





RESEARCH ARTICLE | AUGUST 14 2025

Physics-based machine learning for mantle convection simulations

Siddhant Agarwal ; Ali Can Bekar; Christian Hüttig ; David S. Greenberg ; Nicola Tosi 



Physics of Fluids 37, 086624 (2025)

<https://doi.org/10.1063/5.0281832>



Articles You May Be Interested In

Improving simulations of mantle convection within rocky planets

Sciilight (August 2025)

On fluid flows in precessing spheres in the mantle frame of reference

Physics of Fluids (November 2010)

Quantitative investigation of physical properties of mantle plumes in three-dimensional numerical models

Physics of Fluids (November 2007)



Physics of Fluids

Special Topics Open
for Submissions

[Learn More](#)

Physics-based machine learning for mantle convection simulations

Cite as: Phys. Fluids **37**, 086624 (2025); doi: [10.1063/5.0281832](https://doi.org/10.1063/5.0281832)

Submitted: 21 May 2025 · Accepted: 29 June 2025 ·

Published Online: 14 August 2025






View Online



Export Citation



CrossMark

Siddhant Agarwal,^{1,2,a)}  Ali Can Bekar,² Christian Hüttig,¹  David S. Greenberg,²  and Nicola Tosi¹ 

AFFILIATIONS

¹Institute of Space Research, German Aerospace Center (DLR), Berlin 12489, Germany

²Model-driven Machine Learning, Helmholtz-Zentrum Hereon, Geesthacht 21502, Germany

^{a)}Author to whom correspondence should be addressed: siddhant.agarwal@dlr.de

ABSTRACT

Mantle convection simulations are an essential tool for understanding how rocky planets evolve. However, the poorly known input parameters to these simulations, the non-linear dependence of transport properties on pressure and temperature, and the long integration times in excess of several billion years all pose a computational challenge for numerical solvers. We propose a physics-based machine learning approach that predicts creeping flow velocities as a function of temperature while conserving mass, thereby bypassing the numerical solution of the Stokes problem. A finite-volume solver then uses the predicted velocities to advect and diffuse the temperature field to the next time step, enabling autoregressive rollout at inference. For training, our model requires temperature-velocity snapshots from a handful of simulations (94). We consider mantle convection in a two-dimensional rectangular box with basal and internal heating, and pressure- and temperature-dependent viscosity. Overall, our model is up to 89 times faster than the numerical solver. We also show the importance of different components in our convolutional neural network architecture such as mass conservation, learned paddings on the boundaries, and loss scaling for the overall rollout performance. Finally, we test our approach on unseen scenarios and find that it is able to perform thermal evolution well despite being trained on snapshots from steady-state simulations. However, when additional compressibility effects are included in the energy equation or when the initial condition is too far out of the distribution of the training data, the network fails, leaving room for future improvements.

© 2025 Author(s). All article content, except where otherwise noted, is licensed under a Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>). <https://doi.org/10.1063/5.0281832>

I. INTRODUCTION

A. Motivation

Mantle convection plays a fundamental role in the long-term thermal evolution of rocky bodies, such as Earth, the Moon, Venus, Mercury, and Mars. Key processes that shape the evolution of a planet over billions of years, including its potential habitability, such as volcanism, tectonics, and magnetic field generation, are all intricately related to how the silicate mantle and crust transport heat from the deep interior to the surface (e.g., Schubert *et al.*, 2001; Breuer and Moore, 2015; and Tosi *et al.*, 2014). Mantle rocks behave as a highly viscous fluid over geological timescales in response to buoyancy forces induced by extreme temperatures and pressures. As such, mantle convection is modeled via a system of partial differential equations (PDEs) describing conservation of mass, momentum, and energy, which can be numerically solved using dedicated codes (e.g., Tackley, 2008; Zhong *et al.*, 2008; Hüttig *et al.*, 2013; and Bangerth *et al.*, 2024).

Initial conditions and several parameters for mantle convection simulations are poorly constrained and need to be extensively varied in parameter studies to match sparse observations from satellites, telescopes, and *in situ* measurements (e.g., Tosi and Padovan, 2021). Since simulations are computationally expensive, parameter studies are often carried out using simplified models based on scaling laws. These laws parameterize convective heat fluxes as a function of simulation parameters like the Rayleigh number and the rheological properties of rocks (e.g., Reese *et al.*, 1998; Dumoulin *et al.*, 1999; Deschamps and Sotin, 2001; Korenaga, 2010; Thiriet *et al.*, 2019; and Schulz *et al.*, 2020).

Machine learning has started to overcome the limitations of scaling laws by incorporating more physics, such as temperature- and pressure-based dependence of the viscosity, and by predicting the mean temperature (Shahnas and Pysklywec, 2020), one-dimensional temperature profiles as a function of time (Agarwal *et al.*, 2020) as well as two-dimensional (2D) temperature fields (Agarwal *et al.*, 2021) in

time. [Agarwal et al. \(2021\)](#) modeled the spatiotemporal evolution of the 2D temperature field in a Mars-like planet by first using convolutional autoencoders to compress the data, followed by training a recurrent algorithm to perform time-stepping in the latent space. On the one hand, this model is four orders of magnitude faster at inference than the numerical solver and is able to effectively capture the diverse flow patterns resulting from varying parameter combinations. On the other hand, the authors acknowledged some shortcomings as well: (1) the approach required 10 525 simulations for training and evaluation, making it unfeasible to regenerate such large datasets on high-performance computing systems whenever the simulation setup is modified – for example, when changing the domain geometry, the number of parameters, or the viscosity formulation; (2) the model did not predict other important variables such as pressure and velocity fields; and (3) it failed to accurately capture fine-scale features, such as small-scale downwellings arising from instabilities in the cold upper boundary layer. Recent studies, however, have begun to address this issue by developing models that can resolve such fine-scale features more accurately ([Pathak et al., 2020](#); [Wu et al., 2022](#); and [Yin et al., 2023](#)).

These limitations lead to the following question: *Are machine learning models of mantle convection doomed to subpar predictions, despite training on thousands of simulations?* We tackle this question with a physics-based model that is trained on only 94 simulations and is up to 89 times faster than the numerical solver.

B. Physics-based machine learning

It is worth zooming out of the specific case of mantle convection to consider the wider efforts in physics-based machine learning for PDEs. [Raissi, Perdikaris, and Karniadakis \(2019\)](#) are often cited as one of the seminal papers where automatic differentiation is used to evaluate the PDE terms, which are then incorporated into the loss function. Several follow-up works have proposed improvements, such as for overcoming spectral bias ([Shishehbor et al., 2024](#)), for better shock-capturing in transonic flows ([Wassing et al., 2025](#)), for overcoming local minima in multi-loss objectives ([Liu et al., 2025](#)), and for multi-scale modeling ([Wang et al., 2024b](#)), to name a few.

While it is somewhat a matter of semantics, the term PINN (physics-informed neural networks) is generally associated with methods where the inputs to the neural network are coordinate-based points, and automatic differentiation (AD) is used to calculate the partial derivatives of the outputs with respect to the inputs. However, convolutional neural networks (CNNs) ([Pathak et al., 2020](#); [Cheng et al., 2021](#); [Stachenfeld et al., 2022](#); and [Brandstetter et al., 2023](#)), graph neural networks (GNNs) ([Brandstetter et al., 2023](#); [Horie and Mitsume, 2023](#); [2024](#); [Gladstone et al., 2024](#); and [Lino et al., 2025](#)), and attention based architectures ([Li et al., 2023](#); [Zhou et al., 2024](#); [Hao et al., 2023](#); [2024](#); and [Holzschuh et al., 2025](#)) have also been applied successfully in some cases where accounting for the spatial structure of the inputs is desirable. For example, [Wandel et al. \(2020\)](#) use CNNs to predict flow variables and use a finite difference formulation in the form of convolutional kernels to calculate the partial derivatives instead of using AD. Notably, they do not use any data and solve the system of equations in the training phase. While PINNs excel at solving individual PDE instances through coordinate-based learning, neural operators like DeepONet ([Lu et al., 2021](#); [Wang et al., 2021](#)) and Fourier Neural Operators ([Li et al., 2021](#); [2023a](#); [2023b](#); [2023c](#); and [Wen et al., 2022](#))

take a different approach by learning mappings between function spaces, making them particularly suitable for parametric studies.

Hybrid methods have also gained popularity. Solver-in-the-loop methods ([Um et al., 2021](#); [List et al., 2022](#); [Alieva et al., 2023](#); [Wang et al., 2024](#); and [Kochkov et al., 2024](#)) augment coarse-grid solutions obtained from differentiable PDE solvers with a learned correction to account for high-resolution features. In another hybrid paradigm, computationally expensive components of traditional solvers are substituted with learned approximations. [Tompson et al. \(2017\)](#) replace the expensive pressure projection step in an incompressible Euler solver with an unsupervised formulation to obtain divergence-free velocity fields. The network is trained in the framework of several time-steps where their CNN block can be repeatedly called after each advection step, which helps establishing long-term stability. Other examples of the hybrid solver paradigm are [Bar-Sinai et al. \(2019\)](#), [Greenfeld et al. \(2019\)](#), [Luz et al. \(2020\)](#), and [Kochkov et al. \(2021\)](#). [Agarwal et al. \(2025b\)](#) could be considered a hybrid approach, in which learned one-dimensional temperature profiles serve as optimal initial conditions for a 2D mantle convection solver, allowing it to reach steady- or statistically-steady states 2.8 times faster than with conventional initializations. Achieving this speedup at the cost of only ~ 2 min of training time is appealing. However, this benefit is limited to scenario of the steady-state solution, rather than the full time-stepping process. The latter is crucial in thermal evolution simulations, where the system continues to evolve based on addition and removal of heat. Our new hybrid approach instead targets acceleration of the time-stepping itself, enabling faster simulation of the full temporal evolution.

C. Our approach

We replace the most computationally expensive component of mantle convection simulations – the solution of the mass and momentum conservation equations (i.e., Stokes problem) – with a learned CNN that predicts velocities as a function of temperature and enforces mass conservation by design. The divergence-free velocities obtained from the CNN via a PyTorch model are then fed into the C++ numerical solver GAIA ([Hüttig et al., 2013](#)) via a Python interface to perform a numerically inexpensive advection-diffusion step. In this way, we are able to perform time-stepping without ever learning in time. As we will see later, the mantle convection PDEs provide a strong inductive bias on how to model in time and not accounting for it can make the learning task more challenging. Our contributions in this paper are as follows:

- To the best of our knowledge, this is the first physics-based machine learning model in mantle convection.
- We introduce a scaling for learning velocity fields across several orders of magnitude.
- We use learned paddings on the boundaries for enhanced accuracy.
- We achieve stable predictions over tens of thousands of time-steps without learning in time.
- We evaluate this model on unseen scenarios such as thermal evolution at inference time and demonstrate the strengths and shortcomings of our approach.

We use a similar architecture to the one introduced by [Tompson et al. \(2017\)](#) but need more trainable parameters for our Stokes

problem as opposed to their Poisson problem. In our case, pressure projection is not necessary, as the network conserves mass through constraints. This is similar to how [Wandel et al. \(2020\)](#) enforce mass conservation. However, we adopt a data-driven approach for the momentum equation instead of a purely physics-based one. Not needing training data from the solver is attractive, but reaching the efficiency and accuracy of computational fluid dynamic codes with machine-learning-based solvers is an active area of research ([Grimm et al., 2022](#); [Wandel et al., 2025](#)). Our hybrid approach instead learns on high-quality samples from the solver. All the machine learning code developed and used is available here: https://github.com/agsid-dhant/PBML_Mantle_Convection.

The paper is organized as follows. In Sec. II A, we introduce the governing PDEs and the setup of the simulations. In Sec. II B, we provide an overview of the dataset and discuss the scaling of velocities. We then present the architecture of the CNN used as a Stokes surrogate model in Sec. II C, followed by two baseline models for comparison in Sec. II D. In Sec. III, we discuss our main findings: training details (Sec. III A); velocity predictions from the Stokes model (Sec. III B); time-evolution using a U-net (Sec. III C); performance of our Stokes model across different parameters during rollout (Sec. III D); speedup analysis compared to the numerical solver and other baselines (Sec. III E); ablation studies of key components, where one component at a time is removed to assess its impact on performance (Sec. III F); and performance of our model on some unseen scenarios (Sec. III G). Finally, we conclude by summarizing the main findings of this paper.

II. METHODS

A. Mantle convection equations

The large-scale deformation of crystalline mantle rocks over geological timescales is typically modeled as the dynamics of a viscous fluid with negligible inertia, leading to the so-called Stokes flow. We assume the Boussinesq approximation, whereby the flow is treated as incompressible, and the only density variations considered are those due to temperature changes in the buoyancy force term (e.g., [Schubert et al., 2001](#)). In the non-dimensional form, the conservation equations of mass, momentum, and thermal energy for a creeping fluid heated from below and from within in a 2D Cartesian geometry read

$$\nabla \cdot \mathbf{u} = 0, \quad (1a)$$

$$-\nabla p + \nabla \cdot \left(\eta \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) = Ra T \mathbf{e}_y, \quad (1b)$$

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T = \nabla^2 T + Q. \quad (1c)$$

Here, \mathbf{u} is the velocity vector with horizontal and vertical components u and v , p is the dynamic pressure, i.e., the flow-driving pressure perturbation given by the difference between the total pressure and the hydrostatic pressure that is implicitly used to define a static reference state (see, e.g., Sec. VIA of [Schubert et al., 2001](#)), T is the temperature, \mathbf{e}_y is the unit vector in the vertical direction, Q is the internal heating rate, and η is the dynamic viscosity, which depends on temperature and depth (i.e., hydrostatic pressure) as

$$\eta(T, y) = \exp(-\log(\gamma)T + \log(\beta)(1 - y)), \quad (2)$$

where y is the height ($y = 0$ at the bottom of the domain and $y = 1$ at the top) and the parameters γ and β denote the maximum viscosity

contrasts due to temperature and depth, respectively. Ra is the Rayleigh number,

$$Ra = \frac{\rho g \alpha \delta T D^3}{\eta_0 \kappa}, \quad (3)$$

where ρ is the density, g is the gravitational acceleration, α is the coefficient of thermal expansion, δT is a temperature scale, D is the height of the domain, κ is the thermal diffusivity, and η_0 is a reference (dynamic) viscosity. Ra is thus the ratio of buoyancy forces due to thermal expansion that drive convection to resistive forces due to thermal diffusion and viscosity that inhibit it.

As can be seen from Eq. (1b), for a fluid with negligible inertia, the velocities have no memory of previous time-steps and are fully determined by the temperature field and the resulting viscosity field. These velocities then determine how the temperature field is advanced in time in Eq. (1c) via advection and diffusion. Heat sources and sinks can be optionally added and can even vary in time – for example, due to radioactive decay, which is relevant for planetary interior evolutions.

We solve the aforementioned equations in a 2D rectangular box with an aspect-ratio of four. All boundaries are impermeable (zero normal velocity) and free-slip (zero shear stress). The top and bottom boundaries are isothermal, while the sidewalls are insulating. This translates into the following set of Dirichlet and Neumann boundary conditions:

- Left: $u = 0$, $\frac{\partial v}{\partial x} = 0$, $\frac{\partial T}{\partial x} = 0$
- Right: $u = 0$, $\frac{\partial v}{\partial x} = 0$, $\frac{\partial T}{\partial x} = 0$
- Top: $v = 0$, $\frac{\partial u}{\partial y} = 0$, $T = 0$
- Bottom: $v = 0$, $\frac{\partial u}{\partial y} = 0$, $T = 1$

B. Dataset and scaling

We use the same dataset as in [Agarwal et al. \(2025b\)](#), where the simulations attain a statistical steady-state after typically advancing the solution for tens of thousands of time-steps. While discovering steady-states of the mantle convection simulations is of great interest to the mantle convection community, we note that our model enables fast time-stepping and can be applied to evolution scenarios as well. In fact, we will apply the model learned on this steady-state dataset to an evolution scenario later, where the internal heat source is no longer constant but decays in time.

The dataset is generated with the finite-volume code GAIA on a uniform grid of 128×506 cells. Mass and momentum conservation are solved with the MUMPS direct solver ([Amestoy et al., 2001; 2019](#)), while the energy equation is solved with an iterative solver. The dataset consists of 128 simulations, of which 94 are used for training, 16 for cross-validation, and 18 for testing – all chosen randomly. Three simulation parameters are randomly sampled from a uniform distribution to generate the dataset: Q (between 0 and 10), β (between 1 and 100), and γ (between 10^6 and 10^{10}). The Rayleigh number in Eq. (1b) is set to 1. The vigor of convection is influenced through an effective Rayleigh number through the temperature- and pressure-contrasts. The range of parameters yields effective Rayleigh number values ranging from 10^4 (when $\gamma = 10^6$ and $\beta = 100$) to 10^{10} (when $\gamma = 10^{10}$ and $\beta = 1$).

In Fig. 1, we visualize the range of each field by drawing a line from the minimum to the maximum of each simulation. Temperature

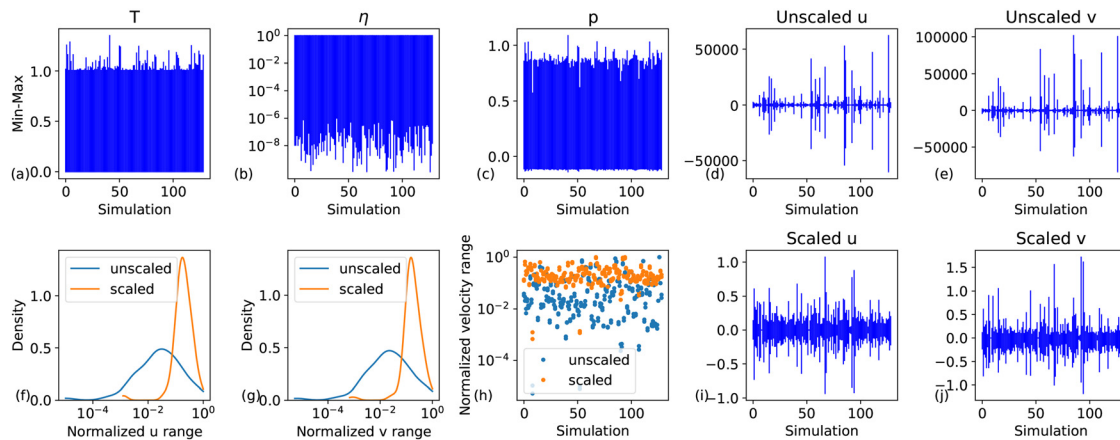


FIG. 1. The ranges of different fields in the simulations (a)–(e). Scaling the velocities brings the ranges across different simulation parameters closer to each other (i) and (j). For ease of visualization, we normalize the u and v range for the scaled and unscaled cases by dividing them by the maximum (f), (g), and (h).

(a), viscosity (b), and pressure (c) are well-behaved. For viscosity, we can take the \log_{10} and divide by 8 so that it varies from approximately -1 to 0 . However, the minimum and maximum velocities (d, e) span different orders of magnitudes – for some simulations, these span large ranges, whereas for the others, these remain much more limited. There are two issues with the standard normalizations such as min-max scaling, mean-standard deviation scaling, and log scaling. First, we do not want to shift the mean of the velocities but only divide by some constant both sides of Eq. (1a), so that mass conservation is not violated. Second, dividing by the overall maximum of the velocities would still leave simulations with values spanning a significantly smaller range compared to the wide ranges spanned by other simulations. As the magnitude of the velocity field is correlated with the simulation parameters, we fit a linear regression model through the maximum values of u and v as a function of the simulation parameters and obtain the following scaling (shown here up to two significant digits):

$$\text{velocity scaler} = 5 e^{0.18Q} \gamma^{0.43} \beta^{-0.46}. \quad (4)$$

Dividing the velocity fields by the above factor brings the simulations within a more homogeneous range. This is shown in Fig. 1(f) for u and Fig. 1(g) for v , where we compute the range as maximum minus minimum value of velocities for all the time-steps in a simulation. We normalize the ranges by the maximum values to facilitate comparison of scaled and unscaled velocities. When unscaled, there is a very small difference between the maximum and minimum value of velocities from each simulation. However, by scaling, we push most of the simulations to have a higher range [Fig. 1(h)] by one order of magnitude.

Our CNN thus learns on scaled velocity fields, which we can simply multiply by Eq. (4) to recover the actual, unscaled magnitudes at inference time. This technique was motivated by our experience with a simpler set of simulations, where only Ra was varied, and a uniform viscosity was used. In this case, velocities and pressure can still vary by several orders of magnitude, but the magnitudes vary approximately linearly with Ra . Scaling the fields by Ra enabled us to obtain good predictions. Thus, this approach is a multi-parameter extension with some modifications: viscosity is no longer constant, and Ra is always 1, but different parameters drive the scales now.

C. Machine learning model

1. Convolutional neural network

CNNs have been widely adopted in a variety of machine learning models for PDEs on uniform grids. As shown in Fig. 2(a), the CNN acts as a surrogate model for the velocities induced by the temperature field. The main input to the CNN is the temperature field, but we further enrich it with the viscosity field calculated via Eq. (2). We also add coordinates of the numerical grid as is sometimes done in coordinate-based neural networks (Serrano *et al.*, 2023; Catalani *et al.*, 2024) and graph neural networks (Lam *et al.*, 2023; Wei and Freris, 2024). Finally, even though the simulation parameters β and γ are already contained in the viscosity field and Q is irrelevant for Eqs. (1a) and (1b), we explicitly pass these three parameters as inputs to the CNN. This is because these parameters are used to scale the velocities as per Eq. (4) and, therefore, provide some additional contexts to the network besides the temperature and viscosity fields on the magnitudes with which it should predict each scaled velocity field. This is useful because the scaling is not perfect, i.e., it is not an exact fit. The predicted velocities are then fed to the numerical solver to advect the temperature field. In this way, the model can be used autoregressively – outputs of the model become the input at the next time step. We use the same architecture as Thompson *et al.* (2017) [Fig. 2(b)], which was designed for modeling linear algebraic systems. First, the input – structured as a tensor of (batch, channels, height, and width) – is turned into a more detailed internal representation through a layer of convolutions. The resulting representation has more number of channels than the number of input variables to the CNN. We downsample this representation five times (creating five levels) by a factor of 2 using average pooling. Average pooling reduces the resolution of data by replacing local regions within a window (2×2 in this case) with their average value, helping the model retain general patterns while discarding fine-grained details. This allows the network to process features of different scales at different levels. The receptive field, or the region of the input that influences a particular output, grows with each level, so that the coarsest level spans almost the entire domain. This is important for PDEs where one point in input can impact any other point in the output. At each resolution (i.e., level), the representation is independently

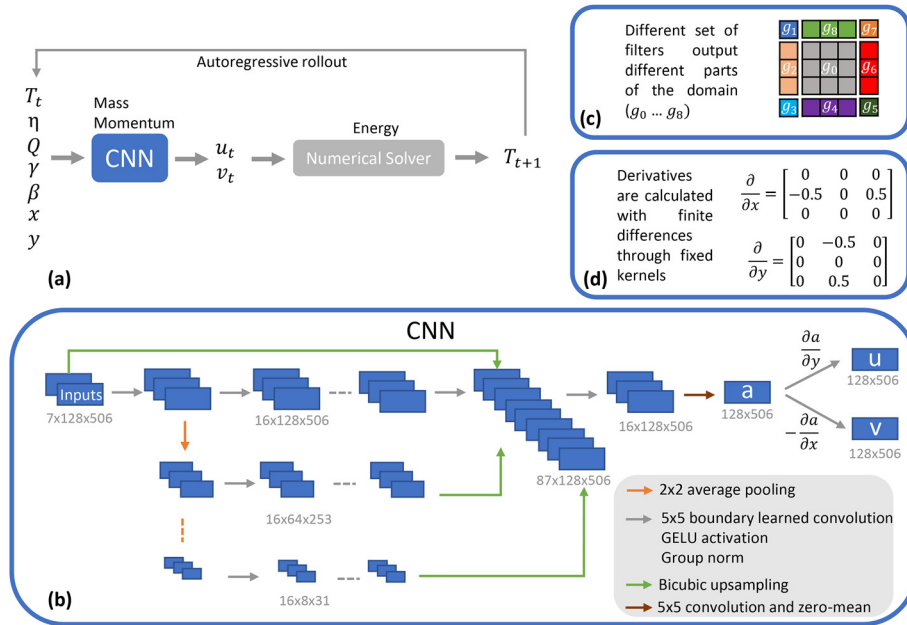


FIG. 2. Our hybrid physics-based model for time-stepping. (a) We use a data-driven approach to model the primary computational bottleneck in mantle convection simulations: solving the Stokes equation. The velocities predicted by the convolutional neural network (CNN) can be fed to the numerical solver for time-stepping. (b) The CNN architecture is inspired by [Tompson et al. \(2017\)](#) for modeling solutions of linear systems as a function of the right-hand-side of the equation. (c) Instead of standard padding methods (such as zero paddings), we use boundary-learned convolutions. A unique set of filters is responsible for predicting each corner and each edge as well as the interior domain. (d) Mass conservation is enforced by predicting a field whose curl yields divergence-free velocity components u and v .

processed with a series of convolutions before being upsampled via bicubic interpolation to the original resolution. Upsampling refers to the process of increasing the spatial resolution of feature maps so that the output matches the original input size. The upsampled features from all levels are concatenated instead of summed, allowing the network to learn how to combine this information through another series of convolutions. This differs slightly from the original U-net architecture where the information is downsampled and upsampled through an autoencoder-style architecture with skip connections between the encoder (that successively decreases resolution) and decoder part (that successively increases resolution) at each resolution ([Ronneberger et al., 2015](#)). Skip connections create direct links between representations from earlier layers to later layers by addition or concatenation. Skip connections link encoder and decoder layers at matching resolutions, allowing fine-grained spatial information lost during downsampling to be reintroduced during upsampling. U-nets favor learning incremental updates to the inputs due to skip connections at each level. This is more sensible when learning time-stepping between states, whereas the architecture of [Tompson et al. \(2017\)](#) seems more efficient for learning output variables (e.g., u, v) as a function of different input variables (e.g., η, T).

We use the Gaussian Error Linear Unit (GELU) as our activation function, which has been empirically shown to outperform ReLU and ELU ([Hendrycks and Gimpel, 2016](#)) on a wide variety of tasks. Unlike ReLU, which zeroes out negative values, or SELU, which is typically used for self-normalizing fully connected architectures, GELU provides stable gradients and expressive nonlinearities better suited to our spatially structured data. GELU is defined as the product of the input x and the cumulative distribution function of a standard Gaussian distribution, which is computed using the error function:

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2} \left(1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right). \quad (5)$$

2. Mass conservation

As in [Wandel et al. \(2020\)](#), we enforce mass conservation in our incompressible flow by predicting a vector potential \mathbf{a} , whose curl provides divergence-free components of velocity. This approach is based on the Helmholtz decomposition (e.g., [Aris, 2012](#)). Working in two dimensions, we need to predict only one component of \mathbf{a} , which corresponds to the stream function, a_z , or simply, a

$$u = \frac{\partial a}{\partial y}, \quad v = -\frac{\partial a}{\partial x}. \quad (6)$$

The derivatives are calculated as central finite differences using fixed convolutional filters [see [Fig. 2\(d\)](#)]. These are conveniently carried out on the Graphics Processing Unit (GPU) as is all the training of the CNN. Since the stream function is defined up to a constant, as in [Wandel et al. \(2020\)](#), we subtract the mean of the output from a to keep the values bounded; this does not change the derivatives of a . We further multiply the output of CNN by an empirically determined factor of 10 (so that a is now ten times the CNN output) to help limit the numerical values of the final layer to a reasonable range, typically not far beyond -1 and 1 . This keeps gradients smaller and activations in the sensitive range of the GELU activation function, making optimization more stable and efficient. We experimented with scaling factors of 1, 10, and 100 and found that a factor of 10 yielded the lowest loss during training. While the optimal range of a is not known *a priori*, this experiment hints that -10 to 10 is approximately the numerical range that a needs to fit this dataset. In the future, one can also consider initializing the multiplicative factor with 10 and then optimizing it with the rest of the network parameters. Different values of dx only serve as a multiplicative factor for [Eq. \(1a\)](#) that have little bearing on mass conservation in this case with a uniform grid. This is also the reason behind the form of the velocity scaling of [Eq. \(4\)](#): divergence-free network velocities multiplied by this factor remain divergence-free. In

practice, with single precision training of the network, multiplying the velocities with large scaling factors might violate mass conservation beyond an acceptable threshold. Hence, we train with double precision, leaving further optimization of training precision for future work. One such strategy could be to use single precision for the trainable weights of the network and to use double precision only for computing the velocities from a using convolutional filters. It would be interesting to see if such a network would predict well across the different magnitudes of velocities that are present in our dataset.

Finally, we note that when computing the derivatives of a with respect to x and y using central finite differences, the resulting velocity fields are reduced in size by two grid points in each respective dimension (one at each boundary) due to the stencil not being defined at the domain edges. We proceed with padding these velocities as per the necessary boundary conditions: we pad with the exact boundary value for Dirichlet boundary conditions, and, for Neumann boundary conditions, we copy the value of the adjacent cell where the derivatives must be zero. This inevitably makes the mass conservation inexact as the velocities at the boundaries are not calculated from a using Eq. (6) but simply padded. Therefore, we compute Eq. (1a) just at the boundaries and plug it into the loss function during training as a soft constraint, while, on the interior domain, mass conservation is already satisfied (down to double machine precision). An alternative to this would be to pad the network output a with either learned values or values derived in a way that the boundary conditions would be satisfied. However, the model was unable to learn a compatible padding. Setting up and solving a linear system would also be an option. Either way, it is not clear if a formulation of a exists that can simultaneously (1) satisfy the free-slip and impermeable boundary conditions, (2) conserve mass at boundaries, and (3) be consistent with a finite-volume based numerical solver. This highlights one of the challenges in physics-based machine learning, namely, learning in a consistent manner with the data from the PDE solver. It is also worth noting that, no matter what values we prescribe at the boundaries, the solver will overwrite them in exactly the same way as we do by either assigning an exact value for a Dirichlet condition or by copying the adjacent value for a Neumann condition. We later assess the effectiveness of a hard constraint vs a soft constraint in rollout performance, i.e., how well the model performs when its predictions are used repeatedly over several time steps. In the soft constraint version, the CNN predicts the velocities directly, and the mass conservation equation calculated on these velocities is plugged into the loss function.

3. Learned boundary paddings

When convolving at the edges of the domain, the input to each layer must be padded if one wishes to maintain the original spatial dimensions after convolution. After a lot of experimentation, we found that typically used paddings such as zeros and replicate (i.e., copying the adjacent value) are detrimental for accuracy on the edges. This shortcoming became evident upon advecting the temperature field with the predicted velocities: errors on the boundaries would give rise to artificial “bubbles” of hot material that would rapidly rise and destabilize the solution. Alguacil *et al.* (2021) showed some advantage of using a spatial context of 0 or 1 to help CNN learn on the boundaries, but we found it to be insufficient when combined with replicate padding.

We employed the “boundary learned convolution” from Innamorati *et al.* (2020), which learns the padding on the boundaries itself by decomposing the domain and learning a separate set of filters responsible for predicting the interior domain as well as all the edges and corners [see Fig. 2(c)]. The learned filters are applied to the spatial context of each subdomain to reproduce the original width and height. The intuition here is that the fields such as temperature and velocities, but also images in general, might have non-trivial yet smooth spatial extrapolations. In this case, adding zeros might create artifacts that the CNN has to learn to overcome. Using a unique set of filters for each subdomain increases the count of trainable parameters by a factor of 9. With the exception of the interior filter, which is applied to the entire domain and produces g_0 , all the “extra” filters are applied to significantly smaller subdomains (edges and corners) and, therefore, do not significantly increase the overall time complexity. Consequently, a network with same total parameter count but regular convolutions would be slower.

4. Loss function

We use mean absolute error (MAE) between the true and predicted fields as our loss function. Mean squared error is another commonly used metric, which, due to squaring of the error, tends to weigh outliers and higher-magnitude deviations more. However, in our task, where the magnitudes of samples differ from each other, we opt for MAE to avoid weighing higher-magnitude samples more as all samples contain physically meaningful information. Furthermore, as we tend to reach small MAE values (below 10^{-3}), squaring this value might make the gradients smaller and slow down convergence of the loss function. Despite the scaling described in Eq. (4), the velocities can still vary significantly. In particular, the high velocity values will now be smaller due to the scaling. Therefore, to ensure that, within each batch, proper weight is given to the lower velocity magnitudes, we scale the loss by dividing it by the range of each example in the batch. We clip this loss scaling at 10 to avoid creating too much imbalance between the examples,

$$S_{\text{norm}} = \text{clip}\left(\frac{1}{\max(x_{\text{true}}) - \min(x_{\text{true}})}, 1, 10\right). \quad (7)$$

For given bounds a and b , the clip function limits the value x , which can take as follows:

$$\text{clip}(x, a, b) = \min(\max(x, a), b). \quad (8)$$

We further add a factor of 10 on the boundaries to penalize errors there more heavily than in the interior of the domain,

$$S_{\text{bc}} = \begin{cases} 1 + 10, & \text{on boundaries,} \\ 1, & \text{otherwise.} \end{cases} \quad (9)$$

These values are somewhat arbitrary, and one could certainly optimize them further at the cost of more computational resources. However, we perform an ablation study to assess the impact of removing the loss scaling terms. The loss is calculated as

$$L(x_{\text{true}}, x_{\text{pred}}) = \begin{cases} \frac{1}{N} \sum |(x_{\text{true}} - x_{\text{pred}}) \cdot S_{\text{norm}} \cdot S_{\text{bc}}|, & \text{if scale loss,} \\ \frac{1}{N} \sum |(x_{\text{true}} - x_{\text{pred}})|, & \text{otherwise.} \end{cases} \quad (10)$$

We add mass conservation to the loss function if mass is conserved as a soft constraint. If the curl-based formulation is used, we only use the mass conservation calculations on the boundaries. The mass conservation term is calculated as

$$L_{\text{mass}} = \frac{1}{N} \sum \left| \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right|. \quad (11)$$

Finally, we found that the central-differences-based curl formulation produced some oscillations in the y direction for u and in the x direction for v . Applying Gaussian filtering to a did not seem to reduce these artifacts. Instead, we learn (i.e., optimize the network's learnable parameters by minimizing the mismatch between the ground truth and the network output) not only on the velocities but also on their derivatives. The intuition behind this is that artifacts become even more prominent when their derivatives are calculated so they can be effectively penalized,

$$L_{\text{deriv}} = \frac{1}{N} \sum \left| \frac{\partial u_{\text{true}}}{\partial y} - \frac{\partial u}{\partial y} \right| + \frac{1}{N} \sum \left| \frac{\partial v_{\text{true}}}{\partial x} - \frac{\partial v}{\partial x} \right|, \quad (12)$$

The overall loss term can now be calculated as

$$L_{\text{total}} = \begin{cases} \frac{L_u + L_v + L_T}{3}, & \text{if } T \text{ is predicted,} \\ \frac{L_u + L_v}{2}, & \text{otherwise,} \end{cases} \quad (13)$$

$$L_{\text{total}} + = \begin{cases} \frac{1}{N} \sum L_{\text{mass}}, & \text{if soft constraint,} \\ \frac{1}{N_{\text{bc}}} \sum L_{\text{mass, bc}}, & \text{if curl - based constraint,} \end{cases} \quad (14)$$

$$L_{\text{total}} + = L_{\text{deriv}}, \quad \text{if derivative loss term is included.} \quad (15)$$

Note that $+ =$ means addition of a new quantity on the right-hand side to an existing quantity on the left-hand side.

D. Baselines

We consider two different prediction models, each with a specific purpose.

1. Numerical solver with suboptimal settings

The velocities predicted by the CNN are relatively accurate but still exhibit some discrepancies compared to the ground truth solution obtained from our numerical PDE solver. Given that advection is still possible using these slightly erroneous velocities, it is natural to ask how an imperfect solver would perform in terms of both accuracy and computational speedup. To this end, we consider as first option solving the mass and momentum equations only every 100th time step. If our model outperforms this benchmark, it would suggest that we can produce meaningful temporal interpolations, despite the training data being available only at every 100th time step. Notably, 100 was also the number of skips required to achieve a speedup comparable to that provided by the CNN. On the one hand, this baseline is somewhat unfair to our best model, since we could likely skip several steps before reaching the error level of the solver with 100 momentum skips. On the other hand, the suboptimal solver is readily available without the need for data or training. We note that “skip” is used in two different

contexts in this work: in the framework of machine learning models, it refers to skip connections in the neural network architecture, whereas in this context, when using the numerical solver, it refers to skipped solutions of the momentum equations and thereby the skipped velocity updates.

As a second option, we consider an iterative solver running on the same GPU used for CNN inference. As the iterative solver is slower than the direct solver, we use an under-relaxation factor of 0.99 to still achieve some speedup with respect to the direct CPU baseline and to evaluate its performance against the CNN.

2. U-net for learning in time

For our second model, we choose a simple U-net architecture to explore how well time-stepping can be learned. In this case, we no longer rely on the numerical advection-diffusion solver but instead learn to predict from the velocities and temperature at a given time step based on the state at the previous time step, the grid coordinates, the viscosity, and the time step dt .

To keep the architecture as close as possible to our Stokes surrogate model, we downsample the original resolution 5 times (encoder part) and upsample it back to the original resolution (decoder part) using the same operations. At every level, the encoder and decoder each contain 3 convolutional layers. Information is passed at each level from encoder to decoder through skip connections via concatenation of channels. We train four different versions:

- (1) U-net-1: We match the overall parameter count of the network to our Stokes surrogate model without our proposed techniques, i.e. boundary-learned convolution and curl-based mass conservation.
- (2) U-net-2: We match the inference time of a single time-step from the network to that of a single u, v prediction of the Stokes model without our techniques.
- (3) U-net-3: We match parameter count and use our techniques.
- (4) U-net-4: We match inference time and use our techniques.

III. RESULTS AND DISCUSSION

A. Training details for the Stokes model

After some trial-and-error of different hyperparameters (parameters that are set before training and affect how the model learns) in our Stokes surrogate model, we found that $16 \times 5 \times 5$ filters, processing features 6 times at each level, provided a good balance between speed and accuracy for predicting the flow velocities. Unless otherwise stated, all results are presented with this architecture. We train on a single GPU as we use a small batch size of 16, which has been shown to improve generalization and contribute to stable rollouts, as observed in [Agarwal et al. \(2021\)](#). We aim for 150 epochs (one pass through the training and validation data) and reduce the learning rate (the step size used to update model parameters during training) by a factor of 0.5 every 20 epochs. In practice, not all networks reach 150 epochs as we sometimes terminate training early to manage training costs. The model we present the results for and refer to as “our best model” required about 5 days of training time to reach 80 epochs on an NVidia V-100 GPU. We note that we do not train on all the snapshots (system states at time-steps) available (training set: 217813, cross-validation (cv) set: 44049, test set: 42872). For each simulation, we pick the first 200

snapshots and then randomly select up to 800 more as large portions of the time-series are in a statistical steady-state and do not have as much variability as the earlier time-steps. By doing so, we effectively pick 1/3 of the stored snapshots, which allows us to load all the data in the CPU memory and train different networks in parallel. Since we only saved every 100th snapshot from the solver to begin with, this means that for each simulation, we use only 1/300 of the snapshots with respect to the original temporal resolution of the solver. One could further speedup training by processing larger batches on multiple GPUs and evaluate generalization performance. Finally, training in single or mixed precision could also help, if mass conservation is still satisfactorily satisfied and if velocities with different orders of magnitude can be learned effectively.

B. Velocity predictions with the Stokes model

In Fig. 3, we show examples of u , v predictions from two different test-set simulations. The errors reach up to 5% of the maximum magnitude of the true velocity in the second sample. This is quite high for

a scientific machine learning study. However, we must keep in mind that a single model is learning on velocities across a few orders of magnitude from a handful of simulations. We deliberately picked two extreme cases from our test set here to demonstrate learning across different magnitudes. This error might also not be the most informative metric, as our goal is to advect the temperature using these velocities. While fitting a larger network could help reducing the errors, it might come at the cost of a lower generalization capability to unseen simulations, potentially leading to unstable rollouts. A larger network would also be slower at inference time (i.e., at predicting after the network has been trained). On average, we find that our best model predicts velocities u_t , v_t that are approximately 14 times more accurate than if one were to simply take the velocities at the last available time step (u_{t-100} , v_{t-100}) from the solver.

C. U-net predictions for time-stepping

As described in Sec. II D 2, we train four different U-nets as baselines to compare against our method. We plot the training curves in

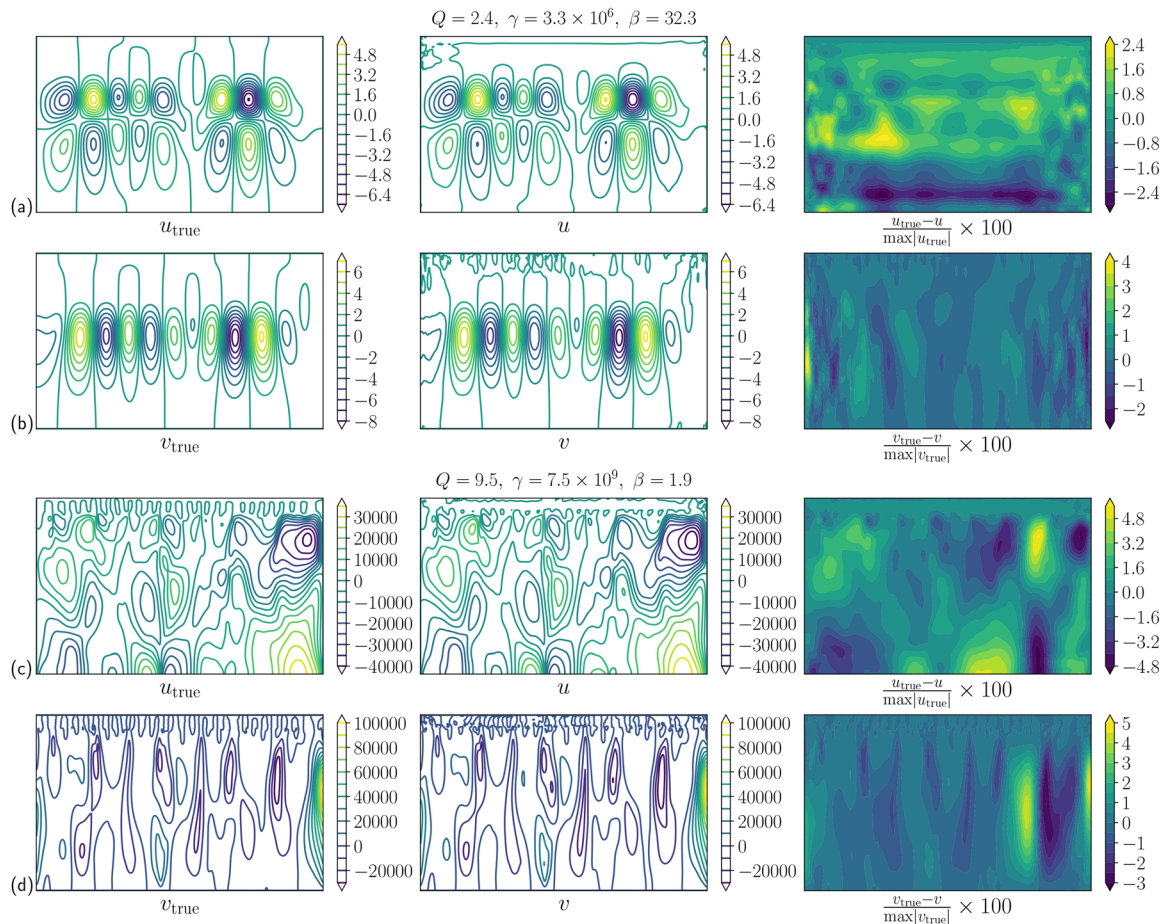


FIG. 3. Examples of velocity predictions compared to ground truth from the direct solver for two different unseen simulations (a) and (b) and (c) and (d) characterized by widely different parameters leading to relatively low (a) and (b) and high velocities (c) and (d). A single model is able to learn across four orders of magnitude. The prediction error is divided by the absolute maximum value and displayed as a percentage error in the right column.

Fig. 4 alongside our best model. Our techniques of boundary-learned convolution and loss scaling improve the loss attained during training compared to their counterparts without these techniques. Without them, we observe some artifacts where the network seems to mix up the fields in certain regions near the top boundary. This could be because the input to the network differs from the output. This highlights the challenge of learning with U-nets beyond incremental updates to the input as the system states evolve in time. However, we are able to overcome these artifacts with the boundary-learned convolution and possibly also due to loss scaling. In fact, U-Net-4 (the same inference time network with our techniques) outperforms our best Stokes surrogate model in terms of training and validation loss on velocities [Fig. 4(b)].

Nevertheless, as we rollout autoregressively during inference time with our best U-net on a mere 16 steps ($\sim 16 \times 100$ simulation time-steps), the solution quickly diverges [Fig. 4(c)]. Several tricks have been suggested for overcoming this accumulation of errors during rollout (e.g., Lippe *et al.*, 2023) such as training on multiple time-steps. As training on multiple time-steps would increase the memory

requirements by tracking a longer graph, the so-called “pushforward” trick does not track gradients in the rollout during training except for the last time step. In this way, the network learns to overcome non-trivial errors that tend to accumulate. This, however, still comes at the cost of increased training time and intuitively makes the learning task more challenging as the network has to learn to de-noise the input in addition to learning a relation between the left-hand side and the right-hand side of a system of PDEs.

Thus, a large array of machine learning methods remains to be explored for directly time-stepping in mantle convection besides the recurrent learning approach of Agarwal *et al.* (2021). These methods are attractive for their significant speedups, ability to process several simulations at once in batch, and end-to-end differentiability with simple models built in open-source frameworks like PyTorch. Nevertheless, we would like to offer an alternate perspective here. From Eq. (1b), we know that the velocities at a given time are dependent on temperature but are not directly a function of velocities at the previous time step – this would not be the case for non-linear, strain-rate dependent rheologies (e.g., Schulz *et al.*, 2020). Learning new

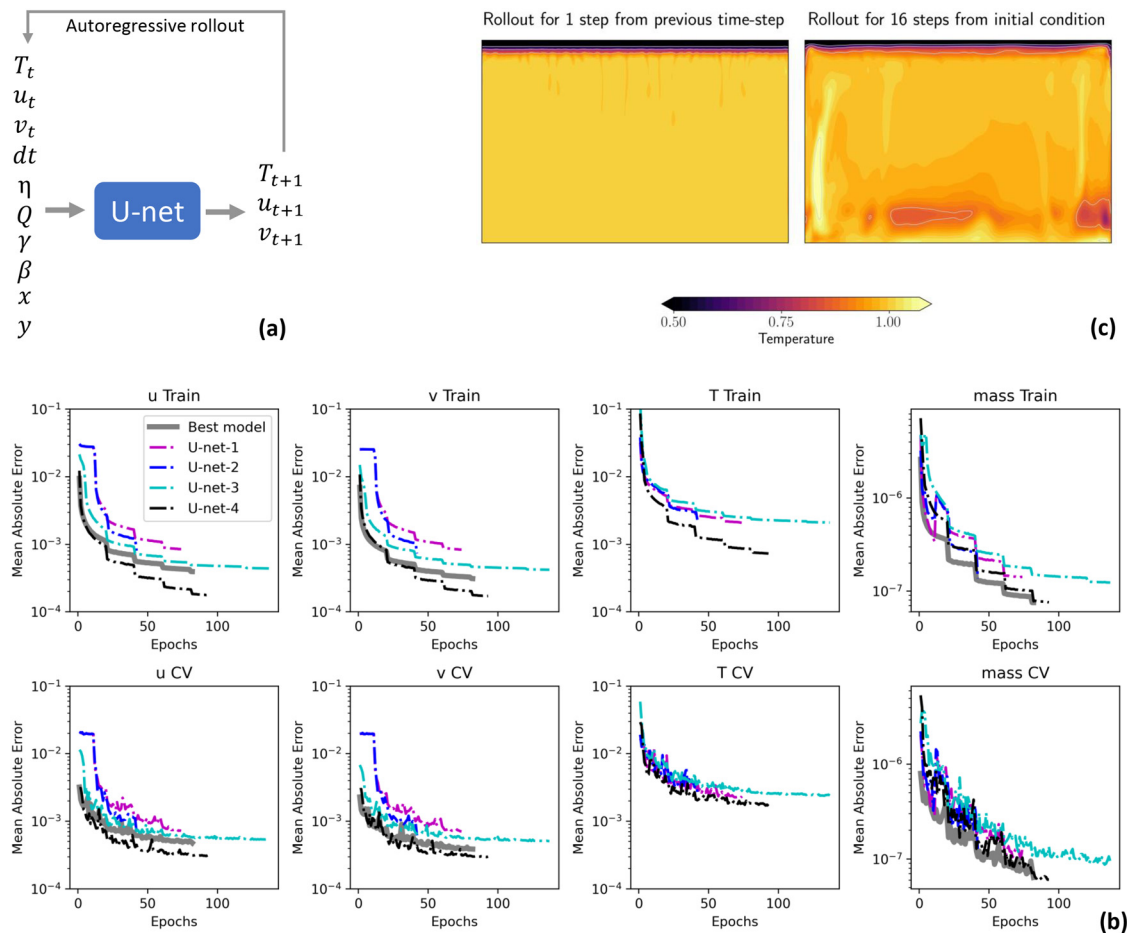


FIG. 4. (a) U-net schematic for predicting from state at time t to state at $t + 1$. (b) Four different U-nets are trained to evaluate if different parameter counts and components like boundary-learned convolution improve the prediction accuracy. These are plotted alongside our best Stokes surrogate model that predicts velocities (solid gray lines). (c) When rolling out autoregressively with the best performing network (dashed black lines), the predictions diverge within 16 steps.

velocities as a function of the previous velocities can, thus, introduce questionable correlations in our model. Although the best U-net outperforms the Stokes surrogate model in single-step velocity predictions, it fails to roll out, perhaps because it is conditioned on and sensitive to previous velocities. A more physics-based improvement to this U-net would be to learn two different networks: one that learns u_t, v_t as a function of T_t (like our Stokes surrogate) and then another network that learns T_{t+1} as a function of T_t, u_t , and v_t . These could be optimized together or separately, but, either way, it would be interesting to explore how well the advection-diffusion operation can be learned. Given that we only have access to time steps sampled at intervals of 100, it remains an open question what the most effective approach is for learning this operator.

Would the network (say f) be more accurate if we trained it using a 100-step rollout – i.e., by repeatedly applying the network to predict each intermediate step from t_1 to t_{100} as $f(f(f(t_1)_1)_{2\dots})_{100}$ – or by directly learning the mapping from the initial state t_1 to the final state t_{100} as $f(t_1)_{100}$ in a single step? This small thought experiment could be further extended to directly learning $T_{t_{100}} = f(T_{t_1})$ but would any features in the network look like velocity fields without explicitly encoding Eq. (1c) and what would be the data-efficiency of such a network? The answer to the first question is no, and to the second, probably poor. Agarwal *et al.* (2021) showed that it is possible to skip velocities and learn this direct mapping from one temperature field to the next 200th, but at the cost of 10 000 training simulations. One can seemingly trade-off data-efficiency and causality for faster time-stepping. For fields where various simulation setups exist in the form of geometries, heat sources, viscosity models, parameter variations, phases transitions, and melting models, data-efficiency can be an important consideration. This can be further viewed through the lens of Buitrago *et al.* (2025) who demonstrate the benefits of learning on multiple time-steps in the past through a “memory” for systems with partially observable (e.g., missing spatial points) or corrupt states (e.g., noise or uncertainty). Agarwal *et al.* (2021) benefited from using 16 time-steps as inputs to their Long-Short-Term-Memory network, possibly because they learn only on T , and the missing velocities as well as the skipped time steps can be seen as an extreme case of missing observables.

D. Rollout performance for different parameters

In contrast to direct time-stepping with the U-net, we are able to produce a stable rollout for tens of thousands of time-steps using our hybrid approach. The velocities predicted by our Stokes model are used to advance the temperature field using the numerical advection-diffusion solver in GAIA. To make all runs comparable, we run the advection-diffusion solver with a Courant number of 1, although the implicit solver allows for higher numbers.

Figure 5 shows three qualitatively different unseen simulations from the test set. These include sluggish convection with low frequency characteristics [Figs. 5(a) and 5(b)], vigorous convection with higher frequency structures and a thinner upper thermal boundary layer [Figs. 5(e) and 5(f)], and an intermediate case in Figs. 5(c) and 5(d), but with a non-monotonic time-series where the mean temperature initially decreases and then increases before stabilizing. With the exception of the sluggish convection case, our model outperforms the suboptimal numerical solver in matching convection patterns such as pronounced undulations in the shape of the stagnant lid as well as in

the horizontally averaged temperature profile. The momentum skips and the resulting errors in the velocities are especially detrimental during the initial transient phase of vigorous convection, causing the intermediate case to miss the trough in the mean temperature curve of row (c) in Fig. 5 – this is undesirable in thermal evolution scenarios, where the history of the planet can contain clues to inferring certain parameters and, thus, requires accurate time-series modeling.

For the sluggish case, the predictions by our model still capture the correct pattern of convection [Fig. 5(b)], but the mean quantities fare slightly worse than the suboptimal solver. This is likely a consequence of the loss scaling of Eq. (7) as it assigns up to 10 times more weight to more vigorously convecting simulations. Reducing the maximum clipping value (e.g., from 10 to 8) could help mitigate this issue and result in a more favorable balance, but this issue ultimately highlights the challenge of learning a single model on disparate scales.

We also found that the prediction accuracy diminishes significantly when extrapolating in the parameter space, highlighting another drawback of our approach. Again, the suboptimal solver performs better than our model when extrapolating. We were able to successfully run 16 out of 18 simulations in our test set. One run produced non-physical lid structures while another diverged after a few thousand steps. We mark these cases in Fig. 6 and notice that this region of the parameter space is quite challenging. This is true not only from a physical standpoint of high Q and γ driving vigorous convection but also from a data point of view, as the failed case (red cross) is clearly beyond our training data range, and the nonphysical case (yellow circle) sits right on the edge of the Q - γ envelope. In the future, we would like to add some additional training points at the exact corners of the parameter space, i.e., 8 more training simulations.

E. Comparison of speedups

To evaluate the speedup of the model, we select an unseen simulation from the test set and run four different models on the same CPU/GPU architecture using the same numerical settings. We run (1) the original direct solver on a CPU, (2) the direct solver with 100 momentum skips on a CPU, (3) an iterative solver (BiCGStab) with under-relaxation factor of 0.99 on a GPU, and (4) our ML model on a GPU. Note that without under relaxation in our setup, the iterative solver on the GPU is actually slower than the direct solver on a CPU. Figure 7(c) shows that our model not only achieves the highest speedup of the four but also has the lowest error in mean quantities with respect to the original solver. This is understandable for the direct solver with momentum skips but rather unexpected for the iterative solver. Indeed, we see that the lid distribution for the iterative solver looks very realistic, but the error in the mean temperature seems to accumulate in this case. This is likely because we use an under-relaxation factor of 0.99, which means that the velocities solved for are not as accurate as those predicted by the direct solver. In other words, we perform an advection-diffusion step in time with a slightly erroneous velocity as we use under relaxation. As in Figs. 5(a) and 5(b), we do not expect this increased accuracy of our method to hold up across the entire parameter range, but it further underscores the promise of machine learning for mantle convection simulations. Direct solvers are rarely used in fine grid resolutions and in three dimensions (3D) due to prohibitive memory requirements. Although it remains to be seen how such a convolutional network would scale to 3D, this result

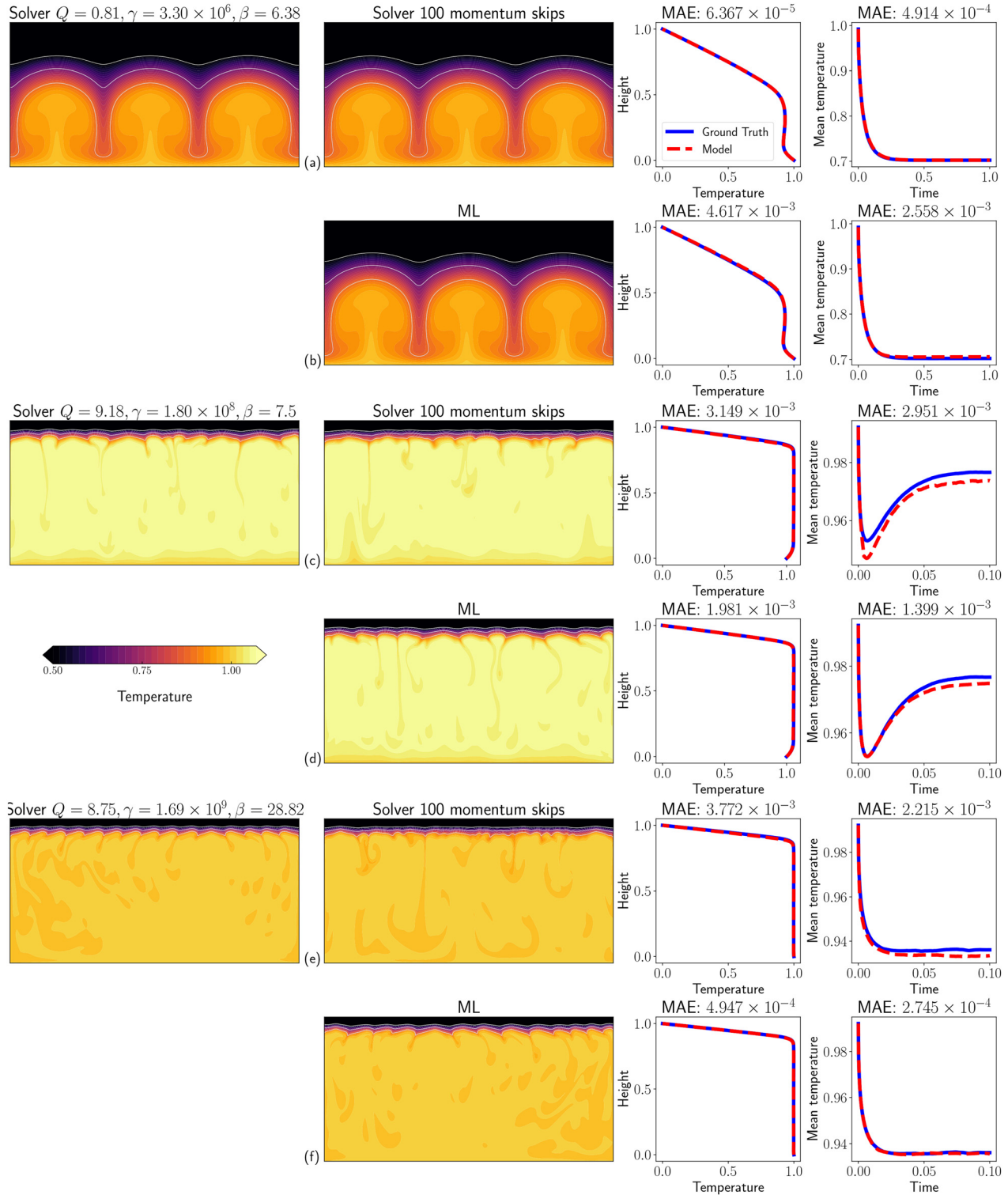


FIG. 5. Evaluation of rollout performance on three qualitatively different simulations (a) and (b), (c) and (d), and (e) and (f). For each simulation, we plot the ground truth (runs with the direct solver) in the first column, followed by the prediction (either baseline of 100 momentum skips or the ML model) in the second, the horizontally average temperature profile in the third, and the mean temperature as a function of time in the fourth.

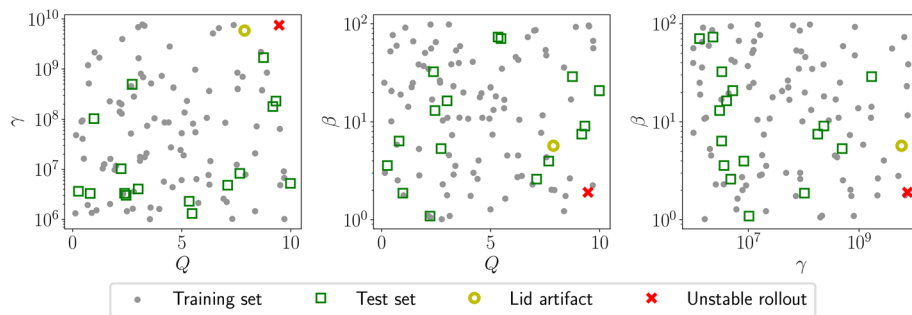


FIG. 6. Parameter space of the training and test sets. We achieve stable rollout on all the simulations in the test set, except when extrapolating in the high Q -high β space (e.g., yellow circle or red cross).

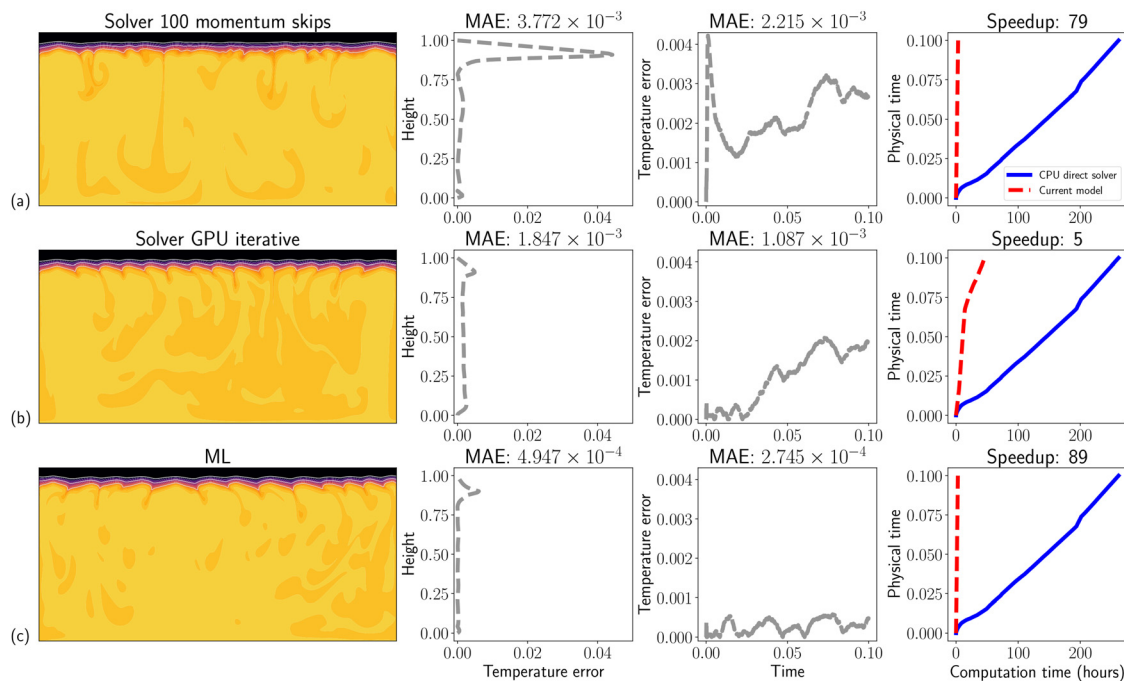


FIG. 7. Comparison of accuracy and speedup with respect to a direct solver on the CPU of (a) direct CPU solver with 100 momentum skips on the CPU, (b) iterative solver with an under-relaxation factor of 0.99 on the GPU, and (c) our model with the ML-based Stokes solver running on the GPU.

suggests that ML for the Stokes problem in 3D has the potential to offer significant cost-accuracy benefits with respect to an iterative solver.

We report the best speedups here when a single run is executed on a V100 GPU. In practice, however, this speedup is throttled by the exchange of data between the Python and C++ code as more runs are launched on a single machine in parallel. This is not ideal for larger campaigns, and we would like to address it in the future. One possibility would be to infer the ML model directly in C++ and achieve even greater speedups. Better GPUs than the V100 can further extend this advantage.

F. Different ablations in the Stokes model

We now present the importance of specific components of our CNN through ablations, where we remove one component at a time

while keeping the rest of the architecture unchanged. Figure 8(a) highlights the importance of enforcing mass conservation for producing a stable rollout. We add mass conservation as a soft constraint to the loss function in Eq. (14) and observe that the overall mass conservation curve in dashed red is going down in Fig. 8(a). However, it is likely just a reflection of the velocity predictions getting better. In our case, where we have ground truth data available at every spatial point in the domain, the mass conservation is not much more than a transformation of data that does not seem to convey any more information during the training. Note that when we enforce mass conservation as a hard constraint using the curl-based formulation, the loss on the interior domain is 0 up to double machine precision. The gray solid curve in Fig. 8(b) has higher values than double-precision zero because of the boundaries. Finding a way to include a curl-based constraint here could further improve accuracy. Nevertheless, the mass conservation error on boundaries

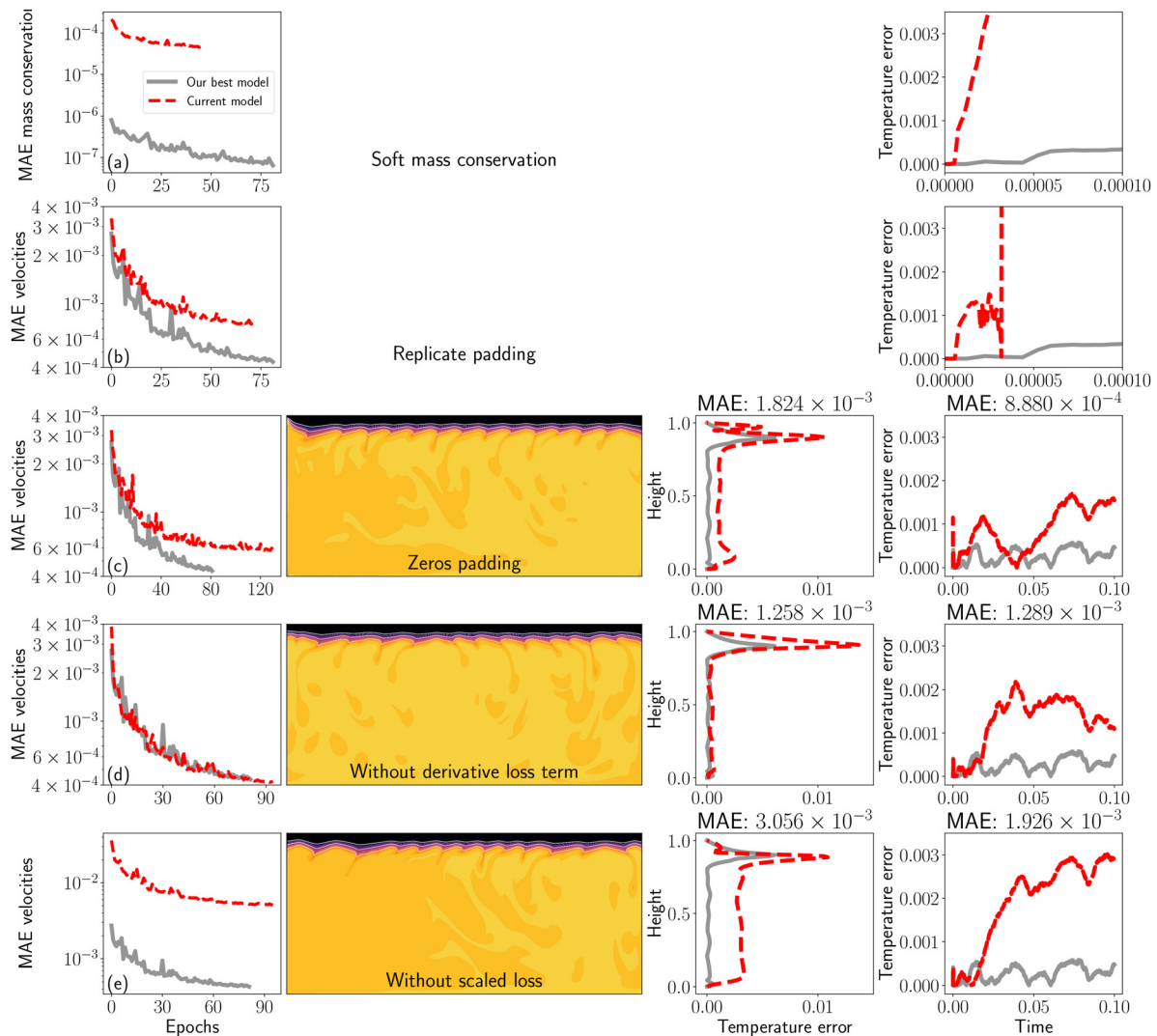


FIG. 8. Results of different ablations, where we change one component of the model at a time. The cross-validation loss is plotted in the first column in solid gray for our best model and the ablated model in dashed red (a)–(e).

decreases as the network predicts better and better velocities. We also tried to multiply the mass conservation loss term with a factor to give it more weight but found that it conflicted with the other terms in that it deteriorated the loss values of velocities. This is a well-known issue with physics-based losses and could be addressed in future work (Liu *et al.*, 2025).

Next, rollout performance was highly sensitive to errors on the boundaries. Zero padding outperforms replicate padding, but we still observe a severe artifact in Fig. 8(d) near the upper-left corner of the domain, where the stagnant lid is “pinched” and some other smaller, more difficult to see inaccuracies appear at the bottom corners. Since the boundary-learned convolution introduces a lot more trainable parameters, we also match the parameter count of the zero padding network by increasing the filter count from 16 to 64. Unfortunately, this deteriorated the performance even more as larger models can be

more prone to diverge during rollout. We calculated the test set MAE on each boundary for the learned padding and compared it those of the zero padding networks. In Table I, we list these as relative

TABLE I. We compute the MAE on the boundaries for every fifth sample in our entire test set for different paddings and plot them with respect to the MAE on the learned padding (boundary-learned convolution).

Relative improvement of learned padding (\uparrow)	Left	Right	Top	Bottom
to zeros padding with 16 filters	8.03	2.78	1.63	2.82
to zeros padding with 64 filters	3.42	2.03	0.99	1.52

improvements and notice that the learned padding is up to 8 times more accurate than zero padding.

Finally, we evaluate the importance of learning on derivatives of the velocities [Eq. (12)] and find modest improvements in the MAE of horizontally averaged temperature profile and the mean temperature over time. When the loss is not scaled [Eq. (7)], the presence of the derivative term seems to create some imbalance in loss terms, but the deterioration in the MAE values stems mainly from the lack of loss scaling. When loss scaling and the derivative term are both taken out, the MAE is 2.804×10^{-3} for the temperature profile and 1.294×10^{-3} for the mean temperature vs time curve. To summarize, both the loss tricks seem to help.

G. Toward a general Stokes model

With our physics-based machine learning approach, we are able to learn from a relatively small training dataset and generate high-quality predictions. We now test the robustness of this approach on the following unseen scenarios, some of which are out-of-distribution (OOD), i.e., predicting on significantly different data than what the network was trained on. To do so, we modify some aspect of the simulation presented in Figs. 5(e) and 5(f).

One, we apply our Stokes surrogate model to an energy equation that differs from the one used to generate the training data. Specifically, we use the Extended Boussinesq Approximation (EBA), where the mass and momentum equations are unchanged, but the energy equation contains two extra terms (adiabatic heating and shear heating) associated with the degree of compressibility in the form of a Dissipation number (Di) (King *et al.*, 2010). We run a simulation with a low $Di = 0.1$, which leads to a small adiabatic temperature gradient between the top and bottom boundary layers and renders this an OOD example. As we see from Figs. 9(a) and 9(b), our model fails to accurately match the ground truth, both in terms of mean quantities and the convection patterns including the noticeable thinning of the lid at the top corners. The solver with momentum skips fares slightly better. Nevertheless, we believe that including some data from EBA runs could alleviate this problem, possibly without introducing additional input variables such as Di to our Stokes surrogate model.

Two, we run a thermal evolution simulation by introducing a time-decaying heat source. We run the simulation until a non-dimensional time $t = 0.056$, which roughly corresponds to 4.5×10^9 years for a Mars-like planet. A decay constant is set to reduce the heat production [Q in Eq. (1c)] by a factor of 10 over the entire evolution. As a consequence, the simulation no longer enters a statistical steady-state but continues evolving with a decreasing mean temperature. Our model outperforms the baseline and produces a very realistic-looking convection pattern. This approach has some parallels to parametrized 1D thermal evolution models, which rely on scaling laws to predict convective heat fluxes that are then used to advance the mean temperature through a global energy balance equation (e.g., Stevenson *et al.*, 1983; Gurnis, 1989; Solomatov, 1995; Korenaga and Jordan, 2003; Grott *et al.*, 2011; and Drilleau *et al.*, 2021). In some respects, our approach can be seen as an extension of these methods in that it enables the prediction of the full spatiotemporal evolution of velocities and temperature from a purely steady-state dataset.

Three, we try different initializations for the same simulation as also done in Agarwal *et al.* (2025b). The model failed completely when initialized with a cold profile (zero temperature everywhere apart from

the bottom boundary), as this lies far outside the distributions of profiles in our training set. The solver with momentum skips, on the other hand, still managed to follow the solution, although not perfectly. As opposed to the cold start, our model fared better when starting with a profile with a temperature of 0.75 everywhere except a small linearly increasing thermal boundary layer of 0.05 on top and bottom. Yet, upon visually inspecting an animation of the simulation, we found that our model does not mimic the solver in the conductive growth of the boundary layers in the initial phase. This is because the minimum internal temperature present in the training set is 0.8, making this case slightly OOD. The fact that our model can still enter the statistical steady-state is interesting. We never teach the network to follow any trajectory or reach a certain end point. This behavior more likely hints at the inherent property of this type of flows: regardless of the initial transient stages, the flow will tend to reach the same statistical steady-state. It remains to be seen if this property can be exploited to discover steady-states even faster. Nevertheless, despite all the physics-based components in our model, it is a sobering reminder that ML models can struggle on OOD examples. In that respect, terminology such as neural solvers or ML-based simulations can be confusing as a certain degree of robustness is expected from solvers. As an example of a study that shows significant OOD capabilities, Ranade *et al.* (2022) take a data-driven approach for learning machine learning models on local domains in space but then infer iteratively like a solver to propagate information globally and arrive at an equilibrium solution of the PDE.

Finally, we use the profile prediction model from Agarwal *et al.* (2025b) to initialize our simulation and our prediction models and find that our model convincingly outperforms the baseline as can be seen in Figs. 9(e) and 9(f). This is encouraging, as our model has never seen this type of initial condition in the training dataset, but the horizontally averaged temperature field is, of course, well within the distribution of the training set. Multiple models can thus be combined to further accelerate mantle convection simulations.

While it is essential to be aware of the limitations of these models, at the same time, these OOD cases provide a template for extending the training dataset. Training data from different initializations and the EBA energy equation can be seamlessly introduced into our existing model. It would also be interesting to extend the setup to different viscosity functions than in Eq. (2). Doing so would require a different way of scaling the velocities – we would like to be independent of the parameters β and γ , as other formulations of viscosity include different parameters. For instance, one could instead scale the velocity based on the minimum value of the viscosity field. The velocity scaling would also need to be explored if more complex physics were introduced, e.g., in the form of pressure- and temperature-dependent thermal expansivity and thermal conductivity, phase transitions, and melting. Higher-order optimizers (e.g., Gupta *et al.* 2018; Vyas *et al.*, 2025) could also be interesting for learning across several orders of magnitude. Finally, in future work, we would like to explore discretization invariance and learning as a function of different geometries. Neural fields (Serrano *et al.*, 2023; Catalani *et al.*, 2024; and Knigge *et al.*, 2024) have shown promise in memory-efficient learning on arbitrary computational grids.

IV. CONCLUSION

We present a novel physics-based machine learning approach for mantle convection simulations. We learn a single model to predict flow velocities spanning four orders of magnitude as a function of

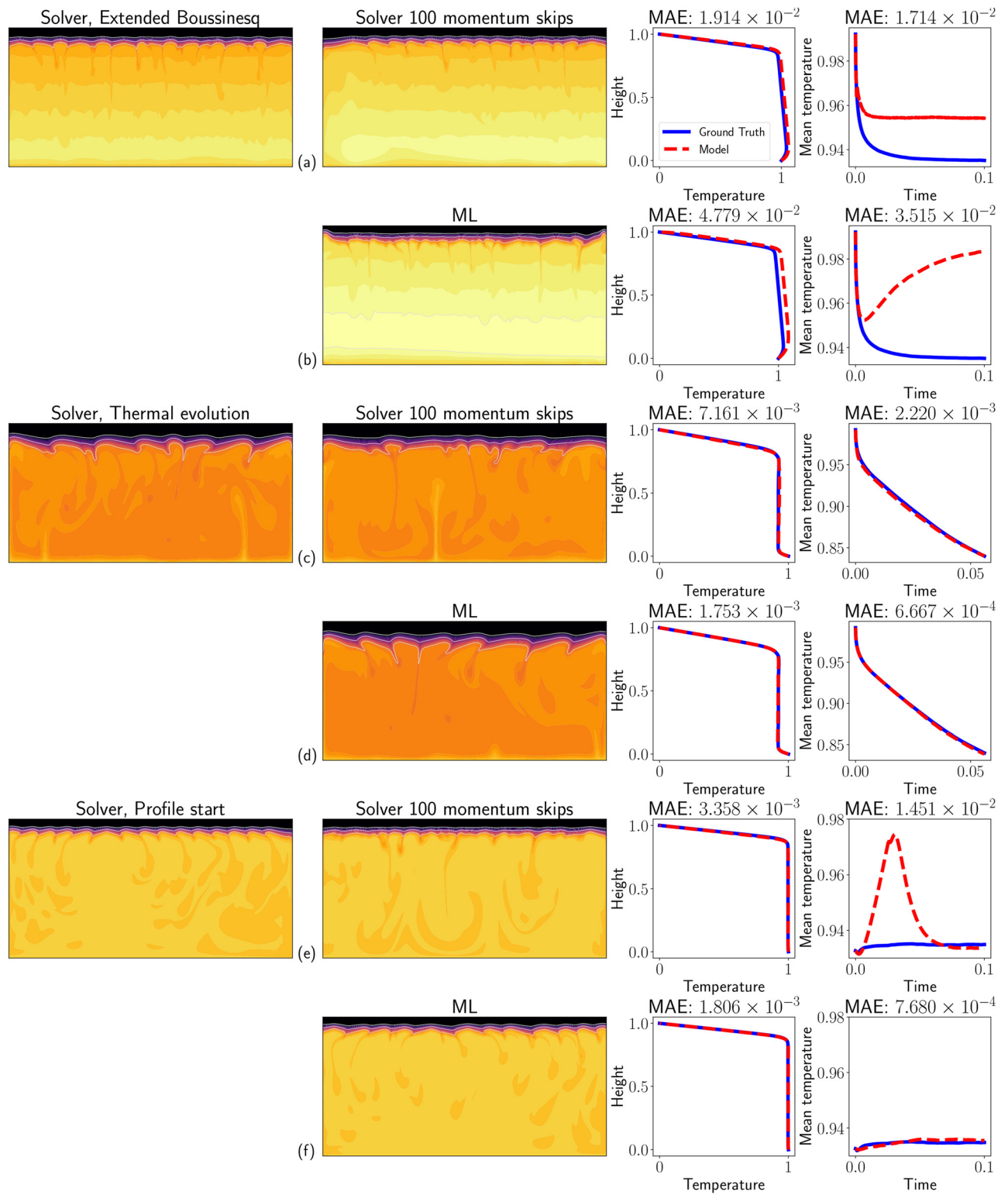


FIG. 9. Performance of the baseline and the ML model on three unseen scenarios: (a) and (b) Extended Boussinesq Approximation (EBA), (c) and (d) thermal evolution, and (e) and (f) starting from the steady-state profile.

three parameters controlling viscosity and internal heating rate. These velocities are then fed to a numerical finite-volume solver to perform advection-diffusion of the temperature field and to obtain the next time step. We found this approach promising for stable predictions over thousands of time-steps, an active area of research in machine learning for PDEs. On the contrary, we were unable to achieve stable rollout by learning in time, despite reaching lower validation losses on the velocities during training. Overall, our model provides a speedup of 89 times, compared to the numerical solution computed with a direct solver.

Our model is stable across the parameter space, except when approaching high values of the internal heating rate (Q) and of the parameter controlling the temperature dependence of the viscosity (γ), as this region of the parameter space is not represented in our sparse training dataset of 94 simulations. We also demonstrated the strengths of our model by removing one component at a time. Mass conservation plays a decisive role. Using boundary-learned convolutions, we obtain up to 8-times more accurate solutions on the boundaries than in the case of zero padding, which further enhances our rollout performance.

Finally, we tested our model on some unseen scenarios. When using a different energy equation including additional compressibility effects, we found that our model fails as the input temperature fields contain an adiabatic gradient in the convecting part, which is not considered in the simulations of our training set. We also tried to start our model from a cold profile but found this was too far away from the training set.

A particularly promising result is the ability to perform a thermal evolution simulation based on a purely steady-state dataset. This shows the versatility of our formulation, as it is already able to generalize in this instance. Physics-based machine learning promises accurate and data-efficient modeling of mantle convection simulations.

ACKNOWLEDGMENTS

This work was funded by the PLAGeS (Physics-based Learning Algorithms for Geophysical flow Simulations) project through the German Ministry of Education and Research BMBF (Project Nos. 16DKWN117A and 16DKWN117B) under the “Deutschen Aufbau- und Resilienzplans” (DARP). DARP is part of NextGenerationEU’s “Aufbau- und Resilienzfähigkeit” (ARF) by the European Union.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Siddhant Agarwal: Conceptualization (lead); Data curation (supporting); Formal analysis (lead); Funding acquisition (supporting); Investigation (lead); Methodology (lead); Software (lead); Validation (lead); Visualization (lead); Writing – original draft (lead); Writing – review & editing (supporting). **Ali Can Bekar:** Conceptualization (supporting); Methodology (supporting); Writing – original draft (supporting); Writing – review & editing (supporting). **Christian Hüttig:** Conceptualization (supporting); Methodology (supporting); Resources (supporting); Software (supporting); Writing – review & editing

(supporting). **David S. Greenberg:** Conceptualization (supporting); Funding acquisition (equal); Project administration (equal); Supervision (supporting); Writing – review & editing (supporting). **Nicola Tosi:** Conceptualization (supporting); Data curation (lead); Funding acquisition (equal); Methodology (supporting); Project administration (equal); Resources (equal); Supervision (lead); Writing – review & editing (lead).

DATA AVAILABILITY

Some sample data from the simulations as well as model weights for the trained networks are available in [Agarwal et al. \(2025a\)](#). These can be used to predict velocities here https://github.com/agsiddhant/PBML_Mantle_Convection. The Github repository contains all the machine learning code used in this paper. The simulations were generated at the German Aerospace Center (DLR). The complete derived data supporting the findings of this study are available from the corresponding author upon reasonable request. Furthermore, GAIA is a proprietary code of the DLR and users interested in working with it should contact Nicola Tosi (nicola.tosi@dlr.de) and Christian Hüttig (christian.huettig@dlr.de).

REFERENCES

- Agarwal, S., Tosi, N., Breuer, D., Padovan, S., Kessel, P., and Montavon, G., “A machine-learning-based surrogate model of Mars’ thermal evolution,” *Geophys. J. Int.* **222**, 1656–1670 (2020).
- Agarwal, S., Tosi, N., Hüttig, C., Greenberg, D., and Bekar, A. (2025a). “PBML_Mantle_Convection,” Zenodo, V. 0.0.1, Dataset <https://doi.org/10.5281/zenodo.15088589>
- Agarwal, S., Tosi, N., Hüttig, C., Greenberg, D. S., and Bekar, A. C., “Accelerating the discovery of steady-states of planetary interior dynamics with machine learning,” *J. Geophys. Res.: Mach. Learn. Comput.* **2**, e2024JH000438, <https://doi.org/10.1029/2024JH000438> (2025b).
- Agarwal, S., Tosi, N., Kessel, P., Breuer, D., and Montavon, G., “Deep learning for surrogate modeling of two-dimensional mantle convection,” *Phys. Rev. Fluids* **6**, 113801 (2021).
- Alguacil, A., Pinto, W. G., Bauerheim, M., Jacob, M. C., and Moreau, S., “Effects of boundary conditions in fully convolutional networks for learning spatio-temporal dynamics,” in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, edited by Y. Dong, N. Kourtellis, B. Hammer, and J. A. Lozano (Springer International Publishing, Cham, 2021), pp. 102–117.
- Alieva, A., Hoyer, S., Brenner, M., Iaccarino, G., and Norgaard, P., “Toward accelerated data-driven Rayleigh–Bénard convection simulations,” *Eur. Phys. J. E* **46**, 64 (2023).
- Amestoy, P., Buttari, A., L’Excellent, J.-Y., and Mary, T., “Performance and scalability of the block low-rank multifrontal factorization on multicore architectures,” *ACM Trans. Math. Software* **45**, 1–26 (2019).
- Amestoy, P., Duff, I. S., Koster, J., and L’Excellent, J.-Y., “A fully asynchronous multifrontal solver using distributed dynamic scheduling,” *SIAM J. Matrix Anal. Appl.* **23**, 15–41 (2001).
- Aris, R., *Vectors, Tensors and the Basic Equations of Fluid Mechanics* (Courier Corporation, 2012).
- Bangerth, W., Dannberg, J., Fraters, M., Gassmoeller, R., Glerum, A., Heister, T., Myhill, R., and Naliboff, J. (2024). “Aspect v3.0.0,” Zenodo. <https://doi.org/10.5281/zenodo.14371679>
- Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P., “Learning data-driven discretizations for partial differential equations,” *Proc. Natl. Acad. Sci. U. S. A.* **116**, 15344–15349 (2019).
- Brandstetter, J., van den Berg, R., Welling, M., and Gupta, J. K., “Clifford neural layers for PDE modeling,” *arXiv:2209.04934* (2023).
- Brandstetter, J., Worrall, D., and Welling, M., “Message passing neural PDE solvers,” *arXiv:2202.03376* (2023).

- Breuer, D. and Moore, W. II, "Dynamics and thermal history of the terrestrial planets, the moon, and io," in *Treatise on Geophysics*, 2nd ed., edited by G. Schubert (Elsevier, Oxford, 2015), Vol. 10, pp. 255–305.
- Buitrago, R., Marwah, T., Gu, A., and Risteski, A., "On the benefits of memory for modeling time-dependent PDEs," in *The Thirteenth International Conference on Learning Representations (ICLR, 2025)*; available at <https://openreview.net/pdf?id=o9kqa5K3tB>.
- Catalani, G., Agarwal, S., Bertrand, X., Tost, F., Bauerheim, M., and Morlier, J., "Neural fields for rapid aircraft aerodynamics simulations," *Sci. Rep.* **14**, 25496 (2024).
- Cheng, L., Illarramendi, E. A., Bogopolsky, G., Bauerheim, M., and Cuenot, B., "Using neural networks to solve the 2d Poisson equation for electric field computation in plasma fluid simulations," *arXiv:2109.13076* (2021).
- Deschamps, F. and Sotin, C., "Thermal convection in the outer shell of large icy satellites," *J. Geophys. Res.* **106**, 5107–5121, <https://doi.org/10.1029/2000JE001253> (2001).
- Drilleau, M., Samuel, H., Rivoldini, A., Panning, M., and Lognonné, P., "Bayesian inversion of the Martian structure using geodynamic constraints," *Geophys. J. Int.* **226**, 1615–1644 (2021).
- Dumoulin, C., Doin, M.-P., and Fleitout, L., "Heat transport in stagnant lid convection with temperature- and pressure-dependent Newtonian or non-Newtonian rheology," *J. Geophys. Res. Solid Earth* **104**, 12759–12777, <https://doi.org/10.1029/1999JB900110> (1999).
- Gladstone, R. J., Rahmani, H., Suryakumar, V., Meidani, H., D'Elia, M., and Zareei, A., "Mesh-based GNN surrogates for time-independent PDEs," *Sci. Rep.* **14**, 3394 (2024).
- Greenfeld, D., Galun, M., Basri, R., Yavneh, I., and Kimmel, R., "Learning to optimize multigrid PDE solvers," in *International Conference on Machine Learning (PMLR, 2019)*, pp. 2415–2423.
- Grimm, V., Heinlein, A., and Klawonn, A., "A short note on solving partial differential equations using convolutional neural networks," in *Domain Decomposition Methods in Science and Engineering XXVII. DD 2022*, Lecture Notes in Computational Science and Engineering Vol. 149, edited by Dostál, Z. et al. (Springer, Cham, 2024).
- Grott, M., Breuer, D., and Laneuville, M., "Thermo-chemical evolution and global contraction of mercury," *Earth Planet. Sci. Lett.* **307**, 135–146 (2011).
- Gupta, V., Koren, T., and Singer, Y., "Shampoo: Preconditioned stochastic tensor optimization," in *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research Vol. 80 (PMLR, 2018), pp. 1842–1850; available at <https://proceedings.mlr.press/v80/gupta18a.html>.
- Gurnis, M., "A reassessment of the heat transport by variable viscosity convection with plates and lids," *Geophys. Res. Lett.* **16**, 179–182 (1989).
- Hao, Z., Su, C., Liu, S., Berner, J., Ying, C., Su, H., Anandkumar, A., Song, J., and Zhu, J., "Dpot: Auto-regressive denoising operator transformer for large-scale PDE pre-training," *arXiv:2403.03542* (2024).
- Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., and Zhu, J., "Gnot: A general neural operator transformer for operator learning," in *International Conference on Machine Learning (PMLR, 2023)*, pp. 12556–12569.
- Hendrycks, D. and Gimpel, K., "Gaussian error linear units (gelus)," *arXiv:1606.08415* (2016).
- Holzschuh, B., Liu, Q., Kohl, G., and Thuerey, N., "PDE-transformer: Efficient and versatile transformers for physics simulations," *arXiv:2505.24717* (2025).
- Horie, M. and Mitsume, N., "Physics-embedded neural networks: Graph neural PDE solvers with mixed boundary conditions," *arXiv:2205.11912* (2023).
- Horie, M. and Mitsume, N., "Graph neural PDE solvers with conservation and similarity-equivariance," *arXiv:2405.16183* (2024).
- Hüttig, C., Tosi, N., and Moore, W., "An improved formulation of the incompressible Navier-Stokes equations with variable viscosity," *Phys. Earth Planet. Inter.* **220**, 11–18 (2013).
- Innamorati, C., Ritschel, T., Weyrich, T., and Mitra, N. J., "Learning on the edge: Investigating boundary filters in CNNs," *Int. J. Comput. Vision* **128**, 773–782 (2020).
- King, S. D., Lee, C., van Keken, P. E., Leng, W., Zhong, S., Tan, E., Tosi, N., and Kameyama, M. C., "A community benchmark for 2-D Cartesian compressible convection in the Earth's mantle," *Geophys. J. Int.* **180**, 73–87 (2010).
- Knigge, D. M., Wessels, D., Valperga, R., Papa, S., Sonke, J.-J., Bekkers, E. J., and Gavves, S., "Space-time continuous PDE forecasting using equivariant neural fields," in *The Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS, 2024)*; available at <https://openreview.net/forum?id=wN5AgP0DJ0>.
- Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S., "Machine learning-accelerated computational fluid dynamics," *Proc. Natl. Acad. Sci. U. S. A.* **118**, e2101784118 (2021).
- Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Klöwer, M., Lottes, J., Rasp, S., Düben, P. et al., "Neural general circulation models for weather and climate," *Nature* **632**, 1060–1066 (2024).
- Korenaga, J., "Scaling of plate tectonic convection with pseudoplastic rheology," *J. Geophys. Res. Solid Earth* **115**, B11405, <https://doi.org/10.1029/2010JB007670> (2010).
- Korenaga, J. and Jordan, T. H., "Physics of multiscale convection in earth's mantle: Onset of sublithospheric convection," *J. Geophys. Res. Solid Earth* **108**, 2333, <https://doi.org/10.1029/2002JB001760> (2003).
- Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Meroze, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P., "Learning skillful medium-range global weather forecasting," *Science* **382**, 1416–1421 (2023).
- Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A., "Fourier neural operator with learned deformations for PDEs on general geometries," *J. Mach. Learn. Res.* **24**, 1–26 (2023a).
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., "Fourier neural operator for parametric partial differential equations," *arXiv:2010.08895* (2021).
- Li, Z., Kovachki, N., Choy, C., Li, B., Kossai, J., Otta, S., Nabian, M. A., Stadler, M., Hundt, C., Azizzadenesheli, K. et al., "Geometry-informed neural operator for large-scale 3D PDEs," *Adv. Neural Inf. Process. Syst.* **36**, 35836–35854 (2023b).
- Li, Z., Shu, D., and Barati Farimani, A., "Scalable transformer for PDE surrogate modeling," *Adv. Neural Inf. Process. Syst.* **36**, 28010–28039 (2023).
- Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A., "Physics-informed neural operator for learning partial differential equations," *arXiv:2111.03794* (2023).
- Lino, M., Pfaff, T., and Thuerey, N., "Learning distributions of complex fluid simulations with diffusion graph networks," *arXiv:2504.02843* (2025).
- Lippe, P., Veeling, B. S., Perdikaris, P., Turner, R. E., and Brandstetter, J., "PDE-refiner: Achieving accurate long rollouts with neural PDE solvers," in *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS, 2023)*; available at <https://openreview.net/forum?id=Qv6468llWS>.
- List, B., Chen, L.-W., and Thuerey, N., "Learned turbulence modelling with differentiable fluid solvers: Physics-based loss functions and optimisation horizons," *J. Fluid Mech.* **949**, A25 (2022).
- Liu, Q., Chu, M., and Thuerey, N., "ConFIG: Towards conflict-free training of physics informed neural networks," in *The Thirteenth International Conference on Learning Representations (ICLR, 2025)*; available at <https://openreview.net/forum?id=APojAzJQiq>.
- Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E., "Learning nonlinear operators via deep Pontryagin based on the universal approximation theorem of operators," *Nat. Mach. Intell.* **3**, 218–229 (2021).
- Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I., "Learning algebraic multigrid using graph neural networks," in *International Conference on Machine Learning (PMLR, 2020)*, pp. 6489–6499.
- Pathak, J., Mustafa, M., Kashinath, K., Motheau, E., Kurth, T., and Day, M., "Using machine learning to augment coarse-grid computational fluid dynamics simulations," *arXiv:2010.00072* (2020).
- Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comp. Phys.* **378**, 686–707 (2019).
- Ranade, R., Hill, D. C., Ghule, L., and Pathak, J., "A composable machine-learning approach for steady-state simulations on high-resolution grids," in *Advances in Neural Information Processing Systems*, edited by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho (Curran Associates Inc., 2022), p. 16.

- Reese, C., Solomatov, V., and Moresi, L.-N., "Heat transport efficiency for stagnant lid convection with dislocation viscosity: Application to Mars and Venus," *J. Geophys. Res.* **103**, 13643–13657, <https://doi.org/10.1029/98JE01047> (1998).
- Ronneberger, O., Fischer, P., and Brox, T., "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention* (Springer, 2015), pp. 234–241.
- Schubert, G., Turcotte, D. L., and Olson, P., *Mantle Convection in the Earth and Planets* (Cambridge University Press, 2001).
- Schulz, F., Tosi, N., Plesa, A.-C., and Breuer, D., "Stagnant-lid convection with diffusion and dislocation creep rheology: Influence of a non-evolving grain size," *Geophys. J. Int.* **220**, 18–36 (2020).
- Serrano, L., Boudec, L. L., Koupai, A. K., Wang, T. X., Yin, Y., Vittaut, J.-N., and Patrick, G., "Operator learning with neural fields: Tackling PDEs on general geometries," in *Thirty-Seventh Conference on Neural Information Processing Systems* (Curran Associates Inc., 2023), p. 31.
- Shahnas, M. H. and Pysklywec, R. N., "Toward a unified model for the thermal state of the planetary mantle: Estimations from mean field deep learning," *Earth Space Sci.* **7**, e2019EA000881 (2020).
- Shishchikhov, M., Hosseinmardi, S., and Bostanabad, R., "Parametric encoding with attention and convolution mitigate spectral bias of neural partial differential equation solvers," *Struct. Multidiscip. Optim.* **67**, 128 (2024).
- Solomatov, V. S., "Scaling of temperature- and stress-dependent viscosity convection," *Phys. Fluids* **7**, 266–274 (1995).
- Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A., "Learned coarse models for efficient turbulence simulation," *arXiv:2112.15275* (2022).
- Stevenson, D. J., Spohn, T., and Schubert, G., "Magnetism and thermal evolution of the terrestrial planets," *Icarus* **54**, 466–489 (1983).
- Tackley, P. J., "Modelling compressible mantle convection with large viscosity contrasts in a three-dimensional spherical shell using the yin-yang grid," *Phys. Earth Planet. Inter.* **171**, 7–18 (2008).
- Thiriet, M., Breuer, D., Michaut, C., and Plesa, A.-C., "Scaling laws of convection for cooling planets in a stagnant lid regime," *Phys. Earth Planet. Inter.* **286**, 138–153 (2019).
- Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K., "Accelerating Eulerian fluid simulation with convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17* (JMLR.org, 2017), pp. 3424–3433.
- Tosi, N., Breuer, D., and Spohn, T., "Evolution of planetary interiors," in *Treatise on Geophysics*, 2nd ed., edited by G. Schubert (Elsevier, 2014), pp. 185–208.
- Tosi, N. and Padovan, S., "Mercury, moon, mars: Surface expressions of mantle convection and interior evolution on stagnant-lid bodies," in *Mantle Convection and Surface Expressions*, edited by H. Marquardt, M. Ballmer, S. Cottar, and J. Konter (AGU Monograph Series, 2021), Chap. XVII.
- Um, K., Brand, R., Yun, F., Holl, P., and Thuerey, N., "Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers," *arXiv:2007.00016* (2021).
- Vyas, N., Morwani, D., Zhao, R., Shapira, I., Brandfonbrener, D., Janson, L., and Kakade, S. M., "SOAP: Improving and stabilizing shampoo using Adam for language modeling," in *The Thirteenth International Conference on Learning Representations* (ICLR, 2025); available at <https://openreview.net/forum?id=IDxZhXrpNf>.
- Wandel, N., Schulz, S., and Klein, R., "Metamizer: A versatile neural optimizer for fast and accurate physics simulations," in *The Thirteenth International Conference on Learning Representations* (ICLR, 2025); available at https://wandeln.github.io/Metamizer_webpage/.
- Wandel, N., Weinmann, M., and Klein, R., "Learning incompressible fluid dynamics from scratch-towards fast, differentiable fluid models that generalize," in *International Conference on Learning Representations* (ICLR, 2020); available at <https://openreview.net/pdf/a304e46c25faf8b1991a7580669d779b3c3e2cd6.pdf>.
- Wang, Q., Ren, P., Zhou, H., Liu, X.-Y., Deng, Z., Zhang, Y., Chengze, R., Liu, H., Wang, Z., Wang, J.-X., Wen, J.-R., Sun, H., and Liu, Y., "P²c²net: PDE-preserved coarse correction network for efficient prediction of spatiotemporal dynamics," in *The Thirty-Eighth Annual Conference on Neural Information Processing Systems* (NeurIPS, 2024); available at <https://openreview.net/forum?id=motlmXq3B1>.
- Wang, S., Wang, H., and Perdikaris, P., "Learning the solution operator of parametric partial differential equations with physics-informed deepnets," *Sci. Adv.* **7**, eabi8605 (2021).
- Wang, Y., Yao, Y., Guo, J., and Gao, Z., "A practical PINN framework for multi-scale problems with multi-magnitude loss terms," *J. Comput. Phys.* **510**, 113112 (2024b).
- Wassing, S., Langer, S., and Bekemeyer, P., "Adopting computational fluid dynamics concepts for physics-informed neural networks," in *AIAA SciTech 2025 Forum* (American Institute of Aeronautics and Astronautics, 2025).
- Wei, L. and Freris, N. M., "Multi-scale graph neural network for physics-informed fluid simulation: Multi-scale graph neural network for physics-informed fluid simulation," *Vis. Comput.* **41**, 1171–1181 (2024).
- Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., and Benson, S. M., "U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow," *Adv. Water Resour.* **163**, 104180 (2022).
- Wu, T., Maruyama, T., and Leskovec, J., "Learning to accelerate partial differential equations via latent global evolution," *arXiv:2206.07681* (2022).
- Yin, Y., Kirchmeyer, M., Franceschi, J.-Y., Rakotomamonjy, A., and Gallinari, P., "Continuous PDE dynamics forecasting with implicit neural representations," *arXiv:2209.14855* (2023).
- Zhong, S., McNamara, A., Tan, E., Moresi, L., and Gurnis, M., "A benchmark study on mantle convection in a 3-D spherical shell using CitcomS," *Geochim. Geophys. Geosyst.* **9**, Q10017 (2008).
- Zhou, H., Ma, Y., Wu, H., Wang, H., and Long, M., "Unisolver: PDE-conditional transformers are universal PDE solvers," *arXiv:2405.17527* (2024).