

# Capturing missing physics in climate model parameterizations using neural differential equations

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

<sup>1</sup>Massachusetts Institute of Technology

<sup>2</sup>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](mailto: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-

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

## 50 **1 Introduction: The parameterization challenge in climate modeling**

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

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

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

78 Embedding multiple parameterizations into a climate model to represent myriad  
79 unresolved scales thus introduces many free parameters that must be jointly estimated.  
80 The parameters may be correlated and point estimates of the optimal parameter values  
81 may be impossible, further complicating the calibration process (Souza et al., 2020). One  
82 common tuning procedure is to modify the parameters in an empirical fashion, but guided  
83 by intuition and understanding, in an effort to tune the climate model to reproduce the  
84 climate of the 20<sup>th</sup> century (Hourdin et al., 2017) during which global observations are  
85 available. Once the model has been calibrated on historical data it is then used to ex-  
86 trapolate into the future. It is clear that such projections must necessarily be compro-  
87 mised due to the uncertainty introduced by parameterizations of unresolved scales.

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

96 Rasp et al. (2018) trained a neural network to represent all atmospheric sub-grid  
97 processes in a climate model with explicit convection. Though the trained model could  
98 potentially surpass the parameterizations that it learned from in speed, it could not im-  
99 prove on their accuracy. Using a single model to substitute for parameterizations of mul-  
100 tiple processes also degrades interpretability because if the neural network behaves in un-  
101 expected ways then it is very difficult to ascertain underlying reasons. If neural networks  
102 do not exactly obey conservation laws, solutions can drift from reality ultimately lead-  
103 ing to numerical instability or blowup. Gentine et al. (2018) improved the approach by  
104 training a neural network to learn atmospheric moist convection from many superparam-  
105 eterized simulations that explicitly resolve moist convection. However, the 2D superpa-  
106 rameterized models do not always faithfully resolve the inherently 3D nature of atmo-  
107 spheric moist convection and their neural network also does not obey conservation laws.

108 O’Gorman and Dwyer (2018) train a random forest to parameterize moist convec-  
109 tion which has the advantage of obeying conservation laws and preserving the non-negativity

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

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

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

141 Our paper is set out as follows. In section 2 we introduce the concept of NDEs and  
142 how they might be used to augment and improve existing parameterization schemes. In  
143 section 3 we set up an NDE which will be used to improve a simple convective adjust-  
144 ment parameterization of free convection. In section 4 we describe how training data can  
145 be generated from high-resolution LES simulations of the ocean boundary layer. Section  
146 5 describes how the NDE is trained on this LES data using two different methods. In  
147 section 6 we compare the fidelity and performance of the trained NDE to commonly used  
148 parameterizations. In section 7 we explore different neural network architectures to in-  
149 vestigate whether nonlocality is important for the representation of free convection and  
150 to investigate the robustness of our approach to changes in neural network architecture.  
151 In section 8 we discuss the advantages NDEs more broadly and discuss ways in which  
152 we might improve the performance and interpretability of our approach. Finally, we sum-  
153 marise and discuss how NDEs might be applied to more complex and challenging pa-  
154 rameterization problems.

## 155 **2 Why use neural differential equations?**

### 156 **2.1 What are neural differential equations?**

157 Machine learning techniques have recently gained great popularity in the model-  
158 ing of physical systems because of their sometimes limited ability to automatically learn  
159 nonlinear relationships from data. However, although very promising, machine learning  
160 techniques have some notable disadvantages, such as their inability to extrapolate out-  
161 side the parameter range of the training data (Barnard & Wessels, 1992) and their lack  
162 of physical interpretation (Fan et al., 2021). Fortunately, recent work has shown that such  
163 concerns can be overcome, or at least ameliorated, by embedding prior structural knowl-  
164 edge into the machine learning framework (Xu et al., 2020). In the context of scientific  
165 models, the universal differential equation (Rackauckas et al., 2020) approach mixes known  
166 differential equation models with universal function approximators (such as neural net-  
167 works) and demonstrates the ability to learn from less data and be more amenable to  
168 generalization. In this paper we will define a neural differential equation (NDE) as a dif-  
169 ferential equation where at least one term contains a neural network. The neural net-  
170 work contains free parameters that must be inferred for the NDE to produce useful pre-  
171 dictions and be a useful model. Because they are phrased as a differential equation model,  
172 NDEs match the mechanistic form of physical laws, allowing for interpretability whilst

173 presenting a learnable structure. Section 3 showcases how this NDE architecture allows  
 174 for combining prior knowledge that inform existing parameterizations directly with neu-  
 175 ral networks. NDEs are trained by performing automatic differentiation through a dif-  
 176 ferential equation solver provided by the Julia scientific machine learning (SciML) soft-  
 177 ware stack (Rackauckas & Nie, 2017).

## 178 **2.2 Data-driven parameterizations using a residual model**

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

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

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

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

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

$$\partial_t T = \nabla \cdot \mathbf{F} \quad (1)$$

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

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

$$\mathbf{F} = \mathbf{F}_{\text{param}} + \mathbf{F}_{\text{missing}} \quad (2)$$

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

$$\mathbf{F}_{\text{missing}} = \text{NN}(\text{prognostic variables, surface fluxes, } \dots) \quad (3)$$

213 will be predicted by the neural network NN using available information such as the  
 214 current state of the prognostic variables, surface fluxes and boundary conditions, etc. Note  
 215 that NN, denoted such since we are using neural networks, need not be a neural network  
 216 and could be any function approximator such as a Gaussian process or a polynomial se-  
 217 ries.

218 The guiding principles taken here are that:

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

222 Note that studying the fluxes predicted by the neural network could lead to an im-  
 223 proved understanding of the missing physics and hence lead to the development of *bet-*  
 224 *ter* structured parameterizations, whether they be theory-driven or data-driven.



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

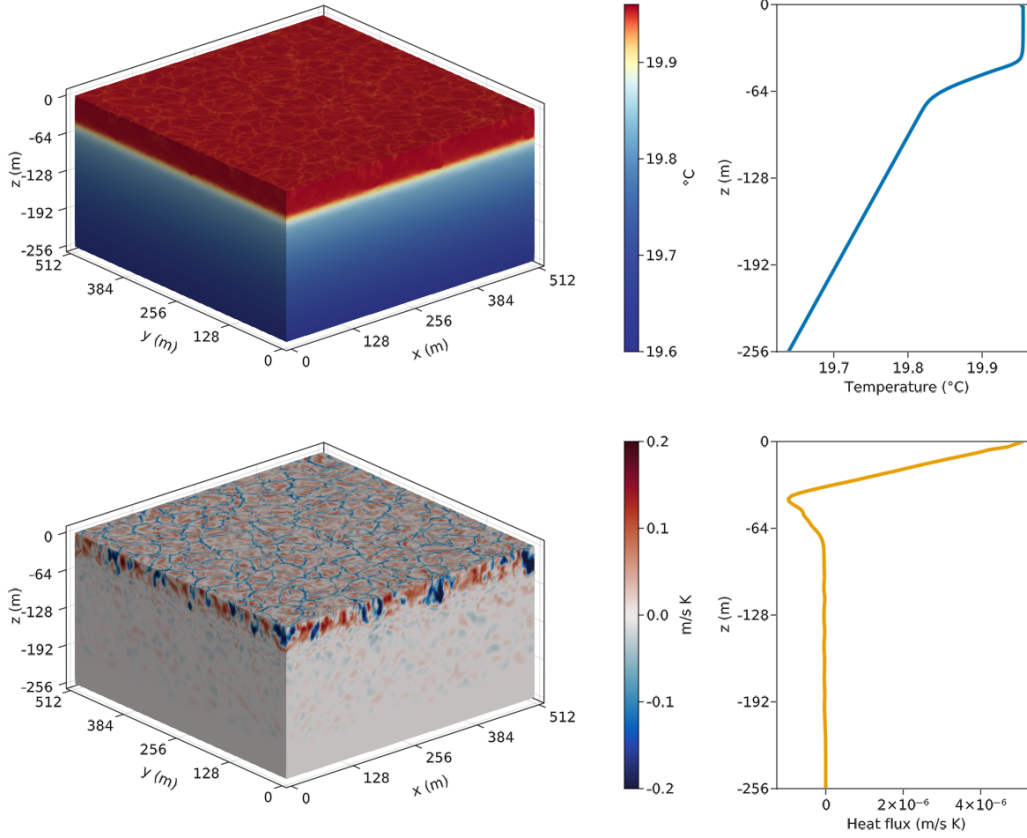
231 In the remainder of our paper we put these ideas in to practice in the context of  
232 free convection within an initially stratified, resting patch of ocean subject to heat loss  
233 at its upper boundary.

### 234 **3 Parameterizing free convection as a neural differential equation**

#### 235 **3.1 Free convection**

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

255 & Follows, 2011, §7.2). Thus accurately predicting the mixed layer depth and fluxes through  
 256 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  $t = 2$  days. (Right) The horizontally-averaged temperature  $\bar{T}(z)$  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).

257 Wind stresses are also present at the ocean’s surface which drive turbulent momen-  
 258 tum fluxes in addition to buoyancy fluxes, further complicating the physics of bound-  
 259 ary layer. That said, free convection is an important physical process which must be cap-  
 260 tured in models. Its accurate representation is non-trivial due to the entrainment of fluid  
 261 through the base of the mixed layer where convective plumes penetrate down into the

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

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

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

$$\partial_t \overline{T} = -\partial_z \overline{w'T'}. \quad (4)$$

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

$$\overline{\phi} = \overline{\phi}(z, t) = \frac{1}{L_x L_y} \int_0^{L_x} \int_0^{L_y} \phi(x, y, z, t) dx dy \quad (5)$$

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

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

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

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

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

### 302 **3.2 Base parameterization: Convective adjustment**

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

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

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

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

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

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

### 328 **3.3 The Neural Differential Equation for free convection**

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

332 We write down the heat flux from the one-dimensional free convection model in-  
 333 cluding convective adjustment thus:

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

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

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

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

338 we identify the missing heat flux as

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

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

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

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

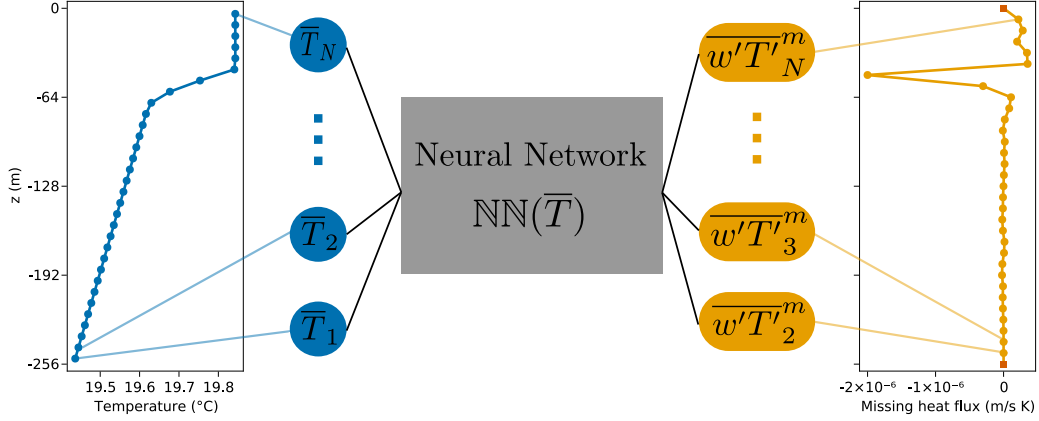
345 Noting that  $\overline{w'T'} = \overline{w'T'}|_{\text{param}} + \overline{w'T'}|_{\text{missing}}$  and substituting the above into the  
 346 PDE, equation (4), we obtain our NDE — a differential equation with a neural network  
 347 representing missing physics:

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

348 The input to the NN is just the current state of the temperature profile  $\overline{T}$ . So the  
 349 job of the NN is to predict  $\overline{w'T'}|_{\text{missing}}$  from the temperature profile  $\overline{T}$ . To train the NDE  
 350 we will non-dimensionalize equation (12) (see appendix D for a derivation of the non-  
 351 dimensional NDE) used in our codes. The NDE is implemented using DifferentialEqua-  
 352 tions.jl and Julia’s scientific machine learning (SciML) software stack (Rackauckas & Nie,  
 353 2017).

### 354 **3.4 Architecture of the Neural Network**

355 The NN predicts the missing (residual) fluxes from the prognostic variables con-  
 356 stituting the state of the model. In the present application, as shown schematically in  
 357 figure 2, we found it sufficient to supply just the temperature profile  $\overline{T}(z)$  as input with  
 358  $\overline{w'T'}|_{\text{missing}}(z)$  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  $\bar{T}(z)$  and  $\overline{w'T'}|_{\text{missing}}(z)$  profiles shown here are one example from the training data. See main text for more details.

359 The input of the neural network is a temperature profile vector  $\bar{T}(z)$  consisting of  
 360  $N = 32$  floating-point values as shown on the leftmost plot of figure 2 and denoted as  
 361  $\bar{T}_1, \bar{T}_2, \dots, \bar{T}_N$ . The  $N$  blue points show the temperature values which are the  $N$  in-  
 362 puts to the neural network, shown as bigger blue circles connected to the values. The  
 363 neural network outputs  $N-1$  values predicting the missing heat flux  $\overline{w'T'}|_{\text{missing}}(z)$   
 364 in the interior of the domain denoted as  $\overline{w'T'}_2^m, \overline{w'T'}_3^m, \dots, \overline{w'T'}_N^m$ . Yellow dots are fluxes  
 365 predicted by the neural network while red squares are boundary fluxes that are prescribed  
 366 and thus do not need to be predicted by the neural network. Knowledge of the bound-  
 367 ary fluxes, together with the form of our NDE, clearly guarantees that conservation is  
 368 obeyed. In all the calculations presented here we set  $N = 32$  so that the resolution of  
 369 the NDE is similar to that of a typical vertical mixing parameterization embedded in an  
 370 ocean model.

371 We use simple feed-forward neural networks in which the input propagates through  
 372 a series of layers with each layer taking an input then transforming it into an output be-  
 373 fore passing the output onto the next layer. The output may be fed through an activa-  
 374 tion function before being passed on. A layer may perform a linear transformation car-  
 375 ried out by an activation function  $\sigma: x \rightarrow \sigma(Wx + b)$ . 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  $\text{NN}$ , a series of fully-connected dense layers in which each layer performs the transformation  $x \rightarrow \text{relu}(Wx + b)$  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 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



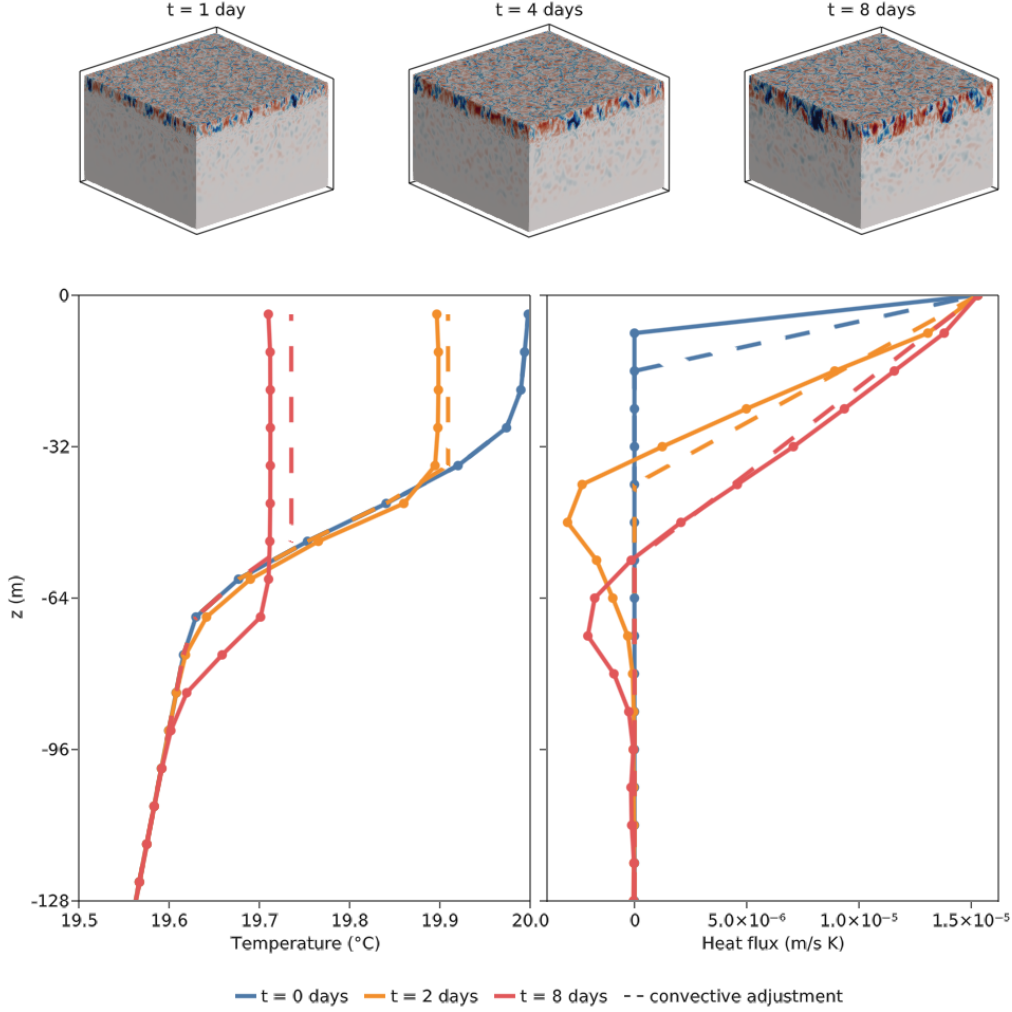
407 resolution required to resolve free convection is  $\mathcal{O}(1\text{ m})$  (Souza et al., 2020) which is far  
 408 beyond current computational capabilities in the context of a global simulation (Fox-Kemper  
 409 et al., 2019). Instead, we deploy the very high resolution LES model shown in figure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

## 413 4.2 Simulation setup

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

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

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

438  $\{1, \dots, N_t\}$ .  $N_z$  is the number of vertical grid elements and  $N_t$  is the number of time  
 439 snapshots. Figure 3 shows a snapshot of the training data from one of the training sim-  
 440 ulations. The predictions from a convective adjustment model are also plotted using dashed

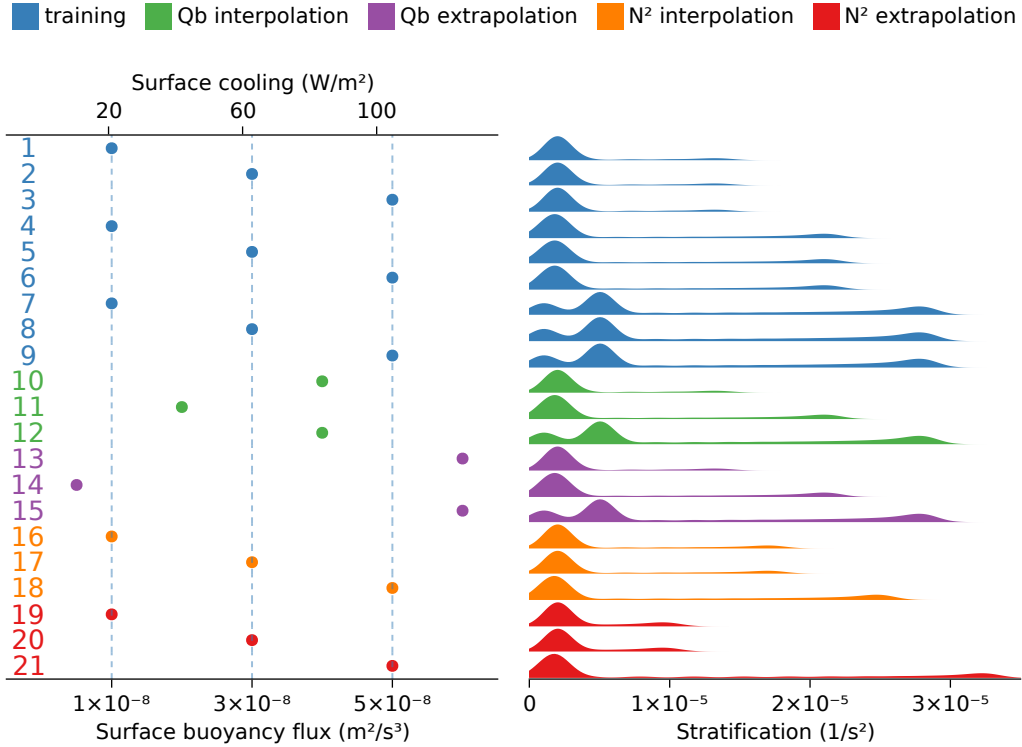
441 lines. The difference between the LES heat flux and the heat flux from convective ad-  
 442 justment is precisely the missing heat flux that the neural network embedded in the NDE  
 443 will learn to predict. In this way the NDE can close the gap between the simple convec-  
 444 tive adjustment parameterization scheme and the full LES simulation. With the neu-  
 445 ral network providing the missing entrainment heat fluxes, we expect the temperature  
 446 profiles to evolve with skill.

### 447 **4.3 Training and validation cases**

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

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

469 To vary different initial  $N^2$  profiles, the thicknesses and stratifications of the sur-  
 470 face and thermocline layers were changed (see appendices B and C for details) and the  
 471 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.

472 tribution of  $N^2$  present in the initial profile. To evaluate interpolation capabilities (sim-  
 473 ulations 16–18, colored orange) the simulation parameters were varied to create an ini-  
 474 tial  $N^2$  profile that is different from that of the training data but which exhibits strat-  
 475 ifications between those of the training simulations. This is evident in the plotted dis-  
 476 tributions (orange kernel density estimates) as their peaks either overlap with or are in  
 477 between the peaks of the distributions for the training simulations. To evaluate extrap-  
 478 olation capabilities (simulations 19–21, colored red) the simulation parameters were var-  
 479 ied to create two very weakly stratified simulations (19 and 20) and one very strongly

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

## 482 5 Training the Neural Differential Equation

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

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

### 499 5.1 Optimization of convective adjustment

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

508 **5.2 Method 1: differential control — training offline on the fluxes**

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

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

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

518 Here  $\mathcal{L}_1$  is the full loss function and  $\mathcal{L}_{1,s}$  is the loss function over simulation  $s$ ,  $N_s =$   
 519 9 is the number of training simulations and  $\boldsymbol{\theta}$  are the parameters being optimized to min-  
 520 imize the loss function, i.e. the neural network weights.

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

535 **5.3 Method 2: integral control — training online on the full fluxes**

536 While differential control can be expected to lead to a trained neural network ca-  
 537 pable of predicting missing fluxes from the temperature profile alone, small discrepan-  
 538 cies in the predicted fluxes may accumulate over time resulting in temperature drift. To  
 539 remedy this we also try to minimize a loss function that includes the term of principal  
 540 interest, the temperature profile itself.

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

545 The loss function in this case takes the form

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

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

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

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

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

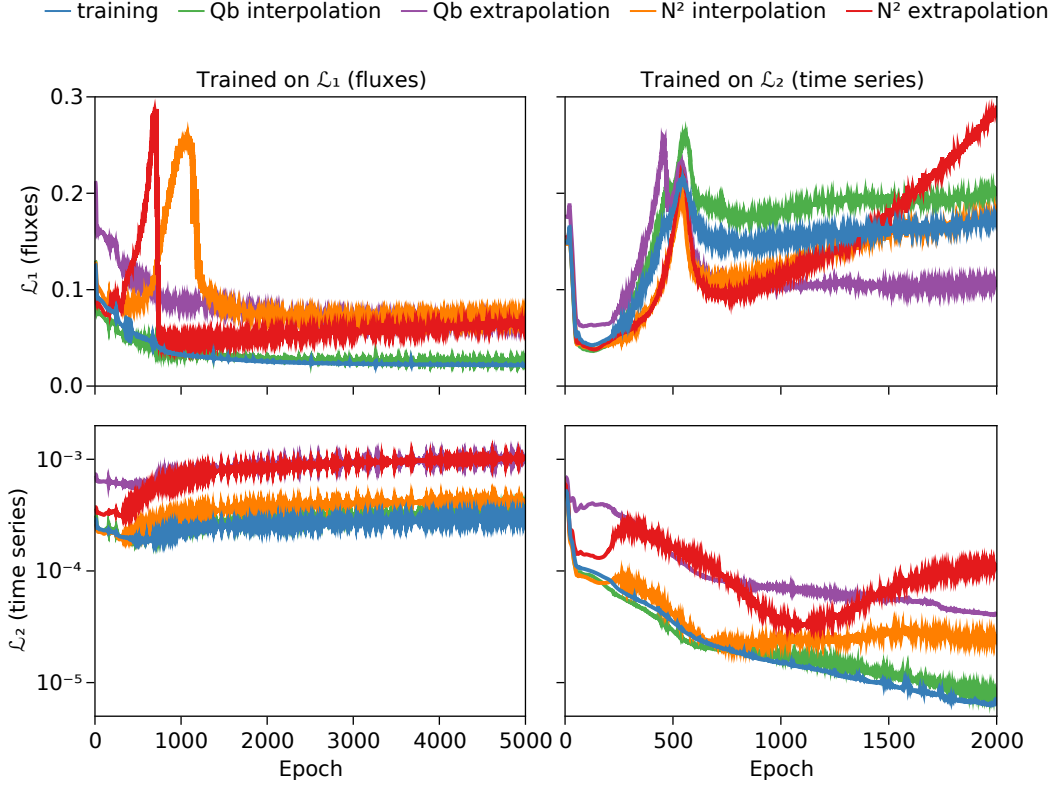
562 set using Glorot uniform initialization but then multiplied by  $10^{-5}$ . This has the effect  
 563 of making the predicted missing flux negligible at epoch zero but the training process  
 564 leads to the NDE slowly improving upon the base parameterization as it is trained. An  
 565 alternative training method, but one we found less simple, is to not train on the full time  
 566 series from epoch zero but rather to train incrementally on longer and longer time se-  
 567 ries. That is by increasing  $\tau$  in equation (14). For example,  $\tau$  can be set to 3 hours for  
 568 25 epochs, then 6 hours for the next 25 epochs, then 12 hours and so on. This has the  
 569 beneficial effect of slowly stabilizing the NDE as it is trained. During the integral con-  
 570 trol training process stabilized explicit time stepping methods of the ROCK family (Abdulle,  
 571 2002) are utilized to efficiently handle the stiffness introduced in the PDE discretization  
 572 to improve the performance and stability of the numerical solution. We find that ROCK4  
 573 provides enough stability for all our training cases.

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

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

592 Inspection of the two panels on the left enables us to evaluate the fidelity of the  
 593 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.

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

## 598 **6 Assessing the fidelity of the trained Neural Differential Equation**

### 599 **6.1 Differential versus integral control**

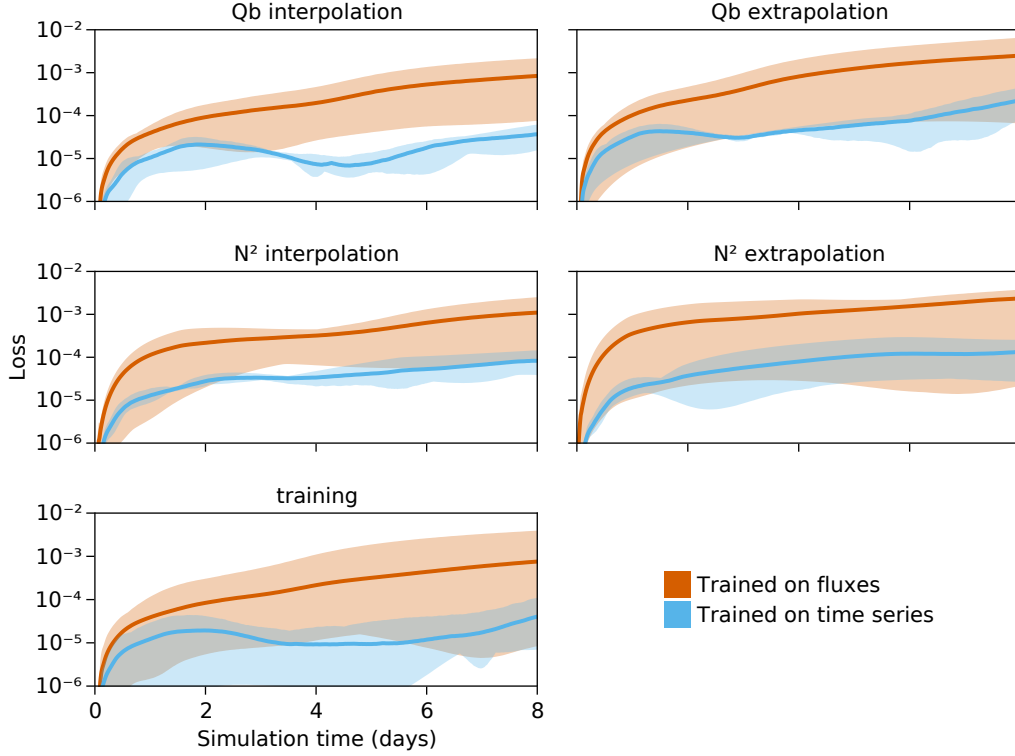
600 In section 5 we described two methods of training the NDE, first via differential  
 601 control and second via integral control. We now address the question of whether train-

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

611 Figure 6 shows how our loss metric compares between our two methods over time.  
 612 Using differential control the loss increases as the simulation progresses in time. This may  
 613 be because, even though the neural network is trained to reproduce the fluxes at snap-  
 614 shots in time, errors are compounded when time-stepping forward leading to divergence.  
 615 With integral control training, instead, the neural network attempts to reproduce the  
 616 time series in its entirety. As a result the error does not generally grow but remains bounded.  
 617 The exception is the slight increase in error near the end of the simulation period at  $t =$   
 618 8 days. The fact that the NDE solution remains congruent with the LES solution, and  
 619 that by  $t = 8$  days the loss when trained on the full time history is smaller by more than  
 620 an order-of-magnitude in both training and validation sets, suggests that the use of in-  
 621 tegral control is superior to the differential control approach. This demonstrates that while  
 622 loss functions can be constructed to train embedded neural networks independently of  
 623 the simulation process, the approach which requires differentiating the simulation greatly  
 624 improves the stability of the learned result.

## 625 **6.2 Comparison with other parameterizations**

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

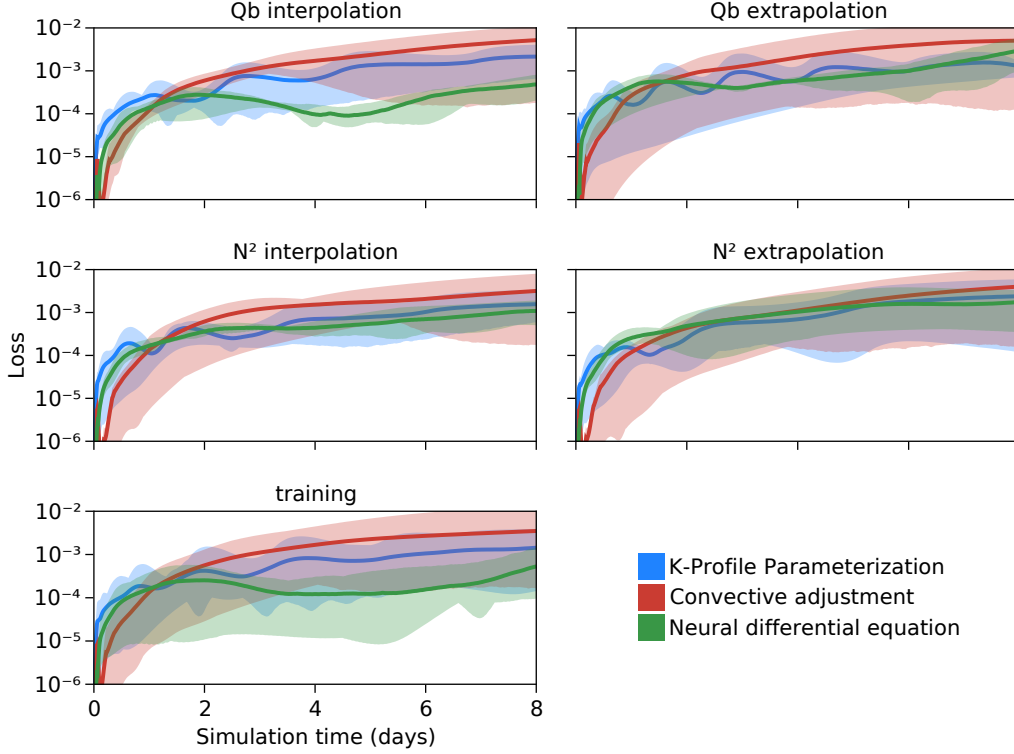


**Figure 6.** Time series of the loss function  $\mathcal{L}_2$  [equation (14)] over the period of mixed layer evolution. In blue is the loss of the NDE trained on the time series (integral control) while in orange is the loss of the NDE trained on the instantaneous fluxes (differential control). The 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.

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

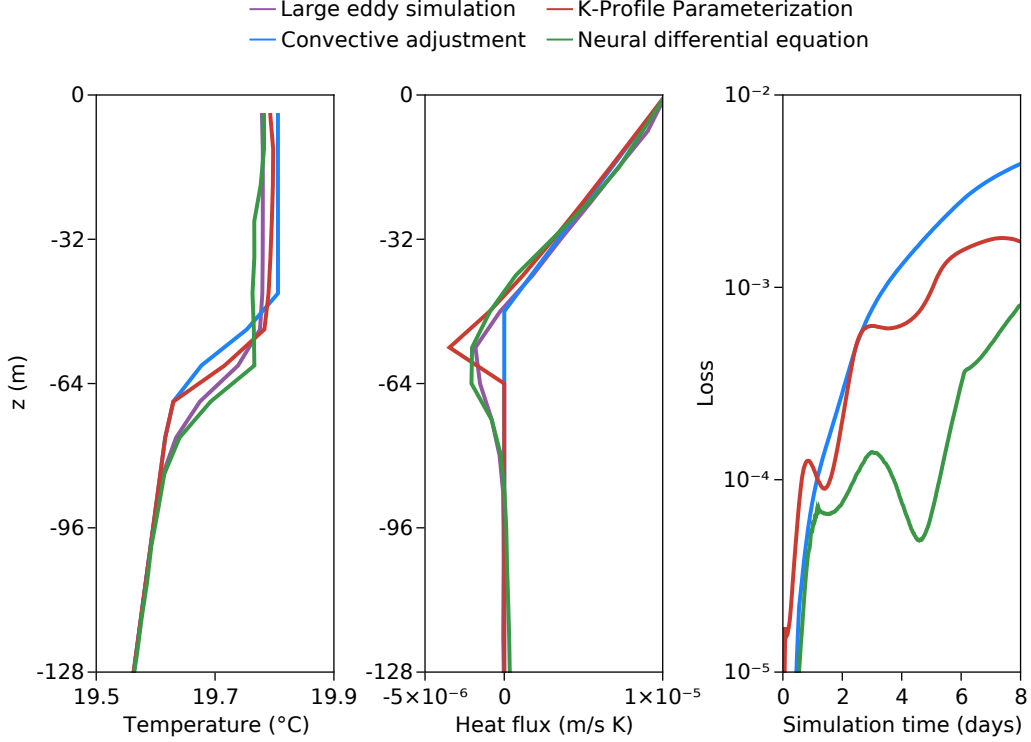
638 Figure 7 shows how each parameterization performs on each of the five simulation  
 639 sets. As expected, we see that in each set our NDE outperforms convective adjustment  
 640 since we can only improve upon the base parameterization. As the simulation proceeds,  
 641 the loss for each parameterization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

645 NDE and KPP perform similarly. The NDE is capable of some extrapolation when tested  
 646 against buoyancy fluxes  $Q_b$  it has not seen before, but shows less skill when the strat-  
 647 ification  $N^2$  is outside the range of the training data. We will further investigate the abil-  
 648 ity to extrapolate using our NDE in section 7 when we modify the architecture of the  
 649 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.

650 Figure 8 updates figure 3 to include the NDE solution under integral control. The  
 651 NDE solution matches the LES solution much more closely than the other parameter-  
 652 izations for both temperature  $\bar{T}(z)$  and heat flux  $\overline{w'T'}(z)$  profiles. Of particular inter-  
 653 est is how closely the NDE heat flux profiles matches up with the profile from the LES  
 654 simulation. The neural network was not trained on the heat flux but rather on the tem-  
 655 perature time series. Nevertheless, it predict a physically meaningful and accurate heat  
 656 flux.



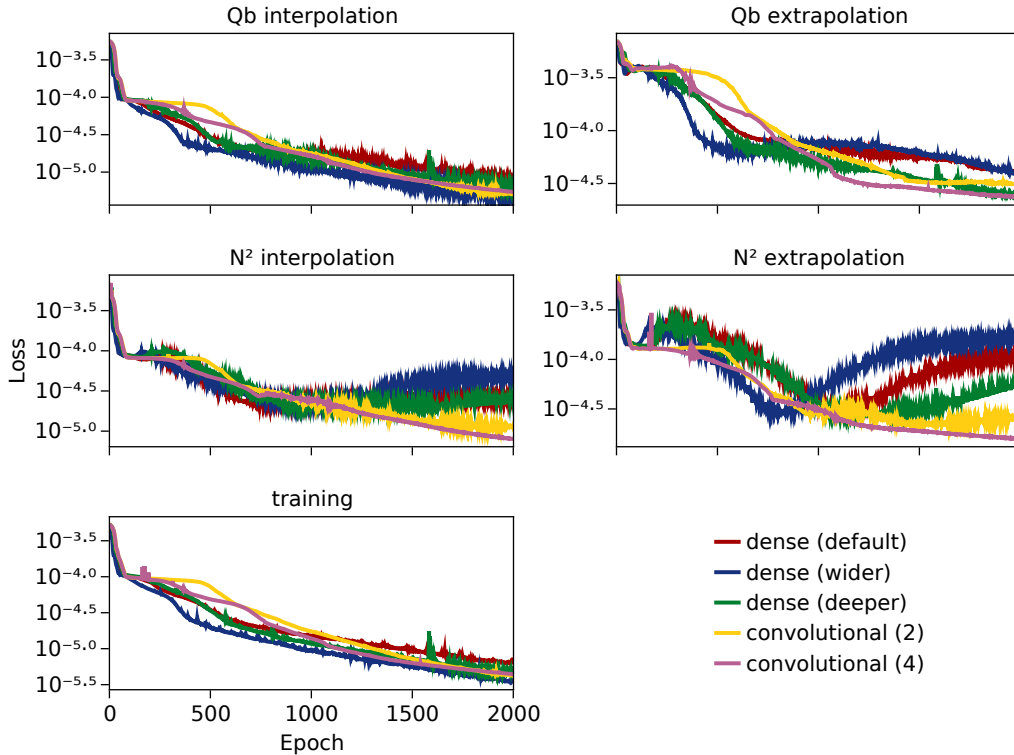
**Figure 8.** Comparison of three different parameterizations against the LES (truth) solution from simulation 11 (see table 1 in appendix C), one of our  $Q_b$  interpolation simulations. (Left) The heat flux profile  $\overline{w'T'}(z)$  predicted by the LES and by each parameterization. (Middle) 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.

## 657 7 Hyperparameter optimization

658 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.

662 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

666 add an extra hidden layer of size  $4N_z$ . We also use two neural networks with convolu-  
 667 tional first layers and filter sizes of 2 and 4. The use of the dense and “convolutional”  
 668 network architectures can be thought of making the assumption that the parameteriza-  
 669 tion is fully nonlocal in the dense case and local in the convolutional case with the fil-  
 670 ter size determining the degree of locality. Appendix G details the architectures used in-  
 671 cluding 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.

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

678 not exhibit non-locality (Souza et al., 2020) suggesting that only local information is re-  
679 quired and that convolutional networks should outperform fully-connected networks. The  
680 convolutional (4) architecture also exhibits less high frequency noise in the loss function,  
681 perhaps indicating that its gradients are less noisy and/or better behaved leading to a  
682 less challenging training process. The convolutional (4) architecture also has the fewest  
683 free parameters, suggesting that fully-connected networks perhaps have too many, lead-  
684 ing to overfitting of the training data. Perhaps designing the architecture around phys-  
685 ical assumptions is more important than designing larger or deeper networks.

686 An encouraging result is that our NDE approach seems robust to the architecture  
687 of the neural network employed in that all five architectures were trainable. This sug-  
688 gesting that the NDE approach could perhaps be used to perform hypothesis testing on  
689 the ideal structure of the parameterization. More investigation is necessary to determine  
690 whether a single architecture can perform best in all cases. A more systematic and pow-  
691 erful approach to hyperparameter optimization might prove useful here (Snoek et al.,  
692 2012) and specialized inference algorithms for large datasets and networks exist (Klein  
693 et al., 2017).

## 694 **8 Discussion and conclusions**

695 Here we have illustrated how NDEs might provide an attractive medium for de-  
696 veloping data-driven parameterizations for climate models. NDEs can be constructed  
697 to augment an existing parameterization scheme whilst remaining faithful to conserva-  
698 tion laws. They can then be, for example, trained on high-resolution LES data within  
699 a differential equation time-stepper. The resulting solutions are stable and faithfully rep-  
700 resent turbulent processes which are challenging to capture using conventional schemes.  
701 For example, we have shown that a simple parameterization of convection — that of con-  
702 vective adjustment — can be improved upon by training the NDE to capture the entrain-  
703 ment fluxes at the base of the well-mixed layer, fluxes that convective adjustment can-  
704 not represent. The augmented parameterization outperforms existing commonly used  
705 parameterizations such as KPP. Training even stiff NDEs (e.g. when convective adjust-  
706 ment is employed) is made possible by the Julia scientific machine learning (SciML) soft-  
707 ware stack.

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

714 We have chosen to focus on convectively-driven turbulence of the ocean boundary  
715 layer since it provides an idealised, proof-of-context setting for our exploration of NDEs.  
716 A number of implementation choices were made for simplicity and could be improved.  
717 The parameterization was trained and tested using a single vertical resolution. Exist-  
718 ing parameterizations such as KPP can be sensitive to model resolution (Souza et al.,  
719 2020) and development of a resolution-independent parameterization would be desirable.  
720 One approach might be to decompose the temperature profile into a Fourier or Cheby-  
721 shev series. The NN could then be trained on polynomial coefficients (Li et al., 2020).  
722 Profiles sampled at different resolutions might then lead to an NDE that effectively be-  
723 comes resolution-independent. Going even further, an autoencoder neural network might  
724 be used to learn the best representation of the data which could, in principle, be very  
725 different from a polynomial expansion, although perhaps less interpretable.

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



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

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

## 765 **A Numerical methods and Oceananigans.jl**

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

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

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

$$\partial_t T + \mathbf{u} \cdot \nabla T = -\nabla \cdot \mathbf{q} \quad (16)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (17)$$

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

$$b = \alpha g (T_0 - T) \quad (18)$$

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

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

$$\tau_{ij} = -2\nu_e \Sigma_{ij} \quad \text{and} \quad \mathbf{q} = -\kappa_e \nabla T \quad (19)$$

782 where  $\Sigma_{ij} = (\partial_i u_j + \partial_j u_i)/2$  is the strain rate tensor and  $\nu_e$  and  $\kappa_e$  are the eddy  
 783 viscosity and eddy diffusivity of temperature. The eddy viscosity  $\nu_e$  and eddy diffusiv-  
 784 ity  $\kappa_e$  in equation (19) are modeled with the anisotropic minimum dissipation (AMD)  
 785 formalism introduced by Rozema et al. (2015) and Abkar et al. (2016), refined by Verstappen  
 786 (2018), and validated and described in detail for ocean-relevant scenarios by Vreugdenhil

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

789 To solve equations (15)–(17) Oceananigans.jl uses a staggered C-grid finite volume  
 790 spatial discretization (Arakawa & Lamb, 1977) with an upwind-biased 5<sup>th</sup>-order weighted  
 791 essentially non-oscillatory (WENO) advection scheme for momentum and tracers (Shu,  
 792 2009). Diffusion terms are computed using centered 2<sup>nd</sup>-order differences. A pressure pro-  
 793 jection method is used to ensure the incompressibility of  $\mathbf{u}$  at every time step (Brown  
 794 et al., 2001). A fast Fourier-transform-based eigenfunction expansion of the discrete second-  
 795 order Poisson operator is used to solve the discrete pressure Poisson equation for the pres-  
 796 sure on a regular grid (Schumann & Sweet, 1988). An explicit 3<sup>rd</sup>-order Runge-Kutta  
 797 method is used to advance the solution in time (Le & Moin, 1991).

## 798 **B Simulation setup and generation of LES training data**

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

804 In all simulations, the model domain is  $512\text{ m} \times 512\text{ m} \times 256\text{ m}$  with 256 grid points  
 805 in the horizontal and 128 grid points in the vertical, giving a 2 m isotropic grid spacing.  
 806 A Coriolis parameter of  $f = 10^{-4}\text{ s}^{-1}$  is used. The fluid is initially at rest ( $\mathbf{u} = \mathbf{0}$ ) and  
 807 the initial condition on  $T$  is horizontally homogeneous and has three layers. At the top  
 808 is a *surface* layer of thickness  $\Delta z_s$  with constant stratification  $N_s^2$ . At the bottom is a  
 809 *deep* layer with constant stratification  $N_d^2$ . In the middle is a *transition* layer or *ther-*  
 810 *mocline* layer of thickness  $\Delta z_t$  with nonlinear stratification determined by fitting a cu-  
 811 bic polynomial for  $T$  between the surface and deep layers preserving continuity and dif-  
 812 ferentiability. The deep layer then has a thickness of  $L_z - \Delta z_s - \Delta z_t$  where  $L_z = 256\text{ m}$   
 813 is the domain depth. Random noise of the form  $\epsilon e^{z/\delta}$  where  $\epsilon \sim \text{Uniform}(0, 1)$  and  $\delta =$   
 814  $8\text{ m}$  is a noise decay length scale is added to the initial  $T$  to stimulate numerical convec-  
 815 tion near the surface. An example of this initial condition can be seen in figure 3.

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

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

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

## 826 C Simulation parameters

827 In table 1 we tabulate the parameters used to generate training and validation sim-  
 828 ulation data. Simulations 1–9 were used for training while simulations 10–21 were used  
 829 for validation. Simulations 1–3 were designed to have a smaller thermocline, while 4–  
 830 6 have a medium thermocline, and 7–9 have a large thermocline. Simulation 10–12 are  
 831 used to test for  $Q_b$  interpolation, 13–15 for  $Q_b$  extrapolation, 16–18 for  $N^2$  interpolation,  
 832 and 19–21 for  $N^2$  extrapolation. Figure 4 shows where in  $Q_b$ – $N^2$  parameter space these  
 833 simulations lie.

## 834 D Derivation of the non-dimensional PDE

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

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

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

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

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

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

$$\hat{t} = \frac{t}{\tau} \quad \text{and} \quad \hat{z} = \frac{z}{L_z} \quad (21)$$

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

$$\frac{\partial \overline{T}}{\partial t} = \frac{\sigma_T}{\tau} \frac{\partial \widehat{\overline{T}}}{\partial \widehat{t}} \quad (22)$$

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

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

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

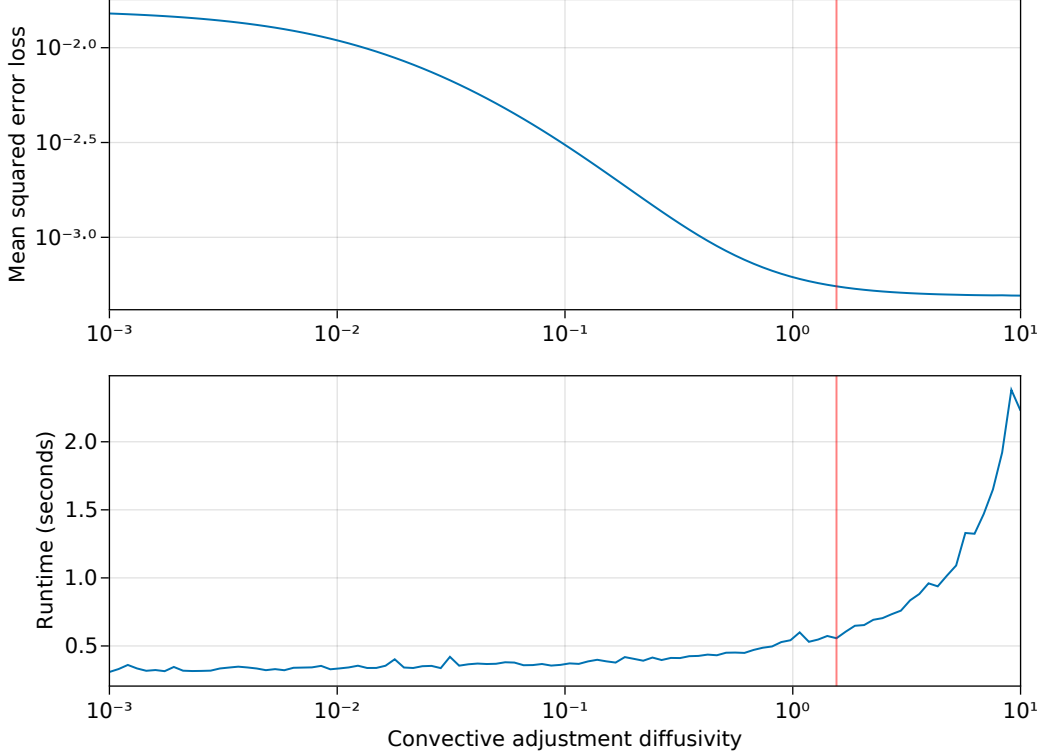
$$\frac{\partial \widehat{\overline{T}}}{\partial \widehat{t}} = -\frac{\sigma_{w'T'}}{\sigma_{\overline{T}}} \left( \frac{\partial \widehat{\overline{w'T'}}}{\partial \widehat{z}} - \widehat{\kappa} \frac{\partial \widehat{\overline{T}}}{\partial \widehat{z}} \right) \quad (24)$$

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

## 853 **E Calibration of convective adjustment scheme**

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

860 We see that increasing  $K_{CA}$  only decreases the loss until it plateaus when  $K_{CA} >$   
 861 1. It would seem natural, then, that we pick a large enough value to readily induce ver-  
 862 tical mixing. However, as the NDE implemented in DifferentialEquations.jl uses the ROCK4  
 863 explicit time-stepper, increasing  $K_{CA}$  also increases the stiffness of the system of differ-  
 864 ential equations necessitating a smaller time step and therefore longtimer runtimes. We  
 865 therefore choose a value of  $K_{CA}$  that minimizes the product of the loss function and the  
 866 runtime. This turns out to correspond to a non-dimensional diffusivity of  $\widehat{K}_{CA} \approx 2$  or  
 867 a dimensional value of  $K_{CA} = (H^2/\tau)\widehat{K}_{CA} \approx 0.2 \text{ m}^2 \text{ s}^{-1}$ . This is a very reasonable  
 868 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.

## 869 **F Calibration of the K-Profile Parameterization**

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

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

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

881 Using reference values as an initial guess, we apply an adaptive differential evolu-  
 882 tion optimization algorithm (Y. Wang et al., 2014) with radius-limited sampling as im-  
 883 plemented in BlackBoxOptim.jl (Feldt, 2018) (`adaptive_de_rand_1_bin_radiuslimited`  
 884 more specifically). This algorithm is suitable for finding global minima in the absence  
 885 of gradient information from automatic differentiation. In this way we train KPP on our  
 886 9 simulations by optimizing our four parameters to minimize the loss function (14).

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

## 893 **G Neural network architectures**

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

## 901 **Open Research**

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

## 905 **Acknowledgements**

906 We thank Keaton Burns for insightful discussions during the development of this  
 907 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)	$D_{N_z \rightarrow 4N_z}, D_{4N_z \rightarrow 4N_z}, D_{4N_z \rightarrow N_z-1}$	24,735	4.89	20.76
dense (wider)	$D_{N_z \rightarrow 8N_z}, D_{8N_z \rightarrow 8N_z}, D_{8N_z \rightarrow N_z-1}$	82,207	6.23	35.05
dense (deeper)	$D_{N_z \rightarrow 4N_z}, D_{4N_z \rightarrow 4N_z},$ $D_{4N_z \rightarrow 4N_z}, D_{4N_z \rightarrow N_z-1}$	41,247	5.77	25.48
convolutional (2)	$C_2, D_{N_z-2+1 \rightarrow 4N_z},$ $D_{4N_z \rightarrow 4N_z}, D_{4N_z \rightarrow N_z-1}$	24,610	8.84	37.97
convolutional (4)	$C_4, D_{N_z-4+1 \rightarrow 4N_z},$ $D_{4N_z \rightarrow 4N_z}, D_{4N_z \rightarrow N_z-1}$	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.

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

910 **References**

- 911 Abdulle, A. (2002). Fourth order Chebyshev methods with recurrence rela-  
 912 tion. *SIAM Journal on Scientific Computing*, *23*(6), 2041–2054. doi:  
 913 10.1137/S1064827500379549
- 914 Abkar, M., Bae, H. J., & Moin, P. (2016). Minimum-dissipation scalar transport  
 915 model for large-eddy simulation of turbulent flows. *Phys. Rev. Fluids*, *1*,  
 916 041701. doi: 10.1103/PhysRevFluids.1.041701
- 917 Arakawa, A., & Lamb, V. R. (1977). Computational Design of the Basic Dynamical  
 918 Processes of the UCLA General Circulation Model. In *Methods in Computa-*  
 919 *tional Physics: Advances in Research and Applications* (Vol. 17, pp. 173–265).  
 920 Elsevier. doi: 10.1016/B978-0-12-460817-7.50009-4
- 921 Barnard, E., & Wessels, L. (1992). Extrapolation and interpolation in neural net-  
 922 work classifiers. *IEEE Control Systems Magazine*, *12*(5), 50–53. doi: 10.1109/  
 923 37.158898
- 924 Besard, T., Foket, C., & De Sutter, B. (2019). Effective Extensible Programming:  
 925 Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed*  
 926 *Systems*, *30*(4), 827–841. doi: 10.1109/TPDS.2018.2872064
- 927 Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Ap-  
 928 proach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. doi: 10.1137/  
 929 141000671
- 930 Bolton, T., & Zanna, L. (2019). Applications of Deep Learning to Ocean Data In-  
 931 ference and Subgrid Parameterization. *Journal of Advances in Modeling Earth*  
 932 *Systems*, *11*(1), 376–399. doi: 10.1029/2018MS001472
- 933 Brown, D. L., Cortez, R., & Minion, M. L. (2001). Accurate Projection Methods  
 934 for the Incompressible Navier–Stokes Equations. *Journal of Computational*  
 935 *Physics*, *168*(2), 464–499. doi: 10.1006/jcph.2001.6715
- 936 DuVivier, A. K., Large, W. G., & Small, R. J. (2018). Argo Observations of  
 937 the Deep Mixing Band in the Southern Ocean: A Salinity Modeling Chal-  
 938 lenge. *Journal of Geophysical Research: Oceans*, *123*(10), 7599–7617. doi:  
 939 10.1029/2018JC014275
- 940 Fan, F.-L., Xiong, J., Li, M., & Wang, G. (2021). On interpretability of artificial  
 941 neural networks: A survey. *IEEE Transactions on Radiation and Plasma Med-*  
 942 *ical Sciences*, *5*(6), 741–760. doi: 10.1109/TRPMS.2021.3066428

- 943 Feldt, R. (2018). *Blackboxoptim.jl*. [https://github.com/robertfeldt/](https://github.com/robertfeldt/BlackBoxOptim.jl)  
 944 [BlackBoxOptim.jl](https://github.com/robertfeldt/BlackBoxOptim.jl). GitHub.
- 945 Fox-Kemper, B., Adcroft, A., Böning, C. W., Chassignet, E. P., Curchitser, E.,  
 946 Danabasoglu, G., ... Yeager, S. G. (2019). Challenges and Prospects  
 947 in Ocean Circulation Models. *Frontiers in Marine Science*, 6. doi:  
 948 10.3389/fmars.2019.00065
- 949 Gentine, P., Pritchard, M., Rasp, S., Reinaudi, G., & Yacalis, G. (2018). Could Ma-  
 950 chine Learning Break the Convection Parameterization Deadlock? *Geophysical*  
 951 *Research Letters*, 45(11), 5742–5751. doi: 10.1029/2018GL078202
- 952 Glorot, X., & Bengio, Y. (2010, 13–15 May). Understanding the difficulty of  
 953 training deep feedforward neural networks. In Y. W. Teh & M. Tittering-  
 954 ton (Eds.), *Proceedings of the thirteenth international conference on artificial*  
 955 *intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sar-  
 956 dinia, Italy: PMLR. Retrieved from [https://proceedings.mlr.press/v9/](https://proceedings.mlr.press/v9/glorot10a.html)  
 957 [glorot10a.html](https://proceedings.mlr.press/v9/glorot10a.html)
- 958 Griffies, S. M., Winton, M., Anderson, W. G., Benson, R., Delworth, T. L., Dufour,  
 959 C. O., ... Zhang, R. (2015). Impacts on Ocean Heat from Transient Mesoscale  
 960 Eddies in a Hierarchy of Climate Models. *Journal of Climate*, 28(3), 952–977.  
 961 doi: 10.1175/JCLI-D-14-00353.1
- 962 Haine, T. W. N., & Marshall, J. (1998). Gravitational, Symmetric, and Baroclinic  
 963 Instability of the Ocean Mixed Layer. *Journal of Physical Oceanography*,  
 964 28(4), 634–658. doi: 10.1175/1520-0485(1998)028<0634:GSABIO>2.0.CO;2
- 965 Hausfather, Z., Drake, H. F., Abbott, T., & Schmidt, G. A. (2020). Evaluating the  
 966 Performance of Past Climate Model Projections. *Geophysical Research Letters*,  
 967 47(1), e2019GL085378. doi: 10.1029/2019GL085378
- 968 Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q.,  
 969 ... Williamson, D. (2017). The Art and Science of Climate Model Tun-  
 970 ing. *Bulletin of the American Meteorological Society*, 98(3), 589–602. doi:  
 971 10.1175/BAMS-D-15-00135.1
- 972 Häfner, D., Jacobsen, R. L., Eden, C., Kristensen, M. R. B., Jochum, M., Nuterman,  
 973 R., & Vinter, B. (2018). Veros v0.1 – a fast and versatile ocean simulator  
 974 in pure Python. *Geoscientific Model Development*, 11(8), 3299–3312. doi:  
 975 10.5194/gmd-11-3299-2018

- 976 Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open*  
 977 *Source Software*, 3(25), 602. doi: 10.21105/joss.00602
- 978 Katz, R. W., Craigmire, P. F., Guttorp, P., Haran, M., Sansó, B., & Stein, M. L.  
 979 (2013). Uncertainty analysis in climate change assessments. *Nature Climate*  
 980 *Change*, 3(9), 769–771. doi: 10.1038/nclimate1980
- 981 Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv.  
 982 doi: 10.48550/ARXIV.1412.6980
- 983 Klein, A., Falkner, S., Bartels, S., Hennig, P., & Hutter, F. (2017). Fast Bayesian  
 984 Optimization of Machine Learning Hyperparameters on Large Datasets. In  
 985 A. Singh & J. Zhu (Eds.), *Proceedings of the 20th international conference on*  
 986 *artificial intelligence and statistics* (Vol. 54, pp. 528–536). PMLR. Retrieved  
 987 from <https://proceedings.mlr.press/v54/klein17a.html>
- 988 Klinger, B. A., Marshall, J., & Send, U. (1996). Representation of convective plumes  
 989 by vertical adjustment. *Journal of Geophysical Research: Oceans*, 101(C8),  
 990 18175–18182. doi: 10.1029/96JC00861
- 991 Large, W. G., McWilliams, J. C., & Doney, S. C. (1994). Oceanic vertical mixing: A  
 992 review and a model with a nonlocal boundary layer parameterization. *Reviews*  
 993 *of Geophysics*, 32(4), 363–403. doi: 10.1029/94RG01872
- 994 Large, W. G., Patton, E. G., & Sullivan, P. P. (2019). Nonlocal Transport and  
 995 Implied Viscosity and Diffusivity throughout the Boundary Layer in LES of  
 996 the Southern Ocean with Surface Waves. *Journal of Physical Oceanography*,  
 997 49(10), 2631–2652. doi: 10.1175/JPO-D-18-0202.1
- 998 Le, H., & Moin, P. (1991). An improvement of fractional step methods for the  
 999 incompressible Navier-Stokes equations. *Journal of Computational Physics*,  
 1000 92(2), 369–379. doi: 10.1016/0021-9991(91)90215-7
- 1001 Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A.,  
 1002 & Anandkumar, A. (2020). *Fourier neural operator for parametric partial*  
 1003 *differential equations*. arXiv. doi: 10.48550/ARXIV.2010.08895
- 1004 Marshall, J., & Plumb, R. A. (2007). *Atmosphere, Ocean and Climate Dynamics:*  
 1005 *An Introductory Text*. Academic Press.
- 1006 Marshall, J., & Schott, F. (1999). Open-ocean convection: Observations, theory, and  
 1007 models. *Reviews of Geophysics*, 37(1), 1–64. doi: 10.1029/98RG02739
- 1008 Mehta, P., Bukov, M., Wang, C.-H., Day, A. G., Richardson, C., Fisher, C. K.,

- 1009 & Schwab, D. J. (2019). A high-bias, low-variance introduction to Ma-  
 1010 chine Learning for physicists. *Physics Reports*, *810*, 1-124. doi: 10.1016/  
 1011 j.physrep.2019.03.001
- 1012 O’Gorman, P. A., & Dwyer, J. G. (2018). Using Machine Learning to Parameterize  
 1013 Moist Convection: Potential for Modeling of Climate, Climate Change, and  
 1014 Extreme Events. *Journal of Advances in Modeling Earth Systems*, *10*(10),  
 1015 2548–2563. doi: 10.1029/2018MS001351
- 1016 Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., ...  
 1017 Edelman, A. (2020). *Universal Differential Equations for Scientific Machine*  
 1018 *Learning*. arXiv. doi: 10.48550/ARXIV.2001.04385
- 1019 Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl – A Performant and  
 1020 Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of*  
 1021 *Open Research Software*, *5*(1), 15. doi: 10.5334/jors.151
- 1022 Ramadhan, A., Lee, X. K., & Piterbarg, U. (2022, September). *ali-*  
 1023 *ramadhan/neuralfreeconvection.jl: v1.0.0*. Zenodo. Retrieved from  
 1024 <https://doi.org/10.5281/zenodo.7108770> doi: 10.5281/zenodo.7108770
- 1025 Ramadhan, A., Wagner, G. L., Hill, C., Campin, J.-M., Churavy, V., Besard, T.,  
 1026 ... Marshall, J. (2020). Oceananigans.jl: Fast and friendly geophysical fluid  
 1027 dynamics on GPUs. *Journal of Open Source Software*, *5*(53), 2018. doi:  
 1028 10.21105/joss.02018
- 1029 Rasp, S., Pritchard, M. S., & Gentine, P. (2018). Deep learning to represent subgrid  
 1030 processes in climate models. *Proceedings of the National Academy of Sciences*,  
 1031 *115*(39), 9684–9689. doi: 10.1073/pnas.1810286115
- 1032 Resplandy, L., Keeling, R. F., Eddebbbar, Y., Brooks, M., Wang, R., Bopp, L., ...  
 1033 Oschlies, A. (2019). Quantification of ocean heat uptake from changes in  
 1034 atmospheric O<sub>2</sub> and CO<sub>2</sub> composition. *Scientific Reports*, *9*(1), 1–10. doi:  
 1035 10.1038/s41598-019-56490-z
- 1036 Roemmich, D., Johnson, G. C., Riser, S., Davis, R., Gilson, J., Owens, W. B., ...  
 1037 Ignaszewski, M. (2009). The argo program: Observing the global ocean with  
 1038 profiling floats. *Oceanography*, *22*(2), 34–43. doi: 10.5670/oceanog.2009.36
- 1039 Rozema, W., Bae, H. J., Moin, P., & Verstappen, R. (2015). Minimum-dissipation  
 1040 models for large-eddy simulation. *Physics of Fluids*, *27*(8), 085107. doi: 10  
 1041 .1063/1.4928700

- 1042 Schneider, T., Teixeira, J., Bretherton, C. S., Brient, F., Pressel, K. G., Schär, C.,  
 1043 & Siebesma, A. P. (2017). Climate goals and computing the future of clouds.  
 1044 *Nature Climate Change*, 7(1), 3–5. doi: 10.1038/nclimate3190
- 1045 Schumann, U., & Sweet, R. A. (1988). Fast Fourier Transforms for Direct Solution  
 1046 of Poisson’s Equation with Staggered Boundary Conditions. *Journal of Com-*  
 1047 *putational Physics*, 75(1), 123–137. doi: 10.1016/0021-9991(88)90102-7
- 1048 Shu, C.-W. (2009). High Order Weighted Essentially Nonoscillatory Schemes for  
 1049 Convection Dominated Problems. *SIAM Review*, 51, 82–126. doi: 10.1137/  
 1050 070679065
- 1051 Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimiza-  
 1052 tion of Machine Learning Algorithms. In F. Pereira, C. Burges, L. Bottou,  
 1053 & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*  
 1054 (Vol. 25). Curran Associates, Inc. Retrieved from [https://proceedings](https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf)  
 1055 [.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper](https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf)  
 1056 [.pdf](https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf)
- 1057 Souza, A. N., Wagner, G. L., Ramadhan, A., Allen, B., Churavy, V., Schloss, J., ...  
 1058 Ferrari, R. (2020). Uncertainty Quantification of Ocean Parameterizations:  
 1059 Application to the K-Profile-Parameterization for Penetrative Convection.  
 1060 *Journal of Advances in Modeling Earth Systems*, 12(12), e2020MS002108. doi:  
 1061 10.1029/2020MS002108
- 1062 Stevens, B., & Bony, S. (2013). What Are Climate Models Missing? *Science*,  
 1063 340(6136), 1053–1054. doi: 10.1126/science.1237554
- 1064 Van Roekel, L., Adcroft, A. J., Danabasoglu, G., Griffies, S. M., Kauffman, B.,  
 1065 Large, W., ... Schmidt, M. (2018). The KPP Boundary Layer Scheme for  
 1066 the Ocean: Revisiting Its Formulation and Benchmarking One-Dimensional  
 1067 Simulations Relative to LES. *Journal of Advances in Modeling Earth Systems*,  
 1068 10(11), 2647–2685. doi: 10.1029/2018MS001336
- 1069 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N.,  
 1070 ... Polosukhin, I. (2017). *Attention is all you need.* arXiv. doi:  
 1071 10.48550/ARXIV.1706.03762
- 1072 Verstappen, R. (2018). How much eddy dissipation is needed to counter-  
 1073 balance the nonlinear production of small, unresolved scales in a large-  
 1074 eddy simulation of turbulence? *Computers & Fluids*, 176, 276–284. doi:

1075 10.1016/j.compfluid.2016.12.016

1076 Vreugdenhil, C. A., & Taylor, J. R. (2018). Large-eddy simulations of stratified  
1077 plane Couette flow using the anisotropic minimum-dissipation model. *Physics*  
1078 *of Fluids*, *30*(8), 085104. doi: 10.1063/1.5037039

1079 Wang, C., Zhang, L., Lee, S.-K., Wu, L., & Mechoso, C. R. (2014). A global per-  
1080 spective on CMIP5 climate model biases. *Nature Climate Change*, *4*(3), 201–  
1081 205. doi: 10.1038/nclimate2118

1082 Wang, Y., Li, H.-X., Huang, T., & Li, L. (2014). Differential evolution based on co-  
1083 variance matrix learning and bimodal distribution parameter setting. *Applied*  
1084 *Soft Computing*, *18*, 232–247. doi: 10.1016/j.asoc.2014.01.038

1085 Williams, R. G., & Follows, M. J. (2011). *Ocean Dynamics and the Carbon Cy-*  
1086 *cle: Principles and Mechanisms*. Cambridge University Press. doi: 10.1017/  
1087 CBO9780511977817

1088 Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-i., & Jegelka, S. (2020).  
1089 *How Neural Networks Extrapolate: From Feedforward to Graph Neural Net-*  
1090 *works*. arXiv. doi: 10.48550/ARXIV.2009.11848

1091 Yuval, J., O’Gorman, P. A., & Hill, C. N. (2021). Use of Neural Networks for  
1092 Stable, Accurate and Physically Consistent Parameterization of Subgrid Atmo-  
1093 spheric Processes With Good Performance at Reduced Precision. *Geophysical*  
1094 *Research Letters*, *48*(6), e2020GL091363. doi: 10.1029/2020GL091363

1095 Zanna, L., & Bolton, T. (2020). Data-driven equation discovery of ocean mesoscale  
1096 closures. *Geophysical Research Letters*, *47*(17), e2020GL088376. doi: 10.1029/  
1097 2020GL088376