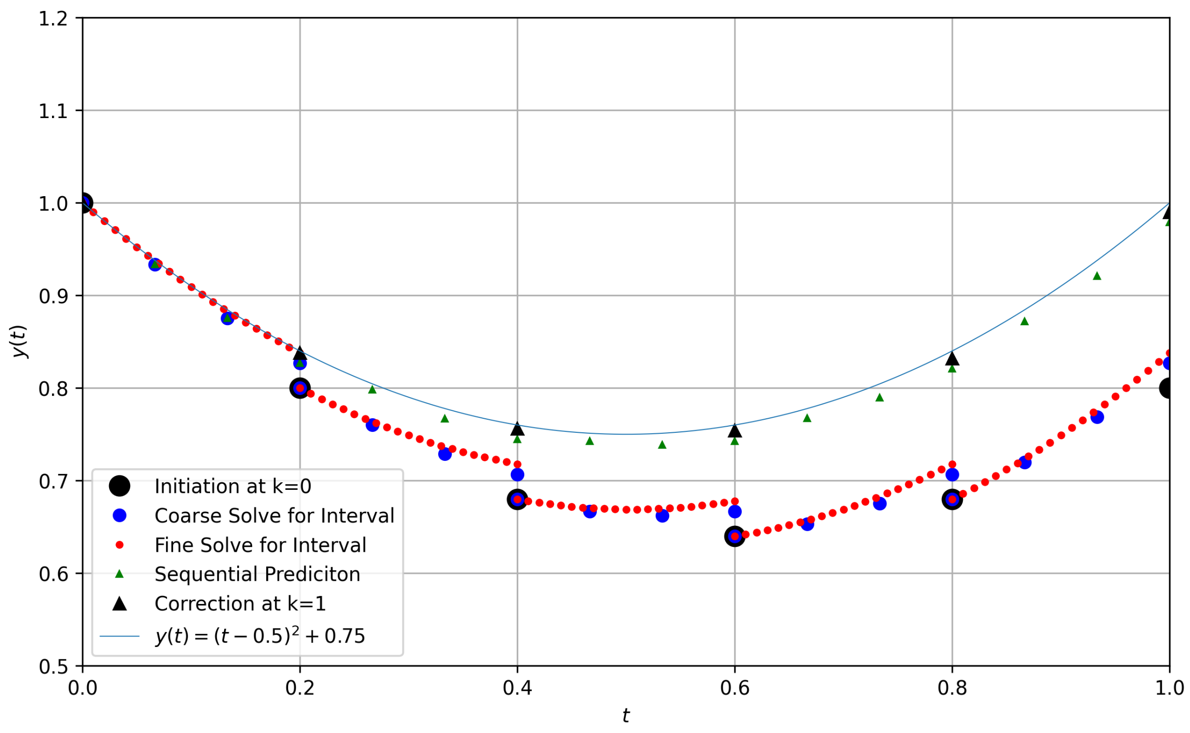


Figure 1: *Approximation of an ODE with the Parareal Algorithm*
Note: Time slices are initialized sequentially with the coarse solver (black dots). Coarse (blue) and fine solver (red) are run in parallel to compute solutions at the end of each time slice. A sequential predictor-corrector step (green) is performed using the delta of the intermediate solutions, yielding the solution after the first iteration (black triangles). This behaviour is repeated iteratively until converged to the exact solution (light blue). Here $\frac{\mathrm{d}y}{\mathrm{d}t} = 2(t - 0.5)$ and $y_0 = 1.0$.

Initialization, $k = 0$. The coarse solver is applied sequentially over the entire time interval $[t_0, T]$, producing initial guesses in all time slices, starting from the initial condition U^0.

$$U_0^0 = U^0, \quad U_{j+1}^0 = \mathcal{G}(t_j, t_{j+1}, U_j^0), \quad j = 0, ..., N-1. \tag{2.14}$$

Iterations, $k > 0$. Per iteration k, the fine solver is applied on all slices in parallel from the previous iterate. For all $k \geq 0$, set $U_0^k = U^0$. Then, the coarse correction is done serially. For $j = 0, ..., N-1$,

$$U_{j+1}^k = \mathcal{G}(t_j, t_{j+1}, U_j^k) + \left(\mathcal{F}(t_j, t_{j+1}, U_j^{k-1}) - \mathcal{G}(t_j, t_{j+1}, U_j^{k-1})\right). \tag{2.15}$$

The algorithm applies both solvers over k iterations and j time slices. For k iterations, all time slices from $j = 0, ..., k$ have been solved equal to a sequential fine solve. Therefore, Parareal will converge for a maximum of N iterations on a bounded time interval, thus converges superlinearly (M. J. Gander & Vandewalle, 2007):

Parareal convergence theorem. Let U^* be the fully serial fine solution, $U^*_{j+1} = \mathcal{F}(t_j, t_{j+1}, U^*_j)$, and $U^*_0 = U^0$. Then for all $k \geq 0$ iterations, $j \leq k$ holds: $U^k_j = U^*_j$. Parareal converges in at most N iterations.

Proof (sketch). For $k = 0, U^0_0 = U^*_0$. Assume true for $k-1$. For $j = 0$:

$$U^k_1 = \mathcal{G}(t_0, t_1, U^k_0) + \big(\mathcal{F}(t_0, t_1, U^{k-1}_0) - \mathcal{G}(t_0, t_1, U^{k-1}_0)\big) = \mathcal{F}(t_0, t_1, U^*_0) = U^*_1.$$

Assume $U^k_j = U^*_j$ for some $j < k$. Since $j < k$, by the induction hypothesis for iteration $k-1$ we also have $U^{k-1}_j = U^*_j$. Then

$$U^k_{j+1} = \mathcal{G}(t_j, t_{j+1}, U^*_j) + \big(\mathcal{F}(t_j, t_{j+1}, U^*_j) - \mathcal{G}(t_j, t_{j+1}, U^*_j)\big) = \mathcal{F}(t_j, t_{j+1}, U^*_j) = U^*_{j+1}.$$

Thus $U^k_j = U^*_j \forall j \leq k$. $\square$

We define a theoretical accuracy-based criterion to stop the iterations of Parareal:

Stopping criterion. Let U^* be the fully serial fine solution. Choose a loss function $L(\cdot, \cdot)$, e.g., mean squared error (MSE) or L^2 error (L2), and a tolerance $\epsilon > 0$. Stop at the minimal k such that

$$J(U^k, U^*) \leq \epsilon. \tag{2.16}$$

In order for Parareal to provide speed-up compared to a sequential fine solve, the number of iterations k needed to converge has to be significantly smaller than the number of sub-intervals, $k \ll N$ (M. J. Gander & Vandewalle, 2007). In ideal the case, where each of the N time slices is assigned one processing unit, the speed-up is limited by the coarse solver τ_{coarse}.

Runtime and speed-up. Initialization costs are of $N\tau_{coarse}$. Each iteration costs $\tau_{fine} + N\tau_{coarse}$.

$$T_{parareal} = N\tau_{coarse} + k(\tau_{fine} + N\tau_{coarse}), \quad T_{fine} = N\tau_{fine}. \tag{2.17}$$

$$S_{parareal} = \frac{T_{fine}}{T_{parareal}} = \frac{N\tau_{fine}}{(k+1)N\tau_{coarse} + k\tau_{fine}} = \frac{1}{(k+1)\frac{\tau_{coarse}}{\tau_{fine}} + \frac{k}{N}}. \tag{2.18}$$

The trade-off that has to be made expects $\tau_{coarse} \ll \tau_{fine}$ and $k \ll N$ (M. J. Gander & Vandewalle, 2007; Minion, 2010). This limitation makes accuracy and costs of the coarse solver the major drivers of the algorithms potential.

Parareal convergence. Consider the scalar test equation $u'(t) = au(t)$ with $a \in \mathbb{C}$. Let $R_{\mathcal{G}}(\Delta t)$ and $R_{\mathcal{F}}(\Delta t)$ be the stability functions of the coarse and fine one-step operators, and $z = a\Delta t$, where Δt is the length of an interval. Assuming $|R_{\mathcal{G}}(z)| < 1$, the asymptotic convergence factor is given by

$$\rho_{convergence} = \frac{R_{\mathcal{F}}(z) - R_{\mathcal{G}}(z)}{1 - |R_{\mathcal{G}}(z)|}, \tag{2.19}$$

so it converges if $\rho_{convergence} < 1$, and stagnates/diverges if $\rho_{convergence} \geq 1$. A proof is found in the work of M. J. Gander and Vandewalle (2007).

In the original publication of Lions et al. (2001), a numerical Euler scheme with large step size Δt was used for coarse propagation, providing a cheap, but also less accurate solution. Nowadays, latest research in the field of machine-learning has come up with techniques outperforming runtime and accuracy of classical numerical methods (Kovachki et al., 2023; Lu et al., 2021). Especially neural operators, such as the Fourier Neural Operator (FNO), reported computations up to three orders of magnitude faster on specific benchmarks compared to traditional PDE solvers (Li et al., 2021), raising the question if an application within Parareal is beneficial.

2.4 Neural Network Fundamentals

Neural networks act as universal function approximators, which means that under certain conditions and inputs, they can approximate any continuous function. This was shown with the Universal Approximation Theorem, making them powerful enough to learn complex mappings of finite dimensions, $f : \mathbb{R}^n \to \mathbb{R}^m$ (Goodfellow et al., 2016; Hornik et al., 1989). The definitions of this section are derived from the work of Goodfellow et al. (2016).

Feedforward Neural Network (FNN). Let $H_{hidden} \in \mathbb{N}$ be the number of network layers and $(d_h)_{h=1}^{H_{hidden}}$ the layer widths, $x \in \mathbb{R}^{d_0}$.

$$\hat{y}^{(0)} = x, \quad z^{(h)} = W^{(h)}\hat{y}^{(h-1)} + b^{(h)}, \quad \hat{y}^{(h)} = \sigma^{(h)}(z^{(h)}), \quad h = 1, ..., H_{hidden}. \tag{2.20}$$

The prediction is given by $\hat{y} = \hat{y}^{(H_{hidden})} \in \mathbb{R}^{d_H}$. $W^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ are learnable weights, $b^{(h)} \in \mathbb{R}^{d_h}$ are learnable biases, and $\sigma^{(h)} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ applied elementwise.

ReLU activation function. $\mathrm{ReLU}(x) = \max\{0, x\}$.

A FNN is trained by updating the learnable model parameters θ such that for a given input the output matches the desired one. The discrepancy is measured by a loss function J, parameters are updated stepwise towards the direction of steepest gradient using a learning rate η to ensure training stability.

Loss and gradient step. For a learning rate $\eta > 0$ and a training loss $J(\theta)$,

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta). \tag{2.21}$$

Early stopping. A validation loss J_{val} is monitored for each epoch e and the training is stopped at the first epoch with $J_{val}(e) \geq J_{val}(e-1)$.

Hyperparameters. Depth H_{hidden}, layer widths d_h, learning rate η, batch size, weight initialization, activation functions, regularization technique.

Although FNNs can represent complex mappings, there exist fundamental limitations when modeling mappings between function spaces: They are unable to capture spatial locality of structured data such as PDEs due to lack of neighbour information, and the number of parameters scales quadratically with input size (Kovachki et al., 2023). This motivates the research on alternative architectures, capable of learning the mappings between function spaces (Kovachki et al., 2023; Li et al., 2021).

2.5 Operator-Learning and the Fourier Neural Operator

Operator learning models aim to learn mappings between function spaces rather than finite-dimensional vectors. Therefore, an input-to-solution Neural Operator (NO) learns a discrete solution of a PDE on a common domain (Kovachki et al., 2023; Lu et al., 2021). This allows for learning an entire family of PDEs rather than solving one instance (Li et al., 2021). The definitions of this section are derived from the work of Kovachki et al. (2023), Li et al. (2021), and Lu et al. (2021).

Operator-learning setup. Let the domain be the periodic box $D = \Omega = [0, L_{spatial}]^d$ and $\mathcal{A}(D) \subset L^2(D; \mathbb{R}^{d_a}), \mathcal{U}(D) \subset L^2(D; \mathbb{R}^{d_u})$ the input and output function spaces. A Neural Operator is a parametric map

$$G_\theta : \mathcal{A}(D) \to \mathcal{U}(D), \quad \theta \in \Theta \subset \mathbb{R}^p. \tag{2.22}$$

Given a dataset with pairs of functions $\mathcal{D} = \{(a_i, u_i)\}_{i=1}^{N_{data}}$ with $a_i \in \mathcal{A}(D), u_i \in \mathcal{U}(D)$, the empirical risk with L^2 loss for the given dataset is

$$J(\theta) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} ||G_\theta(a_i) - u_i||^2_{L^2(D)}. \tag{2.23}$$

In practice, the discrete ℓ^2 approximation is minimized on a grid with n points.

The design of neural operators provides for G_θ acting on functions instead on a fixed grid, using an architecture design that performs operations on continuous representations or interpolation between discretization points. Therefore, some operators transform a function into another by applying an integration kernel to the input. While training still uses discrete samples, the same parameters θ can be applied at different resolutions and the mapping now becomes independent of the grid.

Fourier Neural Operator (FNO). d_a, d_u are widths of input and output channels, d_v is the hidden width. The FNO lifts pointwise to the latent space $P : \mathbb{R}^{d_a} \to \mathbb{R}^{d_v}$, applies $\mathfrak{L}$ Fourier layers, and projects back using $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$:

$$v_0(x) = Pa(x), \quad v_{\ell+1}(x) = \mathrm{ReLU}\left(Wv_\ell(x) + K(v_\ell)(x)\right), \quad \ell = 0, ..., \mathfrak{L}{-}1, u(x) = Qv_{\mathfrak{L}}(x). \tag{2.24}$$

$W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a learned pointwise linear map, K is a Fourier convolution operator, and ReLU is applied elementwise.

Fourier convolution operator. $\mathfrak{F}$ denotes the Fourier transform on D and $\hat{v} = \mathfrak{F}[v]$. $\mathcal{K}$ is the set of the lowest-frequency modes kept in Fourier domain. For each $\kappa \in \mathcal{K}$, the layer learns a complex matrix $\hat{M}(\kappa) \in \mathbb{C}^{d_v \times d_v}$. We define K by

$$(\mathfrak{F}[K(v)])(\kappa) = \begin{cases} \hat{M}(\kappa)\hat{v}(\kappa), & \kappa \in \mathcal{K} \\ 0, & \text{otherwise,} \end{cases} \qquad K(v) = \mathfrak{F}^{-1}[\hat{M} \cdot \hat{v}]. \tag{2.25}$$

The truncation keeps $\mathcal{K}$ symmetric, while all modes outside $\mathcal{K}$ are set to zero.

With the convolution theorem stating that a convolution in physical space corresponds to an elementwise multiplication in Fourier space, the FNO can apply translation-invariant filters globally with quasi-linear computation cost. By keeping the lowest-frequency modes, the long-range dependencies characteristic of PDE solutions are captured.

Per-layer computational cost. $\mathfrak{F}$ and $\mathfrak{F}^{-1}$ use Fast Fourier Transform/inverse Fast Fourier Transform. For a one dimensional grid with n points and $m_{modes} = |\mathcal{K}|$ kept modes, one FNO layer costs roughly

$$O(d_v \; n \log n) \text{ for } \mathfrak{F}/\mathfrak{F}^{-1} \quad + \quad O(m_{modes} \; d_v^2) \text{ for } K \quad + \quad O(n \; d_v^2) \text{ for } Wv. \tag{2.26}$$

For a two dimensional grid with $n \times n$ points and $m_{modes} = |\mathcal{K}|$ kept modes, one FNO layer costs roughly

$$O(d_v \; n^2 \log n) \text{ for } \mathfrak{F}/\mathfrak{F}^{-1} \quad + \quad O(m_{modes} \; d_v^2) \text{ for } K \quad + \quad O(n^2 \; d_v^2) \text{ for } Wv. \tag{2.27}$$

The spectral multiplication within the FNO encodes priors such as smoothness and translation invariance. Therefore, the model generalization relies on continuity and alignment with the learned mapping, for instance through training on periodic domains.

Given the moderate cost of an FNO forward pass, numerical experiments have shown a superior speed compared to classical numerical methods, while maintaining low error rates for non-linear PDEs such as the Burgers' equation and the Navier-Stokes equation (Kovachki et al., 2023; Li et al., 2021). Furthermore, recent applications have reported practical benefits when using the FNO for physical simulations, in particular when utilizing it as a coarse propagator within Parareal (Ibrahim et al., 2024; Pamela et al., 2024).

2.6 Related Work

Physical simulations often rely on computationally intensive calculations, making algorithmic optimization inevitable (M. Gander & Halpern, 2007; Rudi et al., 2015). In their recent publications, Ibrahim et al. (2024) and Pamela et al. (2024) have explored a novel approach that combines a classical numerical solver with machine-learning-technique. They built on the Parareal algorithm proposed by Lions et al. (2001), which divides long-time simulations into smaller time slices, making it suitable for in-time parallelization.

In the studies of Ibrahim et al. (2024) and Pamela et al. (2024), the Parareal coarse solver was replaced by a Fourier Neural Operator (FNO), which learns mappings between infinite-dimensional function spaces. It has shown superior speed compared to traditional numerical methods for time-dependent problems, while maintaining low error (Li et al., 2021). This hybrid combination of both methods proposes speed-up for physical simulations (Gorynina et al., 2022; Ibrahim et al., 2024; Pamela et al., 2024). In the approach of Pamela et al. (2024), the framework additionally exploits the large amount of data that is inherently produced by Parareal for further training of the model, thereby demonstrating the influence of FNO quality on Parareal speed-up.

Analyses of the original Parareal algorithm show that the efficiency is limited by accuracy and cost of the coarse method (M. J. Gander & Vandewalle, 2007; Minion, 2010). This results in boundaries shown in Equation 2.18 and Equation 2.19. Exceeding demands of time and computation power for physical simulations drive the research on algorithmic optimization. This is addressed with work of Ibrahim et al. (2024) and Pamela et al. (2024), which extends possibilities even further. Although recent applications combining machine-learning methods with classical numerical solvers have shown potential, the drivers of benefit are yet entirely understood.

We have chosen to rebuild this hybrid approach of Ibrahim et al. (2024) and Pamela et al. (2024) as it seemed to be well-suited for a small-scale numerical experiment. Due to its novelty, literature is quite sparse, often tightly bound to a specific field, and lacks investigations under basic or normalized conditions. This work seeks to contribute to the domain of machine-learning-accelerated solvers for time-dependent partial differential equations. It is an exploration that aims to provide an intuition on such a hybrid approach. Rather than replicating the original study's specific application to fusion research, our study uses the proposed approach to perform a more fundamental investigation.

3 Methodology

3.1 Experimental Hypotheses

This study aims to evaluate the effectiveness of a hybrid Parareal algorithm that utilizes a Fourier Neural Operator (FNO) as a data-driven coarse solver in the Parareal framework in a small scale experimental setup. Our first experiment investigates the relationship between the accuracy of the FNO and the Parareal algorithm's convergence rate. A more accurate FNO, which has been trained on more data and for longer training time, should provide a superior initial guess for the Parareal iterations, thereby reducing the number of iterations required to reach a satisfactory solution. Based on this expectation, we formulate the following hypotheses:

- H_0: There is no significant correlation between the quality of the FNO coarse solver (increasing training samples and epochs) and the convergence speed of the Parareal algorithm.
- H_A: Parareal simulations using a higher-quality FNO coarse solver (trained with more samples and over more epochs) will require fewer iterations to converge, yielding a significantly faster convergence rate.

The second experiment focuses on the benefits of the hybrid approach in terms of runtime, by measuring the achieved computational speed-up. It explores how a high-quality coarse solver translates into a faster simulation compared to a traditional sequential solver. We expect the speed-up to be limited by the coarse updates which are sequentially applied within Parareal. Therefore, we formulate the following hypotheses:

- H_0: The computational speed-up provided by the Parareal algorithm is not significantly affected by the quality of the FNO coarse solver.
- H_A: A higher quality FNO coarse solver will provide greater computational speed-up for Parareal simulations. The speed-up is expected to increase with the number of parallel intervals.

We now turn to the implementation of our experimental setup.

3.2 Setup and Implementation

Software Stack and Architecture

The technical implementation was organized in three stages, (i) the data sampling, (ii) the training of the FNO model, and (iii) the Parareal simulations. The experimental pipeline was implemented in Python, primarily using `JAX`. For data sampling, we used `apebench`[1] and `exponax`[2], two state-of-the-art libraries for PDE emulation. The model was implemented in `flax.nnx`, which provides simple state management. We used a `PyTorch` data loader and `orbax` for checkpointing. Parareal combined the FNO model as coarse solver and ETDRK method as fine solver provided by the `exponax` library. Both solvers were integrated into Parareal. Iterative loops were realized with `jax.vmap`. To ensure reproducibility, one global seed was passed to all random number generators in the pipeline. The evaluation of simulation outputs was done in a separate script. A link to the program code is provided in the appendix.

Hardware

All three stages were performed on a server cluster with the following specifications: 8 $\times$ NVIDIA A100 GPUs with 40 GB memory per node, and 21 nodes in total. Each node had 2 $\times$ AMD EPYC 7302 CPUs, with 3.0 GHz and 16 cores. There was a total RAM of 1TB DDR4-3200 installed. The cluster used SLURM scheduling[3]. All experimental operations were performed with a precision of `float64`, while FNO spectral convolution uses `complex128` weights.

PDEs, Domains, and Discretization

We studied the 2D linear advection-diffusion equation given in Equation 2.5, and the 1D non-linear viscous Burgers' equation given in Equation 2.6. The problems were examined under default conditions provided by the `APEBench` library, defined with periodic boundary conditions over a domain of $\Omega = [0.0, 1.0]^d$, and a time horizon of $T = 5.0$. It comes with a built-in data generator for trajectories of various PDEs, which we used for generation of initial conditions. The spatial discretization has 256 discrete grid points for 1D scenarios, and 256×256 for 2D scenarios. Training and test trajectories shared the same grid. Unless stated otherwise, $\Delta t_{fine} = \Delta t_{coarse} = 0.1$, resulting in 50 steps (51 states including the initial condition). Parameters were fixed per PDE and kept constant during training and inference. All data is already limited to $[-1, 1]$ by APEBench a priori, so no additional normalization was applied.

[1]The APEBench library is a benchmark suite for evaluating neural operators and was developed by Koehler et al. (2024).

[2]The Exponax fine solver utilized in our experiments is hosted at https://github.com/Ceyron/exponax.

[3]For job submission, we used a tool hosted at https://github.com/2bys/submit.

3.3 Model and Training Pipeline

Fine Solver

The fine solver used for data sampling and Parareal was an ETDRK method. We used an implementation from the Exponax library which is based on the publication of Kassam and Trefethen (2005). The particular solver came with a built-in dealiasing of 2/3. Stability and adaptivity were handled internally. Aside from using a fixed step-size controller, we used default settings.

FNO Coarse Solver

The FNO performs multiple spectral convolutions (see Equation 2.25). As part of our experiments, we implemented convolutions in 1D and 2D in separate models. The algorithmic procedure of a 2D convolution in Fourier space is shown in Figure 2.

```
1: Notation:
2:     $B$: batch size
3:     $C_{in}$: number of input channels
4:     $C_{out}$: number of output channels
5:     $N_x, N_y$: spatial dimensions (height, width)
6:     $W_{pos}, W_{neg}$: Complex weights for positive and negative frequency band
7:     $modes_x, modes_y$: number of Fourier modes to keep
8: procedure FNOConv2D($x, W_{pos}, W_{neg}, modes_x, modes_y$)
9:     $(B, C_{in}, N_x, N_y) \leftarrow$ shape($x$)
10:    $x_{ft} \leftarrow$ rfft2($x$)                                          ▷ Forward FFT
11:    $out_{ft} \leftarrow \mathbf{0}$ of shape $(B, C_{out}, N_y, N_y/2+1)$    ▷ Initialize output spectrum
12:    $x_{pos} \leftarrow x_{ft}[:, :, 1 : modes_x, 1 : modes_y]$            ▷ Positive frequency band
13:    $y_{pos}[b, c_{out}, i, j] \leftarrow \sum_{c_{in}} x_{pos}[b, c_{in}, i, j]\ W_{pos}[c_{out}, c_{in}, i, j]$   ▷ Multiply kept modes
14:    $out_{ft}[:, :, 1 : modes_x, 1 : modes_y] \leftarrow y_{pos}$
15:    $x_{neg} \leftarrow x_{ft}[:, :, N_x - modes_x : N_x, 1 : modes_y]$    ▷ Negative frequency band
16:    $y_{neg}[b, c_{out}, i, j] \leftarrow \sum_{c_{in}} x_{neg}[b, c_{in}, i, j]\ W_{neg}[c_{out}, c_{in}, i, j]$   ▷ Multiply kept modes
17:    $out_{ft}[:, :, N_x - modes_x : N_x, 1 : modes_y] \leftarrow y_{neg}$
18:    $x \leftarrow$ irfft2($out_{ft}, (N_x, N_y)$)                          ▷ Inverse FFT
19:    return $x$
20: end procedure
```

Figure 2: *2D Fourier Neural Operator Convolution*
The convolution uses a Fast Fourier Transform (FFT) to lift the input into the frequency domain. A small set of low-frequency modes is kept and multiplied with learned complex weights. The result is then transformed back with an inverse FFT and returned.

In our study we used a model architecture with 4 Fourier layers, similar to the one provided in the original publication of Li et al. (2021). For 1D problems, we kept $m_{modes} = 16$ modes with a latent width of $d_v = 64$. For 2D problems, we kept $(m_{modes}, m_{modes}) = (12, 12)$ modes with a latent width of $d_v = 32$. Activations were performed with GELU. Weights were seeded via a global seed, initialized as complex with Xavier-initialization, and scaled by the number of channels. In our experimental setup, the FNO acted as one-step operator, mapping from an input with a single temporal state to an output at the next temporal state, $u^n \rightarrow u^{n+1}$. It was trained on the same grid used for inference and a single channel without extra conditioning on coefficients such as v or step size Δt.

A model outline is provided in the appendix. Note that our implementation flattened the temporal domain. This might have an effect on learning time dependent problems when using a temporal domain greater than 1, since a genuine Neural Operator for D-dimensional problems would encode an additional D+1 dimension for temporal interpolation (Li et al., 2021). Due to the one-step predictor role, our approach was well suited for an implementation that only encodes spatial domains. This helped us to maintain a relatively small number of parameters, resulting in $287,425$ parameters for the 1D models, and $1,186,177$ for the 2D models.

Training Protocol

The FNO was trained with the mean squared error (MSE) per batch as training loss function. We used Adam optimization, combined with a fixed learning rate of $1e-3$ and a batch size of 32. Checkpointing was performed every 10 epochs, enabling Parareal simulations at multiple training states (checkpoints). The model was trained on GPU using the `CUDA` API.

Datasets and Splits

All training and simulation data was generated using the fine solver. Initial conditions were sampled from APEBench with default settings, creating periodic fields with 5 Fourier modes, a zero mean, and a maximum within $[-1, 1]$. Training data was computed using the fine solver. Up to 500 training/validation trajectories and 30 test trajectories were generated, each over 50 time steps. This led to a total number of $50 \times 500 = 25,000$ training pairs and $50 \times 30 = 1,500$ test pairs. An 80/20 split was employed for the training and validation sets on the training pairs. Initial conditions of the test trajectories were equal to the ones used in Parareal simulations. All generated data was stored in `.npy` files and loaded as `jax`-arrays during training.

Parareal Configuration

In the original publication of (Lions et al., 2001), Parareal was intended to be used with a numerical coarse, and a numerical fine solver. To ensure compatibility with different types of solvers, we have implemented the Parareal algorithm in `JAX`. In our simulations, the initialization of the N time slices was done with the coarse solver via rollout of N FNO forward passes. Each time slice was assigned to one processing unit and computed in parallel. Coarse updates were done serially.

3.4 Metrics and Evaluation

FNO Training Evaluation

For each scenario, models with three different sizes of data were trained. For comparability of the quality impact, model architectures were fixed across training runs, only varying data size. Per scenario, we decided for data sizes of 50, 100, and 200 samples (full trajectories), where the default number of samples intended by APEBench was the smallest one. Each sample equalled 50 one-step input-output pairs. For each configuration, $n = 30$ models were trained over 500 epochs using different seeds for weight initialization and data splits. During training, loss for training, validation, and test set was tracked using the mean squared error (MSE).

The primary evaluation protocol was validation-best: For each seed, we have selected the checkpoint that minimized the validation-loss. We report the training loss and test loss at this checkpoint. This allows for computation of the generalization gap per seed, which is defined by the delta of the test loss and training loss are the validation-selected best epoch. The generalization gap indicates overfitting or generalization of the model.

As a secondary evaluation, we used a test-selected protocol to pick the best checkpoint based on test loss. Because the best epoch can differ by seed, we report the average epoch index and the aggregated mean test loss value. We choose this evaluation protocol to select and highlight the models that are used in our follow-up experiments. Furthermore, time per epoch are reported along with the total training times per data set size. Reported times do not include testing passes.

We show figures for training loss vs. epochs and test loss vs. epochs, where the mean test-best epoch is marked. We provide both, tables for our validation-best selection protocol featuring the generalization gap, as well as a training summary stating the best models and times. All reported metrics are aggregated and reported with the mean (M) and standard deviation (SD) over the $n = 30$ seeds.

Simulations Experiment 1

In our first experiment, we aimed to quantify the impact of the coarse solver quality on Parareal convergence. We therefore used the serial fine solver as a baseline for ground truth comparison, while varying data sizes (50 / 100 / 200) and the number of epochs (100 / 200 / 300 / 400 / 500) of the coarse solver model. For each unique configuration (data size $\times$ epoch) of data size and epoch, simulations were run for all 30 initial conditions from the test set. Spatial discretization and time step were identical to the ones used for training over all runs. To match the number of 8 GPUs for parallel computation, simulations were run over 8 time steps (or 8 Parareal intervals). This also sets the maximum number of Parareal iterations to 8. In practice, one would stop iterating after a certain stopping criterion, i.e. when solution values are no longer changing beyond a threshold with further iterations. For our error metrics, it is necessary to compare the intermediate results of all iterations with the ground-truth solution. Therefore, we always run simulations for all Parareal iterations.

Our simulation run error was computed per Parareal iteration at each interval. The error is the MSE over the spatial domain relative to the fine solution at every time step, averaged across the 30 initial conditions. As all simulations were run with the models trained with 30 different seeds for each unique checkpoint configuration, this results in $30 \times 3 \times 5 \times 30 = 13,500$ solved trajectories or 450 runs for each scenario. Reported metrics therefore always have been aggregated over 30 trajectories and 30 seeds.

Convergence and accuracy of simulation were measured entire trajectories using (i) the Area Under Curve (AUC) over iterations, and (ii) the share of iterations below a threshold. While the AUC is used as an indicator of relative convergence speed (lower AUC means faster convergence), the share-of-iterations allows for an absolute tuning to our specific case. We tied thresholds to training performance to emphasize the hybrid method rather than the model alone. Therefore, thresholds $\epsilon_{scenario}$ were set for each scenario specifically, always one order of magnitude lower than the mean test loss of the best model identified during training. In addition, we also show simulations after the Parareal coarse initialization at iteration 0 (the full rollout of the models), and simulations after the first iteration for indicating full time-parallel computation, if below threshold $\epsilon_{scenario}$.

Statistical analysis is performed with correlation tests, using the number of training samples and the number of epochs as predictors. For each unique configuration (samples $\times$ epoch) we computed seed-aggregated metrics. Both AUC and share-below-threshold were tested for p-values with a one-sided Spearman correlation ($\alpha = .05$). For completeness, we also report p-values for Pearson r, although we do not expect the correlation to be linear in neither case.

Before further processing, all simulation results were scanned for outliers to prevent instability from skewing statistics. As we are predicting output values within $[-1, 1]$, a threshold of 1.0 was set for filtering. If any seed exhibited any interval at any iteration with an MSE above this threshold, the entire configuration was considered as unstable and was excluded from analysis.

For the simulations, we show MSE vs. Parareal iteration curves by data size and by epoch. We prepare heat maps for Parareal initialization and iteration 1. We also provide a heat map indicating the AUC over the entire Parareal simulation. We report tables for correlation test results per metric and predictor.

Simulations Experiment 2

Our second, smaller experiment aimed to quantify the benefit of the hybrid setup in terms of time. We therefore used times of sequential fine solver and model roll out as a baseline for comparison, while varying data sizes (50 / 100 / 200) of the coarse solver model and the number of Parareal intervals (1 / 2 / 4 / 8) on our 2D Advection-Diffusion problem. For each seed, we selected the saved checkpoint whose epoch is closest to the mean best-epoch (across 30 seeds) and report its test loss. We used the same spatial discretization and step size as in the first experiment. This ensures that the number of solver steps is consistent regardless of the number of time slices. So for instance, when using 4 intervals, each solver had to perform 2 sequential steps. Parallelization was achieved by explicitly requesting 1, 2, 4, or 8 GPUs.

For each unique configuration data size and intervals, simulations were again run with all 30 initial conditions from the test set. Runtimes were tracked with the `time` module and markers are placed around function calls. Host synchronization was enforced explicitly. Parareal runtimes were calculated as follows: For each seed, the error was tracked per iteration using the fine solver as baseline. We then again used the share-below-threshold metric from experiment one to determine the number of iterations with satisfactory error. This was then used to compute the relative time to threshold convergence. We report the mean (M) and standard deviation (SD) per simulation run across seeds.

Baseline runtimes of coarse and fine solver were calculated and averaged across seeds. Although warm-up calls were executed prior to each simulation run, sequential solvers were run multiple times and runtimes are again averaged to make sure to keep the overhead of a sequential solve small, ensuring comparison. We used sequential fine/coarse runtimes for 8 intervals with 8 steps each for comparison.

Our 2D FNO models had $1,186,177$ parameters. For one model forward pass (time step), 1.598 GFLOPS were expected ($4 \times \mathfrak{F}$ and $\mathfrak{F}^{-1}$, $4\times$ summations on kept modes K, $4\times$ skip connections W, pointwise linear layers, GELU activations). For the `exponax` ETDRK method, we were not able to obtain a valid estimation on FLOPS per time step. Thus, only times are reported.

We provide a derivation of the net algorithm runtime boundary using tracked times from both solvers. We show figures for Parareal simulation times vs. the number of samples the model is trained on for all configurations of data size and intervals.

3.5 Sanity Checks and Reproducibility

Prior to our numerical experiments, sanity checks were performed on external libraries for our selected parameters. To ensure numerical accuracy, we have compared analytic solutions from the `Exponax` fine solver with those from different explicit and implicit solvers. In addition, all data samples created by the fine solver were tested on min, max, and non-finite values, ensuring model training and simulation over normalized data in the range of $[-1, 1]$. By running simulations over all iterations, we were able to monitor Parareal convergence, as the solution is always equal to the one of the fine solver at the last iteration. We treated runs with a n error below 1.0×10^{-32} at the final iteration as converged.

Our experiments were made to be reproducible and deterministic by setting a global seed which is used at all points of random number generation, i.e. model weight initialization and data splits. Training data generation itself was done with fixed seeds, thus does not vary with models. Library versions and code references are listed in the appendix.

We have performed small snapshot tests prior to our experiments in order to verify basic applicability of the setup. We have tested different model architectures, simulation parameters, and/or problems using the 1D/2D Diffusion equation and the 2D Navier-Stokes equation. Due to the limited scope, these are neither reported nor used for our main claims.

4 Experiment 1: FNO Quality Impact

4.1 FNO Training Results

For both scenarios, we observe increasing stability with more data, as indicated by a decrease in the test-loss standard deviation. We also observe a decreasing generalization gap with increasing data in both scenarios. The training time per epoch increases approximately linearly with data size. Some models trained on larger datasets exhibit overfitting in later epochs, whereas models trained on 50 samples maintain a higher but monotonically decreasing test loss.

4.1.1 2D Advection-Diffusion Scenario

For each configuration, models for the 2D Advection-Diffusion scenario were trained for 500 epochs on datasets comprising 50, 100, or 200 samples. Figure 3 shows training loss versus epoch for each configuration and solid lines denote the mean across seeds, and the translucent bands indicate the range across seeds. We did not observe major instabilities during training.

Under the validation-selected protocol, the mean best-epoch occurred at 492.70 (50 samples), 477.33 (100 samples), and 479.40 (200 samples). The generalization gap decreased with sample size: mean (SD) = 4.60×10^{-5} (2.64×10^{-5}) for 50 samples, 3.99×10^{-6} (1.39×10^{-6}) for 100 samples, and 9.41×10^{-7} (4.02×10^{-7}) for 200 samples. A larger generalization gap indicates overfitting, whereas a smaller gap indicates better generalization to unseen data. This trend is illustrated in Figure 4. The detailed values of validation-selected training loss, test losses, and generalization gap are found in the appendix.

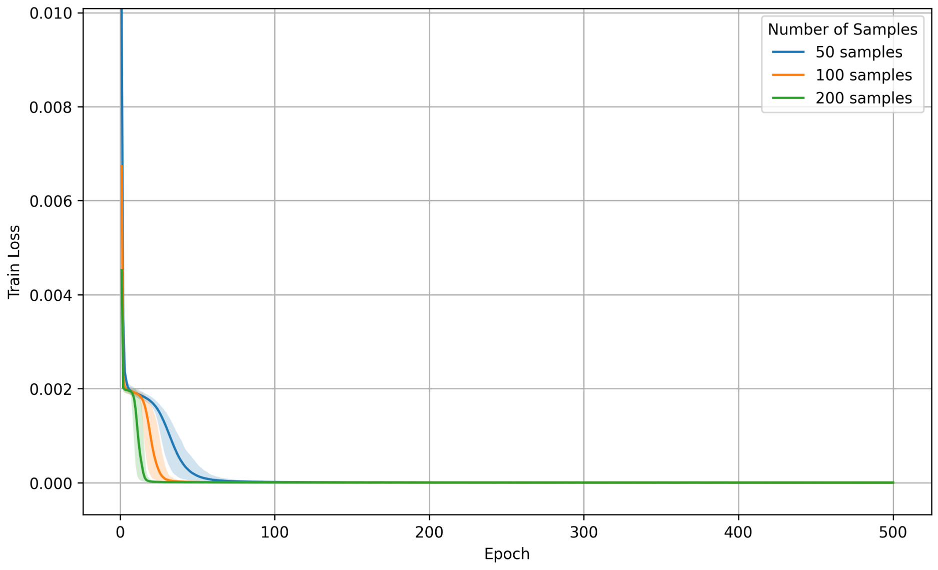


Figure 3: *Training Loss vs. Epochs for 2D Advection-Diffusion Scenario*
Note: The lines indicate the mean training loss vs. epochs for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models

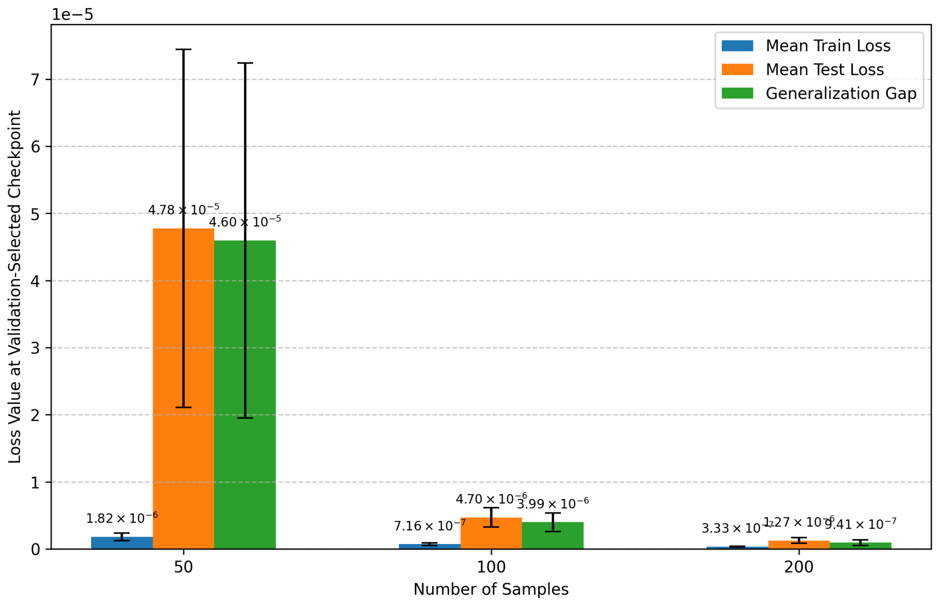


Figure 4: *Generalization Gap for 2D Advection-Diffusion Scenario (Validation-Selected)*
Note: Generalization improves with increasing training data size. Mean and Standard Deviation are computed over 30 models

With test-selected checkpoints, the best-epoch variability increased with data size: mean (SD) best epoch = 492.33 (8.28) for 50 samples, 478.03 (16.09) for 100 samples, and 471.60 (25.96) for 200 samples. The corresponding test losses became more stable with more data: mean (SD) = 4.53×10^{-5} (2.49×10^{-5}), 4.56×10^{-6} (1.42×10^{-6}), and 1.25×10^{-6} (4.24×10^{-7}) for 50, 100, and 200 samples, respectively. Figure 5 shows mean test loss versus epoch and dots mark the mean best epoch (test-selected) for each data size. Dots indicate the mean best-epoch (from a test-selected best checkpoint) for each data size. Average time per epoch and total training time increased with sample size: 0.73 s per epoch (6.07 min total) for 50 samples, 1.59 s (13.24 min) for 100 samples, and 3.95 s (32.90 min) for 200 samples. For the model trained on 200 samples, test loss increases between epochs 400 and 500 (see Figure 5). Table 1 shows a training summary including training times and test losses at these test-selected best checkpoints.

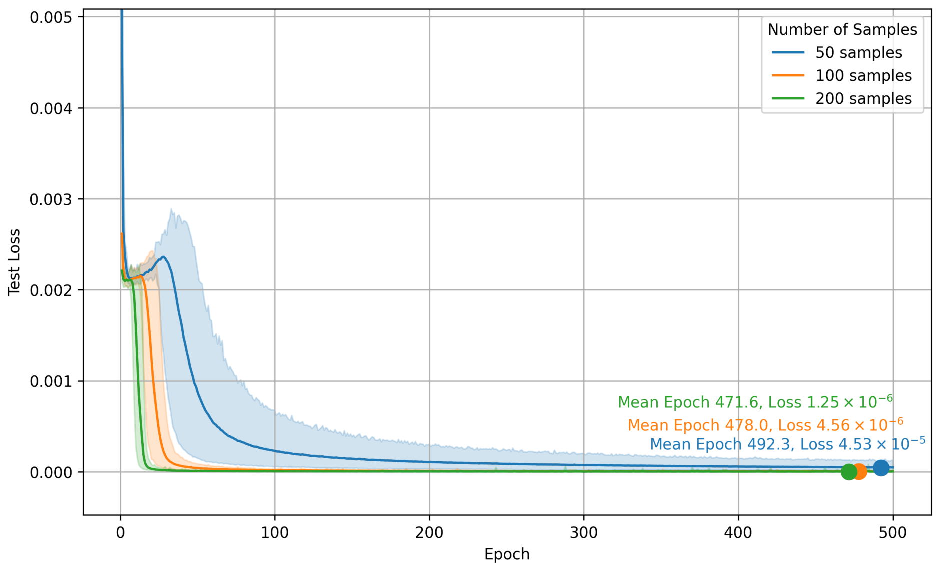


Figure 5: *Test Loss vs. Epochs for 2D Advection-Diffusion Scenario*
Note: Lines show the mean training loss versus epoch for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models. Dots indicate the mean best epoch for each data size.

Table 1: *Training Summary for 2D Advection-Diffusion Scenario (Test-Selected)*
Note: Mean and standard deviation over 30 models.

Samples	n	Mean Epoch Time (s)	Mean Total Time (min)	Epoch Mean	Epoch SD	Test Loss Mean	Test Loss SD
50	30	0.73	6.07	492.33	8.28	4.53×10^{-5}	2.49×10^{-5}
100	30	1.59	13.24	478.03	16.09	4.56×10^{-6}	1.42×10^{-6}
200	30	3.95	32.90	471.60	25.96	1.25×10^{-6}	4.24×10^{-7}

4.1.2 1D Burgers' Scenario

For this scenario, we trained for 500 epochs on datasets with 50, 100, and 200 trajectories. Figure 7 shows the training loss over the epochs. Solid curves are the mean across $n = 30$ seeds and translucent bands indicate the range. Although some runs showed minor instabilities, we did not observe major instabilities during training.

Under the validation-selected protocol, the mean best epoch occurred at 471.27 (50 samples), 413.23 (100), and 253.93 (200). At those checkpoints, the generalization gap

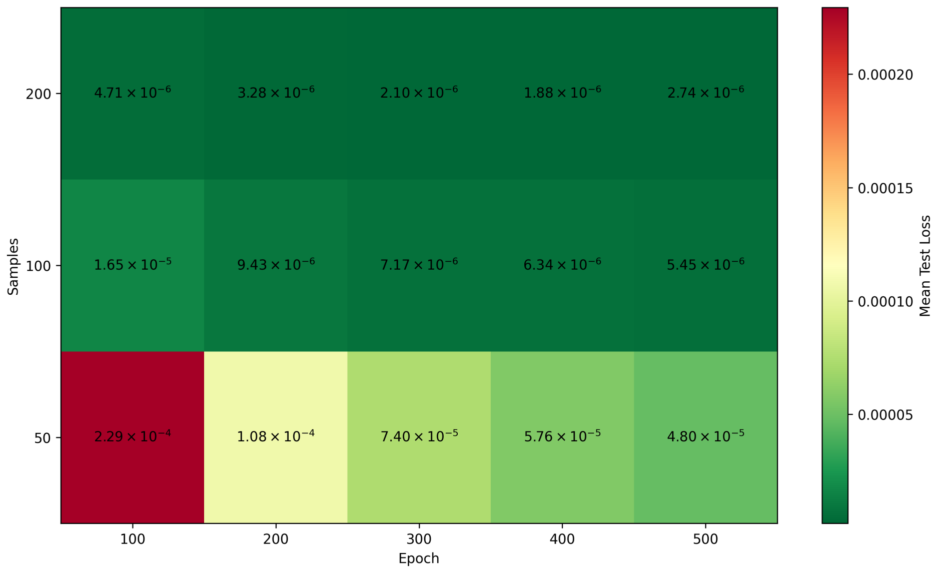


Figure 6: *Test Losses of Checkpoints for Experiment 1 with 2D Advection-Diffusion Scenario*
Note: Test losses generally decrease with larger training sets and/or longer training. The increase at 200 samples and 500 epochs indicates overfitting. Means computed over 30 models

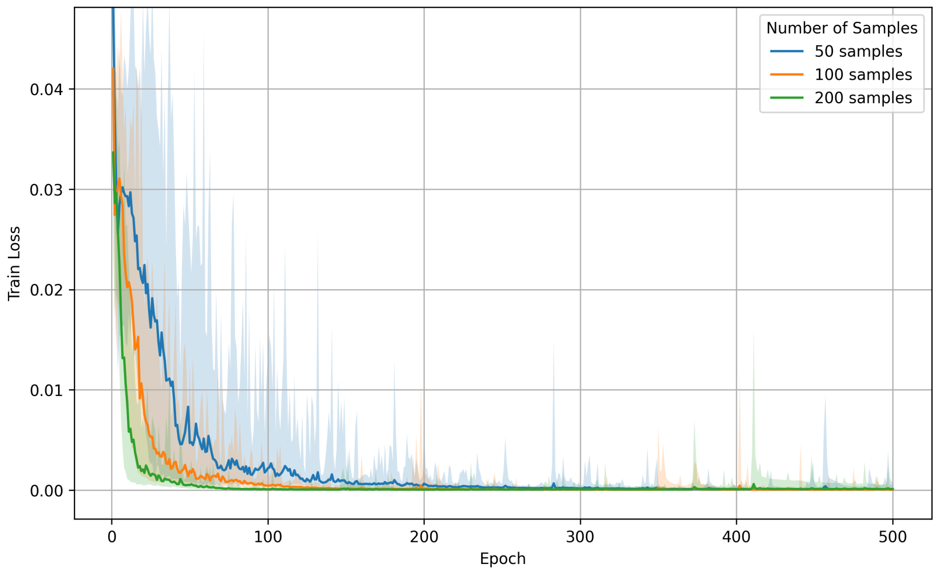


Figure 7: *Training Loss vs. Epochs for 1D Burgers' Scenario*
Note: The lines indicate the mean training loss vs. epochs for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models

was slightly negative—test loss below train loss: mean (SD) $= -5.21 \times 10^{-7}$ (1.28×10^{-5}), -1.13×10^{-5} (1.32×10^{-5}), and -1.54×10^{-5} (1.36×10^{-5}) for 50, 100, and 200 samples, respectively.This suggests the models occasionally perform better on the test set than on the training set, likely due to stochasticity. The trend is visualized in Figure 8. Detailed numbers are provided in the appendix.

With test-selected checkpoints, the mean (SD) best epoch was 473.87 (40.97), 422.70

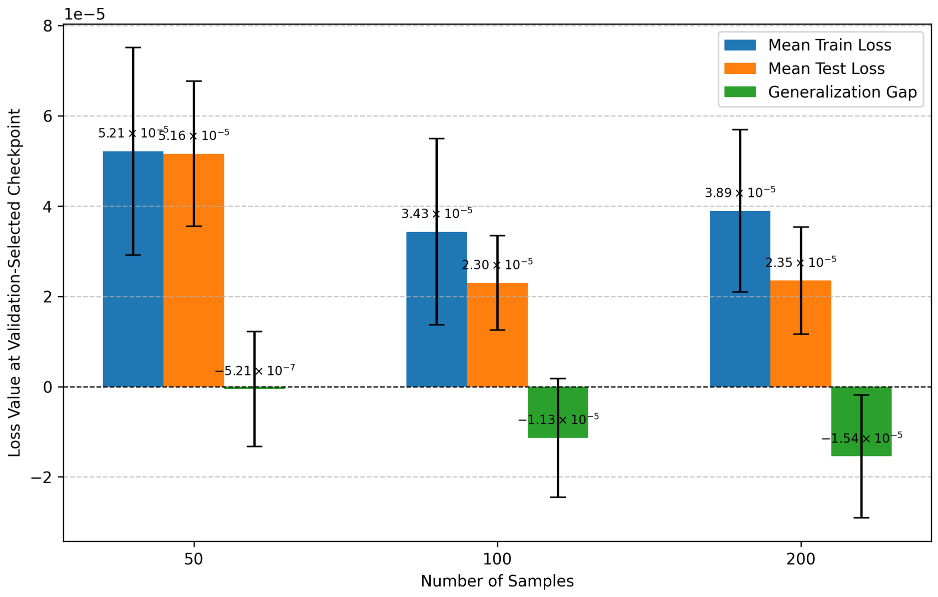


Figure 8: *Generalization Gap for 1D Burgers' Scenario (Validation-Selected)*
Note: A better generalization to unknown data is achieved when increasing the training data size. The negative values are likely caused by stochasticity. Mean and Standard Deviation are computed over 30 models

(73.81), and 267.00 (90.80) for 50, 100, and 200 samples, respectively. The corresponding test losses decreased and became more stable with more data: mean (SD) = 5.07×10^{-5} (1.53×10^{-5}), 2.27×10^{-5} (1.04×10^{-5}), and 2.19×10^{-5} (8.74×10^{-6}). Average time per epoch and total training time were 0.32 s (2.67 min), 0.64 s (5.32 min), and 1.25 s (10.44 min) for 50, 100, and 200 trajectories, respectively. The mean test loss over epochs is displayed in Figure 9. A summary of the test-selected results is provided in Table 2. Figure 9 shows the mean test losses of checkpoints chosen for the simulation experiments. The models trained on 100 and 200 samples indicate overfitting, with test loss increasing in the second half of training.

Table 2: *Training Summary for 1D Burgers' Scenario (Test-Selected)*
Note: Mean and standard deviation over 30 models.

Samples	n	Mean Epoch Time (s)	Mean Total Time (min)	Epoch Mean	Epoch SD	Test Loss Mean	Test Loss SD
50	30	0.32	2.67	473.87	40.97	5.07×10^{-5}	1.53×10^{-5}
100	30	0.64	5.32	422.70	73.81	2.27×10^{-5}	1.04×10^{-5}
200	30	1.25	10.44	267.00	90.80	2.19×10^{-5}	8.74×10^{-6}

4.2 Parareal Simulation Results

We evaluated $3 \times 5 = 15$ coarse-solver configurations. Several runs of the 1D Burgers' scenario were noticeably unstable, whereas the 2D Advection-Diffusion simulations

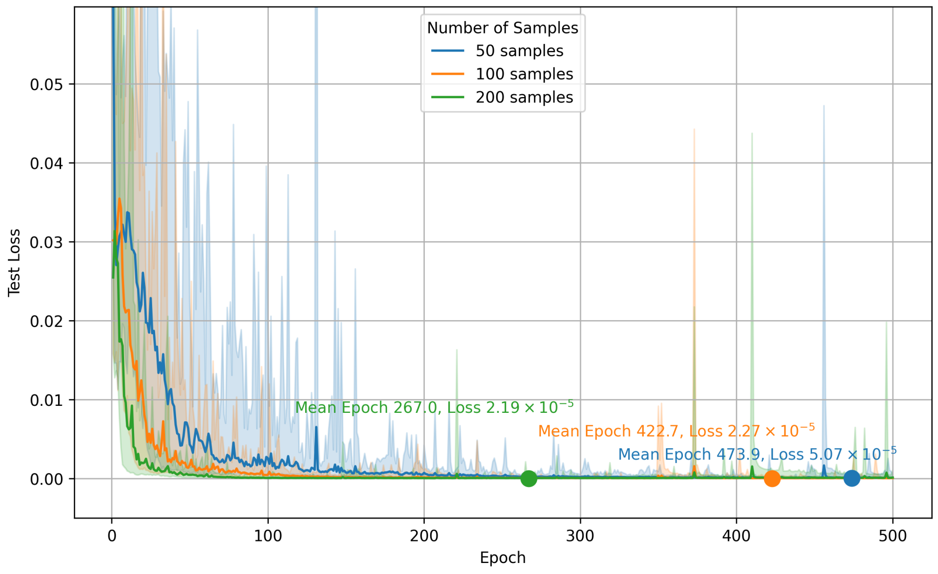
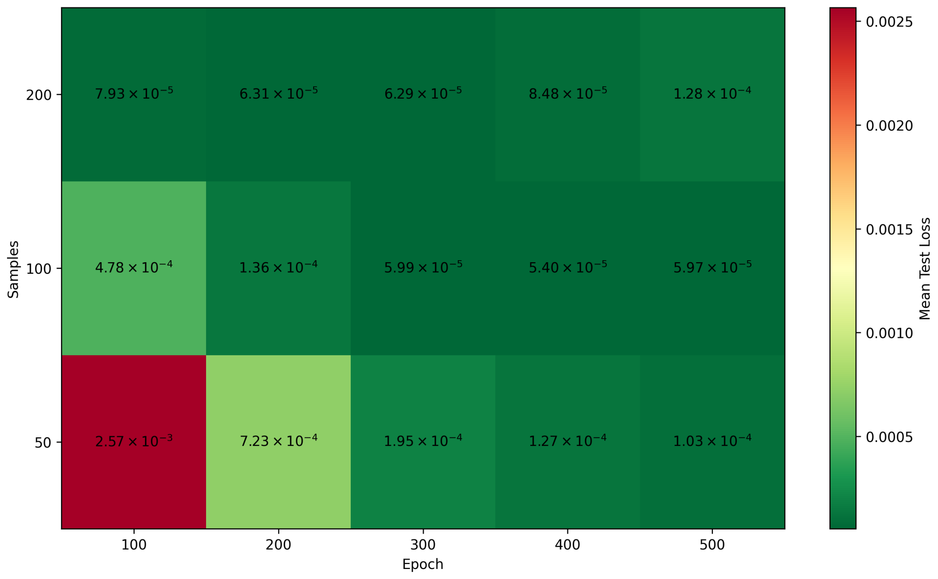


Figure 9: *Test Loss vs. Epochs for 1D Burgers' Scenario*
Note: The lines indicate the mean training loss vs. epochs for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models. Dots indicate the mean best epoch for each data size.



Samples \ Epoch	100	200	300	400	500
200	7.93×10^{-5}	6.31×10^{-5}	6.29×10^{-5}	8.48×10^{-5}	1.28×10^{-4}
100	4.78×10^{-4}	1.36×10^{-4}	5.99×10^{-5}	5.40×10^{-5}	5.97×10^{-5}
50	2.57×10^{-3}	7.23×10^{-4}	1.95×10^{-4}	1.27×10^{-4}	1.03×10^{-4}

Figure 10: *Test Losses of Checkpoints for Experiment 1 with 1D Burgers' Scenario*
Note: Test losses generally decrease with larger training sets and/or longer training. The increase at 200 samples and 400/500 epochs indicates overfitting. Means computed over 30 models

showed no such instabilities. Based on model test loss, we set evaluation thresholds $\epsilon_{ad} = 1.0 \times 10^{-7}$ and $\epsilon_{bg} = 1.0 \times 10^{-6}$. In the 2D Advection-Diffusion case, Parareal improves monotonically with data size and shows a weaker improvement with the number of epochs. Despite numerical instabilities, the 1D Burgers' simulations show partial improvement with the number of samples and a weaker or uncertain effect of the number of epochs.

4.2.1 2D Advection-Diffusion Scenario

Parareal simulations for the 2D Advection-Diffusion scenario produced no outliers or non-finite values. Therefore, all $3 \times 5 = 15$ unique configurations were included in the evaluation. Figure 11 and Figure 12 show mean squared error (averaged over seeds) stratified by training samples and epochs. At Parareal iteration 0, the MSE corresponds to the coarse initialization (i.e., the full sequential model rollout). Errors change most at the first and last iterations. Intermediate iterations are comparatively stable. Plateau levels reflect the coarse-solver configuration, and dispersion shrinks with increasing iteration count. At the final iteration, Parareal solutions are equal to the fine solve.

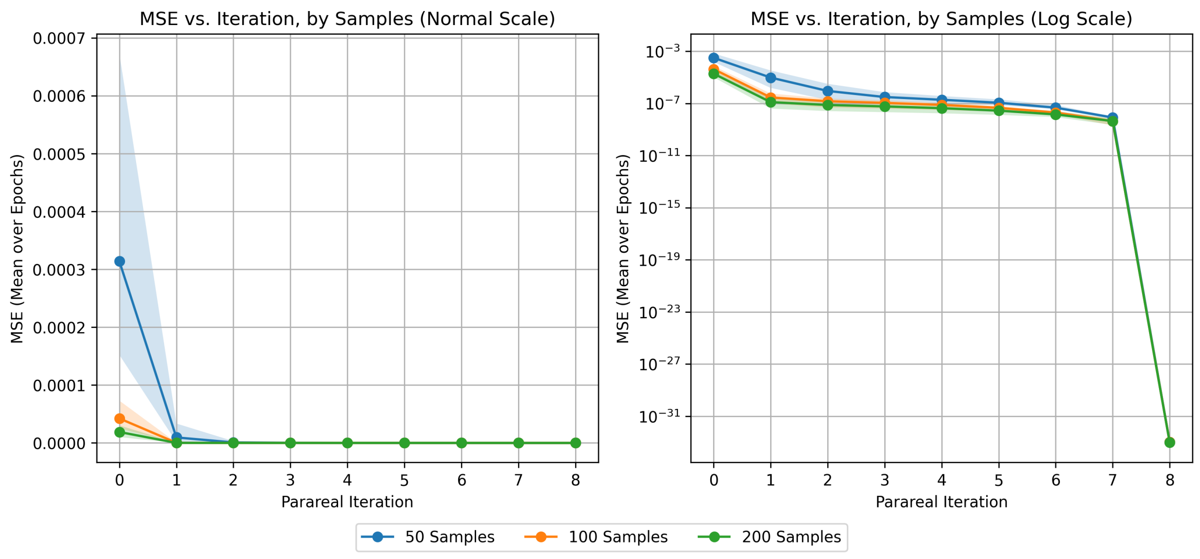


Figure 11: *Simulation MSE by Samples vs. Iteration for 2D Advection-Diffusion Scenario*
Note: Major drops in MSE occur at the first iterations (impact of the coarse solver accuracy) and at the last iterations (final convergence to the ground truth solution). A model trained on more samples yields a higher accuracy at initiation, resulting in a higher accuracy over the iterations (lower plateau). Mean and Standard Deviation over 30 Simulation runs, one run simulates 30 trajectories.

Figure 13 illustrates the error at Parareal iteration 0 (the model's coarse rollout). Although error generally decreases with model quality, models trained on 200 samples for 500 epochs show a renewed error increase. This translates to the error after the first predictor-corrector step (iteration $k = 1$). These errors were obtained from a fully time-parallel computation of the trajectories. The effect is also visible in the AUC (Figure 14). Overall, AUC improves with both samples and epochs.

The AUC was significantly negatively correlated with the number of training samples: Spearman $\rho = -0.926$, $p < .001$ and Pearson $r = -0.603$, $p = .009$. For the number of training epochs, we observed a non-significant trend: Spearman $\rho = -0.338$, $p = .109$ and Pearson $r = -0.367$, $p = .089$. Table 3 shows the results of the AUC correlation tests.

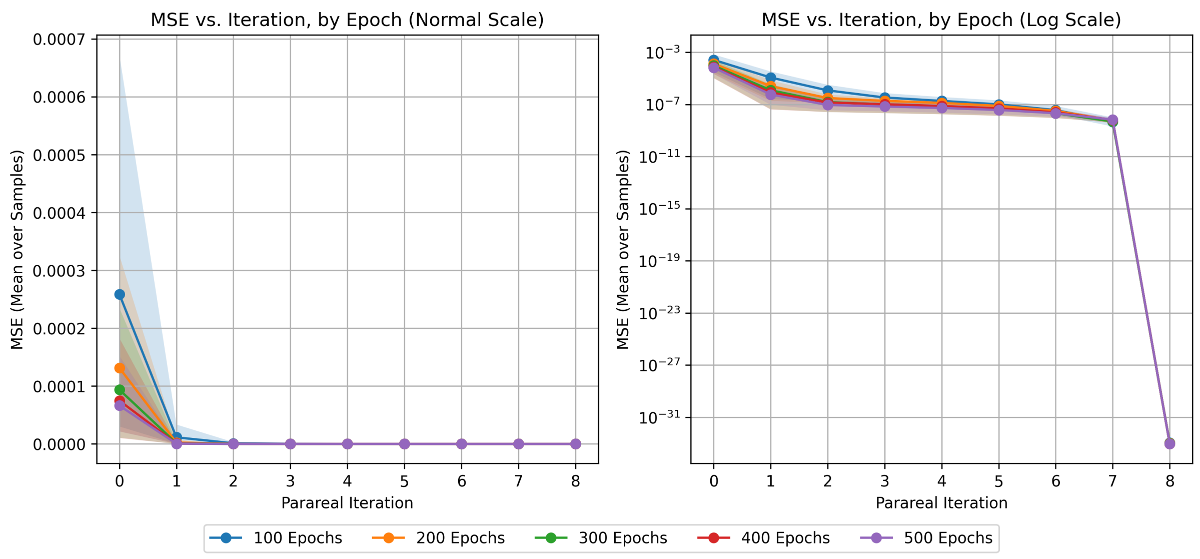


Figure 12: *Simulation MSE by Epoch for 2D Advection-Diffusion Scenario*
Note: Major drops in MSE occur at the first iterations (impact of the coarse solver accuracy) and at the last iterations (final convergence to the ground truth solution). A model trained for more epochs yields a higher accuracy at initiation, resulting in a higher accuracy over the iterations (lower plateau). Mean and Standard Deviation over 30 Simulation runs, one run simulates 30 trajectories.

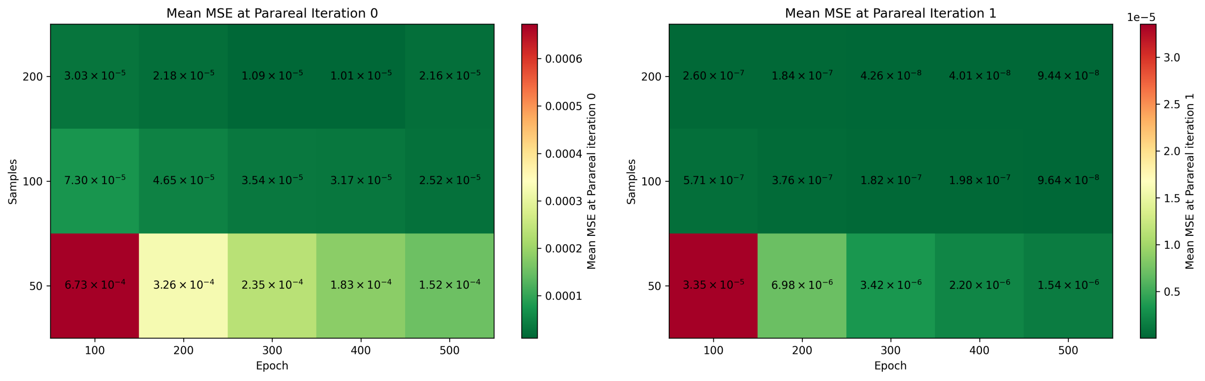


Figure 13: *Simulation MSE by Epoch and Samples at Parareal Iteration* 0 *and* 1 *for 2D Advection-Diffusion Scenario*
Left: MSE values of Parareal simulations after the coarse initialization. This equals the error of an independent model rollout over the trajectories in the test set. Right: MSE values of Parareal simulations after the first Parareal iteration. This equals the error after an entire parallel computation without iterative refinement. Mean over 30 Simulation runs, one run simulates 30 trajectories.

For the share-below-threshold metric ($\epsilon_{ad} = 1.0 \times 10^{-7}$), correlations with training samples were significant: Spearman $\rho = 0.850$, $p < .001$ and Pearson $r = 0.774$, $p < .001$. For training epochs, Pearson was significant ($r = 0.454$, $p = .044$), whereas Spearman indicated a trend ($\rho = 0.436$, $p = .052$). Throughout, we use $\alpha = .05$. Results are listed in Table 4.

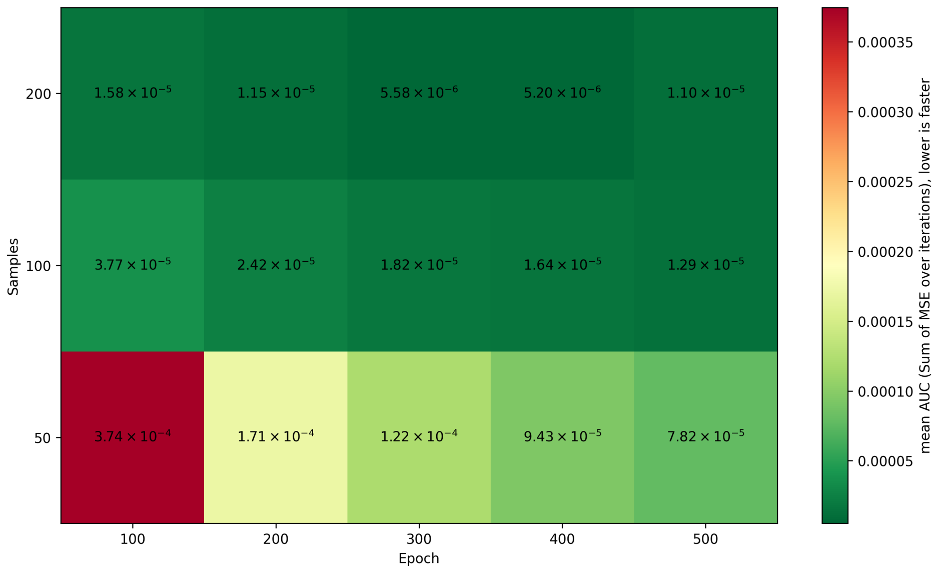


Figure 14: *AUC by Epoch and Samples for 2D Advection-Diffusion Scenario*
Note: A lower AUC value indicates higher speed (i.e., a satisfactory error can be achieved faster with a better model). The increase in AUC at 500 epochs and 200 samples is caused by a model that overfits. Mean over 30 Simulation runs, one run simulates 30 trajectories.

Table 3: *Correlation Simulation AUC with Model Training for 2D Advection-Diffusion Scenario*
Note: Correlation is computed over 15 configurations, where each was run with 30 models.

Correlation Pair	n	*df*	ρ / r	p	α
Samples vs. AUC (Spearman ρ)	15	13	-0.926	<.001*	.05
Epoch vs. AUC (Spearman ρ)	15	13	-0.338	.109	.05
Samples vs. AUC (Pearson r)	15	13	-0.603	.009*	.05
Epoch vs. AUC (Pearson r)	15	13	-0.367	.089	.05

Table 4: *Correlation Simulation Share-Below-Threshold with Model Training for 2D Advection-Diffusion Scenario*
Note: $\epsilon_{ad} < 1.0 \times 10^{-7}$. Correlation is computed over 15 configurations, where each was run with 30 models.

Correlation Pair	n	*df*	ρ / r	p	α
Samples vs. Share below ϵ (Spearman ρ)	15	13	0.850	<.001*	.05
Epoch vs. Share below ϵ (Spearman ρ)	15	13	0.436	.052	.05
Samples vs. Share below ϵ (Pearson r)	15	13	0.774	<.001*	.05
Epoch vs. Share below ϵ (Pearson r)	15	13	0.454	.044*	.05

4.2.2 1D Burgers' Scenario

During preprocessing of the Burgers' scenario, we observed that a subset of runs produced extremely large and sometimes non-finite values. Of the 15 configurations, 5 were excluded, reducing the set available for analysis. This had an impact on our reports, as some configurations were no longer available for analysis. Figure 15 and Figure 16 show how the error is reduced over the number of Parareal iterations, while the slope decreases. Numerical accuracy varied largely over the first iterations for the models that were trained on 100 samples.

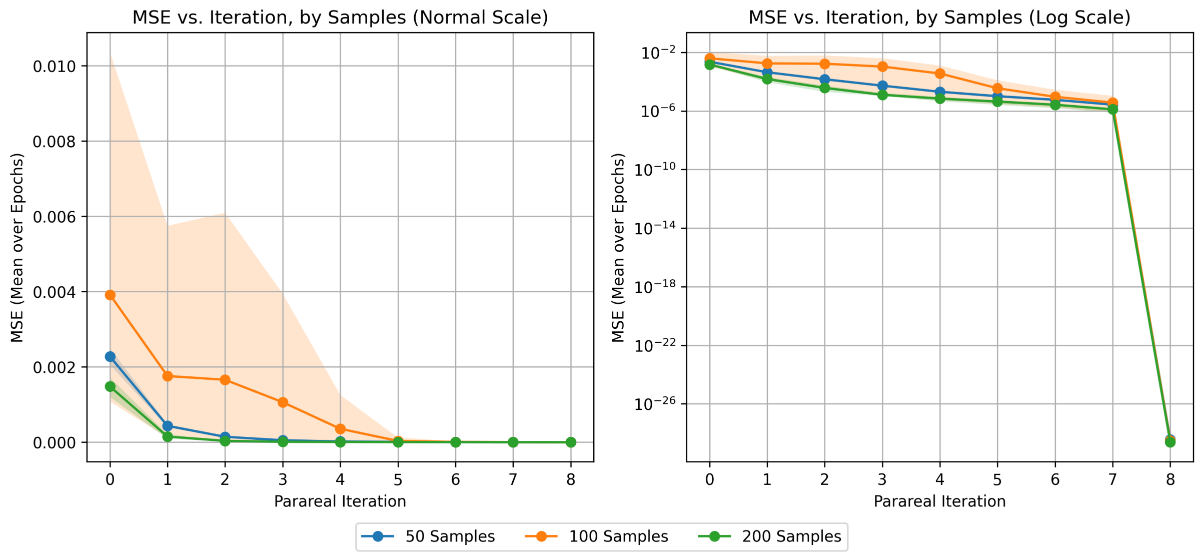


Figure 15: *Simulation MSE by Samples vs. Iteration for 1D Burgers' Scenario*
Note: Drops in MSE occur at the first iterations (impact of the coarse solver accuracy) and at the last iterations (final convergence to the ground truth solution). Deviations of simulations with models trained with 100 samples are caused by low quality model. Mean and Standard Deviation over 30 Simulation runs, one run simulates 30 trajectories.

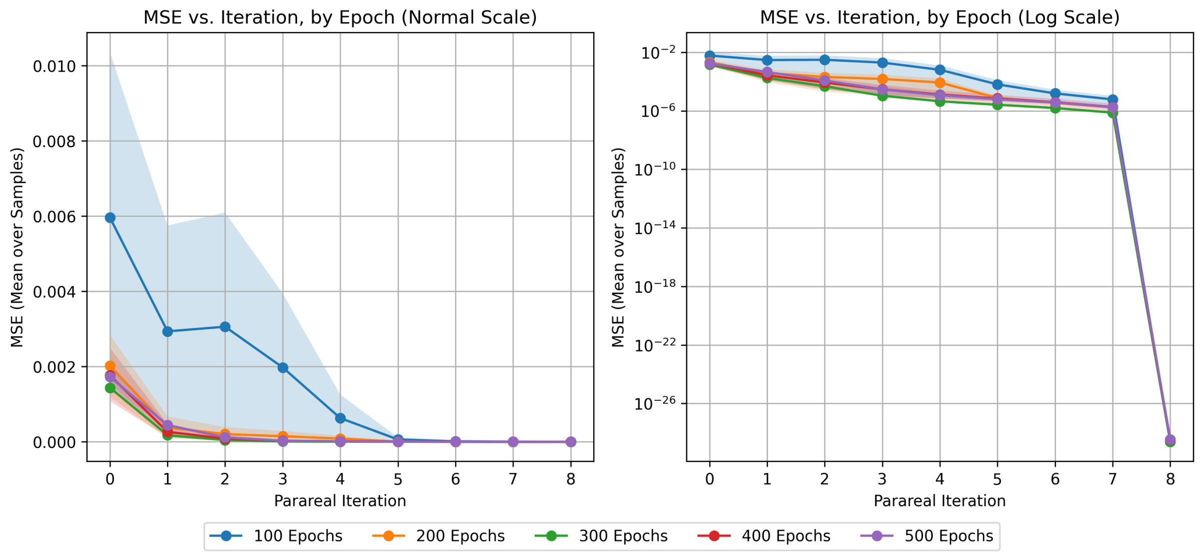


Figure 16: *Simulation MSE by Epoch vs. Iteration for 1D Burgers' Scenario*
Note: Drops in MSE occur at the first iterations (impact of the coarse solver accuracy) and at the last iterations (final convergence to the ground truth solution). Deviations of simulations with models trained over 100 epochs are caused by low quality model. Mean and Standard Deviation over 30 Simulation runs, one run simulates 30 trajectories.

The impact of numerical instabilities is also reflected in the AUC speed metric (see Figure 17). Empty tiles illustrate configurations that had to be excluded as considered unstable. While simulations using models trained with 200 samples seemed to work best, there was no clear tendency. Our indicated a modest but significant negative association between AUC and training samples (Spearman $\rho = -0.571$, $p = .042$). Other than a tendency of a mild linear correlation of the AUC with the number of epochs ($r = -0.482$, $p = .079$), tests have not yielded any other notable results (Table 5). Error values of simulation runs with initialization and after the first iteration are found in the appendix.

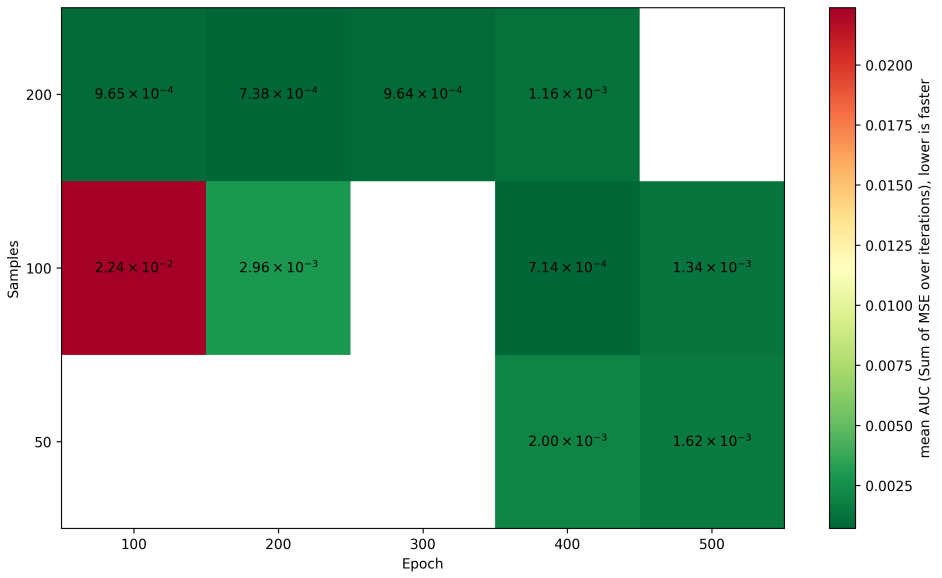


Figure 17: *AUC by Epoch and Samples for 1D Burgers' Scenario*
Note: A lower AUC value indicates higher speed (i.e., a satisfactory error can be achieved faster with a better model). Missing values stem from excluded outlier configurations. The increase in AUC at 400 and 500 epochs for 200 samples is caused by models that overfit. Mean over 30 Simulation runs, one run simulates 30 trajectories.

Table 5: *Correlation Simulation AUC with Model Training for 1D Burgers' Scenario*
Note: Correlation is computed over 10 configurations, where each was run with 30 models.

Correlation Pair	n	*df*	ρ / r	p	α
Samples vs. AUC (Spearman ρ)	10	8	-0.571	.042*	.05
Epoch vs. AUC (Spearman ρ)	10	8	-0.081	.413	.05
Samples vs. AUC (Pearson r)	10	8	-0.222	.269	.05
Epoch vs. AUC (Pearson r)	10	8	-0.482	.079	.05

A similar result is at hand for the share-below-threshold tests using $\epsilon_{bg} = 1.0 \times 10^{-6}$. We observed a positive correlation with the number of samples: Spearman $\rho = 0.610$, $p = .031$ (significant at $\alpha = .05$) and Pearson $r = 0.476$, $p = .082$ (trend level). In contrast, there was no evidence for an association with training epochs (Spearman $\rho = 0.198$, $p = .292$ and Pearson $r = 0.297$, $p = .202$, see Table 6 for details).

Table 6: *Correlation Simulation Share-Below-Threshold with Model Training for 1D Burgers' Scenario*
Note: $\epsilon_{bg} < 1.0 \times 10^{-6}$. Correlation is computed over 10 configurations, where each was run with 30 models.

Correlation Pair	n	*df*	ρ / r	p	α
Samples vs. Share below ϵ (Spearman ρ)	10	8	0.610	.031*	.05
Epoch vs. Share below ϵ (Spearman ρ)	10	8	0.198	.292	.05
Samples vs. Share below ϵ (Pearson r)	10	8	0.476	.082	.05
Epoch vs. Share below ϵ (Pearson r)	10	8	0.297	.202	.05

4.3 Investigation of Outliers

Out of our experiments with the non-linear 1D Burgers' scenario, five configurations were excluded from analysis. A more detailed analysis shows divergent behaviour during Parareal simulations. Figure 18 and Figure 19 show errors increasing with interval index during intermediate iterations until convergence at the final iteration. Later intervals (i.e., later time steps) are particularly affected. We attribute this to inferior model quality combined with the non-linearity of Burgers' equation and rule out numerical-precision issues, as our setup uses `float64` throughout.

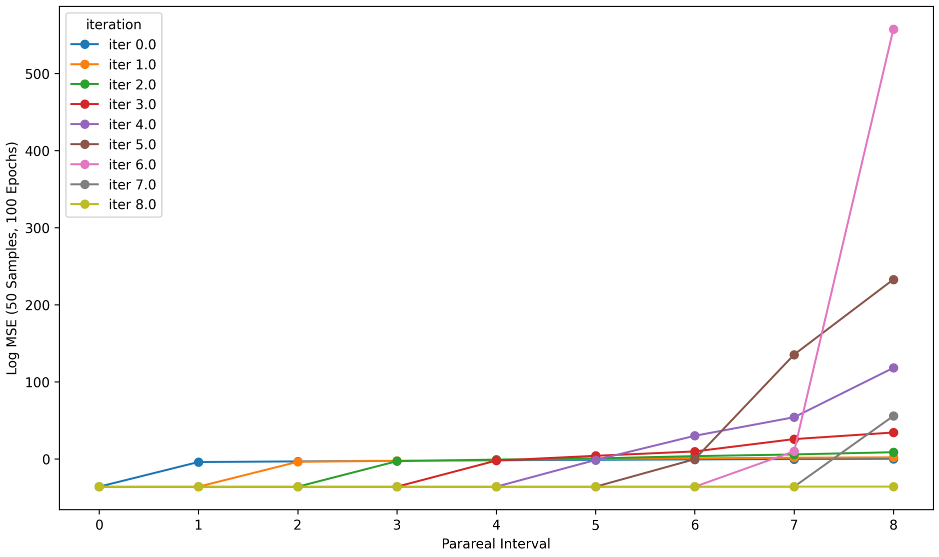


Figure 18: *Diverging Simulation for 1D Burgers' Scenario*
Note: Outliers show a significant increase in error during later iterations (where the model input is expected to show greater difference from the training data) and at the last intervals (where coarse model inference becomes less accurate). Eventually, the algorithm converges, as the solution becomes equal to the one of a fine solve. Values are for an example outlier configuration at 50 samples and 100 epochs. Mean over 30 Simulation runs, one run simulates 30 trajectories.

During Parareal, later intervals depend on solutions from earlier ones. At early iterations, solutions at later intervals come almost entirely from the coarse solver. In the predictor-corrector step, deviations can be magnified, increasing with the number of in-

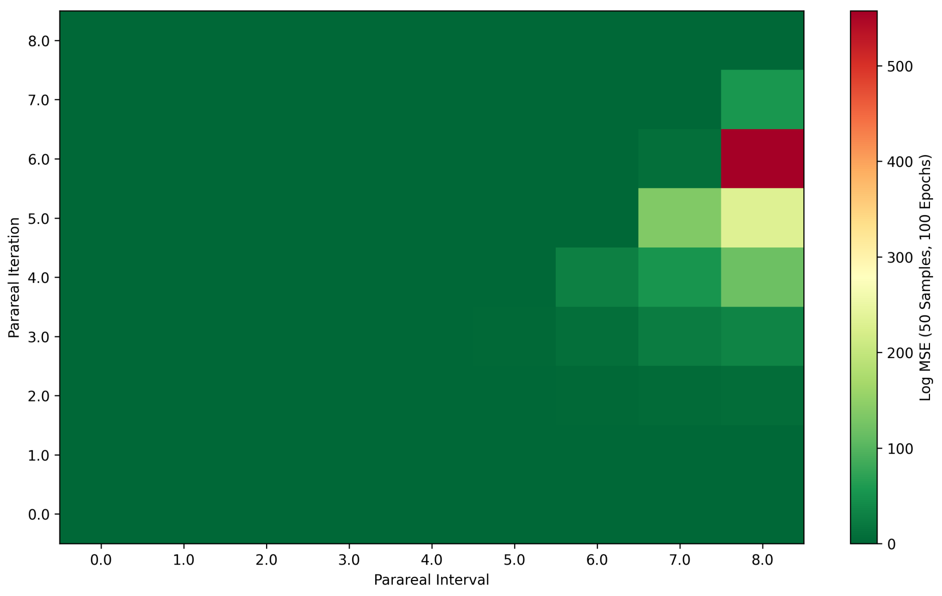


Figure 19: *Blow-Up of Simulation for 1D Burgers' Scenario*
Note: Large error values occur especially at later intervals and during later iterations. Values are for an example outlier configuration at 50 samples and 100 epochs. Mean over 30 Simulation runs, one run simulates 30 trajectories.

tervals. If the coarse solution is not sufficiently close to the fine solution, updates may increase rather than reduce the error in intermediate iterations, even though the method converges eventually. This behaviour is consistent with a convergence factor similar to the ρ proposed by M. J. Gander and Vandewalle (2007), see Equation 2.19.

In addition, Burgers' equation features non-linearity and shocks. Although we simulate the viscous case (preventing shock formation), intermediate Parareal solutions may include physical features the model does not yet represent due to limited training, or modes absent from the training data, leading to divergent propagation over the trajectory.

Steiner et al. (2015) showed a dependence of Parareal convergence for the non-linear Navier-Stokes equation on the Reynolds number (i.e., viscosity) and spatial discretization. This may also apply to the 1D viscous Burgers' scenario we used, which is closely related via its diffusive and convective terms. With the APEBench default viscosity $\nu = 3.0 \times 10^{-4}$ and a spatial discretization of 256 points, the same factors could have contributed to the observed behaviour.

4.4 Discussion

Our results indicate a strong dependence on coarse-solver model quality within Parareal. The model impacts convergence speed (AUC), numerical accuracy (share-below-threshold), and numerical stability. We cannot fully generalise across scenarios. However, both cases clarify what drives the benefit of the hybrid setup. Differences appear to stem more from linearity vs. non-linearity than from dimensionality.

Data Size vs. Epochs

Results give clear indications in terms of quantitative factors. When holding epochs fixed and increasing data size, AUC decreases and share-below-threshold increases to a significant extent using Spearman tests. This is true for both, linear and non-linear scenarios, while linear scenarios also show a significant correlation for the Pearson test. Conversely, holding data size fixed and varying epochs, correlations show a weaker, sometimes ambiguous effect (clearer in the linear case, mixed in the nonlinear case). This suggests representation quality and data size the main drivers of applicability of the hybrid approach. More samples help robustness of the solver, whereas longer training yields diminishing returns once it saturates, resulting in insignificant Pearson correlation. It aligns with the findings of Ibrahim et al. (2024), where the Fourier Neural Operator is an effective coarse model, but longer training time improves Parareal's convergence only marginally. It also motivates the approach of Pamela et al. (2024), where the the model was further trained on the data generated with simulations. Furthermore, this aligns partly with our hypothesis: Effects are clearly noticeable, however still not entirely generalizable throughout our setup and beyond. Moreover, it implies a limit to the extent the FNO accuracy can accelerate Parareal, beyond which more training yields little additional speed-up.

Transfer from Training Metrics to Parareal Utility

Model test loss is a useful indicator of Parareal performance, but not stability. In practice, instead of simulating all combinations, one can use model test loss as a quality indicator. Our experiment therefore confirms that a lower test loss and a smaller generalization gap translate into faster convergence (lower AUC) and higher accuracy (greater share-below-threshold). Nevertheless, it shows that a good stand-alone FNO perspective by test loss may still be an unreliable coarse solver in vanilla Parareal. Although more samples expand mode coverage during training, Parareal may generate intermediate states with frequencies beyond what the coarse solver model represents, propagating large errors. For non-linear scenarios, our outlier analysis shows this behaviour and implies the need for close alignment between coarse and fine solvers and the target scenario. Taking a look at convergence factor theorem proposed by M. J. Gander and Vandewalle (2007), we propose the model test loss as an informal indicator of the coarse solver's performance to a certain extent. However, to be quantified for the given problem individually, as it does not guarantee overall stability.

Monotonicity vs. Over-Training Artifacts

Artifacts of over-trained models impact simulation behaviour. Although we can confirm a monotone improvement for a model trained on more data in the linear case, training remains highly sensitive in terms of problem and extent. In the 2D Advection-Diffusion, initialization and iteration-1 error increase at 200 samples $\times$ 500 epochs compared to earlier checkpoint suggests mild overfitting. Although Parareal eventually converges, the overall initialization quality degrades performance in terms of a larger AUC and a smaller share-below-threshold metric over the entire simulation. When comparing with model test losses, configurations of poor simulation runs can be linked to inferior model quality. Again, we propose the model test loss as an indicator for Parareal simulation speed and accuracy, but not for stability of the setup.

Iterative Behaviour

Results align with Parareal's efficiency bound. Most gains across iterations occur from iteration 0 to 1 and near the final iteration, while intermediate iterations show plateaus. This reflects the limited information propagation per correction step and again highlights the key role of the coarse initialization for practicality. In the linear case, larger data sizes increase the chance that an accuracy criterion is already met after the first iteration, thereby enabling potential speed-up (in our showcase even entire in-time parallelization). In our non-linear scenario, runs benefit less uniformly, yet show a relationship to coarse solver robustness.

The large plateaus themselves are typical for Parareal. However, they lead to numerous computations of less impact, thus being less useful, unless desired anyway. In a productive setting, additional simulation data created during intermediate iterations inherently could be beneficial. As shown by Pamela et al. (2024), an iterative model retraining on fine solver results of all iterations leads to higher accuracy in future simulations, thus more speed-up. This aligns with our findings, showing an increase in speed and accuracy for a larger data size the model has been trained on.

Recursive Dependency

Parareal's success relies on representative training data, simultaneously producing it. In our setting, a representative model has strong impact on the provided speed-up. The discovered dependency on variability in training data is not new, but rather a common pattern with Deep Neural Networks. For neural operators in specific, acquisition of training data can become very expensive, as it necessitates intensive computation. Thus, this again brings up the problem that neural operators initially were intended for avoiding it (Chen et al., 2025; Lu et al., 2022). With Parareal conversely, a vast amount of simulation data is generated inherently (although most often discarded). This makes it a good fit for specific simulations, as shown in the advanced approach of Pamela et al. (2024), where the FNO is iteratively retrained on the by-product data, resulting in additional acceleration.

Hypothesis

Overall, our results support H_A for the linear PDE and partially for the non-linear PDE. Better models trained on more diverse data require fewer Parareal iterations to reach a fixed accuracy, whereas additional epochs offer diminishing returns and can even harm stability.

4.5 Limitations

We evaluate Parareal in its original version proposed by Lions et al. (2001). All results are obtained on synthetic, normalized data provided by the APEBench library with fixed grids and Δt. Cross-resolution shifts, initial condition distribution shifts, and different spatial discretizations remain untested. Our coarse architecture is fixed to the one proposed by Li et al. (2021), thus we do not explore any alternative inductive biases (e.g., additional input channels for coefficients and/or step size). This is also the case for our Parareal setup, as the number of intervals was matched to available GPUs, which had an impact on scalability of our experiments. We also do not assess multi-step coarse rollouts, which could impact accuracy trade-offs.

As thresholds $\epsilon_{scenario}$ are tied to the scenario, a comparison across problems becomes more complicated. This emphasises within-problem conclusions, while conclusions from linear results to non-linear ones cannot be made. In a production setup, thresholds need to be aligned with acceptance criteria of the problem. Furthermore, outlier filtering in the Burgers' scenario reduces instability bias on the one hand, but also reduces statistical power favouring more stable configurations. We use MSE, AUC, and share-below-threshold, while alternative metrics could change the utility of the approach.

5 Experiment 2: Parareal Speed-Up

Simulations of the second experiment are run using the closest to test-selected best checkpoints saved at 490 epochs for 50 samples, 480 epochs at 100 samples, and 470 epochs at 200 samples (see Table 1), while varying the number of Parareal intervals N (1 / 2 / 4 / 8). Models of experiment 2 therefore had the mean times-to-epoch and test losses given in Table 7. A detailed description of training results is provided in subsection 4.1.

Table 7: *Models utilized as Coarse Solver in Experiment 2*
Note: Mean and standard deviation are computed across 30 models.

Samples	Epoch	n	Mean Time-to-Epoch (s)	Test Loss M	Test Loss SD
50	490	30	272.36	4.89×10^{-5}	2.57×10^{-5}
100	480	30	1103.08	6.33×10^{-6}	2.27×10^{-6}
200	470	30	1870.82	1.74×10^{-6}	7.40×10^{-7}

5.1 Lower Boundary Derivation

Parareal speed-up in terms of time is achieved for a significantly smaller number of iterations than intervals, $k \ll N$, and a significantly shorter interval runtime of coarse solver compared to the fine one, $\tau_{coarse} \ll \tau_{fine}$ (M. J. Gander & Vandewalle, 2007; Minion, 2010). We define the within-algorithm runtime for both solvers T_{fine} and T_{coarse} from Equation 2.18. P is the number of parallel processing units available,

$$T_{coarse} = (k+1)N\tau_{coarse}, \quad T_{fine} = k\frac{N\tau_{fine}}{P}.$$

The additional $k+1$ iteration of the coarse solver is introduced with the sequential coarse initialization. We therefore present an estimate for the theoretical minimum runtime that could be achieved with full parallelization using individual solver runtimes.

Table 8: Sequential Runtimes of Solvers τ_{coarse} and τ_{fine}

	M	SD
Sequential Coarse Runtime per Interval (s)	0.0118	0.0007
Sequential Fine Runtime per Interval (s)	0.0071	0.0005

Mean runtimes of our solvers per interval (i.e., per time step Δt) are shown in Table 8. We therefore expect the lower bound on the net runtime per simulation run (30 trajectories) for $k = 8, N = 8, P = 8$ at

$$T_{lower} = 30 \cdot \left((k+1)N\tau_{coarse} + k\frac{N\tau_{fine}}{P} \right) \approx 25.61\text{s}.$$

This boundary does not reflect communication and synchronization overhead, but only the net runtimes of the solvers within Parareal. In our case, the fine solver is faster than the coarse one. As it is applied sequentially and does not depend on the number of processing units, its runtime becomes the main driver of our lower boundary value.

5.2 Runtime Results

Runtime results highlight a trade-off between the degree of parallelization and the accuracy of the coarse solver. As shown in the left panel of Figure 20, the Parareal runtime-to-threshold (green bars) generally decreases as the number of samples, S, increases (from 50 to 100 to 200) for a fixed number of intervals. This trend is particularly visible when comparing the runtimes for N=8 across the three sample sizes. A more accurate coarse solver provides a better initial guess, reducing the number of Parareal iterations required to reach the convergence threshold.

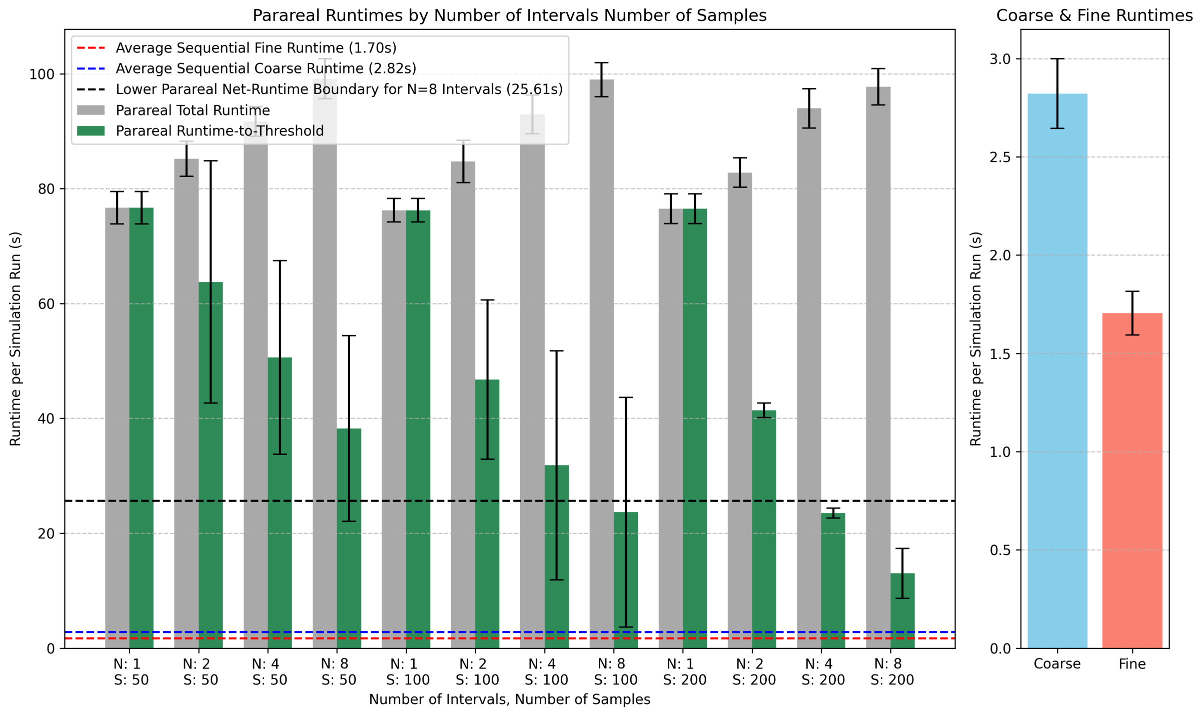


Figure 20: *Simulation Runtimes by Number of Intervals and Number of Samples*
Left: Grey bars illustrate the total Parareal runtime in seconds for each configuration of data size (S) and Parareal intervals (N). Green bars illustrate the Parareal runtime needed to achieve a simulation run error below threshold τ_{ad}. The blue (coarse) and red (fine) dotted lines indicate the mean sequential solver runtime. The black dotted line indicates the lower Parareal runtime boundary, where Parareal would have been executed for 8 iterations with parallelization over 8 processing units, ignoring synchronization overhead. Right: Detailed view of sequential runtimes for the coarse (blue) and the fine (red) solver. Note: The fine solver is significantly faster than the coarse one. For no configuration of intervals and epochs, Parareal speed-up is achieved when compared to a sequential fine solver. Mean and Standard Deviation over 30 Simulation Runs, one run simulates 30 trajectories.

The results also illustrate a significant increase in total Parareal runtime (grey bars) as the number of intervals increases from N=1 to N=8 for a given sample size. This is because a higher number of intervals introduces more communication and overhead between processors, which can slow down the overall computation. The lower Parareal net-runtime boundary represents the theoretical minimum total runtime without overhead. It is clear that the Parareal runtimes for N=8 are significantly higher than this boundary, suggesting that overheads become a dominant factor at higher levels of parallelization with our solver times. A comparison of the individual sequential solver times (right-hand panel) shows a significant discrepancy, with the coarse solver being outperformed by the fine one, M=2.8223s (SD=0.1772s) and M=1.7045s (SD=0.1114s). As the runtime-to-threshold does not undercut a sequential fine application for any of our configurations, the overall algorithm application becomes obsolete.

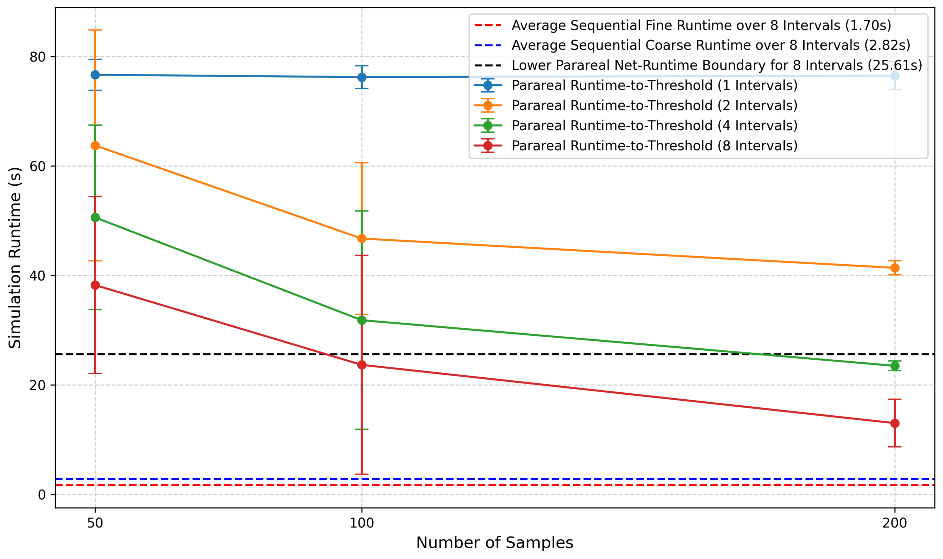


Figure 21: *Simulation Runtimes vs. Number of Samples*
Note: Lines illustrate the Parareal runtime in seconds needed to achieve a simulation run error below threshold τ_{ad} for the number of intervals by number of samples the coarse model was trained on. The blue (coarse) and red (fine) dotted lines indicate the mean sequential solver runtime. The black dotted line indicates the lower Parareal runtime boundary. For no configuration of intervals and epochs, Parareal speed-up is achieved when compared to a sequential fine solver. Mean and Standard Deviation over 30 Simulation Runs, one run simulates 30 trajectories.

Figure 21 shows the same trend. At a fixed number of intervals, increasing the number of training samples reduces runtime-to-threshold This reflects the better-trained coarse solver providing a stronger initialization. The effect is especially clear for 4 and 8 intervals, where the runtime-to-threshold drops noticeably when the number of samples increases from 50 to 200. Again, no runtime advantage is realized overall because the fine solver is significantly faster than the coarse solver.

5.3 Discussion

In conclusion, the performance of the Parareal algorithm is highly dependent on the balance between parallelization and the quality of the coarse solver. However, in our setup Parareal does not outperform the sequential fine solver baseline in any configuration. We mainly relate this to the high speed of the fine solver. Although the number of Parareal iterations needed to match our accuracy criterion is reduced noticeably with a higher-quality coarse solver, this behaviour does not result in a runtime reduction to a beneficial level.

Solver Trade-Off

The combination of solvers in our setup did not meet the speed-up condition. As stated in Equation 2.18, the coarse solver runtime needs to be significantly smaller than the fine one in order to provide speed-up (M. J. Gander & Vandewalle, 2007; Minion, 2010). This stems from the coarse solver being applied serially in every iteration, while only the parallelization of the fine solver is possible. During preparation of our work, we have identified the ETDRK method provided by the `exponax` library as a critical factor. This made the fine solver faster than the coarse model forward pass, making it less suitable for runtime comparison on our problems. Because our scenarios are relatively simple, we could not demonstrate a wall-clock benefit from Parareal. Given the additional overhead the algorithm introduces, the gap between the baseline fine solver runtime and Parareal becomes even larger for our solvers.

Quality Impact

Simulation acceleration is closely tied to model training. In terms of absolute runtimes, runs using models trained on more samples outperform those trained on fewer samples. Referring back to the test losses of the utilized models in Table 7, we again propose lower test loss as an indicator for shorter wall-clock time. However, test loss alone is not sufficient to predict Parareal success. Instead, it always has to be considered in relation to the fine solver applied.

Accuracy Tuning

Coarse-solver accuracy is not directly proportional to simulation runtime. With our fine solver, higher accuracy can be achieved via finer temporal discretisation per interval, which increases computations and runtime (Kassam & Trefethen, 2005). By contrast, our neural model is tied to the step size it was trained on, making accuracy primarily a function of training. For runs over eight intervals, runtime-to-threshold drops by only a factor of around 3 across sample counts, whereas time-per-epoch increases by nearly 7 times. Training time is therefore a critical design choice, data acquisition to be prioritized.

Hypothesis

Under our current implementation, τ_{coarse} is not sufficiently smaller than τ_{fine}. Hence, we do not observe a wall-clock speed-up and cannot support H_A. Future work should target the ratio of the solver runtimes and communication overhead to reach speed-up through Parareal.

5.4 Limitations

Again, we evaluate the Parareal algorithm performance using synthetic, normalized data provided by the APEBench library, which can limit the direct applicability of our findings to other problems. The effectiveness of the Parareal algorithm is highly dependent on the relative speed of the solvers and our decision for the threshold. Our choice led to the case that the fine solver was unexpectedly performant, which made it challenging to observe any speed-up conditions.

The experiments were constrained by the specific cluster architecture and the number of available GPUs. The observed runtimes may be subject fluctuations in capacity, thus may not be directly reproducible. Despite the effort to mitigate timing variability by warm-up calls and aggregating results from multiple runs, slight deviations in runtime may occur. While these are not expected to impact the fundamental conclusion, they are still to be considered when interpreting the results.

6 Conclusion

Overall, our empirical results confirm that FNO quality matters most via data size, while the number of training epochs is secondary. A larger training set consistently led to improved Parareal convergence (lower AUC, higher share below the threshold).Additional epochs had diminishing effects and sometimes even impaired performance, as indicated by overfitting at high epochs for large datasets. For the linear scenario, evidence for the impact of model quality is context-dependent, while for the non-linear scenario, the evidence is insufficient. In all tested settings, results show a reduced simulation runtime with models trained on more data. Nevertheless, our setup did not show a wall-clock speed-up because the chosen fine solver was too fast and overheads dominated. A lower FNO test MSE generally predicted faster Parareal convergence and better accuracy, but did not guarantee stability of the hybrid approach. For our non-linear scenario, divergence appeared especially in later intervals and iterations, and its occurrence became more frequent with weaker coarse models.

Thus, we answer our questions with: Convergence, accuracy, **and stability** of the hybrid Parareal setup are strongly influenced by the FNO in terms of training data size and variety. The approach may only be beneficial in settings where representative data is available and must be fine-tuned to the given circumstances, as was done by Ibrahim et al. (2024) and Pamela et al. (2024). We can neither confirm any beneficial application in terms of speed-up, nor reject it. A speed increase exists with a higher-quality model; nevertheless, it is still limited by the methods used within Parareal.

We again emphasize the impact of training data and point to the iterative approach of Pamela et al. (2024). Using intermediate fine solutions for training to expand FNO mode coverage could eliminate the limitations we have discovered. This may also reduce the dependence on chosen initial conditions, thereby allowing us to draw more generalizable conclusions. We also suggest to revisit the speed-up with a slower fine solver. Our results show how a model trained on more data affects the algorithm's wall-time. Since the model inference time does not change with training but only with its architecture, a proper test under the speed-up condition might reveal a beneficial application, similar to the findings of Ibrahim et al. (2024).

Secondly, we recommend a replication exploring different FNO variants, for instance, additional training on step size and coefficients, or using multi-step rollout. Such approaches could further strengthen stability and yield higher accuracy.

The hybrid combination of the Fourier Neural Operator within the Parareal algorithm shows a strong dependence on the amount and variety of data the model was trained on. Its general usefulness is strongly scenario dependent. A stand-alone FNO remains a strong surrogate, but making the hybrid setup practical in terms of wall-time likely requires high-quality training data, iterative retraining, and a much faster ratio between the coarse and fine solver.

References

Chen, W., Song, J., Ren, P., Subramanian, S., Morozov, D., & Mahoney, M. W. (2025). Data-efficient operator learning via unsupervised pretraining and in-context learning. https://arxiv.org/abs/2402.15734

Evans, L. C. (2010). *Partial differential equations* (2nd ed.). American Mathematical Society.

Franck, E., Hoelzl, M., Lessig, A., & Sonnendrücker, E. (2015). Energy conservation and numerical stability for the reduced mhd models of the non-linear jorek code. https://arxiv.org/abs/1408.2099

Gander, M., & Halpern, L. (2007). Minisymposium 5: Space-time parallel methods for partial differential equations. In O. B. Widlund & D. E. Keyes (Eds.), *Domain decomposition methods in science and engineering XVI* (pp. 265–265). Springer. https://doi.org/10.1007/978-3-540-34469-8_30

Gander, M. J., & Vandewalle, S. (2007). Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, *29*(2), 556–578. https://doi.org/10.1137/05064607X

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

Gorynina, O., Legoll, F., Lelièvre, T., & Perez, D. (2022). Combining machine-learned and empirical force fields with the parareal algorithm: Application to the diffusion of atomistic defects. https://arxiv.org/abs/2212.10508

Hairer, E., Nørsett, S. P., & Wanner, G. (1993). *Solving ordinary differential equations i: Nonstiff problems* (2nd rev. ed.). Springer-Verlag. https://doi.org/10.1007/978-3-540-78862-1

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Ibrahim, A. Q., Götschel, S., & Ruprecht, D. (2024). Space-time parallel scaling of parareal with a Fourier neural operator as coarse propagator. https://doi.org/10.48550/arXiv.2404.02521

Kassam, A.-K., & Trefethen, L. N. (2005). Fourth-order time-stepping for stiff PDEs. *SIAM Journal on Scientific Computing*, *26*(4), 1214–1233. https://doi.org/10.1137/S1064827502410633

Koehler, F., Niedermayr, S., Westermann, R., & Thuerey, N. (2024). Apebench: A benchmark for autoregressive neural emulators of PDEs. https://arxiv.org/abs/2411.00180

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2023). Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, *24*(89), 1–97. http://jmlr.org/papers/v24/21-1524.html

Legoll, F., Lelièvre, T., & Sharma, U. (2022). An adaptive parareal algorithm: Application to the simulation of molecular dynamics trajectories. *SIAM Journal on Scientific Computing*, *44*(1), B146–B176. https://doi.org/10.1137/21M1412979

LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems*. Society for Industrial & Applied Mathematics.

Li, Z., Kovachki, N., Azizzadenesheli, K., Stuart, A., Anandkumar, A., Liu, B., & Bhattacharya, K. (2021). Fourier neural operator for parametric partial differential equations. *Proceedings of the International Conference on Learning Representations (ICLR)*. https://doi.org/10.48550/arXiv.2010.08895

Lions, J.-L., Maday, Y., & Turinici, G. (2001). Résolution d'edp par un schéma en temps « parareéel » [“parareal” in time discretization of PDEs]. *Comptes Rendus de l'Académie des Sciences – Series I – Mathematics*, *332*, 661–668. https://doi.org/10.1016/S0764-4442(00)01793-6

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, *3*(3), 218–229. https://doi.org/10.1038/s42256-021-00302-5

Lu, L., Pestourie, R., Johnson, S. G., & Romano, G. (2022). Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, *4*(2), 023210. https://doi.org/10.1103/PhysRevResearch.4.023210

Minion, M. L. (2010). A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, *5*(2), 265–300.

Montanelli, H., & Bootland, N. (2020). Solving periodic semilinear stiff PDEs in 1D, 2D and 3D with exponential integrators. https://arxiv.org/abs/1604.08900

Pamela, S. J. P., Carey, N., Brandstetter, J., Akers, R., Zanisi, L., Buchanan, J., Gopakumar, V., Hoelzl, M., Huijsmans, G., Pentland, K., James, T., Antonucci, G., & JOREK Team. (2024). Neural-parareal: Dynamically training neural operators as coarse solvers for time-parallelisation of fusion MHD simulations. https://doi.org/10.48550/arXiv.2405.01355

Rudi, J., Malossi, A. C. I., Isaac, T., Stadler, G., Gurnis, M., Staar, P. W. J., Ineichen, Y., Bekas, C., Curioni, A., & Ghattas, O. (2015). An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in Earth's mantle. *Proceedings of*

SC '15: International Conference for High Performance Computing, Networking, Storage and Analysis. https://doi.org/10.1145/2807591.2807675

Saltz, J. H., & Naik, V. K. (1988). Towards developing robust algorithms for solving partial differential equations on MIMD machines. *Parallel Computing*, *6*, 19–44. https://doi.org/10.1016/0167-8191(88)90003-8

Steiner, J., Ruprecht, D., Speck, R., & Krause, R. (2015). Convergence of parareal for the Navier–Stokes equations depending on the Reynolds number. In A. Abdulle, S. Deparis, D. Kressner, F. Nobile, & M. Picasso (Eds.), *Numerical mathematics and advanced applications — enumath 2013* (pp. 195–202). Springer International Publishing.

Appendix

FNOModel2D Forward Pass

```
**Notation:**
    $B$: batch size, $N_x, N_y$: spatial sizes
    $T_h$: time_history, $T_f$: time_future, $C_{in}$: input channels per step, $C$: model width
    Input $x \in \mathbb{R}^{B\times N_x\times N_y\times T_h\times C_{in}}$
    Pointwise linear layers: $\mathrm{fc0} : \mathbb{R}^{T_h C_{in}+2} \to \mathbb{R}^{C}$, $\mathrm{fc1} : \mathbb{R}^{C} \to \mathbb{R}^{2C}$, $\mathrm{fc2} : \mathbb{R}^{2C} \to \mathbb{R}^{T_f}$
    Fourier blocks: FNOConv2D($C \to C$) uses rfft2/irfft2 and low-mode weights
    Skip blocks: $1\times1$ convs $\mathrm{w0}, \mathrm{w1}, \mathrm{w2}, \mathrm{w3}$ act on channels-first tensors
    GELU($\cdot$) is applied elementwise
**procedure** FNOMODEL2D($x$)

    // — **Preprocess and add grid** —
    $(B, N_x, N_y, T_h, C_{in}) \leftarrow \mathrm{shape}(x)$
    $x \leftarrow \mathrm{reshape}(x,\ B, N_x, N_y,\ T_h \cdot C_{in})$        ▷ flatten time and channels
    $\mathrm{grid} \leftarrow \mathrm{stack}\big(\mathrm{linspace}(0,1,N_y), \mathrm{linspace}(0,1,N_x)\big)$ broadcast to $(B, N_x, N_y, 2)$
    $x \leftarrow \mathrm{concat}(x, \mathrm{grid}) \in \mathbb{R}^{B\times N_x\times N_y\times(T_h C_{in}+2)}$
    $x \leftarrow \mathrm{fc0}(x)$        ▷ pointwise: last dim $(T_h C_{in} + 2) \to C$
    $x \leftarrow \mathrm{transpose}(x,\ B, C, N_x, N_y)$        ▷ to channels-first for FNO/conv blocks

    // — **4 Fourier blocks with skip connections** —
    $x \leftarrow \mathrm{GELU}\big(\mathrm{FNOConv2D}_0(x) + \mathrm{w0}(x)\big)$
    $x \leftarrow \mathrm{GELU}\big(\mathrm{FNOConv2D}_1(x) + \mathrm{w1}(x)\big)$
    $x \leftarrow \mathrm{GELU}\big(\mathrm{FNOConv2D}_2(x) + \mathrm{w2}(x)\big)$
    $x \leftarrow \mathrm{FNOConv2D}_3(x) + \mathrm{w3}(x)$        ▷ no activation after last block

    // — **Project to future steps** —
    $x \leftarrow \mathrm{transpose}(x,\ B, N_x, N_y, C)$        ▷ back to channels-last for pointwise MLP
    $x \leftarrow \mathrm{fc1}(x);\ x \leftarrow \mathrm{GELU}(x)$
    $x \leftarrow \mathrm{fc2}(x)$        ▷ $B \times N_x \times N_y \times T_f$
    **return** $x$
**end procedure**
```

Figure 22: *2D FNO Model Forward Pass*

FNO Training

2D Advection-Diffusion Scenario

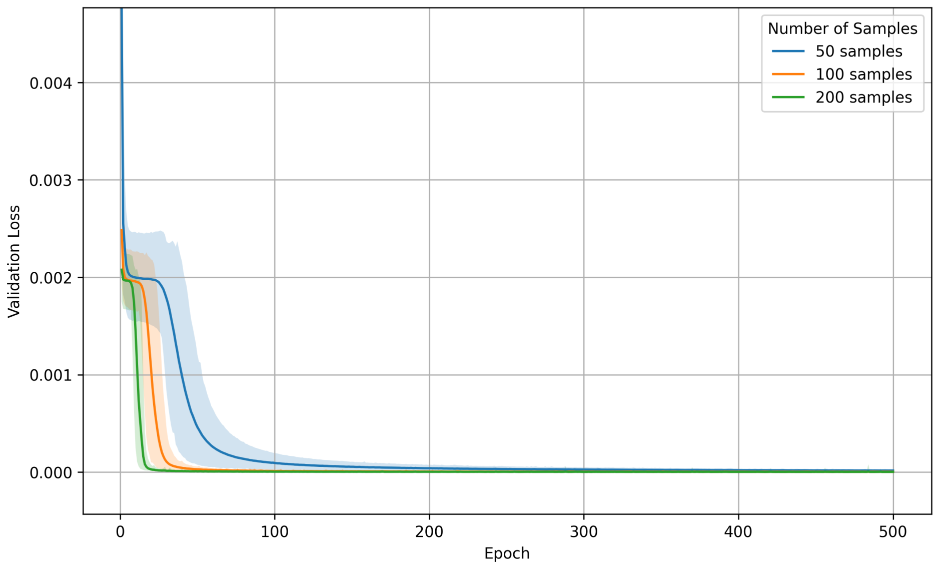


Figure 23: *Validation Loss vs. Epochs for 2D Advection-Diffusion Scenario*
Note: The lines indicate the mean training loss vs. epochs for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models

Table 9: *Generalization Gap for 2D Advection-Diffusion Scenario (Validation-Selected)*
Note: A lower generalization gap indicates better model performance to unseend data. Mean and standard deviation over 30 models.

Samples	n	Mean Epoch	Train Loss M	Train Loss SD	Test Loss M	Test Loss SD	Gap M	Gap SD
50	30	492.70	1.82×10^{-6}	5.44×10^{-7}	4.78×10^{-5}	2.67×10^{-5}	4.60×10^{-5}	2.64×10^{-5}
100	30	477.33	7.16×10^{-7}	1.76×10^{-7}	4.70×10^{-6}	1.45×10^{-6}	3.99×10^{-6}	1.39×10^{-6}
200	30	479.40	3.33×10^{-7}	6.41×10^{-8}	1.27×10^{-6}	4.41×10^{-7}	9.41×10^{-7}	4.02×10^{-7}

1D Burger's Scenario

Table 10: *Generalization Gap for 1D Burgers' Scenario (Validation-Selected)*
Note: A lower generalization gap indicates better model performance to unseend data. Mean and standard deviation over 30 models

Samples	n	Mean Epoch	Train Loss M	Train Loss SD	Test Loss M	Test Loss SD	Gap M	Gap SD
50	30	471.27	5.21×10^{-5}	2.30×10^{-5}	5.16×10^{-5}	1.61×10^{-5}	-5.21×10^{-7}	1.28×10^{-5}
100	30	413.23	3.43×10^{-5}	2.06×10^{-5}	2.30×10^{-5}	1.04×10^{-5}	-1.13×10^{-5}	1.32×10^{-5}
200	30	253.93	3.89×10^{-5}	1.80×10^{-5}	2.35×10^{-5}	1.19×10^{-5}	-1.54×10^{-5}	1.36×10^{-5}

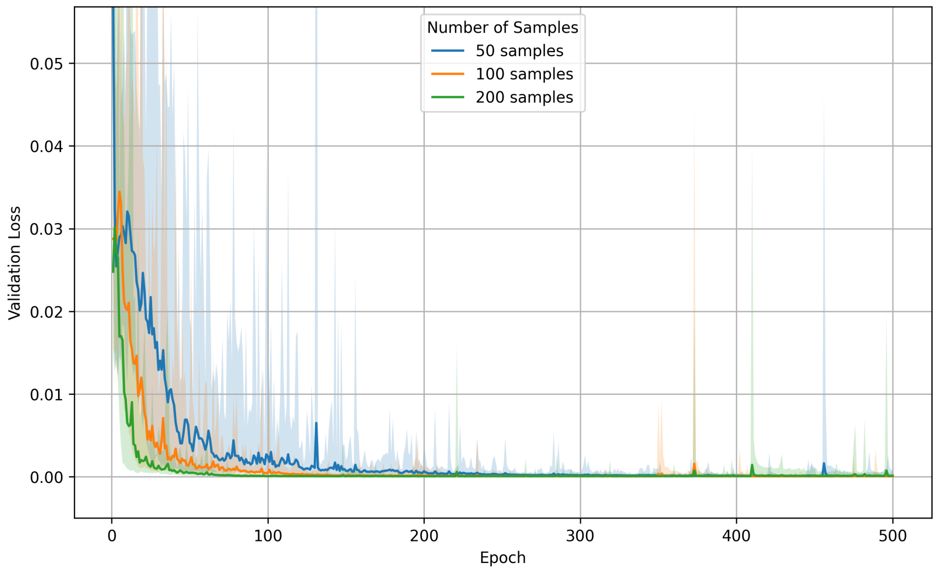


Figure 24: *Validation Loss vs. Epochs for 1D Burgers' Scenario*
Note: The lines indicate the mean training loss vs. epochs for models trained with different data sizes. The translucent band indicates the standard deviation. Values are computed over 30 models

Experiment 1

2D Advection-Diffusion Scenario

1D Burger's Scenario

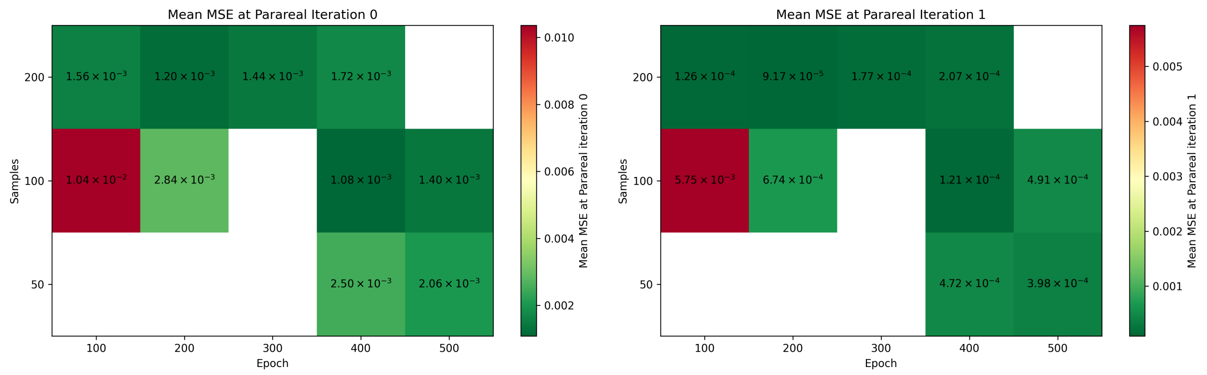


Figure 25: *Simulation MSE by Epoch and Samples at Parareal Iteration* 0 *and* 1 *for for 1D Burgers' Scenario*
Left: MSE values of Parareal simulations after the coarse initialization. This equals the error of an independent model rollout over the trajectories in the test set. Right: MSE values of Parareal simulations after the first Parareal iteration. This equals the error after an entire parallel computation without iterative refinement. Mean over 30 Simulation runs, one run simulates 30 trajectories.

Technical Setup

Program Code

Public repository link: https://github.com/cleesee/fno_parareal.git
Final commit hash: 3332ce71fc2b33a176c7b9108603f4a0e3aa0d2c

Environment and Versions

Table 11: *Cluster environment*

Component	**Version / Value**
Container runtime	singularity-ce 3.11.5-1.el8
OS inside container	Debian GNU/Linux 13 (trixie)
Python	3.12.11 (GCC 14.2.0, glibc 2.41)
Deep-learning build	PyTorch 2.7.0+cu126 (CUDA 12.6)

Table 12: *Python Packages installed in the Singularity Container*

Package	**Version**
`absl-py`	2.3.1
`annotated-types`	0.7.0
`apebench`	0.1.1
`certifi`	2025.8.3
`charset-normalizer`	3.4.3
`chex`	0.1.90
`click`	8.2.1
`contourpy`	1.3.3
`cycler`	0.12.1
`diffrax`	0.7.0
`einops`	0.8.1
`equinox`	0.13.0
`etils`	1.13.0
`exponax`	0.1.0
`filelock`	3.19.1
`flax`	0.10.6
`fonttools`	4.59.1
`fsspec`	2025.7.0
`gitdb`	4.0.12
`GitPython`	3.1.45
`humanize`	4.12.3

Package	**Version**
idna	3.10
importlib_resources	6.5.2
jax	0.6.0
jax-cuda12-pjrt	0.6.0
jax-cuda12-plugin	0.6.0
jaxlib	0.6.0
jaxtyping	0.3.2
Jinja2	3.1.6
kiwisolver	1.4.9
lineax	0.0.8
markdown-it-py	4.0.0
MarkupSafe	3.0.2
matplotlib	3.10.5
mdurl	0.1.2
ml_dtypes	0.5.3
mpmath	1.3.0
msgpack	1.1.1
nest-asyncio	1.6.0
networkx	3.5
numpy	2.2.4
nvidia-cublas-cu12	12.6.4.1
nvidia-cuda-cupti-cu12	12.6.80
nvidia-cuda-nvcc-cu12	12.9.86
nvidia-cuda-nvrtc-cu12	12.6.77
nvidia-cuda-runtime-cu12	12.6.77
nvidia-cudnn-cu12	9.12.0.46
nvidia-cufft-cu12	11.3.0.4
nvidia-cufile-cu12	1.11.1.6
nvidia-curand-cu12	10.3.7.77
nvidia-cusolver-cu12	11.7.1.2
nvidia-cusparse-cu12	12.5.4.2
nvidia-cusparselt-cu12	0.6.3
nvidia-nccl-cu12	2.26.2
nvidia-nvjitlink-cu12	12.6.85
nvidia-nvtx-cu12	12.6.77
opt_einsum	3.4.0
optax	0.2.4

Package	Version
optimistix	0.0.10
orbax-checkpoint	0.11.13
packaging	25.0
pandas	2.3.1
pdequinox	0.1.2
pillow	11.3.0
pip	25.2
platformdirs	4.3.8
protobuf	6.32.0
psutil	7.0.0
pydantic	2.11.7
pydantic_core	2.33.2
Pygments	2.19.2
pyparsing	3.2.3
python-dateutil	2.9.0.post0
pytz	2025.2
PyYAML	6.0.2
requests	2.32.5
rich	14.1.0
scipy	1.16.1
seaborn	0.13.2
sentry-sdk	2.35.0
setproctitle	1.3.6
setuptools	80.9.0
simplejson	3.20.1
six	1.17.0
smmap	5.0.2
sympy	1.14.0
tensorstore	0.1.76
toolz	1.0.0
torch	2.7.0
tqdm	4.67.1
trainax	0.0.2
treescope	0.1.10
triton	3.3.0
typing_extensions	4.14.1
typing-inspection	0.4.1

Package	Version
`tzdata`	2025.2
`urllib3`	2.5.0
`wadler_lindig`	0.1.7
`wandb`	0.20.1
`zipp`	3.23.0

List of Global Seeds

40 / 41 / 42 / 43 / 44 / 45 / 46 / 47 / 48 / 49 / 50 / 51 / 52 / 53 / 54 / 55 / 56 / 57 / 58 / 59 / 60 / 61 / 62 / 63 / 64 / 65 / 66 / 67 / 68 / 69