Capturing missing physics in climate model parameterizations using neural differential equations

Ali Ramadhan[1], John Marshall[1], Andre Souza[1], Xin Kai Lee[1,2], Ulyana Piterbarg[1], Adeline Hillier[1], Gregory LeClaire Wagner[1], Christopher Rackauckas[1], Chris Hill[1], Jean-Michel Campin[1], Raffaele Ferrari[1]

[1]Massachusetts Institute of Technology

[2]Imperial College London

Key Points:

- We describe a data-driven parameterization framework wherein a base parameterization is augmented with a neural network and trained online.
- We demonstrate that a neural differential equation can outperform existing models to parameterize vertical mixing in free convection.
- The approach is conservative, stable in time, independent of the time-stepper, and allows for the capture of upgradient and nonlocal fluxes.

Corresponding author: Ali Ramadhan, `alir@mit.edu`

Abstract

We explore how neural differential equations (NDEs) may be trained on highly resolved fluid-dynamical models of unresolved scales providing an ideal framework for data-driven parameterizations in climate models. NDEs overcome some of the limitations of traditional neural networks (NNs) in fluid dynamical applications in that they can readily incorporate conservation laws and boundary conditions and are stable when integrated over time. We advocate a method that employs a 'residual' approach, in which the NN is used to improve upon an existing parameterization through the representation of residual fluxes which are not captured by the base parameterization. This reduces the amount of training required and providing a method for capturing up-gradient and nonlocal fluxes. As an illustrative example, we consider the parameterization of free convection of the oceanic boundary layer triggered by buoyancy loss at the surface. We demonstrate that a simple parameterization of the process — convective adjustment — can be improved upon by training a NDE against highly resolved explicit models, to capture entrainment fluxes at the base of the well-mixed layer, fluxes that convective adjustment itself cannot represent. The augmented parameterization outperforms existing commonly used parameterizations such as the K-Profile Parameterization (KPP). We showcase that the NDE performs well independent of the time-stepper and that an online training approach using differentiable simulation via the Julia scientific machine learning software stack improves accuracy by an order-of-magnitude. We conclude that NDEs provide an exciting route forward to the development of representations of sub-grid-scale processes for climate science, opening up myriad new opportunities.

Plain Language Summary

Even with today's immense computational resources, climate models cannot resolve every cloud in the atmosphere or eddying swirl in the ocean. However, collectively these small-scale turbulent processes play a key role in setting Earth's climate. Climate models attempt to represent unresolved scales via surrogate models known as parameterizations. However, these parameterizations have limited fidelity and can exhibit structural deficiencies. Here we demonstrate that neural differential equations (NDEs) may be trained on highly resolved fluid-dynamical models of unresolved scales and act as data-driven parameterizations in an ocean model. NDEs overcome limitations of traditional neural networks in fluid dynamical applications in that they can incorporate conserva-

tion laws and are stable when integrated for long times. We argue that NDEs provide a new route forward to the development of surrogate models for climate science, opening up exciting new opportunities.

1 Introduction: The parameterization challenge in climate modeling

As is often the case in science and engineering problems which address complex systems, numerical models have become central to the study of Earth's climate (Hourdin et al., 2017). Climate models have historically been skillful at projecting changes in global mean surface temperature (Hausfather et al., 2020). However, they often have regional deficiencies and biases which compromise future projections, especially on regional and local scales (C. Wang et al., 2014). Unfortunately, more certain regional and local information is precisely what is needed to make better decisions designed to mitigate and adapt to the effects of climate change (Katz et al., 2013).

A major source of uncertainty in climate *projections* is due to *missing physics*, that is small-scale physical processes that cannot be resolved in climate models, such as cloud formation in the atmosphere (Stevens & Bony, 2013) and small-scale boundary layer turbulence (DuVivier et al., 2018) and mixing by mesoscale eddies in the ocean (Griffies et al., 2015). Such unresolved physical processes must be represented somehow if one is to faithfully model the evolution of Earth's climate. Instead of explicitly resolving such phenomena, which is computationally not feasible (Schneider et al., 2017), a more computationally efficient surrogate model, or *parameterization*, is employed to represent their transport properties.

Parameterization schemes are typically developed through guidance from theory and observations but necessarily have an empirical flavor to them. For example, small-scale oceanic boundary layer turbulence cannot be resolved due to prohibitive computational costs; the coarse-grained equations are not closed and so empirical choices must be made in developing parametric representations. In the case of cloud formation, cloud microphysics and entrainment processes are not fully understood and so again, empirical choices must be made. Each parameterization thus inevitably has associated with it uncertain parameters — such as mixing length scales or an exponent in a scaling law — whose values must be prescribed but which are often difficult to infer from observations and associated with large uncertainties.

Embedding multiple parameterizations into a climate model to represent myriad unresolved scales thus introduces many free parameters that must be jointly estimated. The parameters may be correlated and point estimates of the optimal parameter values may be impossible, further complicating the calibration process (Souza et al., 2020). One common tuning procedure is to modify the parameters in an empirical fashion, but guided by intuition and understanding, in an effort to tune the climate model to reproduce the climate of the 20^{th} century (Hourdin et al., 2017) during which global observations are available. Once the model has been calibrated on historical data it is then used to extrapolate into the future. It is clear that such projections must necessarily be compromised due to the uncertainty introduced by parameterizations of unresolved scales.

Previous data-driven approaches to improving the fidelity of parameterizations have been undertaken. Souza et al. (2020) attempted to automatically calibrate and quantify the uncertainty in a parameterization of oceanic convection using simulated data from high-resolution large-eddy simulations (LES). They learned that the parameterization of interest was structurally deficient because the optimal value of the parameters depended on the physical scenario. This finding suggests that some parameterizations may not be calibrated due to structural deficiencies. In such a case, developing a new parameterization would seem desirable.

Rasp et al. (2018) trained a neural network to represent all atmospheric sub-grid processes in a climate model with explicit convection. Though the trained model could potentially surpass the parameterizations that it learned from in speed, it could not improve on their accuracy. Using a single model to substitute for parameterizations of multiple processes also degrades interpretibility because if the neural network behaves in unexpected ways then it is very difficult to ascertain underlying reasons. If neural networks do not exactly obey conservation laws, solutions can drift from reality ultimately leading to numerical instability or blowup. Gentine et al. (2018) improved the approach by training a neural network to learn atmospheric moist convection from many superparameterized simulations that explicitly resolve moist convection. However, the 2D superparameterized models do not always faithfully resolve the inherently 3D nature of atmospheric moist convection and their neural network also does not obey conservation laws.

O'Gorman and Dwyer (2018) train a random forest to parameterize moist convection which has the advantage of obeying conservation laws and preserving the non-negativity

of precipitation. However, random forests make predictions by matching current inputs to previously trained-upon inputs using an ensemble of decision trees. Such an approach can lead to a very large memory footprint and so be computationally demanding if trained on copious data. Furthermore, random forests do not readily generalize outside the training set. They find that training on a warm climate leads to skillful predictions for a colder climate because the extra-tropics of the warm climate provide training data for the tropics of the control climate. This reminds us of the need to train on a wide range of physical scenarios. Yuval et al. (2021) extend the approach but switch to using a neural network. They train the neural network to predict coarse-grained subgrid fluxes rather than tendencies and so are able to obey conservation laws. They find that the NN performs similarly to the random forest while using much less memory.

Bolton and Zanna (2019) train a 2D convolution neural network on subgrid eddy momentum forcings from an idealized high-resolution quasi-geostrophic ocean model mimicing a double gyre setup such as the Gulf Stream in the North Atlantic. They find it is capable of generalizing to differing viscosities and wind forcings. However, global momentum conservation must be enforced via a post-processing step to obtain optimal results. Zanna and Bolton (2020) take a different approach and attempt to learn analytic closed-form expressions for eddy parameterizations with embedded conservation laws. The learned parameterization resembles earlier published closures but is less stable than the convolutional neural network approach. Thus, it not yet clear that equation discovery will necessarily lead to improved parameterization schemes, or that training on quasi-geostrophic models can be transferred to ocean models based on more complete equation sets.

Taking a different approach to previous studies, here we describe a new route to developing data-driven parameterization in which neural differential equations (NDEs) are used to augment simple and robust existing parameterizations. The NDEs are trained on high-resolution simulations of turbulence in the surface boundary layer of the ocean and embedded into an ocean model. We describe an encouraging proof-of-concept exploration applied to free convection in the turbulent oceanic boundary layer similar in spirit to Souza et al. (2020). We outline next steps and suggest a strategy to tackle more difficult parameterization problems.

Our paper is set out as follows. In section 2 we introduce the concept of NDEs and how they might be used to augment and improve existing parameterization schemes. In section 3 we set up an NDE which will be used to improve a simple convective adjustment parameterization of free convection. In section 4 we describe how training data can be generated from high-resolution LES simulations of the ocean boundary layer. Section 5 describes how the NDE is trained on this LES data using two different methods. In section 6 we compare the fidelity and performance of the trained NDE to commonly used parameterizations. In section 7 we explore different neural network architectures to investigate whether nonlocality is important for the representation of free convection and to investigate the robustness of our approach to changes in neural network architecture. In section 8 we discuss the advantages NDEs more broadly and discuss ways in which we might improve the performance and interpretability of our approach. Finally, we summarise and discuss how NDEs might be applied to more complex and challenging parameterization problems.

2 Why use neural differential equations?

2.1 What are neural differential equations?

Machine learning techniques have recently gained great popularity in the modeling of physical systems because of their sometimes limited ability to automatically learn nonlinear relationships from data. However, although very promising, machine learning techniques have some notable disadvantages, such as their inability to extrapolate outside the parameter range of the traning data (Barnard & Wessels, 1992) and their lack of physical interpretation (Fan et al., 2021). Fortunately, recent work has shown that such concerns can be overcome, or at least ameliorated, by embedding prior structural knowledge into the machine learning framework (Xu et al., 2020). In the context of scientific models, the universal differential equation (Rackauckas et al., 2020) approach mixes known differential equation models with universal function approximators (such as neural networks) and demonstrates the ability to learn from less data and be more amenable to generalization. In this paper we will define a neural differential equation (NDE) as a differential equation where at least one term contains a neural network. The neural network contains free parameters that must be inferred for the NDE to produce useful predictions and be a useful model. Because they are phrased as a differential equation model, NDEs match the mechanistic form of physical laws, allowing for interpretability whilst

presenting a learnable structure. Section 3 showcases how this NDE architecture allows for combining prior knowledge that inform existing parameterizations directly with neural networks. NDEs are trained by performing automatic differentiation through a differential equation solver provided by the Julia scientific machine learning (SciML) software stack (Rackauckas & Nie, 2017).

2.2 Data-driven parameterizations using a residual model

The NDE approach has great promise, we believe, because many parameterizations of unresolved processes in climate modeling can be posed as a partial differential equation (PDE) with known terms and boundary conditions, but also unknown or incompletely known terms. Our approach is to include the known physics explicitly but to employ neural networks to represent all remaining terms. The neural network is trained on data—either observational or, as here, synthetic data from high-resolution simulations—that resolves the missing processes.

One might be tempted to use a neural network to capture all the turbulent fluxes of unresolved processes. Indeed perhaps a large enough network could be trained on enough data for this to be accomplished. However, the amount of data required may be prohibitively large and the resulting trained network might be larger than necessary. Instead, it is more efficient to harness as much physical knowledge as possible and deploy the neural network to learn physics that is difficult to parameterize. Such an approach also greatly reduces the burden on the neural network in terms of its training, which can be a great burden on computational resources.

The approach we take here, is to adopt an existing theory-driven *base parameterization* that encapsulates the robust physics in a simple, interpretable manner and augment it with a neural network that predicts the unresolved physics. In this way the NDE can only improve upon the parameterization adopted.

It is important that the neural network and the NDE in to which it is embedded, is constructed in such a manner that it obeys all pertinent physical laws, such as any conservation principles and boundary conditions. For example, a base parameterization might be responsible for predicting the flux of some quantity, for example temperature T which, in the application here, is the turbulent flux of heat. The neural network should not add or remove any heat from the domain, but only redistribute it within the interior. To en-

code this idea as a conservation law, we write the right-hand-side (RHS) of the temperature tendency equation as the divergence of a flux $\boldsymbol{F}$ thus

$$\partial_t T = \nabla \cdot \boldsymbol{F} \tag{1}$$

enabling us to guarantee conservation by applying appropriate boundary conditions at the edges of the domain. The neural network responsible for predicting a component of the fluxes then only redistributes properties, and does not act as a source or a sink. Surface fluxes can then be prescribed as boundary conditions.

Since we know some of the robust physics that enters the RHS, but not all of the physics, we separate the flux out as

$$\boldsymbol{F} = \boldsymbol{F}_{\text{param}} + \boldsymbol{F}_{\text{missing}} \tag{2}$$

where $\boldsymbol{F}_{\text{param}}$ will be computed using an existing base parameterization while

$$\boldsymbol{F}_{\text{missing}} = \mathbb{NN}(\text{prognostic variables}, \text{surface fluxes}, \cdots) \tag{3}$$

will be predicted by the neural network $\mathbb{NN}$ using available information such as the current state of the prognostic variables, surface fluxes and boundary conditions, etc. Note that $\mathbb{NN}$, denoted such since we are using neural networks, need not be a neural network and could be any function approximator such as a Gaussian process or a polynomial series.

The guiding principles taken here are that:

- $\boldsymbol{F}_{\text{param}}$ should be *simple* and based on robust well-known physics, and that
- the neural network is only employed to address the *complicated stuff* or *missing physics* we do not know how to represent.

Note that studying the fluxes predicted by the neural network could lead to an improved understanding of the missing physics and hence lead to the development of *better* structured parameterizations, whether they be theory-driven or data-driven.

One major advantage of taking such a *residual* parameterization approach is that an existing base parameterization can be readily used to provide an excellent first guess using, for example, down-gradient mixing assumptions. The neural network can then focus on predicting (hopefully smaller) residual fluxes, reducing the amount of training required. Moreover, the neural network can potentially capture up-gradient fluxes and non-local effects which are typically difficult to model using existing structured closures.

In the remainder of our paper we put these ideas in to practice in the context of free convection within an initially stratified, resting patch of ocean subject to heat loss at its upper boundary.

3 Parameterizing free convection as a neural differential equation

3.1 Free convection

Free convection occurs when a stably stratified fluid is cooled at its upper surface, or acted upon by some other destabilizing buoyancy flux, such as ejection of salt in ice formation. This loss of buoyancy at the surface leads to fluid parcels being made dense relative to their surroundings and so they sink, displacing buoyant fluid upwards. In this way a convectively-driven turbulent mixed layer is created that deepens with time (Marshall & Schott, 1999). Such a process occurs everywhere in the ocean as solar radiation and clouds cause the atmosphere to warm and cool the ocean surface over the diurnal and seasonal cycles: see the review of (Marshall & Plumb, 2007). The depth of the mixed layer and the strength of the convectively-driven mixing are important factors in setting the global climate: they determine, for example, how much heat, carbon, and other soluble gases are sequestered from the atmosphere into the ocean because exchanges between the two fluids occur through this mixed layer. The surface mixed layer alone contains as much carbon as in the atmosphere. Moreover, the ocean absorbs the vast majority (over 90%) of the excess heat in the climate system induced by greenhouse gases (Resplandy et al., 2019). The mixed layer is also crucial for ocean biochemistry as it sets the vertical scale over which many marine organisms yo-yo up and down and thus the amount of sunlight they have access to and hence their growth rate. For example nutrient replenishment in the wintertime North Atlantic is associated with deep convection and to springtime blooms when the mixed layer is shallow and light is plentiful (Williams

& Follows, 2011, §7.2). Thus accurately predicting the mixed layer depth and fluxes through it is critical for ensuring the fidelity of the models used to make climate projections.

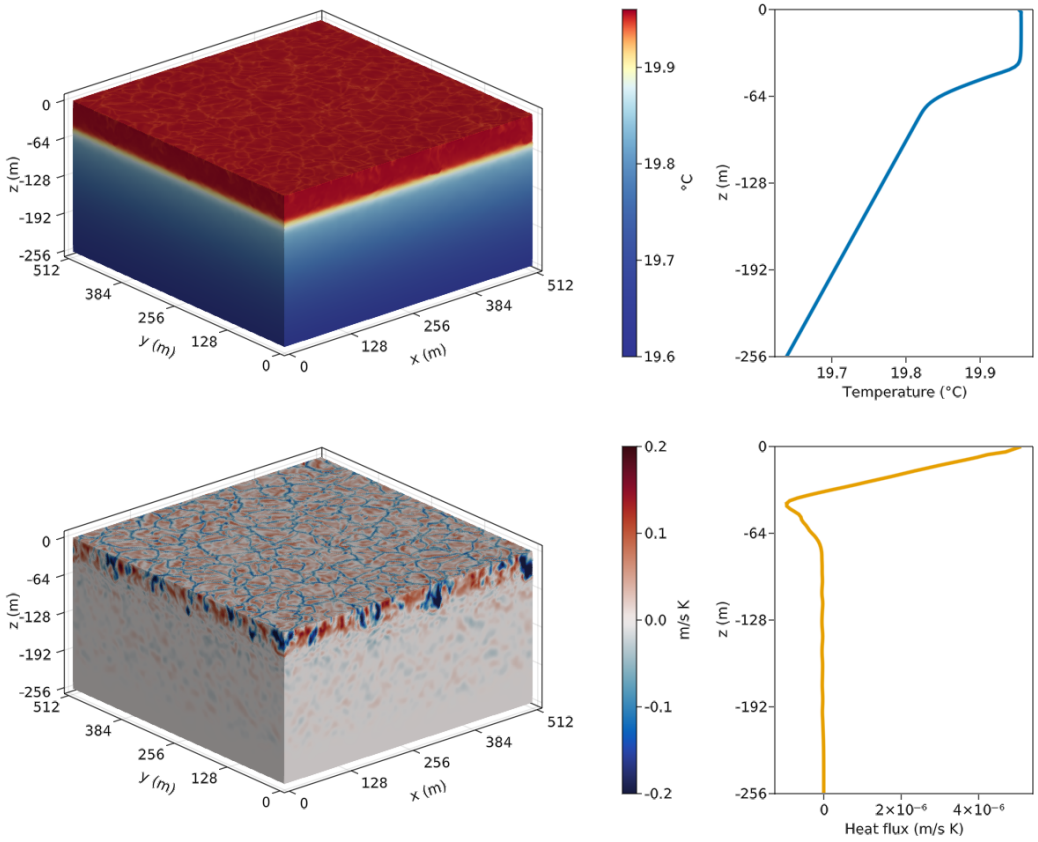


Figure 1. Snapshot from an LES simulation of small-scale oceanic boundary layer turbulence forced by a surface cooling in a horizontally periodic domain using $256^2 \times 128$ cells and simulated using the Oceananigans.jl software package. (Left) Snapshots of the temperature field T (top left) and turbulent vertical temperature flux field $w'T'$, proportional to the heat flux (bottom left) at $t = 2\,\text{days}$. (Right) The horizontally-averaged temperature $\overline{T}(z)$ and turbulent vertical heat flux $\overline{w'T'}(z)$ at the same time snapshot. Note that the heat flux profile (bottom right) includes the sub-grid scale eddy diffusivity $\overline{\kappa_e \partial_z T}(z)$ (see appendix A for details).

Wind stresses are also present at the ocean's surface which drive turbulent momentum fluxes in addition to buoyancy fluxes, further complicating the physics of boundary layer. That said, free convection is an important physical process which must be captured in models. Its accurate representation is non-trivial due to the entrainment of fluid through the base of the mixed layer where convective plumes penetrate down into the

stratified fluid below. This leads to the development of an entrainment layer below the mixed layer. This entrainment physics is very difficult to represent and parameterize — see Souza et al. (2020); Van Roekel et al. (2018). Instead of attempting to develop another theory-driven parameterization of these entrainment fluxes, we will deploy a neural network within the framework of NDEs to capture them.

Figure 1 shows a snapshot from a large eddy simulation of oceanic free convection. The mixed layer is the region of vertically-uniform (mixed) temperature spanning roughly the upper 50 m of the domain. Over this layer the upward heat flux decreases linearly from its surface value down toward zero at the base of the layer. The entrainment region can readily be seen at the base of the mixed layer, at a depth of roughly 60 m, where the heat flux becomes negative.

Explicitly simulating free convection requires solving the three-dimensional Boussinesq equations. However a look at figure 1 suggests that the turbulence is horizontally homogeneous and that we may be able to predict its vertical effects without simulating the extra horizontal dimensions. A one-dimensional model of free convection in a column of water can be obtained by Reynolds averaging the advection-diffusion equation for temperature [see equation (16) in appendix A] to obtain the following one-dimensional PDE for the temperature

$$\partial_t \overline{T} = -\partial_z \overline{w'T'}. \tag{4}$$

Here an overline indicates a horizontal average of a three-dimensional time-varying quantity

$$\overline{\phi} = \overline{\phi}(z,t) = \frac{1}{L_x L_y} \int_0^{L_x} \int_0^{L_y} \phi(x,y,z,t)\,\mathrm{d}x\,\mathrm{d}y \tag{5}$$

and a prime indicates a departure from the horizontal mean $\phi'(x,y,z,t) = \phi(x,y,z,t) - \overline{\phi}(z,t)$: L_x and L_y are the domain lengths along the x and y dimensions: $\overline{w'T'}$ is the turbulent vertical heat flux responsible for redistributing heat and is the focus of our attention.

In the absence of phase changes, the fluid dynamics of the upper ocean are well described by the Boussinesq equations which can be solved numerically. Thus LES sim-

ulations can be run to provide reliable training data for $\overline{w'T'}$. In LES simulations the major part of the turbulent heat flux is achieved by resolved motions and an LES closure is used to represent any sub-grid fluxes through, in the present study, an eddy diffusivity. Thus equation (4) can be written as

$$\partial_t \overline{T} = -\partial_z \overline{w'T'} = -\partial_z (\overline{w'T'}|_{\text{advective}} - \overline{\kappa_e \partial_z T}). \tag{6}$$

Thus $\overline{w'T'}$ is explicitly represented via advection of temperature by the resolved flow $\overline{w'T'}|_{\text{advective}}$ (computed as the resolved vertical velocity anomaly w' multiplied by the resolved temperature anomaly T') while sub-grid heat fluxes are accounted for via an eddy diffusivity $\kappa_e(x, y, z)$ modeled using an LES closure (see appendix A for more details). The diffusive eddy heat fluxes are thus a small fraction of the advective heat fluxes. In this one-dimensional model of free convection we neglect background or molecular diffusion of temperature since the molecular thermal diffusivity coefficient of seawater [$\kappa \sim \mathcal{O}(1 \times 10^{-7}\,\mathrm{m^2\,s^{-1}})$] is orders of magnitude smaller than the typical eddy diffusivity deployed in the forward model [$\kappa_e \sim \mathcal{O}(1 \times 10^{-2}\,\mathrm{m^2\,s^{-1}})$ in the turbulent mixed layer].

3.2 Base parameterization: Convective adjustment

It is well known that, in the invicid limit, if dense fluid lies above a lighter fluid, gravitational instability will ensue of the kind seen in figure 1, leading to vertical mixing (Haine & Marshall, 1998). This vertical mixing occurs due to inherently three-dimensional small-scale free convection with length scales $\mathcal{O}(1 \times 10^{-1}\,\mathrm{m})$ which cannot be resolved by global ocean models whose vertical grid spacing is $\mathcal{O}(1\,\mathrm{m})$ and horizontal grid spacing is $\mathcal{O}(10\,\mathrm{km})$ and thus the vertical mixing must be parameterized. If the mixing is not resolved numerically the model will produce statically unstable water columns with dense fluid atop lighter fluid.

As a simple, physically-plausible base parameterization we choose convective adjustment. This will capture most of the resolved flux, but not all of it. Convective adjustment is a simple parameterization that represents this mixing via a large vertical diffusivity if vertical buoyancy gradients imply static instability (Klinger et al., 1996). It can readily capture the vertical structure of the boundary layer except for the entrainment region at its base since it makes no attempt to account for entrainment processes.

If a neural network can be trained to accurately predict the turbulent fluxes associated with entrainment then augmenting convective adjustment with such a neural network will likely increase the skill of the parameterization.

In this paper convective adjustment is implemented as a diffusive process in which a large diffusivity K_{CA} is turned on in regions of the water column that are statically unstable:

$$\overline{w'T'}(z,t) \approx -\kappa_{\text{CA}}(z,t)\partial_z\overline{T}(z,t) \quad \text{where} \quad \kappa_{\text{CA}}(z,t) = \begin{cases} K_{\text{CA}}, & \text{if } \partial_z\overline{T}(z,t) < 0 \\ 0, & \text{otherwise} \end{cases}. \tag{7}$$

The one free parameter K_{CA} must be chosen which is done here by calibrating convective adjustment against the same training simulations as the NDE. We find that the optimal convective adjustment diffusivity is $K_{\text{CA}} \approx 0.2\,\text{m}^2\,\text{s}^{-1}$ (see appendix E for details). This value is adopted and kept at the same constant value across all our experiments.

3.3 The Neural Differential Equation for free convection

As convective adjustment does not account for entrainment that occurs at the base of the mixed layer, we will now augment it with a neural network which will be trained to address the entrainment fluxes.

We write down the heat flux from the one-dimensional free convection model including convective adjustment thus:

$$\overline{w'T'} = \overline{w'T'}|_{\text{CA}} + \overline{w'T'}|_{\text{missing}} = -\kappa_{\text{CA}}\partial_z\overline{T} + \overline{w'T'}|_{\text{missing}}. \tag{8}$$

where $\overline{w'T'}|_{\text{CA}}$ is the turbulent vertical heat flux predicted by convective adjustment and $\overline{w'T'}|_{\text{missing}}$ is the missing portion.

Comparing the form of $\overline{w'T'}$ above to that obtained by horizontally averaging the equations of the LES model,

$$\overline{w'T'} = \overline{w'T'}|_{\text{advective}} - \overline{\kappa_e\partial_z T} \tag{9}$$

we identify the missing heat flux as

$$\overline{w'T'}|_{\text{missing}} = \overline{w'T'}|_{\text{advective}} - \overline{\kappa_e \partial_z T} + \kappa_{\text{CA}} \partial_z \overline{T}. \tag{10}$$

The quantity on the left-hand side, $\overline{w'T'}|_{\text{missing}}$, must be learnt, while the quantities on the right are all known, being provided by the LES data (for $\overline{w'T'}|_{\text{advective}} - \overline{\kappa_e \partial_z T}$) and the base parameterization ($\kappa_{\text{CA}} \partial_z \overline{T}$). We use a neural network to represent the missing flux,

$$\overline{w'T'}|_{\text{missing}}(z) = \mathbb{NN}\left[\overline{T}(z)\right], \tag{11}$$

by training it to learn the relationship between equation (10) and the temperature profile $\overline{T}$.

Noting that $\overline{w'T'} = \overline{w'T'}|_{\text{param}} + \overline{w'T'}|_{\text{missing}}$ and substituting the above into the PDE, equation (4), we obtain our NDE — a differential equation with a neural network representing missing physics:

$$\partial_t \overline{T} = -\partial_z \overline{w'T'} = -\partial_z \left[\mathbb{NN}\left(\overline{T}\right) - \kappa_{\text{CA}} \partial_z \overline{T}\right] \tag{12}$$

The input to the NN is just the current state of the temperature profile $\overline{T}$. So the job of the NN is to predict $\overline{w'T'}|_{\text{missing}}$ from the temperature profile $\overline{T}$. To train the NDE we will non-dimensionalize equation (12) (see appendix D for a derivation of the non-dimensional NDE) used in our codes. The NDE is implemented using DifferentialEquations.jl and Julia's scientific machine learning (SciML) software stack (Rackauckas & Nie, 2017).

3.4 Architecture of the Neural Network

The NN predicts the missing (residual) fluxes from the prognostic variables constituting the state of the model. In the present application, as shown schematically in figure 2, we found it sufficient to supply just the temperature profile $\overline{T}(z)$ as input with $\overline{w'T'}|_{\text{missing}}(z)$ as the output.

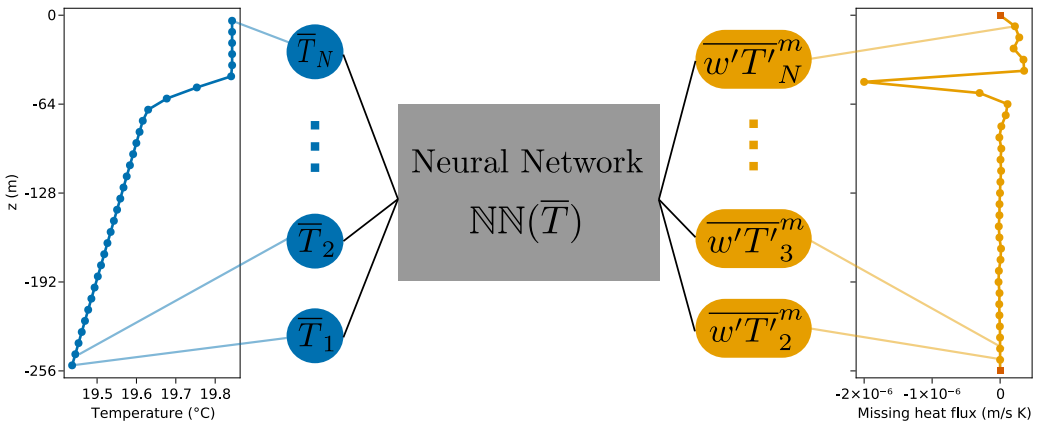


Figure 2. Schematic representation of the architecture of the neural network. Temperature values are colored in blue while predicted missing (equivalently, residual) fluxes are colored yellow. A temperature profile consisting of N values is fed into the neural network which predicts $N-1$ values for the missing heat flux in the interior of the domain. The $\overline{T}(z)$ and $\overline{w'T'}|_{\text{missing}}(z)$ profiles shown here are one example from the training data. See main text for more details.

The input of the neural network is a temperature profile vector $\overline{T}(z)$ consisting of $N = 32$ floating-point values as shown on the leftmost plot of figure 2 and denoted as $\overline{T}_1, \overline{T}_2, \cdots, \overline{T}_N$. The N blue points show the temperature values which are the N inputs to the neural network, shown as bigger blue circles connected to the values. The neural network outputs $N-1$ values predicting the missing heat flux $\overline{w'T'}|_{\text{missing}}(z)$ in the interior of the domain denoted as $\overline{w'T'}^m_2, \overline{w'T'}^m_3, \cdots, \overline{w'T'}^m_N$. Yellow dots are fluxes predicted by the neural network while red squares are boundary fluxes that are prescribed and thus do not need to be predicted by the neural network. Knowledge of the boundary fluxes, together with the form of our NDE, clearly guarantees that conservation is obeyed. In all the calculations presented here we set $N = 32$ so that the resolution of the NDE is similar to that of a typical vertical mixing parameterization embedded in an ocean model.

We use simple feed-forward neural networks in which the input propagates through a series of layers with each layer taking an input then transforming it into an output before passing the output onto the next layer. The output may be fed through an activation function before being passed on. A layer may perform a linear transformation carried out by an activation function σ: $x \rightarrow \sigma(Wx + b)$. The layer can also perform a

convolution or any other transformation on its input. The parameters of the neural network comprise entries of the weight matrix W and bias vector b for each layer. They are optimized using specialized optimization algorithms, many of which use various forms of gradient descent to minimize a loss function. In this way we ensure that the neural network learns some relationship between the inputs and outputs. For an introduction to neural networks and how to train them, see Mehta et al. (2019).

We choose to begin exploration by adopting a simple architecture for the neural network, denoted here by $\mathbb{NN}$, a series of fully-connected dense layers in which each layer performs the transformation $x \to \text{relu}(Wx+b)$ and where the activation function is a rectified linear unit (relu) defined as $\text{relu}(x) = \max(0, x)$. In §7 we will consider different $\mathbb{NN}$ architectures including wider and deeper networks as well as convolutional layers. The architectures used are catalogued in appendix G. The Julia package Flux.jl (Innes, 2018) is used to set up the neural networks and train them.

The neural network takes N input values and outputs $N-1$ values. This is because the one-dimensional model of free convection is implemented using a finite volume method on a staggered grid — sometimes called the Arakawa C-grid after Arakawa and Lamb (1977) — so there are N cell centers where the temperature is located but $N+1$ cell interfaces where the turbulent heat fluxes are located. The top-most and bottom-most cell interfaces correspond to the upper and lower boundaries where heat fluxes are prescribed and therefore known. This is why the neural network only predicts the remaining fluxes at the $N-1$ cell interfaces in the interior of the domain.

4 Generation of training data using large eddy simulations

4.1 Training data from high resolution simulations of free convection

Ideally one would use observations of temperature profiles and turbulent heat fluxes from the real ocean to provide training data. While ocean observations exist in many regions of the ocean especially thanks to Argo floats (Roemmich et al., 2009), they are not available at sufficiently high spatial and temporal resolutions in the mixed layer, and are subject to measurement error. Furthermore, observations of the contemporaneous surface fluxes are not usually available.

We could employ a very high-resolution global ocean simulation from which training data could be extracted covering a wide range of physical scenarios. However, the

resolution required to resolve free convection is $\mathcal{O}(1\,\mathrm{m})$ (Souza et al., 2020) which is far beyond current computational capabilities in the context of a global simulation (Fox-Kemper et al., 2019). Instead, we deploy the very high resolution LES model shown in figure 1. By varying the surface buoyancy flux and the initial stratification (by setting an initial temperature profile) the box simulations can span a large range of physical scenarios and provide a rich variety of training data in a controlled and well-defined setting.

4.2 Simulation setup

Simulations of free convection are run by numerically solving the Boussinesq equations with an LES closure to represent sub-grid fluxes. The LES is initialized with a stratified fluid at rest with an idealized vertical structure consisting of a weakly-stratified surface layer and a weakly-stratified abyssal/interior layer separated by a strongly-stratified thermocline. A constant surface cooling is applied via a surface buoyancy flux leading to gravitational instability and the formation of a deepening mixed layer. The LES simulations are performed using the Oceananigans.jl software package. The numerical methods employed are described in appendix A and the simulation setup is described in more detail in appendix B. Horizontally-averaged output taken at regular time intervals from these simulations is used as training data. A snapshot from a typical solution is shown in figure 1.

The LES simulations are run using $256^2 \times 128$ elements with a uniform size of 1 m. However, the parameterization is run in one dimension at a coarser resolution chosen, typical of that used in general circulation ocean models (GCM). This is because the parameterization is intended to eventually embed into a global GCM which typically employ coarser grids. Ocean GCMs often use a vertically stretched grid which is coarser deeper in the ocean, but reaches $\mathcal{O}(10\,\mathrm{m})$ near the surface. We use a constant vertical resolution of $\Delta z = 8\,\mathrm{m}$ yielding $N_z = 32$ vertical levels. Thus the horizontally-averaged simulation output is coarse-grained from a vertical grid of 256 elements to one of 32 elements to provide training data.

The training data consists of a time series of horizontally-averaged temperature $\overline{T}(z,t)$ and horizontally-averaged turbulent vertical heat flux $\overline{w'T'}(z,t)$ from each simulation. They are discretized in space and time denoted, for example, by $\overline{T}_{k,n} = \overline{T}(z_k, t_n)$ where z_k is the k^{th} vertical coordinate ($k \in \{1, \cdots, N_z\}$) and t_n is the n^{th} time snapshot ($n \in$

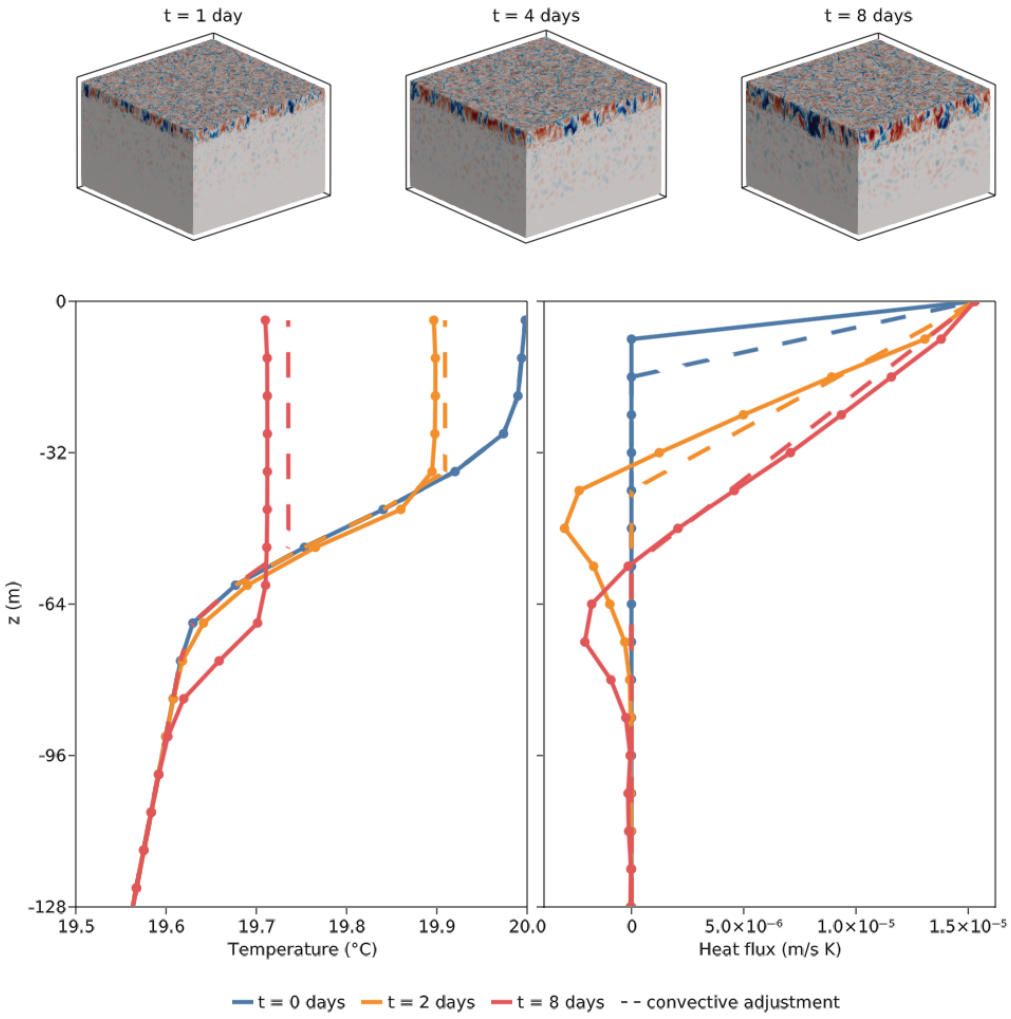


Figure 3. Snapshots of temperature and heat flux profiles during the evolution of a convectively driven boundary layer driven by cooling from the surface. Solid lines show the true (LES) solution at different times and the points show the location of coarse-grained temperature values used by the neural network. The dashed lines show the solution produced by the convective adjustment parameterization. The heat flux profile (bottom right) includes the diffusive contribution $\overline{\kappa_e \partial_z T}(z)$ (see appendix A for more information). The profiles are taken from simulation 5 of table 1 in appendix C. Note that only half the vertical domain is shown but the full solution extends down to 256 m.

$\{1, \cdots, N_t\}$). N_z is the number of vertical grid elements and N_t is the number of time snapshots. Figure 3 shows a snapshot of the training data from one of the training simulations. The predictions from a convective adjustment model are also plotted using dashed

lines. The difference between the LES heat flux and the heat flux from convective adjustment is precisely the missing heat flux that the neural network embedded in the NDE will learn to predict. In this way the NDE can close the gap between the simple convective adjustment parameterization scheme and the full LES simulation. With the neural network providing the missing entrainment heat fluxes, we expect the temperature profiles to evolve with skill.

4.3 Training and validation cases

Our NDE is to be designed to perform well across a wide range of surface buoyancy fluxes and stratifications. We must necessarily train our NDE on a limited set of simulations, however, but can evaluate its ability to generalize by assessing how it performs when asked to make predictions for heat fluxes and stratifications it has not been trained on. We can test whether the NDE can interpolate — that is make a correct prediction when the physical scenario is distinct but not more extreme than scenarios it has been trained on. We can also test whether the NDE can extrapolate — that is perform well in scenarios outside of its training space. Interpolation and extrapolation is tested by varying both the surface buoyancy flux, Q_b, and the stratification, N^2. In the present study we used 9 training simulations and 12 testing/validation simulations split into 4 sets of 3 simulations: Q_b interpolation, Q_b extrapolation, N^2 interpolation, and N^2 extrapolation, as set in figure 4.

Figure 4 shows the position of the 21 simulations in parameter space and whether they were used for training or validation. The simulation parameters used are tabulated in table 1 in appendix C. The NDE was trained on 3 different values of Q_b (blue points) and 3 different initial N^2 profiles, leading to the 9 training simulations (colored blue). To evaluate interpolation capabilities, values of Q_b in between the three training values were chosen (simulations 9–12, colored green): to evaluate extrapolation capabilities, values smaller and larger than the training values were chosen (simulations 13–15, colored purple). Initial N^2 profiles were taken from the training simulations so that only Q_b was varied.

To vary different initial N^2 profiles, the thicknesses and stratifications of the surface and thermocline layers were changed (see appendices B and C for details) and the range of initial N^2 are visualized as kernel density estimates in figure 4 showing the dis-

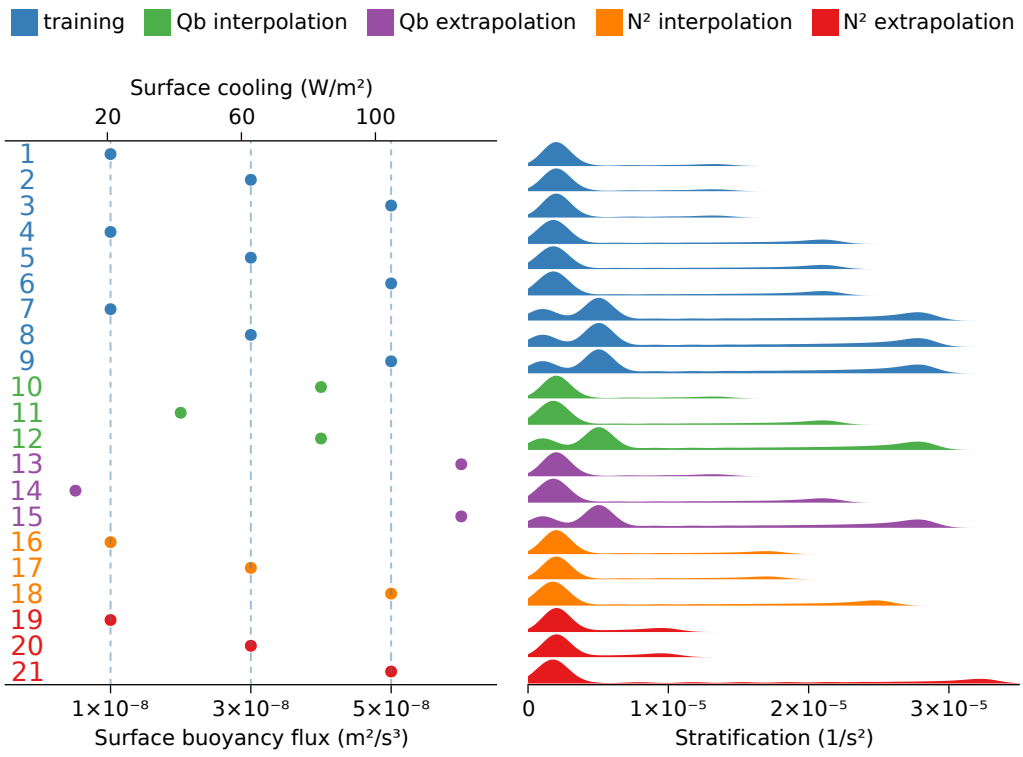


Figure 4. Position in parameter space of simulations used for training and validation. Each simulation differs in the assumed surface heat/buoyancy flux Q_b and initial stratification N^2 set by the initial vertical temperature distribution. Simulations 1–9 were used for training (colored in blue) while simulations 10–21 were used to test interpolation and extrapolation capabilities. (Left) The surface buoyancy flux of each simulation with dashed lines indicating the three values which provided training data. (Right) Kernel density estimates of the initial stratification distribution of each simulation.

tribution of N^2 present in the initial profile. To evaluate interpolation capabilities (simulations 16–18, colored orange) the simulation parameters were varied to create an initial N^2 profile that is different from that of the training data but which exhibits stratifications between those of the training simulations. This is evident in the plotted distributions (orange kernel density estimates) as their peaks either overlap with or are in between the peaks of the distributions for the training simulations. To evaluate extrapolation capabilities (simulations 19–21, colored red) the simulation parameters were varied to create two very weakly stratified simulations (19 and 20) and one very strongly

stratified simulation (21). Values of Q_b were chosen from those used in training so that only the initial N^2 profile was varied.

5 Training the Neural Differential Equation

In section 2 we discussed the key idea behind NDEs in which unknown terms are represented by neural networks and trained to learn the unknowns. This can be done in two different ways. The neural network can be trained independently of the NDE using profiles of the missing fluxes and then brought into the NDE. This first method can be called *differential control* as the neural network is trained to predict fluxes at instances in time and so it learns derivatives or rates at instances in time. Alternatively, the NDE can be trained directly on the entire temperature time series with the neural network embedded within it. This second approach can be called *integral control* as the neural network learns to predict the temperature integrated over time. Integral control is more computationally intensive and requires sophisticated automatic differentiation software since it involves back-propagation through the differential equation solver. In this section we will use both training methods and compare them.

Before going on it should be noted that another descriptor for our two methods could be used: *differential control* is *offline* training and *integral control* is *online* training. The distinction being that online training is performed while the parameterization is being run. Sometimes we switch between the two.

5.1 Optimization of convective adjustment

Before training the NDE, it is prudent to optimize our base convective adjustment parameterization. This ensures that convective adjustment performs optimally, reducing the burden on the neural network. This is done by calibrating convective adjustment against the set of training simulations described in the previous section. In the convective adjustment scheme, equation (7), there is a single free parameter, K_{CA}. By scanning through plausible values of K_{CA} across our suite of simulations, we find the optimal value to be $K_{\mathrm{CA}} \approx 0.2\,\mathrm{m^2\,s^{-1}}$ (see appendix E for details): K_{CA} was subsequently kept at a constant value of $0.2\,\mathrm{m^2\,s^{-1}}$ in all calculations.

5.2 Method 1: differential control — training offline on the fluxes

Since the neural network maps $\overline{T}(z)$ profiles to $\overline{w'T'}|_{\text{missing}}(z)$ profiles (see figure 2), we can simply train the neural network to learn this relationship from appropriate training data. Training to replicate the fluxes can be thought of as a form of differential control as it is learning the instantaneous $\overline{T} \to \overline{w'T'}|_{\text{missing}}(z)$ relationship. Once the neural network is trained, it can be brought into the NDE which can be immediately used to make predictions.

We seek to minimize the difference between the $\overline{w'T'}|_{\text{missing}}(z)$ profile predicted by the neural network and the profile diagnosed from LES data. Thus we aim to minimize a loss function of the form

$$\mathcal{L}_1(\boldsymbol{\theta}) = \sum_{s=1}^{N_s} \mathcal{L}_{1,s}(\boldsymbol{\theta}) \quad \text{where} \quad \mathcal{L}_{1,s}(\boldsymbol{\theta}) = \frac{1}{L_z} \int_{-L_z}^{0} \left| \overline{w'T'}_s(z;\boldsymbol{\theta})|^{NN}_{\text{missing}} - \overline{w'T'}_s(z)|^{LES}_{\text{missing}} \right|^2 \mathrm{d}z. \tag{13}$$

Here $\mathcal{L}_1$ is the full loss function and $\mathcal{L}_{1,s}$ is the loss function over simulation s, $N_s = 9$ is the number of training simulations and $\boldsymbol{\theta}$ are the parameters being optimized to minimize the loss function, i.e. the neural network weights.

The neural network is trained over 5000 epochs. Each epoch is defined as one training iteration involving a full pass through the training data. The training is performed using the ADaptive Moment Estimation (ADAM) algorithm which is based on stochastic gradient descent and utilizes running averages of the gradients and its second moments (Kingma & Ba, 2014). A learning rate (or step size) of $\eta = 10^{-3}$ is used. The top left panel of figure 5 shows the value of the loss function (13) for the training and validation simulations as the neural network is trained (see appendix G for training times). We see that during training the loss decreases for most of the simulation sets. The loss for the N^2 interpolation and extrapolation sets sharply rises then decreases before slowly increasing, possibly indicating some amount of overfitting to the training data during earlier epochs. However, the bottom left panel of figure 5 shows that when embedded into the NDE and evaluated using a second loss function (introduced in the next subsection), training the neural network on fluxes does not actually lead to improved predictions for the temperature profile.

5.3 Method 2: integral control — training online on the full fluxes

While differential control can be expected to lead to a trained neural network capable of predicting missing fluxes from the temperature profile alone, small discrepancies in the predicted fluxes may accumulate over time resulting in temperature drift. To remedy this we also try to minimize a loss function that includes the term of principal interest, the temperature profile itself.

Training to reproduce the time series can be thought of as a form of integral control as it is learning to predict behavior over the time history of the simulation, and not just at singular instants in time, and backpropagating through the differential equation solver as the NDE is simulated.

The loss function in this case takes the form

$$\mathcal{L}_2(\boldsymbol{\theta}) = \sum_{s=1}^{N_s} \mathcal{L}_{2,s}(\boldsymbol{\theta}) \quad \text{where} \quad \mathcal{L}_{2,s}(\boldsymbol{\theta}) = \frac{1}{\tau L_z} \int_0^{\tau} \int_{-L_z}^{0} \left| \overline{T}_{NDE}(z,t;\boldsymbol{\theta}) - \overline{T}_{LES}(z,t) \right|^2 \mathrm{d}z \, \mathrm{d}t \tag{14}$$

where τ is the length of simulation time (or window) over which $\overline{T}(z,t)$ is included in the loss function in case we do not want to train on the full time series.

For training we again find that ADAM with a learning rate of $\eta = 10^{-3}$ leads to good results. One might want to decrease η as the loss decreases, taking smaller steps as the global minimum is approached. However, we found that incrementally or exponentially decreasing η as a function of the epoch number did not lead to faster training or lower loss values.

Neural network weights are often initialized using schemes designed to speed up training and avoid issues of vanishing gradients during training. The neural network architectures used here are initialized in Flux.jl using Glorot uniform initialization (Glorot & Bengio, 2010). This works well with differential control training but can compromise integral control training.

Because of our formulation, the neural network is capturing a residual from a base parameterization. Therefore the starting hypothesis is that the physics parameterization is correct, which corresponds to the neural network outputting zero for everything. This is done by setting the default waits sufficiently close to zero. The initial weights are

set using Glorot uniform initialization but then multiplied by 10^{-5}. This has the effect of making the predicted missing flux negligible at epoch zero but the training process leads to the NDE slowly improving upon the base parameterization as it is trained. An alternative training method, but one we found less simple, is to not train on the full time series from epoch zero but rather to train incrementally on longer and longer time series. That is by increasing τ in equation (14). For example, τ can be set to 3 hours for 25 epochs, then 6 hours for the next 25 epochs, then 12 hours and so on. This has the beneficial effect of slowly stabilizing the NDE as it is trained. During the integral control training process stabilized explicit time stepping methods of the ROCK family (Abdulle, 2002) are utilized to efficiently handle the stiffness introduced in the PDE discretization to improve the performance and stability of the numerical solution. We find that ROCK4 provides enough stability for all our training cases.

Figure 5 shows the value of both loss functions (13) and (14) for the different simulation sets. They show high-frequency noise due to the ADAM algorithm constantly attempting to ensure that the optimization does not remain in local minima. From the bottom right panel we see that augmenting convective adjustment with the trained neural network improves the loss by almost 2 orders of magnitude on the training set. The loss on the validation simulations where Q_b was varied improves monotonically, perhaps indicating that overfitting is not a concern. In validation simulations where N^2 was varied the loss fluctuates somewhat more, potentially indicating that overfitting is occuring. Since both loss functions increase in the N^2 extrapolation case, overfitting is indeed likely occurring here.

Comparing the two panels on the right, we see that both loss functions drop very quickly in the first 50 epochs suggesting that training on the time series leads to learning of both the time series and the fluxes. However, as the NDE is trained more its ability to predict fluxes worsens over time. One possible explanation for this surprising result is that the neural network is being trained on instantaneous fluxes (snapshots in time) which may be noisy. Much noise is removed through horizontal averaging, however some remains. A potential remedy is to train the neural network on time-averaged horizontally-averaged fluxes.

Inspection of the two panels on the left enables us to evaluate the fidelity of the NDE trained using differential control. We see that while it performs well in predicting

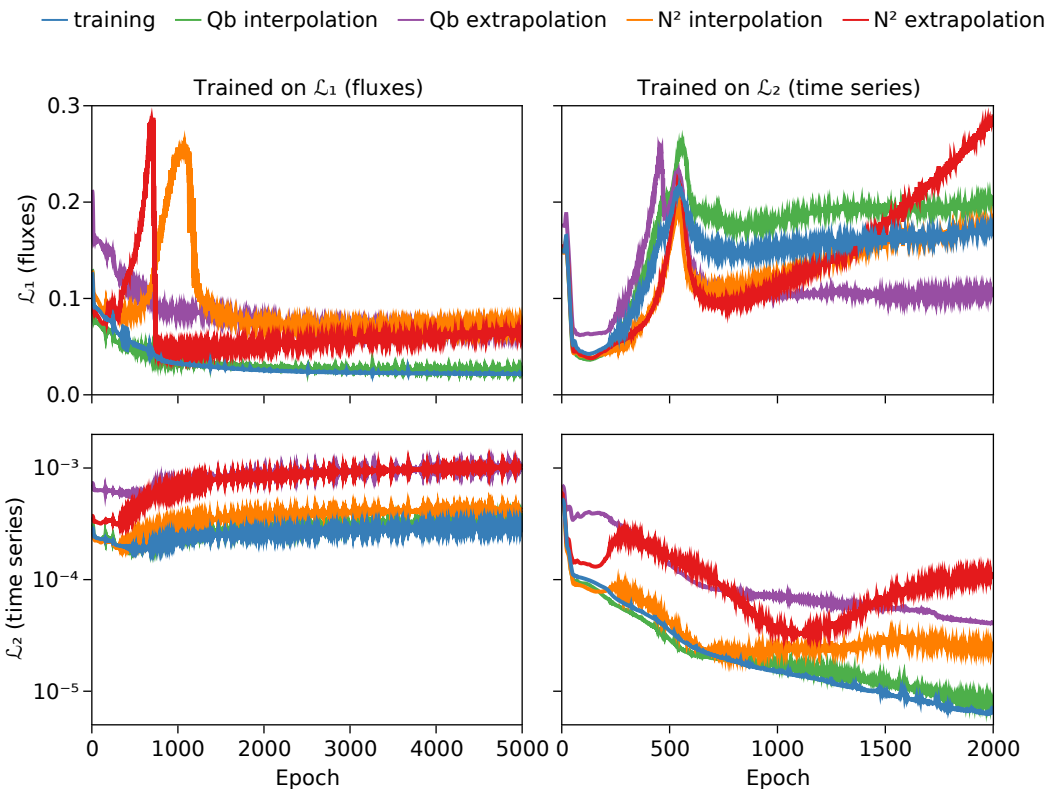


Figure 5. Loss history of our two training methods evaluated using $\mathcal{L}_1$ [equation (13)] and $\mathcal{L}_2$ [equation (14)]. The left column shows both loss functions when training on the fluxes given by $\mathcal{L}_1$. The right column shows both loss functions when training on the time series given by $\mathcal{L}_2$. The top row shows both methods assessed using loss function $\mathcal{L}_1$ while the bottom row shows both methods assessed using $\mathcal{L}_2$. Different colors are used to indicate the training set and each testing set of simulations. The solid lines show the mean loss across all simulations in the same set.

the fluxes (top left), the prediction of the temperature time series does not improve (bottom left). Indeed, in this case the NDE does worse at predicting the time series. This may again be associated with noise in the instantaneous fluxes and could perhaps be ameliorated somewhat by using time-averaged fluxes.

6 Assessing the fidelity of the trained Neural Differential Equation

6.1 Differential versus integral control

In section 5 we described two methods of training the NDE, first via differential control and second via integral control. We now address the question of whether train-

ing an NDE on the full time series (integral control) provides any benefits over training on instantaneous flux snapshots (differential control). For the comparisons in this section the NDE is solved as part of a 1D column model simulated using Oceananigans.jl. While the NDE was trained using the explicit ROCK4 time-stepper, it performs just as well embedded in Oceananigans.jl which utilizes a split-explicit time-stepper with a second-order Adams-Bashforth explicit time-stepper and implicit Backward Euler time-stepper for convective adjustment. This demonstrates the independence of the NDE on the time stepper used operationally or during training since a neural network is only used to learn a source term for a PDE.

Figure 6 shows how our loss metric compares between our two methods over time. Using differential control the loss increases as the simulation progresses in time. This may be because, even though the neural network is trained to reproduce the fluxes at snapshots in time, errors are compounded when time-stepping forward leading to divergence. With integral control training, instead, the neural network attempts to reproduce the time series in its entirety. As a result the error does not generally grow but remains bounded. The exception is the slight increase in error near the end of the simulation period at $t = 8\,\text{days}$. The fact that the NDE solution remains congruent with the LES solution, and that by $t = 8\,\text{days}$ the loss when trained on the full time history is smaller by more than an order-of-magnitude in both training and validation sets, suggests that the use of integral control is superior to the differential control approach. This demonstrates that while loss functions can be constructed to train embedded neural networks independently of the simulation process, the approach which requires differentiating the simulation greatly improves the stability of the learned result.

6.2 Comparison with other parameterizations

The main goal of developing data-driven parameterizations with NDEs is to improve upon existing parameterizations. Thus we now explore the skill of the trained NDE with existing parameterizations. In particular we compare the NDE with the base parameterization — convective adjustment — and a much more sophisticated scheme known as the K-Profile Parameterization (or KPP for short) described by Large et al. (1994). This is a popular vertical mixing model used by many ocean GCMs. Almost by construction, our NDE can only improve upon convective adjustment so we will focus on the comparison with KPP. The latter scheme performs well in the representation of free convec-

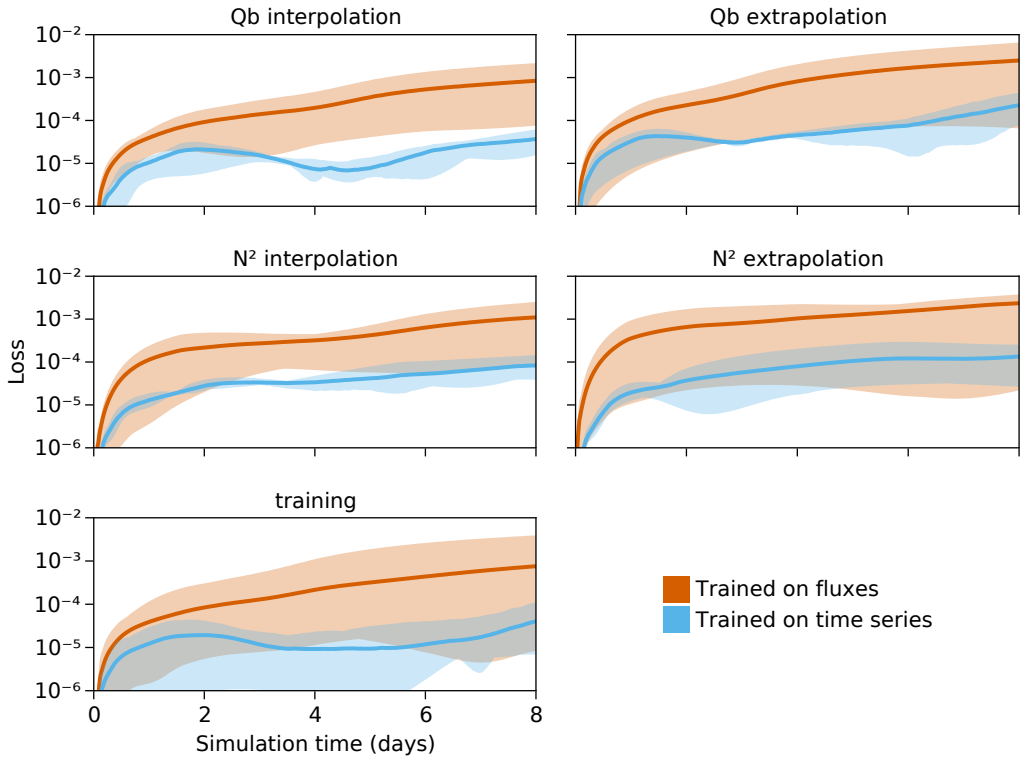


Figure 6. Time series of the loss function $\mathcal{L}_2$ [equation (14)] over the period of mixed layer evolution. In blue is the loss of the NDE trained on the time series (integral control) while in orange is the loss of the NDE trained on the instantaneous fluxes (differential control). The solid lines show the mean loss across all simulations in the same set while the shaded area shows the minimum and maximum loss across all simulations.

tion, except that it cannot easily be calibrated to work equally well for all background stratifications (Souza et al., 2020). For a challenging comparison, therefore, we have also tuned KPP's parameters to perform well against our training simulations (see appendix F for details).

Figure 7 shows how each parameterization performs on each of the five simulation sets. As expected, we see that in each set our NDE outperforms convective adjustment since we can only improve upon the base parameterization. As the simulation proceeds, the loss for each parameterization increases indicating that they diverge from the LES solution over time. However, our NDE generally outperforms both convective adjustment and KPP. Convective adjustment shows skill initially but deteriorates over time. For training and interpolation cases our NDE outperforms KPP while for extrapolation cases the

NDE and KPP perform similarly. The NDE is capable of some extrapolation when tested against buoyancy fluxes Q_b it has not seen before, but shows less skill when the stratification N^2 is outside the range of the training data. We will further investigate the ability to extrapolate using our NDE in section 7 when we modify the architecture of the neural network.

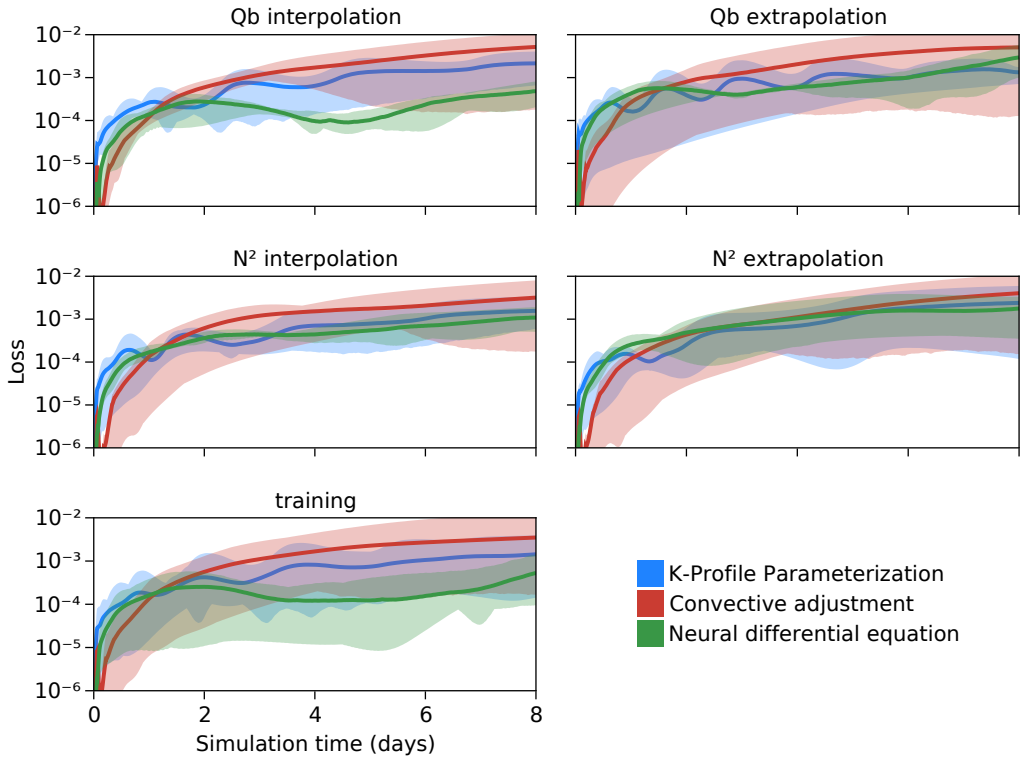


Figure 7. The time series loss $\mathcal{L}_2$ [equation (14)] of three different parameterizations as a function of simulation time, for different sets of training and testing simulations. The solid lines show the mean loss across all simulations in the same set, while the shaded area shows the minimum and maximum loss across all simulations.

Figure 8 updates figure 3 to include the NDE solution under integral control. The NDE solution matches the LES solution much more closely than the other parameterizations for both temperature $\overline{T}(z)$ and heat flux $\overline{w'T'}(z)$ profiles. Of particular interest is how closely the NDE heat flux profiles matches up with the profile from the LES simulation. The neural network was not trained on the heat flux but rather on the temperature time series. Nevertheless, it predict a physically meaningful and accurate heat flux.

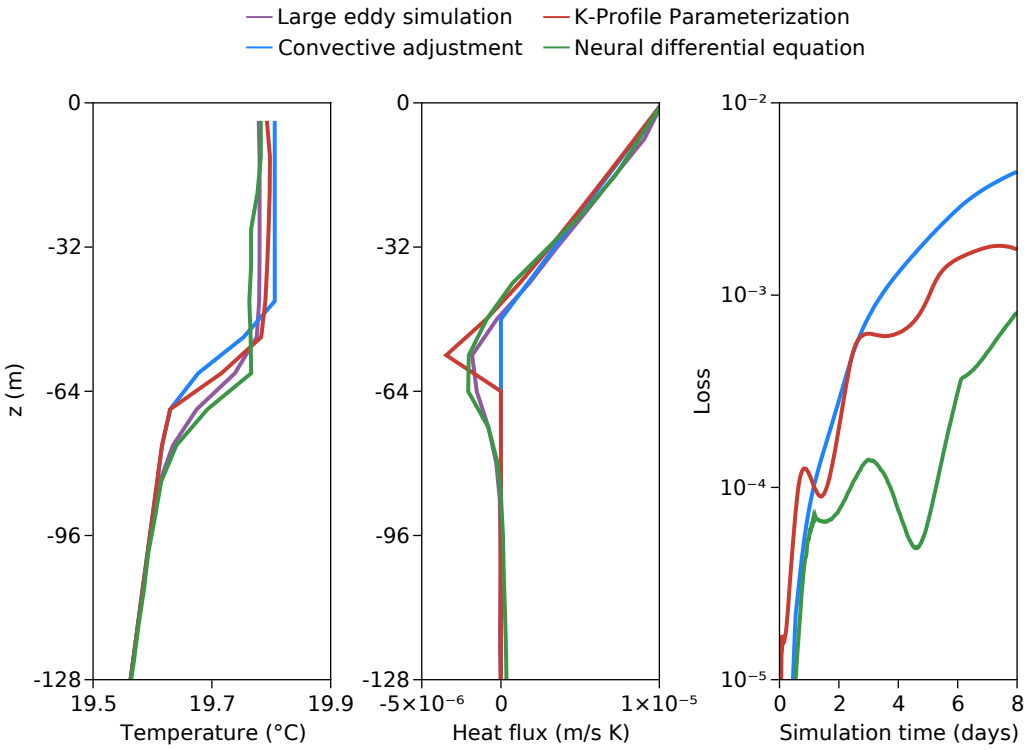


Figure 8. Comparison of three different parameterizations against the LES (truth) solution from simulation 11 (see table 1 in appendix C), one of our Q_b interpolation simulations. (Left) The heat flux profile $\overline{w'T'}(z)$ predicted by the LES and by each parameterization. (Middle) The predicted temperature profiles $\overline{T}(z)$. (Right) The loss between the LES (true) solution and the predicted temperature profiles for each parameterization. Note that only half the vertical domain is shown to emphasize the structure of the mixed layer.

7 Hyperparameter optimization

Thus far we have only trained the NDE using a dense, fully-connected neural network. We will now change the architecture of the neural network in an attempt at hyperparameter optimization, i.e., optimizing the architecture of the neural network itself to improve the skill of the NDE.

There are two main reasons to optimize the architecture of the neural network: improve the fidelity of the NDE itself, and to carry out hypothesis testing using different architectures. Here we experiment with four additional architectures. For a *wider* network we increase the size of the hidden layers from $4N_z$ to $8N_z$; for a *deeper* network we

add an extra hidden layer of size $4N_z$. We also use two neural networks with convolutional first layers and filter sizes of 2 and 4. The use of the dense and "convolutional" network architectures can be thought of making the assumption that the parameterization is fully nonlocal in the dense case and local in the convolutional case with the filter size determining the degree of locality. Appendix G details the architectures used including the number of free parameters (or weights) in each one.

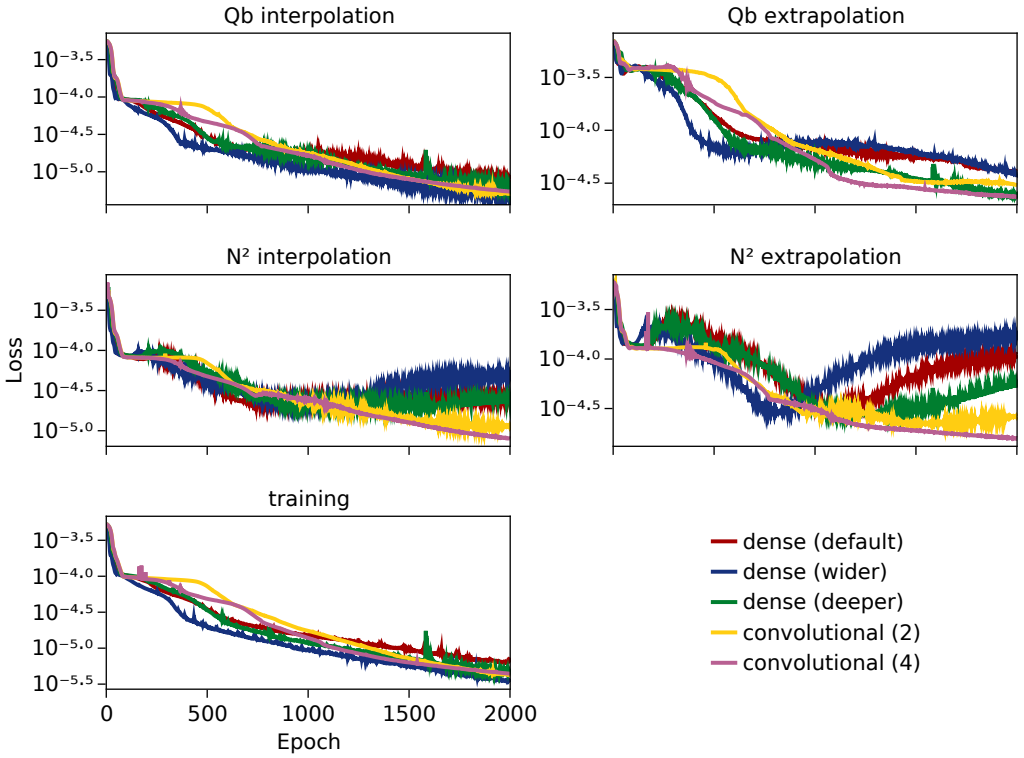


Figure 9. Time series of loss $\mathcal{L}_2$ [equation (14)] for various neural network architectures for each training and testing simulation, as a function of the training epoch. Each colored curve corresponds to one of the architectures used and each panel shows how the different architectures performed on each set of simulations.

Figure 9 shows how the different architectures performed on the different sets of simulations. The architectures all perform similarly on the training set. Considering the interpolation and extrapolation capabilities we see that the convolutional (4) architecture consistently outperform the others. In the N^2 interpolation and extrapolation cases we see that by epoch 2000 all the architectures are likely overfitting the training data except for convolutional (4). This is perhaps not surprising since free convection does

not exhibit non-locality (Souza et al., 2020) suggesting that only local information is required and that convolutional networks should outperform fully-connected networks. The convolutional (4) architecture also exhibits less high frequency noise in the loss function, perhaps indicating that its gradients are less noisy and/or better behaved leading to a less challanging training process. The convolutional (4) architecture also has the fewest free parameters, suggesting that fully-connected networks perhaps have too many, leading to overfitting of the training date. Perhaps designing the architecture around physical assumptions is more important than designing larger or deeper networks.

An encouraging result is that our NDE approach seems robust to the architecture of the neural network employed in that all five architectures were trainable. This suggesting that the NDE approach could perhaps be used to perform hypothesis testing on the ideal structure of the parameterization. More investigation is necessary to determine whether a single architecture can perform best in all cases. A more systematic and powerful approach to hyperparameter optimization might prove useful here (Snoek et al., 2012) and specialized inference algorithms for large datasets and networks exist (Klein et al., 2017).

8 Discussion and conclusions

Here we have illustrated how NDEs might provide an attractive medium for developing data-driven parameterizations for climate models. NDEs can be constructed to augment an existing parameterization scheme whilst remaining faithful to conservation laws. They can then be, for example, trained on high-resolution LES data within a differential equation time-stepper. The resulting solutions are stable and faithfully represent turbulent processes which are challenging to capture using conventional schemes. For example, we have shown that a simple parameterization of convection — that of convective adjustment — can be improved upon by training the NDE to capture the entrainment fluxes at the base of the well-mixed layer, fluxes that convective adjustment cannot represent. The augmented parameterization outperforms existing commonly used parameterizations such as KPP. Training even stiff NDEs (e.g. when convective adjustment is employed) is made possible by the Julia scientific machine learning (SciML) software stack.

We can usefully describe the method we are advocating as a 'residual' approach, in which the NN is used to improve upon an existing parameterization through the representation of (residual) fluxes which are not captured by the base parameterization. All of this is elegantly enabled through the use of NDEs. We see no reason why a similar residual approach cannot be used to improve upon many other existing parameterizations exploiting the attractive form of NDEs.

We have chosen to focus on convectively-driven turbulence of the ocean boundary layer since it provides an idealised, proof-of-context setting for our exploration of NDEs. A number of implementation choices were made for simplicity and could be improved. The parameterization was trained and tested using a single vertical resolution. Existing parameterizations such as KPP can be sensitive to model resolution (Souza et al., 2020) and development of a resolution-independent parameterization would be desirable. One approach might be to decompose the temperature profile into a Fourier or Chebyshev series. The NN could then be trained on polynomial coefficients (Li et al., 2020). Profiles sampled at different resolutions might then lead to an NDE that effectively becomes resolution-independent. Going even further, an autoencoder neural network might be used to learn the best representation of the data which could, in principle, be very different from a polynomial expansion, although perhaps less interpretable.

The learning methods and tooling employed here are rather standard in the ML community and we have made no attempt to optimize the NDE's performance for either prediction time or training time. Neural network pruning can be used to eliminate unnecessary free parameters in the architecture leading to faster predictions with minimal impact on accuracy. A more thorough investigation into hyper-parameter optimization could reveal smaller and faster architectures that perform just as well or even better than, for example, those explored in section 7. Moreover, reducing the amount of training data could lead to similar accuracy yet with shorter training times. In the present study the networks employed were too small to benefit from the speedup provided by GPUs. That said, larger networks might benefit from GPU resources and offer large speedups compared to CPUs. Efficient GPU-enabled NDEs might then be readily embedded into fast GPU-enabled ocean models such as Oceananigans (Ramadhan et al., 2020) and Veros (Häfner et al., 2018).

Data-driven parameterizations, residual or not, can also be used to perform hypothesis testing for closures. For example, to test whether a local or nonlocal closure is needed to model some complex physics, two neural networks may be trained: one using dense, fully-connected layers to furnish a fully nonlocal closure and another using convolutional layers for a local closure. Comparing the accuracy of the two could indicate whether a nonlocal closure is required, or provide insight into which physical scenarios exhibit nonlocal physics. There are a vast number of possibilities that could be explored: recurrent neural networks can be used to test whether including time history improves the closure, long-term short-memory (LSTM) architectures and transformers (Vaswani et al., 2017) to test whether including lagged information improves the closure, and so on. Custom architectures with new physical properties can also readily be constructed, for example, to create a data-driven parameterization that is only nonlocal. Physical insights from such hypothesis testing with data-driven parameterizations may be valuable in and of themselves, and can also guide the development of improved theory-driven parameterizations.

Future work should explore whether the NDE framework can be applied to more complex turbulent processes in both the ocean and atmosphere. Having tackled free convection, it would be natural to consider the more realistic case of a water column forced by both buoyancy and momentum surface fluxes. The parameterization of mesoscale eddy transport in the ocean is also a crucial and inadequately-represented turbulent process for which NDEs might improve upon existing parameterizations. We considered physical scenarios with constant surface fluxes but future work should also investigate whether NDEs are capable of performing well in the presence of spatially- and time-varying surface fluxes. Since surface fluxes are prescribed here as boundary conditions, rather than being fed into the neural network, the NDE approach could generalize rather well to less idealised forcing scenarios.

A Numerical methods and Oceananigans.jl

Oceananigans.jl (Ramadhan et al., 2020) is open source software for ocean studies written in the Julia programming language (Bezanson et al., 2017). It runs on both CPUs or GPUs using Julia's native GPU compiler (Besard et al., 2019).

For the large eddy simulations performed in this paper, Oceananigans.jl is configured to solve the spatially filtered, incompressible Boussinesq equations with a temperature tracer. Letting $\boldsymbol{u} = (u, v, w)$ be the three-dimensional, spatially filtered velocity field, T be the temperature, p be the kinematic pressure, f be the Coriolis parameter, and $\boldsymbol{\tau}$ and $\boldsymbol{q}$ be the stress tensor and temperature flux due to sub-filter turbulent diffusion, the equations of motion are

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \boldsymbol{f} \times \boldsymbol{u} + \nabla p = \boldsymbol{b} - \nabla \cdot \boldsymbol{\tau} \tag{15}$$

$$\partial_t T + \boldsymbol{u} \cdot \nabla T = -\nabla \cdot \boldsymbol{q} \tag{16}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{17}$$

The buoyancy $\boldsymbol{b} = b\hat{\boldsymbol{z}}$ appearing in equation (15) is related to temperature by a linear equation of state,

$$b = \alpha g (T_0 - T) \tag{18}$$

where $T_0 = 20\,^\circ\mathrm{C}$ is a reference temperature, $\alpha = 2 \times 10^{-4}\,\mathrm{K}^{-1}$ is the thermal expansion coefficient, and $g = 9.81\,\mathrm{m\,s}^{-2}$ is gravitational acceleration at the Earth's surface.

The sub-filter stress and momentum fluxes are modeled with downgradient closures, such that

$$\tau_{ij} = -2\nu_e \Sigma_{ij} \quad \text{and} \quad \boldsymbol{q} = -\kappa_e \nabla T \tag{19}$$

where $\Sigma_{ij} = (\partial_i u_j + \partial_j u_i)/2$ is the strain rate tensor and ν_e and κ_e are the eddy viscosity and eddy diffusivity of temperature. The eddy viscosity ν_e and eddy diffusivity κ_e in equation (19) are modeled with the anisotropic minimum dissipation (AMD) formalism introduced by Rozema et al. (2015) and Abkar et al. (2016), refined by Verstappen (2018), and validated and described in detail for ocean-relevant scenarios by Vreugdenhil

and Taylor (2018). AMD is accurate on anisotropic grids (Vreugdenhil & Taylor, 2018) and is relatively insensitive to resolution (Abkar et al., 2016).

To solve equations (15)–(17) Oceananigans.jl uses a staggered C-grid finite volume spatial discretization (Arakawa & Lamb, 1977) with an upwind-biased 5^{th}-order weighted essentially non-oscillatory (WENO) advection scheme for momentum and tracers (Shu, 2009). Diffusion terms are computed using centered 2^{nd}-order differences. A pressure projection method is used to ensure the incompressibility of $\boldsymbol{u}$ at every time step (Brown et al., 2001). A fast Fourier-transform-based eigenfunction expansion of the discrete second-order Poisson operator is used to solve the discrete pressure Poisson equation for the pressure on a regular grid (Schumann & Sweet, 1988). An explicit 3^{rd}-order Runge-Kutta method is used to advance the solution in time (Le & Moin, 1991).

B Simulation setup and generation of LES training data

To generate training data we run a suite of high-resolution LES simulations using Oceananigans.jl. The simulations capture various combinations of surface buoyancy flux and stratification profiles keeping everything else constant. Parameter values are chosen so that the mixed layer depth does not extend beyond approximately half of the domain depth by the end of the simulation, thus minimizing significant finite size effects.

In all simulations, the model domain is $512\,\text{m} \times 512\,\text{m} \times 256\,\text{m}$ with 256 grid points in the horizontal and 128 grid points in the vertical, giving a 2 m isotropic grid spacing. A Coriolis parameter of $f = 1 \times 10^{-4}\,\text{s}^{-1}$ is used. The fluid is initially at rest ($\boldsymbol{u} = \boldsymbol{0}$) and the initial condition on T is horizontally homogeneous and has three layers. At the top is a *surface* layer of thickness Δz_s with constant stratification N_s^2. At the bottom is a *deep* layer with constant stratification N_d^2. In the middle is a *transition* layer or *thermocline* layer of thickness Δz_t with nonlinear stratification determined by fitting a cubic polynomial for T between the surface and deep layers preserving continuity and differentiability. The deep layer then has a thickness of $L_z - \Delta z_s - \Delta z_t$ where $L_z = 256\,\text{m}$ is the domain depth. Random noise of the form $\epsilon e^{z/\delta}$ where $\epsilon \sim \text{Uniform}(0, 1)$ and $\delta = 8\,\text{m}$ is a noise decay length scale is added to the initial T to stimulate numerical convection near the surface. An example of this initial condition can be seen in figure 3.

The surface buoyancy flux Q_b is implemented as a surface temperature flux boundary condition $Q_\theta = Q_b/(\alpha g)$. A gradient boundary condition on T is imposed at the

bottom of the domain to maintain the bottom stratification. A sponge layer of the form $\partial_t \boldsymbol{u} = -\tau^{-1} e^{-(z+L_z)/\ell} \boldsymbol{u}$, where $\tau = 15\,\text{minutes}$ and $\ell = L_z/10$, is used to relax the velocities to zero near the bottom, in part to absorb internal waves that could otherwise bounce around the domain. The temperature T is similarly relaxed to the initial condition with the form $\partial_t T = -\tau^{-1} e^{-(z+L_z)/\ell} [T - T_0(z)]$ near the bottom to maintain the bottom stratification where τ and ℓ have the same value as before.

Horizontally-averaged output is written to disk every 10 minutes of simulation time. Each simulation is run for 8 days.

C Simulation parameters

In table 1 we tabulate the parameters used to generate training and validation simulation data. Simulations 1–9 were used for training while simulations 10–21 were used for validation. Simulations 1–3 were designed to have a smaller thermocline, while 4–6 have a medium thermocline, and 7–9 have a large thermocline. Simulation 10–12 are used to test for Q_b interpolation, 13–15 for Q_b extraoplation, 16–18 for N^2 interpolation, and 19–21 for N^2 extrapolation. Figure 4 shows where in $Q_b - N^2$ parameter space these simulations lie.

D Derivation of the non-dimensional PDE

The numerical values of $\overline{T} \sim \mathcal{O}(1 \times 10^1\,^\circ\text{C})$ and $\overline{w'T'} \sim \mathcal{O}(1 \times 10^{-5}\,\text{m}\,\text{s}^{-1}\,\text{K}^{-1})$ vary across six orders of magnitude. When training the neural network or performing a gradient descent search, having huge disparities in values may make it difficult to find optimal step sizes and thus difficult to train. Thus we perform a feature scaling on the values of $\overline{T}$ and $\overline{w'T'}$ to normalize the data before processing and training. We use a zero-mean unit-variance scaling

$$\widehat{\overline{T}} = \frac{\overline{T} - \mu_{\overline{T}}}{\sigma_{\overline{T}}} \quad \text{and} \quad \widehat{\overline{w'T'}} = \frac{\overline{w'T'} - \mu_{\overline{w'T'}}}{\sigma_{\overline{w'T'}}} \tag{20}$$

where $\widehat{\cdot}$ indicates a normalized quantity and μ_α and σ_α are the mean and standard deviation of α evaluated over the entire training datasets. Using this scaling both the inputs and outputs of the neural network are dimensionless and $\mathcal{O}(1)$.

ID	Δz_s (m)	Δz_t (m)	Q_b ($\mathrm{m^2\,s^{-3}}$)	Q_b ($\mathrm{W\,m^{-2}}$)	N_s^2 ($\mathrm{s^{-2}}$)	N_d^2 ($\mathrm{s^{-2}}$)
1	48	24	1×10^{-8}	20.9	2×10^{-6}	2×10^{-6}
2	48	24	3×10^{-8}	62.8	2×10^{-6}	2×10^{-6}
3	48	24	5×10^{-8}	104	2×10^{-6}	2×10^{-6}
4	24	48	1×10^{-8}	20.9	1×10^{-6}	2×10^{-6}
5	24	48	3×10^{-8}	62.8	1×10^{-6}	2×10^{-6}
6	24	48	5×10^{-8}	104	1×10^{-6}	2×10^{-6}
7	24	64	1×10^{-8}	20.9	1×10^{-6}	5×10^{-6}
8	24	64	3×10^{-8}	62.8	1×10^{-6}	5×10^{-6}
9	24	64	5×10^{-8}	104	1×10^{-6}	5×10^{-6}
10	48	24	4×10^{-8}	83.8	2×10^{-6}	2×10^{-6}
11	24	48	2×10^{-8}	41.9	1×10^{-6}	2×10^{-6}
12	24	64	4×10^{-8}	83.8	1×10^{-6}	5×10^{-6}
13	48	24	6×10^{-8}	126	2×10^{-6}	2×10^{-6}
14	24	48	0.5×10^{-8}	10.5	1×10^{-6}	2×10^{-6}
15	24	64	6×10^{-8}	126	1×10^{-6}	5×10^{-6}
16	36	36	1×10^{-8}	20.9	2×10^{-6}	2×10^{-6}
17	36	36	5×10^{-8}	104	2×10^{-6}	2×10^{-6}
18	24	56	3×10^{-8}	62.8	1×10^{-6}	2×10^{-6}
19	36	36	1×10^{-8}	20.9	2×10^{-6}	2×10^{-6}
20	36	36	5×10^{-8}	104	2×10^{-6}	2×10^{-6}
21	24	56	3×10^{-8}	62.8	1×10^{-6}	2×10^{-6}

Table 1. Simulation parameters used to generate training and validation data. Parameters changed are described in appendix B.

As the neural network now deals in dimensionless quantities, the NDE being solved, equation (12), must also be non-dimensionalized. This is also important to ensure numerical stability reasons. We non-dimensionalize time t and the vertical coordinate z as

$$\widehat{t} = \frac{t}{\tau} \quad \text{and} \quad \widehat{z} = \frac{z}{L_z} \tag{21}$$

where $\tau = 8\,\text{days}$ is the duration of the simulation and $L_z = 256\,\text{m}$ is the depth of the domain. The time derivative of $\overline{T}$ becomes

$$\frac{\partial \overline{T}}{\partial t} = \frac{\sigma_T}{\tau} \frac{\partial \widehat{\overline{T}}}{\partial \widehat{t}} \tag{22}$$

and the vertical derivative of $\overline{w'T'}$ becomes

$$\frac{\partial \overline{w'T'}}{\partial z} = \frac{\sigma_{\overline{w'T'}}}{L_z} \frac{\partial \widehat{\overline{w'T'}}}{\partial \widehat{z}} \tag{23}$$

Using equations (20)–(23) we can rewrite the NDE, equation (12), non-dimensionally as

$$\frac{\partial \widehat{\overline{T}}}{\partial \widehat{t}} = -\frac{\sigma_{\overline{w'T'}}}{\sigma_{\overline{T}}} \left(\frac{\partial}{\partial \widehat{z}} \widehat{\overline{w'T'}} - \widehat{\kappa} \frac{\partial \widehat{\overline{T}}}{\partial \widehat{z}} \right) \tag{24}$$

which is solved numerically during the training process as described in §2.

E Calibration of convective adjustment scheme

Before training the NDE, the base parameterization must be calibrated. In the case of convective adjustment there is only one free parameter, the convective adjustment diffusivity coefficient K_{CA} in equation (7). Since one can readily run simulations of convective adjustment to compare with LES data we evaluate the loss function (14) over the set of training simulations for values of $10^{-3} \leq K_{\text{CA}} \leq 10^{1}$ to determine the optimal value of K_{CA} (see figure 10).

We see that increasing K_{CA} only decreases the loss until it plateaus when $K_{\text{CA}} > 1$. It would seem natural, then, that we pick a large enough value to readily induce vertical mixing. However, as the NDE implemented in DifferentialEquations.jl uses the ROCK4 explicit time-stepper, increasing K_{CA} also increases the stiffness of the system of differential equations necessitating a smaller time step and therefore longtimer runtimes. We therefore choose a value of K_{CA} that minimizes the product of the loss function and the runtime. This turns out to correspond to a non-dimensional diffusivity of $\widehat{K}_{\text{CA}} \approx 2$ or a dimensional value of $K_{\text{CA}} = (H^2/\tau)\widehat{K}_{\text{CA}} \approx 0.2\,\text{m}^2\,\text{s}^{-1}$. This is a very reasonable value and is in the range discussed by Klinger et al. (1996).

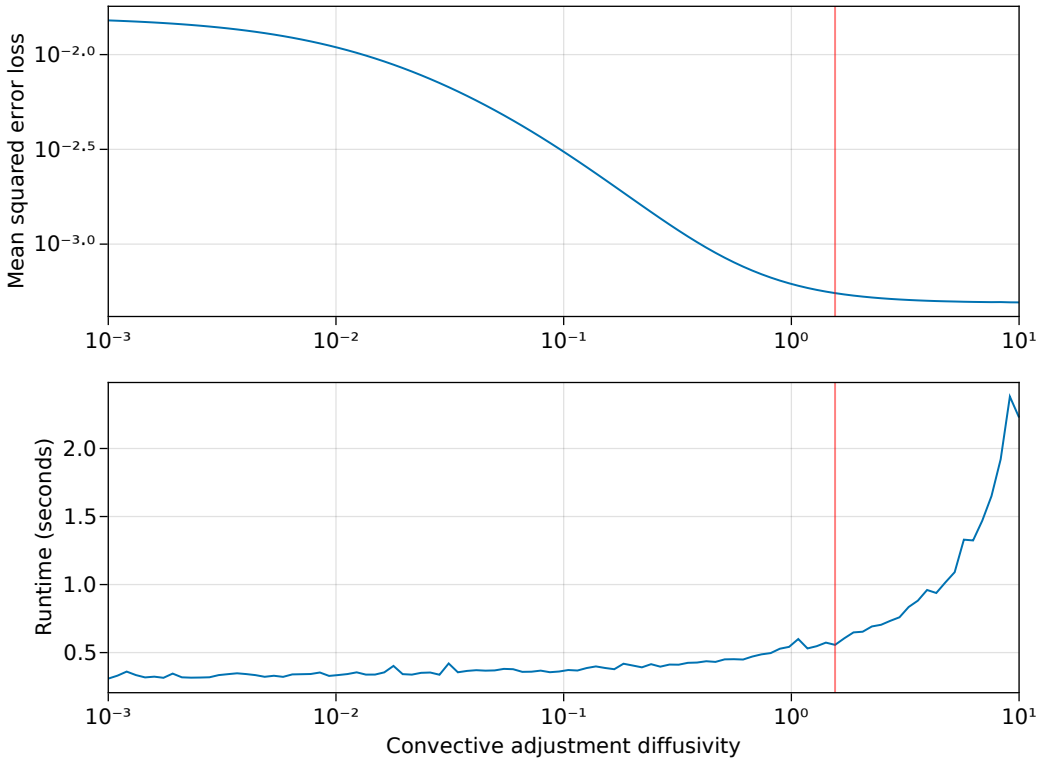


Figure 10. (Top) The value of the loss function (14) evaluated over the set of training simulations as a function of the convective adjustment diffusivity K_{CA}. (Bottom) The median wall clock time of the evaluations at each value of K_{CA}. The red line indicates the value of K_{CA} that minimizes the product of the loss and runtime.

F Calibration of the K-Profile Parameterization

For the comparison between the NDE and KPP parameterizations described in section 6.2, both should be trained on the same set of simulations. This is especially true because KPP may not be optimized for free convection, whilst the NDE is.

We use the KPP implementation provided by OceanTurb.jl and described by Souza et al. (2020). It is a mathematically identical algebraic reorganization of the original formulation proposed by Large et al. (1994) to reduce the number of free parameters from six to just four $\boldsymbol{\mathcal{C}} = (\mathcal{C}^S, \mathcal{C}^H, \mathcal{C}^D, \mathcal{C}^N)$ in the case of free convection. $\mathcal{C}^S$ is a surface layer fraction, $\mathcal{C}^H$ is a mixing depth parameter, $\mathcal{C}^D$ is a flux scaling parameter for downgradient fluxes, and $\mathcal{C}^N$ is a flux scaling for non-local fluxes. The reference parameters as

given by Large et al. (2019), rephrased by the four parameters of Souza et al. (2020), are $\mathcal{C}_0 = (0.1, 0.96, 1.36, 6.33)$.

Using reference values as an initial guess, we apply an adaptive differential evolution optimization algorithm (Y. Wang et al., 2014) with radius-limited sampling as implemented in BlackBoxOptim.jl (Feldt, 2018) (`adaptive_de_rand_1_bin_radiuslimited` more specifically). This algorithm is suitable for finding global minima in the absence of gradient information from automatic differentiation. In this way we train KPP on our 9 simulations by optimizing our four parameters to minimize the loss function (14).

The reference parameters lead to a loss of $\mathcal{L}_2(\mathcal{C}_0) = 1.26 \times 10^{-4}$. The optimization algorithm is given the following box constraints following Souza et al. (2020): $0 \leq \mathcal{C}^S \leq 1$, $0 \leq \mathcal{C}^H \leq 8$, $0 \leq \mathcal{C}^D \leq 8$, $0 \leq \mathcal{C}^N \leq 8$. After roughly $10,000$ iterations, the differential evolution algorithm converges on the parameters $\mathcal{C}^\star = (\frac{2}{3}, 8, 0.16, 5)$ with loss $\mathcal{L}_2(\mathcal{C}^\star) = 5.33 \times 10^{-5}$ which corresponds to an improvement of roughly $2.4\times$. $\mathcal{C}^\star$ is used when comparing KPP against other parameterizations in section 6.2.

G Neural network architectures

In section 7 we trained the NDE against the same training simulations but using five different architectures, as detailed in table 2. They are composed of a number of sequential layers, either fully-connected dense layers denoted by $D_{n\to m}$ with n inputs and m outputs or one-dimensional convolutional layers denoted C_n with filter size n. The total number of free parameters for each architecture is included in the table, together with the training time per epoch for both training methods. The size of the dense layers were typically in multiples of $N_z = 32$, the number of vertical grid points.

Open Research

Julia code to produce the simulation data, train the NDE, and plot all the figures can be found in a GitHub repository (`https://github.com/ali-ramadhan/NeuralFreeConvection.jl`) archived at Zenodo (Ramadhan et al., 2022).

Acknowledgements

We thank Keaton Burns for insightful discussions during the development of this study. Our work is supported by the generosity of Eric and Wendy Schmidt by recom-

Name	architecture	N_p	$t_{\text{train}}^{\text{fluxes}}$ (s)	$t_{\text{train}}^{\text{timeseries}}$ (s)
dense (default)	$D_{N_z \to 4N_z}, D_{4N_z \to 4N_z}, D_{4N_z \to N_z - 1}$	24,735	4.89	20.76
dense (wider)	$D_{N_z \to 8N_z}, D_{8N_z \to 8N_z}, D_{8N_z \to N_z - 1}$	82,207	6.23	35.05
dense (deeper)	$D_{N_z \to 4N_z}, D_{4N_z \to 4N_z}, D_{4N_z \to 4N_z}, D_{4N_z \to N_z - 1}$	41,247	5.77	25.48
convolutional (2)	$C_2, D_{N_z - 2 + 1 \to 4N_z}, D_{4N_z \to 4N_z}, D_{4N_z \to N_z - 1}$	24,610	8.84	37.97
convolutional (4)	$C_4, D_{N_z - 4 + 1 \to 4N_z}, D_{4N_z \to 4N_z}, D_{4N_z \to N_z - 1}$	24,356	8.89	38.83

Table 2. Neural network architectures used in section 7. N_p is the total number of free parameters for the architecture. $t_{\text{train}}^{\text{fluxes}}$ and $t_{\text{train}}^{\text{timeseries}}$ are the training times per epoch (in seconds) using loss functions $\mathcal{L}_1$ [equation (13)] and $\mathcal{L}_2$ [equation (14)] respectively.

mendation of the Schmidt Futures program, by the National Science Foundation under grant AGS-6939393 and by the MIT-GISS collaborative agreement of NASA.

References

Abdulle, A. (2002). Fourth order Chebyshev methods with recurrence relation. *SIAM Journal on Scientific Computing*, *23*(6), 2041–2054. doi: 10.1137/S1064827500379549

Abkar, M., Bae, H. J., & Moin, P. (2016). Minimum-dissipation scalar transport model for large-eddy simulation of turbulent flows. *Phys. Rev. Fluids*, *1*, 041701. doi: 10.1103/PhysRevFluids.1.041701

Arakawa, A., & Lamb, V. R. (1977). Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model. In *Methods in Computational Physics: Advances in Research and Applications* (Vol. 17, pp. 173–265). Elsevier. doi: 10.1016/B978-0-12-460817-7.50009-4

Barnard, E., & Wessels, L. (1992). Extrapolation and interpolation in neural network classifiers. *IEEE Control Systems Magazine*, *12*(5), 50–53. doi: 10.1109/37.158898

Besard, T., Foket, C., & De Sutter, B. (2019). Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *30*(4), 827–841. doi: 10.1109/TPDS.2018.2872064

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. doi: 10.1137/141000671

Bolton, T., & Zanna, L. (2019). Applications of Deep Learning to Ocean Data Inference and Subgrid Parameterization. *Journal of Advances in Modeling Earth Systems*, *11*(1), 376–399. doi: 10.1029/2018MS001472

Brown, D. L., Cortez, R., & Minion, M. L. (2001). Accurate Projection Methods for the Incompressible Navier–Stokes Equations. *Journal of Computational Physics*, *168*(2), 464-499. doi: 10.1006/jcph.2001.6715

DuVivier, A. K., Large, W. G., & Small, R. J. (2018). Argo Observations of the Deep Mixing Band in the Southern Ocean: A Salinity Modeling Challenge. *Journal of Geophysical Research: Oceans*, *123*(10), 7599–7617. doi: 10.1029/2018JC014275

Fan, F.-L., Xiong, J., Li, M., & Wang, G. (2021). On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, *5*(6), 741-760. doi: 10.1109/TRPMS.2021.3066428

Feldt, R. (2018). *Blackboxoptim.jl.* `https://github.com/robertfeldt/BlackBoxOptim.jl`. GitHub.

Fox-Kemper, B., Adcroft, A., Böning, C. W., Chassignet, E. P., Curchitser, E., Danabasoglu, G., ... Yeager, S. G. (2019). Challenges and Prospects in Ocean Circulation Models. *Frontiers in Marine Science*, *6*. doi: 10.3389/fmars.2019.00065

Gentine, P., Pritchard, M., Rasp, S., Reinaudi, G., & Yacalis, G. (2018). Could Machine Learning Break the Convection Parameterization Deadlock? *Geophysical Research Letters*, *45*(11), 5742–5751. doi: 10.1029/2018GL078202

Glorot, X., & Bengio, Y. (2010, 13–15 May). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from `https://proceedings.mlr.press/v9/glorot10a.html`

Griffies, S. M., Winton, M., Anderson, W. G., Benson, R., Delworth, T. L., Dufour, C. O., ... Zhang, R. (2015). Impacts on Ocean Heat from Transient Mesoscale Eddies in a Hierarchy of Climate Models. *Journal of Climate*, *28*(3), 952–977. doi: 10.1175/JCLI-D-14-00353.1

Haine, T. W. N., & Marshall, J. (1998). Gravitational, Symmetric, and Baroclinic Instability of the Ocean Mixed Layer. *Journal of Physical Oceanography*, *28*(4), 634–658. doi: 10.1175/1520-0485(1998)028⟨0634:GSABIO⟩2.0.CO;2

Hausfather, Z., Drake, H. F., Abbott, T., & Schmidt, G. A. (2020). Evaluating the Performance of Past Climate Model Projections. *Geophysical Research Letters*, *47*(1), e2019GL085378. doi: 10.1029/2019GL085378

Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., ... Williamson, D. (2017). The Art and Science of Climate Model Tuning. *Bulletin of the American Meteorological Society*, *98*(3), 589–602. doi: 10.1175/BAMS-D-15-00135.1

Häfner, D., Jacobsen, R. L., Eden, C., Kristensen, M. R. B., Jochum, M., Nuterman, R., & Vinter, B. (2018). Veros v0.1 – a fast and versatile ocean simulator in pure Python. *Geoscientific Model Development*, *11*(8), 3299–3312. doi: 10.5194/gmd-11-3299-2018

Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*, *3*(25), 602. doi: 10.21105/joss.00602

Katz, R. W., Craigmile, P. F., Guttorp, P., Haran, M., Sansó, B., & Stein, M. L. (2013). Uncertainty analysis in climate change assessments. *Nature Climate Change*, *3*(9), 769–771. doi: 10.1038/nclimate1980

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv. doi: 10.48550/ARXIV.1412.6980

Klein, A., Falkner, S., Bartels, S., Hennig, P., & Hutter, F. (2017). Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In A. Singh & J. Zhu (Eds.), *Proceedings of the 20th international conference on artificial intelligence and statistics* (Vol. 54, pp. 528–536). PMLR. Retrieved from `https://proceedings.mlr.press/v54/klein17a.html`

Klinger, B. A., Marshall, J., & Send, U. (1996). Representation of convective plumes by vertical adjustment. *Journal of Geophysical Research: Oceans*, *101*(C8), 18175–18182. doi: 10.1029/96JC00861

Large, W. G., McWilliams, J. C., & Doney, S. C. (1994). Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Reviews of Geophysics*, *32*(4), 363–403. doi: 10.1029/94RG01872

Large, W. G., Patton, E. G., & Sullivan, P. P. (2019). Nonlocal Transport and Implied Viscosity and Diffusivity throughout the Boundary Layer in LES of the Southern Ocean with Surface Waves. *Journal of Physical Oceanography*, *49*(10), 2631–2652. doi: 10.1175/JPO-D-18-0202.1

Le, H., & Moin, P. (1991). An improvement of fractional step methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, *92*(2), 369-379. doi: 10.1016/0021-9991(91)90215-7

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). *Fourier neural operator for parametric partial differential equations.* arXiv. doi: 10.48550/ARXIV.2010.08895

Marshall, J., & Plumb, R. A. (2007). *Atmosphere, Ocean and Climate Dynamics: An Introductory Text.* Academic Press.

Marshall, J., & Schott, F. (1999). Open-ocean convection: Observations, theory, and models. *Reviews of Geophysics*, *37*(1), 1–64. doi: 10.1029/98RG02739

Mehta, P., Bukov, M., Wang, C.-H., Day, A. G., Richardson, C., Fisher, C. K.,

& Schwab, D. J. (2019). A high-bias, low-variance introduction to Machine Learning for physicists. *Physics Reports*, *810*, 1-124. doi: 10.1016/j.physrep.2019.03.001

O'Gorman, P. A., & Dwyer, J. G. (2018). Using Machine Learning to Parameterize Moist Convection: Potential for Modeling of Climate, Climate Change, and Extreme Events. *Journal of Advances in Modeling Earth Systems*, *10*(10), 2548–2563. doi: 10.1029/2018MS001351

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., ... Edelman, A. (2020). *Universal Differential Equations for Scientific Machine Learning.* arXiv. doi: 10.48550/ARXIV.2001.04385

Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, *5*(1), 15. doi: 10.5334/jors.151

Ramadhan, A., Lee, X. K., & Piterbarg, U. (2022, September). *ali-ramadhan/neuralfreeconvection.jl: v1.0.0.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.7108770` doi: 10.5281/zenodo.7108770

Ramadhan, A., Wagner, G. L., Hill, C., Campin, J.-M., Churavy, V., Besard, T., ... Marshall, J. (2020). Oceananigans.jl: Fast and friendly geophysical fluid dynamics on GPUs. *Journal of Open Source Software*, *5*(53), 2018. doi: 10.21105/joss.02018

Rasp, S., Pritchard, M. S., & Gentine, P. (2018). Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, *115*(39), 9684–9689. doi: 10.1073/pnas.1810286115

Resplandy, L., Keeling, R. F., Eddebbar, Y., Brooks, M., Wang, R., Bopp, L., ... Oschlies, A. (2019). Quantification of ocean heat uptake from changes in atmospheric O_2 and CO_2 composition. *Scientific Reports*, *9*(1), 1–10. doi: 10.1038/s41598-019-56490-z

Roemmich, D., Johnson, G. C., Riser, S., Davis, R., Gilson, J., Owens, W. B., ... Ignaszewski, M. (2009). The argo program: Observing the global ocean with profiling floats. *Oceanography*, *22*(2), 34–43. doi: 10.5670/oceanog.2009.36

Rozema, W., Bae, H. J., Moin, P., & Verstappen, R. (2015). Minimum-dissipation models for large-eddy simulation. *Physics of Fluids*, *27*(8), 085107. doi: 10.1063/1.4928700

Schneider, T., Teixeira, J., Bretherton, C. S., Brient, F., Pressel, K. G., Schär, C., & Siebesma, A. P. (2017). Climate goals and computing the future of clouds. *Nature Climate Change*, *7*(1), 3–5. doi: 10.1038/nclimate3190

Schumann, U., & Sweet, R. A. (1988). Fast Fourier Transforms for Direct Solution of Poisson's Equation with Staggered Boundary Conditions. *Journal of Computational Physics*, *75*(1), 123–137. doi: 10.1016/0021-9991(88)90102-7

Shu, C.-W. (2009). High Order Weighted Essentially Nonoscillatory Schemes for Convection Dominated Problems. *SIAM Review*, *51*, 82–126. doi: 10.1137/070679065

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems* (Vol. 25). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf`

Souza, A. N., Wagner, G. L., Ramadhan, A., Allen, B., Churavy, V., Schloss, J., ... Ferrari, R. (2020). Uncertainty Quantification of Ocean Parameterizations: Application to the K-Profile-Parameterization for Penetrative Convection. *Journal of Advances in Modeling Earth Systems*, *12*(12), e2020MS002108. doi: 10.1029/2020MS002108

Stevens, B., & Bony, S. (2013). What Are Climate Models Missing? *Science*, *340*(6136), 1053–1054. doi: 10.1126/science.1237554

Van Roekel, L., Adcroft, A. J., Danabasoglu, G., Griffies, S. M., Kauffman, B., Large, W., ... Schmidt, M. (2018). The KPP Boundary Layer Scheme for the Ocean: Revisiting Its Formulation and Benchmarking One-Dimensional Simulations Relative to LES. *Journal of Advances in Modeling Earth Systems*, *10*(11), 2647–2685. doi: 10.1029/2018MS001336

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). *Attention is all you need.* arXiv. doi: 10.48550/ARXIV.1706.03762

Verstappen, R. (2018). How much eddy dissipation is needed to counterbalance the nonlinear production of small, unresolved scales in a large-eddy simulation of turbulence? *Computers & Fluids*, *176*, 276–284. doi:

10.1016/j.compfluid.2016.12.016

Vreugdenhil, C. A., & Taylor, J. R. (2018). Large-eddy simulations of stratified plane Couette flow using the anisotropic minimum-dissipation model. *Physics of Fluids*, *30*(8), 085104. doi: 10.1063/1.5037039

Wang, C., Zhang, L., Lee, S.-K., Wu, L., & Mechoso, C. R. (2014). A global perspective on CMIP5 climate model biases. *Nature Climate Change*, *4*(3), 201–205. doi: 10.1038/nclimate2118

Wang, Y., Li, H.-X., Huang, T., & Li, L. (2014). Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Applied Soft Computing*, *18*, 232–247. doi: 10.1016/j.asoc.2014.01.038

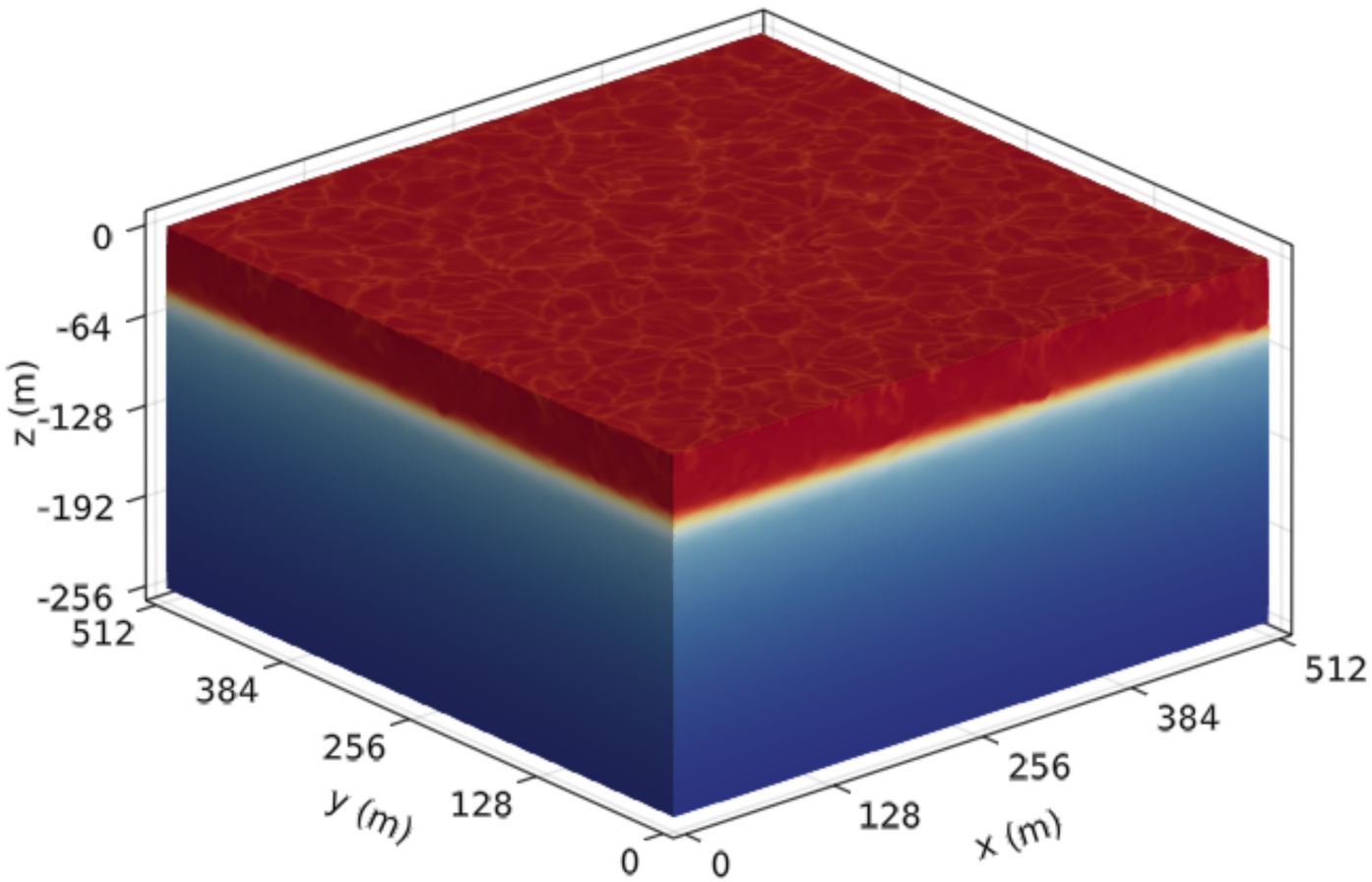
Williams, R. G., & Follows, M. J. (2011). *Ocean Dynamics and the Carbon Cycle: Principles and Mechanisms*. Cambridge University Press. doi: 10.1017/CBO9780511977817

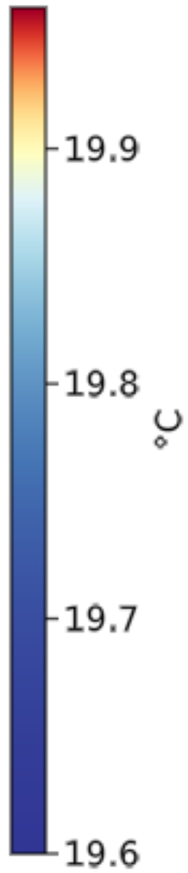
Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-i., & Jegelka, S. (2020). *How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks*. arXiv. doi: 10.48550/ARXIV.2009.11848

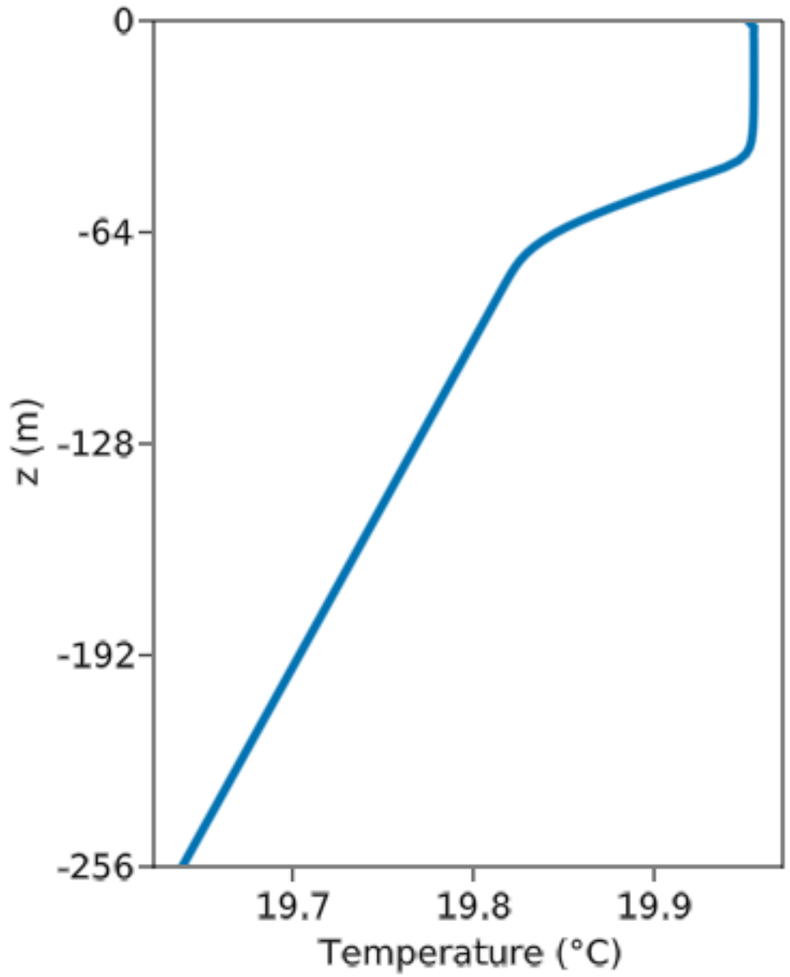
Yuval, J., O'Gorman, P. A., & Hill, C. N. (2021). Use of Neural Networks for Stable, Accurate and Physically Consistent Parameterization of Subgrid Atmospheric Processes With Good Performance at Reduced Precision. *Geophysical Research Letters*, *48*(6), e2020GL091363. doi: 10.1029/2020GL091363

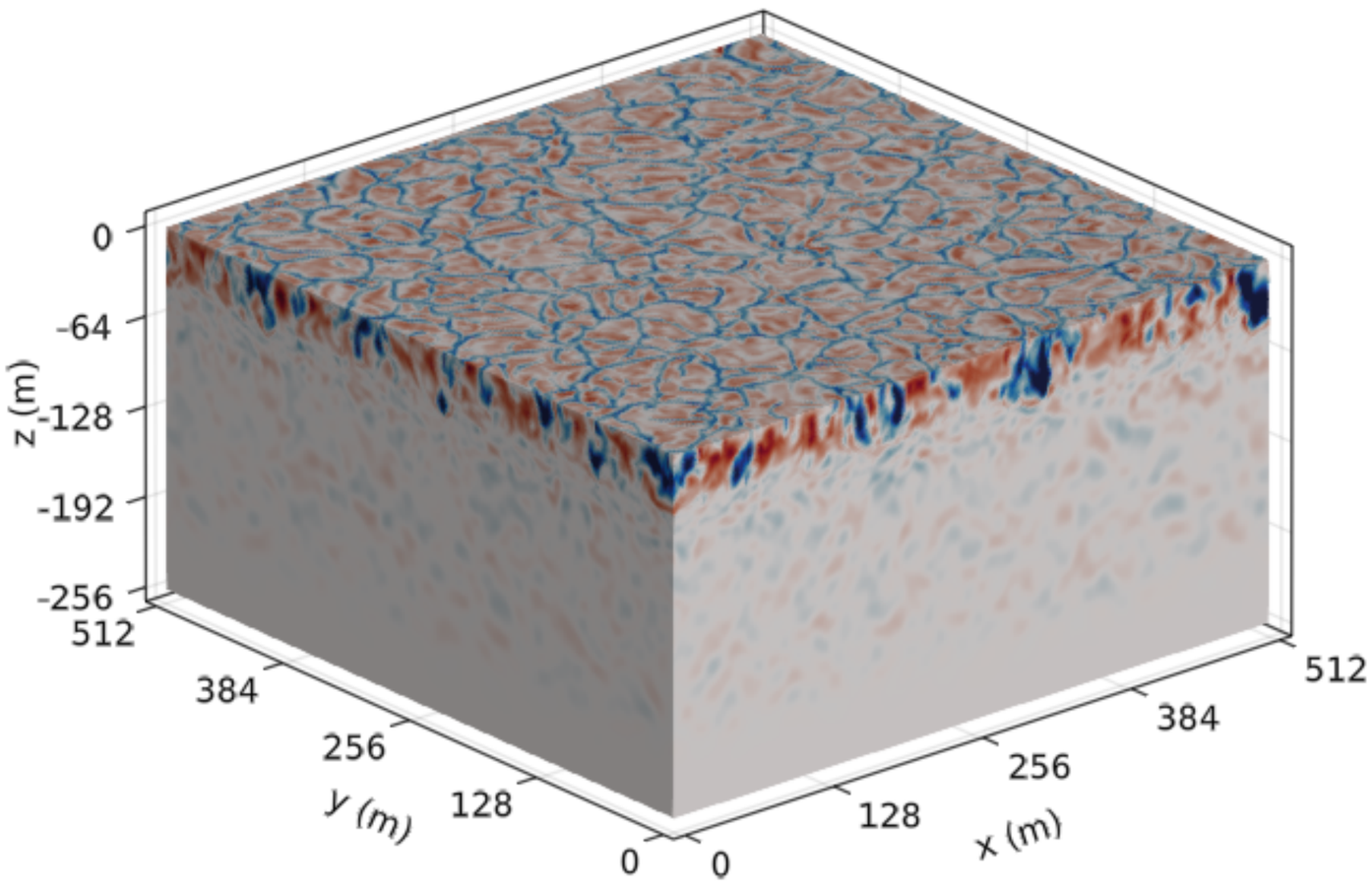
Zanna, L., & Bolton, T. (2020). Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters*, *47*(17), e2020GL088376. doi: 10.1029/2020GL088376

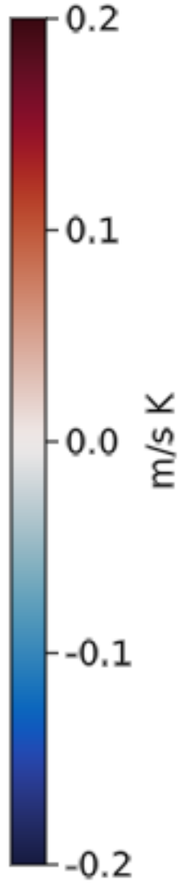
Figure 1.


0
-64
-128
-192
-256
z (m)
512
384
256
128
0
y (m)
0
128
256
384
512
x (m)


19.9
19.8
19.7
19.6
°C


0
-64
-128
-192
-256
z (m)
19.7
19.8
19.9
Temperature (°C)


0
-64
-128
-192
-256
z (m)
512
384
256
128
0
y (m)
0
128
256
384
512
x (m)


0.2
0.1
0.0
-0.1
-0.2
m/s K

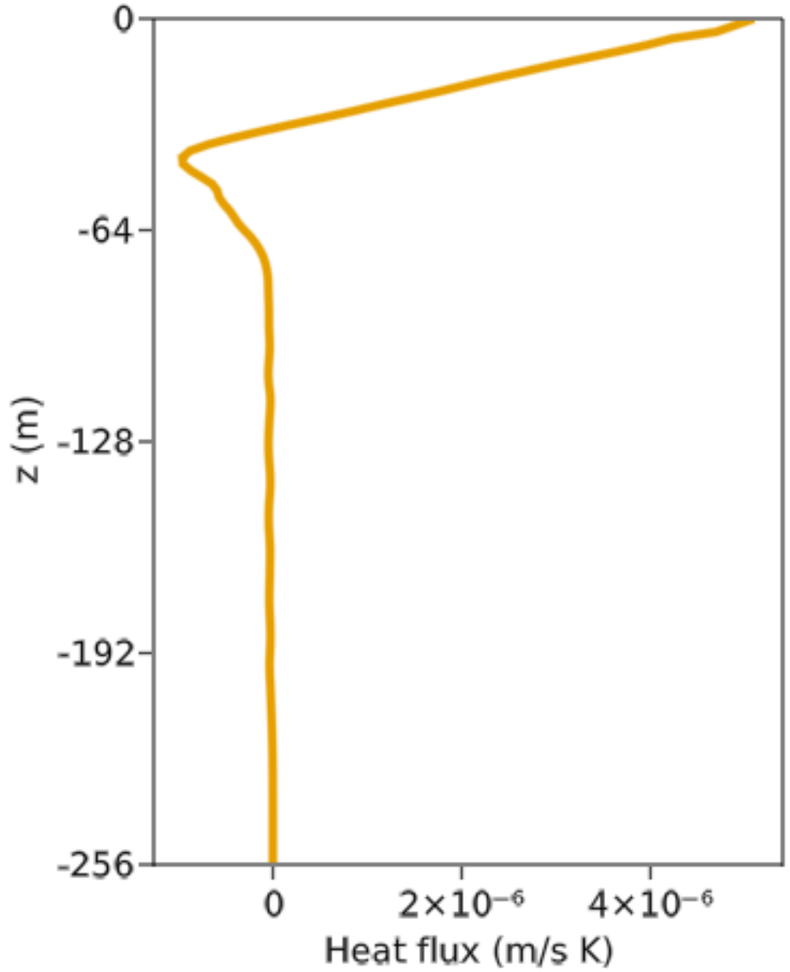

0
-64
-128
-192
-256
z (m)
0
2×10⁻⁶
4×10⁻⁶
Heat flux (m/s K)

Figure 2.

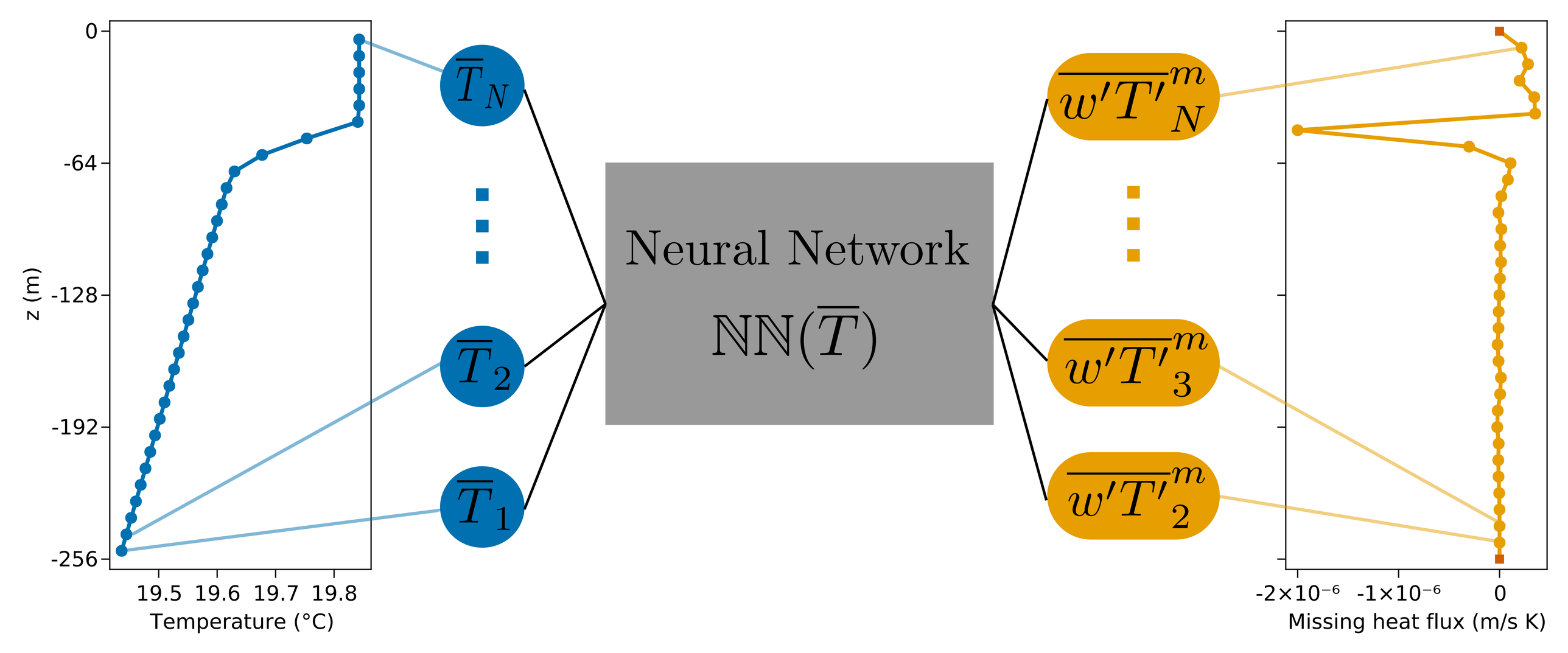

0
-64
-128
-192
-256
z (m)
19.5
19.6
19.7
19.8
Temperature (°C)
$\overline{T}_N$
$\overline{T}_2$
$\overline{T}_1$
Neural Network
$\mathbb{NN}(\overline{T})$
$\overline{w'T'}^m_N$
$\overline{w'T'}^m_3$
$\overline{w'T'}^m_2$
-2×10⁻⁶
-1×10⁻⁶
0
Missing heat flux (m/s K)

Figure 3.

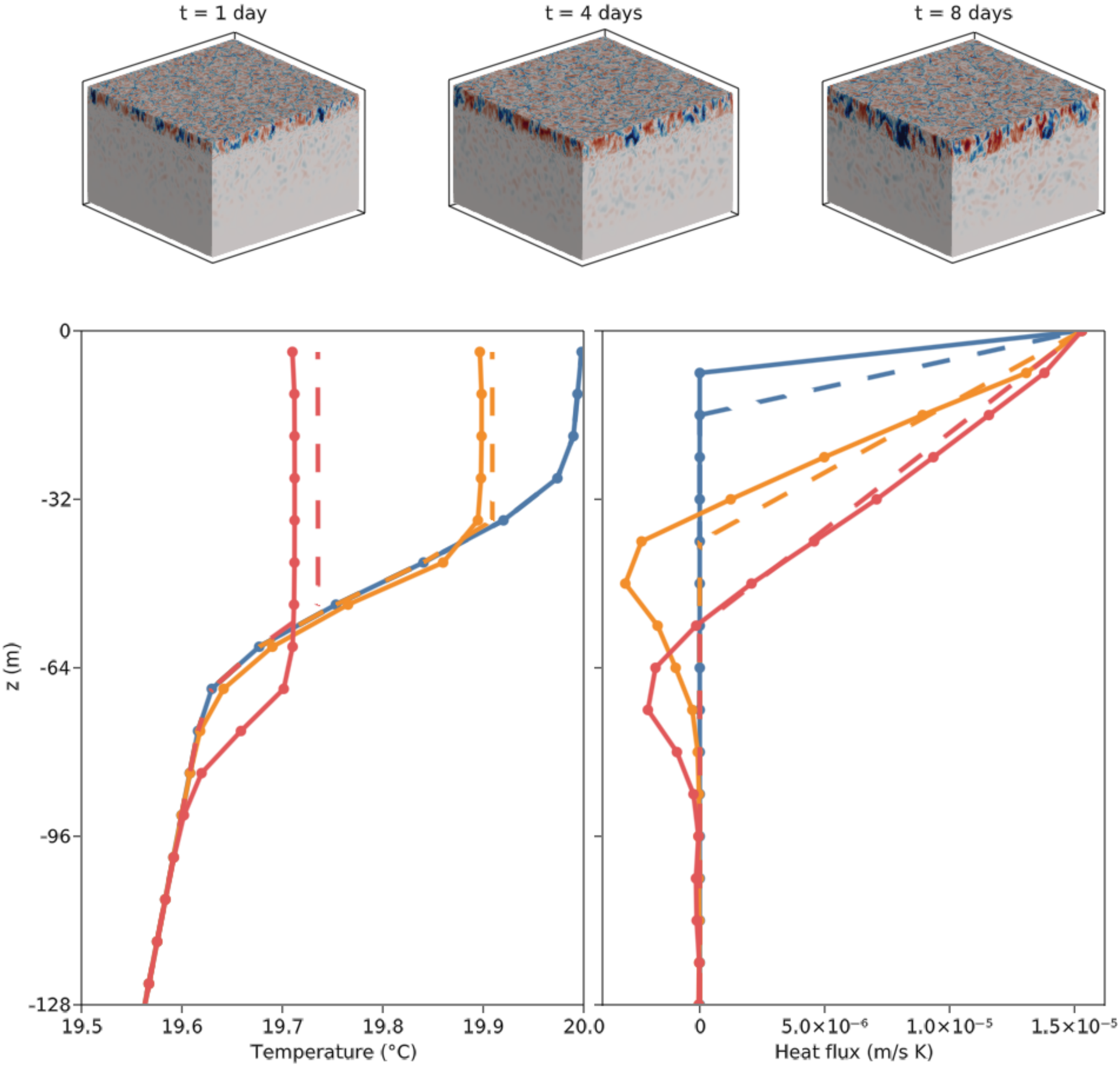

t = 1 day
t = 4 days
t = 8 days
0
-32
-64
-96
-128
z (m)
19.5
19.6
19.7
19.8
19.9
20.0
Temperature (°C)
0
5.0×10⁻⁶
1.0×10⁻⁵
1.5×10⁻⁵
Heat flux (m/s K)
t = 0 days
t = 2 days
t = 8 days
convective adjustment

Figure 4.

training
Qb interpolation
Qb extrapolation
N² interpolation
N² extrapolation

Surface cooling (W/m²)
20
60
100

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

1×10^{-8}
3×10^{-8}
5×10^{-8}
Surface buoyancy flux (m²/s³)

0
1×10^{-5}
2×10^{-5}
3×10^{-5}
Stratification (1/s²)

Figure 5.

training
Qb interpolation
Qb extrapolation
N² interpolation
N² extrapolation

Trained on $\mathcal{L}_1$ (fluxes)

Trained on $\mathcal{L}_2$ (time series)

$\mathcal{L}_1$ (fluxes)

0.3
0.2
0.1
0.0

$\mathcal{L}_2$ (time series)

10^{-3}
10^{-4}
10^{-5}

0
1000
2000
3000
4000
5000

0
500
1000
1500
2000

Epoch

Epoch

Figure 6.

Qb interpolation

Qb extrapolation

N² interpolation

N² extrapolation

training

Loss

10^{-2}

10^{-3}

10^{-4}

10^{-5}

10^{-6}

0

2

4

6

8

Simulation time (days)

Trained on fluxes

Trained on time series

Figure 7.

Qb interpolation

Qb extrapolation

N² interpolation

N² extrapolation

training

Loss

10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}

0
2
4
6
8

Simulation time (days)

K-Profile Parameterization
Convective adjustment
Neural differential equation

Figure 8.

Large eddy simulation
K-Profile Parameterization
Convective adjustment
Neural differential equation

z (m)
0
-32
-64
-96
-128
19.5
19.7
19.9
Temperature (°C)

0
-32
-64
-96
-128
-5×10⁻⁶
0
1×10⁻⁵
Heat flux (m/s K)

Loss
10⁻²
10⁻³
10⁻⁴
10⁻⁵
0
2
4
6
8
Simulation time (days)

Figure 9.

Qb interpolation

Qb extrapolation

N² interpolation

N² extrapolation

training

Loss

$10^{-3.5}$
$10^{-4.0}$
$10^{-4.5}$
$10^{-5.0}$
$10^{-5.5}$

0
500
1000
1500
2000

Epoch

dense (default)
dense (wider)
dense (deeper)
convolutional (2)
convolutional (4)

Figure 10.

Mean squared error loss

$10^{-2.0}$

$10^{-2.5}$

$10^{-3.0}$

10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1}

Runtime (seconds)

2.0

1.5

1.0

0.5

10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1}

Convective adjustment diffusivity