

# 1 High-level, high-resolution ocean modeling at all scales 2 with Oceananigans

3 Gregory L. Wagner<sup>1</sup>, Simone Silvestri<sup>1</sup>, Navid C. Constantinou<sup>2,3</sup>,  
4 Ali Ramadhan<sup>4</sup>, Jean-Michel Campin<sup>1</sup>, Chris Hill<sup>1</sup>, Tomás Chor<sup>5</sup>,  
5 Jago Strong-Wright<sup>6</sup>, Xin Kai Lee<sup>1</sup>, Francis Poulin<sup>7</sup>, Andre Souza<sup>1</sup>,  
6 Keaton J. Burns<sup>1,8</sup>, John Marshall<sup>1</sup>, and Raffaele Ferrari<sup>1</sup>

7 <sup>1</sup>Massachusetts Institute of Technology, Cambridge, MA, USA

8 <sup>2</sup>University of Melbourne, Parkville, VIC, Australia

9 <sup>3</sup>ARC Center of Excellence for the Weather of the 21st Century, Australia

10 <sup>4</sup>atdepth MRV, Cambridge, MA, USA

11 <sup>5</sup>University of Maryland, College Park, MD, USA

12 <sup>6</sup>University of Cambridge, Cambridge, United Kingdom

13 <sup>7</sup>University of Waterloo, Waterloo, ON, Canada

14 <sup>8</sup>Flatiron Institute, New York, NY, USA

## 15 Key Points:

- 16 Oceananigans provides a powerful interface for simulating oceanic motion at all scales  
with novel parameterizations and numerical methods.
- 17 Combining simple numerics with GPU-enabled high-resolution permits accurate simulations  
with accessible code.
- 18 High-level programmable interfaces are crucial to maximize both user and developer  
productivity.

22 **Abstract**

23 We describe the vision, user interface, governing equations, and numerical methods  
 24 that underpin new ocean modeling software called “Oceananigans”. Oceananigans is being  
 25 developed by the Climate Modeling Alliance as part of a larger project to build a trainable  
 26 climate model with quantifiable uncertainty. We argue that Oceananigans status as a popular,  
 27 capable modeling system realizes a vision for accelerating progress in Earth system modeling  
 28 that balances demands for model accuracy and performance, needed for state-of-the-art  
 29 science, against accessibility, which is needed to accelerate development. This vision combines  
 30 three cooperative elements: *(i)* a relatively simple finite volume algorithm *(ii)* optimized for  
 31 high-resolution simulations on GPUs which is *(iii)* exposed behind an expressive, high-level  
 32 user interface (using the Julia programming language in our case). We offer evidence for  
 33 the vision’s potential by illustrating the creative potential of our user interface, showcasing  
 34 Oceananigans physics with example simulations that range from simple classroom problems to  
 35 a realistic global ocean simulation spanning all scales of oceanic fluid motion, and describing  
 36 advances in parameterization, numerical methods, and computational efficiency.

37 **Plain Language Summary**

38 This paper introduces Oceananigans, a new software tool for simulating ocean currents  
 39 and fluid motion. Unlike most existing software for ocean modeling, Oceananigans is written  
 40 in Julia, a modern programming language that makes it easier to install, learn, and use.  
 41 Rather than relying on rigid configuration files, users write scripts, giving them more flexibility  
 42 and creative control over their simulations. Moreover, Oceananigans can simulate everything  
 43 from tiny millimeter-scale turbulence in a small box to planetary-scale ocean circulation.  
 44 It is also fast and efficient, taking advantage of graphics processing units (GPUs) to run  
 45 high-resolution simulations at speeds comparable to lower-resolution models in other software.  
 46 Our goal is not just to provide a tool for scientists. Our approach to combine simple numerics  
 47 on GPUs with a powerful user interface can accelerate the pace of model development, and  
 48 therefore accelerate the pace of scientific progress.

49 **1 Introduction**

50 Computation is fundamental to ocean and climate science, such that software is rate-  
 51 limiting for scientific progress. Since the first general circulation models ran on primitive  
 52 computers (Phillips, 1956; Bryan, 1969), advances in hardware, numerical methods, and the  
 53 approximate parameterization of otherwise unresolved processes have improved the fidelity  
 54 of ocean simulations (Griffies et al., 2015). Yet as technology advances, the gap between  
 55 potential and practice in ocean modeling is stagnant or widening, to the point that most  
 56 software today *(i)* can no longer use the world’s fastest computers, *(ii)* relies on outdated  
 57 user interfaces, and *(iii)* is still useful for only a limited subset of the wide variety of ocean  
 58 modeling problems.

59 This paper describes new ocean modeling software written in the Julia programming  
 60 language (Bezanson et al., 2017) called Oceananigans. Oceananigans is being developed  
 61 by the Climate Modeling Alliance (along with heroic external collaborators) as part of  
 62 a larger effort to develop a climate model automatically-calibrated to observations and  
 63 high resolution simulations, and with quantified uncertainty. Oceananigans development  
 64 is motivated primarily by the need for new capabilities. The most materially pressing  
 65 is the need to implement a hierarchical approach to climate model development (Held,  
 66 2005), wherein nonhydrostatic large eddy simulations are used to generate synthetic data  
 67 for calibrating parameterizations, followed by the refinement of parameters in a hydrostatic  
 68 global context against observations. Starting from scratch also allowed us to target GPUs  
 69 and CPUs and to lower the bar for future accelerator support, by leveraging the performance  
 70 portability offered by Julia’s KernelAbstractions (Churavy, 2024). Using GPUs reduces

the computational expense of ensemble calibration, enables higher resolution simulations, supports the next-generation of AI-based parameterizations, and makes ocean modeling cheaper and more accessible. Finally, and perhaps most important, we required a tool that was easy to use — not only for conducting creative science, but for quickly prototyping new parameterizations (Wagner, Hillier, et al., 2025), new numerical methods (Silvestri, Wagner, Campin, et al., 2024), and new algorithms for scaling simulations up to hundreds of GPUs (Silvestri, Wagner, Constantinou, et al., 2024). Our ultimate goal is to accelerate the *process* of model development and therefore, through a longer process of collective effort, accelerate progress in ocean and climate science.

### 1.1 From millimeters to millennia

The evolution of ocean circulation over millennia is controlled by turbulent mixing with scales that range down to millimeters. Two distinct systems have evolved to model and understand this huge range of oceanic motion: “GCMs” (general circulation models) for hydrostatic regional-to-global scale simulations, and simpler software for nonhydrostatic large eddy simulations (LESs) with meter-scale resolution that are high-fidelity but limited in duration and extent. Compared to LES, GCMs usually invoke more elaborate numerical methods and parameterizations to cope with the global ocean’s complex geometry and the more significant impacts of unresolved subgrid processes.

Oceananigans began as software for LES (Ramadhan et al., 2020), by perfecting an approach for hybrid hydrostatic/nonhydrostatic dynamical cores pioneered by MITgcm (Marshall, Adcroft, et al., 1997) for GPUs. Our nonhydrostatic LES algorithm was then adapted and optimized for a hydrostatic GCM (Silvestri, Wagner, Constantinou, et al., 2024). At the same time, we developed LES-inspired, minimally-dissipative numerical methods for turbulence-resolving simulations (Silvestri, Wagner, Campin, et al., 2024) that automatically adapt to changing resolution. The result is a computationally efficient modeling system suited to brute force, resolution-forced approach to accuracy for all scales of oceanic motion. Such a “LES the ocean” strategy is appealingly simple compared to alternatives relying on explicit dissipation, generalized vertical coordinates (Shchepetkin & McWilliams, 2005; Leclair & Madec, 2011; Petersen et al., 2015), Lagrangian vertical advection (Halliwell, 2004; Griffies et al., 2020), or unstructured horizontal grids (Ringler et al., 2013; Danilov et al., 2017; Korn et al., 2022). We hypothesize that “resolution everywhere” alleviates the need for unstructured targeted resolution and will reduce the spurious numerical mixing that pollutes the fidelity of lower-resolution simulations (Griffies et al., 2000), while yielding a plethora of additional improvements (Chassaignet & Xu, 2017, 2021; Kiss et al., 2020). At the same time, using simple algorithms preserves the accessibility of our source code and maximizes the benefits of the Julia programming language.

### 1.2 Why programmable interfaces matter

In 1984, Cox published the first description of generalizable ocean modeling software (Cox, 1984; Griffies et al., 2015). The “Cox model” is written in FORTRAN 77 and features a multi-step user interface for building new models: first, source code modifications are written to determine, for example, domain geometry and boundary conditions, emplaced into the “base code”, and compiled. Next, a text-based namelist file is used to determine parameters like the stop iteration, mixing coefficients, and solver convergence criteria. Cox (1984) provided three example model configurations to illustrate the user interface.

With forty years of progress in software engineering, numerical methods, and parameterization of unresolved processes, and more than a billion times more computational power, today’s ocean models bear little resemblance to the Cox model — *except* for their user interfaces. Current interfaces, though obviously more advanced than Cox’s, still impose multi-step workflows that invoke several programming paradigms. These multi-step workflows typically require the generation of input data using a separate scripting language, configuration of

121 numerous namelists, and source code modifications to change the model equations in ways  
 122 not accessible through a change of parameters.

123 One of our most important contributions is the development of a fundamentally different,  
 124 programmable user interface that provides a seamless workflow for numerical experiments  
 125 including setup, execution, analysis, and visualization using a single script. Programmable  
 126 interfaces written in scripting languages like Python and Julia are the interface of choice  
 127 and engine of progress in countless fields from visualization to machine learning, and their  
 128 benefits transfer to ocean modeling. A particularly inspiring example of a productive user  
 129 interface for computational fluid dynamics is provided by Dedalus (Burns et al., 2020), a  
 130 CPU-based spectral framework for solving partial differential equations in simple geometries.

131 A programmable interface shines for simple problems — but doesn't just help new users.  
 132 More importantly, this workflow accelerates the implementation of new numerical methods  
 133 and parameterizations by experienced developers. It facilitates writing and relentlessly  
 134 refactoring comprehensive test suites. It enables fast prototyping with tight implementation-  
 135 evaluation iterations. It makes it easier to collaborate by communicating concise but evocative  
 136 code snippets. It makes Oceananigans fun to use. Leveraging this programmable interface  
 137 together with the intrinsic productivity of the Julia programming language, Oceananigans  
 138 has progressed from a simple system for serial nonhydrostatic modeling (Ramadhan et  
 139 al., 2020) to parallelized software with capabilities at all scales up to global hydrostatic  
 140 simulations with breakthrough performance (Silvestri, Wagner, Constantinou, et al., 2024),  
 141 using innovative numerical methods (Silvestri, Wagner, Campin, et al., 2024) and new,  
 142 automatically-calibrated vertical mixing parameterizations (Wagner, Hillier, et al., 2025).  
 143 Users benefit too.

144 The Julia programming language, which is compiled and productive, has a lot to do  
 145 with the feasibility of our design. Unlike functions in pure Python, for example, Julia  
 146 functions implemented by users for forcing and boundary conditions can operate even in  
 147 high performance contexts on GPUs. Julia enables unique Oceananigans features, such as  
 148 interactivity, extensibility, automatic installation on any system, and portability to laptops  
 149 and GPUs through advanced Julia community tools (Besard et al., 2018; Churavy, 2024).  
 150 Oceananigans achieves breakthrough performance by using GPUs, but remains accessible to  
 151 students using personal laptops running Windows or Mac OS. Easy installation on personal  
 152 computers facilitates creative computation, since complex numerical experiments can be  
 153 prototyped productively in a comfortable personal environment before transferred to a high  
 154 performance environment for production runs.

155 Productive interfaces are only as powerful as the capability they expose. Oceananigans  
 156 combines a range of capabilities offered by other systems: a numerical design for modeling  
 157 across scales from MITgcm (Marshall, Adcroft, et al., 1997; Marshall, Hill, et al., 1997), a  
 158 simple and performant algorithm for LES from PALM and PyCLES (Pressel et al., 2015),  
 159 and GPU capabilities like Veros (Häfner et al., 2021), and scripting like Thetis (Kärnä et al.,  
 160 2018). Oceananigans assembles these diverse features behind an expressive programmable  
 161 interface.

### 162 1.3 Outline of this paper

163 This paper introduces the concepts that underpin Oceananigans' user interface and  
 164 illustrates how a productive user interface can be designed to harness wide-ranging capabilities  
 165 for high-resolution modeling of any scale of oceanic motion. Our aim is to evidence and  
 166 explain Oceananigans tripartite achievement: performance, flexibility, and friendliness at the  
 167 same time. We do not attempt to document the specifics of the user interface in detail or to  
 168 provide a comprehensive description of all features, however: for that we refer the reader to  
 169 Oceananigans documentation.

170       Section 2 begins by explicating the basic innovations of Oceananigans' programmable  
 171       interface using two classroom examples: two-dimensional turbulence, and a forced passive  
 172       tracer advected by two-dimensional turbulence. In section 3, we write down the governing  
 173       equations that underpin Oceananigans' nonhydrostatic and hydrostatic models. We build our  
 174       case for Oceananigans innovations by progressing from simple direct numerical simulations  
 175       of freshwater cabbeling and flow around a cylinder, to realistic tidally-forced large eddy  
 176       simulations over a headland, to a 1/12<sup>th</sup> degree eddying global ocean simulation.

177       Section 4 provides a primer to the finite volume spatial discretization that Oceananigans  
 178       uses to solve the nonhydrostatic and hydrostatic equations. This section establishes  
 179       Oceananigans' unique suitability for turbulence-resolving simulations that have minimal,  
 180       implicitly dissipative advection schemes based on Weighted Essentially Non-Oscillatory  
 181       (WENO) reconstruction. We conclude in section 6 by outlining future development work and  
 182       anticipating the next major innovations in ocean modeling which, we hope, will someday  
 183       render the present work obsolete.

## 184       2 Oceananigans, the library

185       Oceananigans is fundamentally a *library* of tools for building models by writing programs  
 186       called “scripts”. This departs from the usual framework wherein software provides pre-written  
 187       monolithic programs that are configured with parameters. For writing scripts, Oceananigans  
 188       syntax combines mathematical symbols with natural language. Our goal is to enable evocative  
 189       scripting that approaches the effectiveness of writing for communicating computational  
 190       science.

### 191       2.1 Hello, ocean

192       The way to learn new ocean modeling software is by building simulations with it. Our  
 193       first example in listing 1 sets up, runs, and visualizes a simulation of two-dimensional  
 194       turbulence. The 22 lines of listing 1 illustrate one of Oceananigans' main achievements: a  
 195       numerical experiment may be completely described by a single script. To execute the code in  
 196       listing 1, we need to copy into a file (call this, for example, `hello_ocean.jl`) and executed  
 197       by typing `julia hello_ocean.jl` at a terminal.

198       Oceananigans scripts organize into four sections. The first three define the “grid” “model”,  
 199       and “simulation”, and conclude with execution of the simulation. The fourth section, often  
 200       implemented separately for complex or expensive simulations, performs post-processing and  
 201       analysis. In listing 1, the grid defined on lines 4–7 determines the problem geometry, spatial  
 202       resolution, and machine architecture. To use a CPU instead of a GPU, one writes `CPU()` in  
 203       place of `GPU()` on line 5: no other changes to the script are required.

204       Lines 9–12 define the model, which solves the Navier–Stokes equations in two dimensions  
 205       with a 9th-order Weighted, Essentially Non-Oscillatory (WENO) advection scheme (see  
 206       section 4 for more information about WENO). The velocity components  $u, v$  are initialized  
 207       with uniformly distributed random numbers within  $[-1, 1]$ . The model definition can also  
 208       encompass forcing, boundary conditions, and the specification of additional terms in the  
 209       momentum and tracer equations such as Coriolis forces or turbulence closures.

210       Line 14 builds a `Simulation` with a time-step  $\Delta t = 0.01$  which will run until  $t = 10$   
 211       (Oceananigans does not assume dimensionality by default, so time is non-dimensional via  
 212       user input in this case). `Simulation` can be used to inject arbitrary user code into the  
 213       time-stepping loop in order to log simulation progress or write output to disk. Lines 17–19  
 214       analyze the final state of the simulation by computing vorticity, illustrating Oceananigans'  
 215       toolbox for building expression trees of discrete calculus and arithmetic operations. The same  
 216       tools may be used to define online diagnostics to be periodically computed and saved to disk  
 217       while the simulation runs. Line 22 concludes the numerical experiment with a visualization.  
 218       The result is shown in figure 1.

```

1 using Oceananigans
2
3 # The third dimension is "flattened" to reduce the domain from three to two dimensions.
4 topology = (Periodic, Periodic, Flat)
5 architecture = GPU() # CPU() works just fine too for this small example.
6 x = y = (0, 2π)
7 grid = RectilinearGrid(architecture; size=(256, 256), x, y, topology)
8
9 model = NonhydrostaticModel(; grid, advection=WENO(order=9))
10
11 ε(x, y) = 2rand() - 1 # Uniformly-distributed random numbers between [-1, 1].
12 set!(model, u=ε, v=ε)
13
14 simulation = Simulation(model; Δt=0.01, stop_time=10)
15 run!(simulation)
16
17 u, v, w = model.velocities
18 ζ = ∂x(v) - ∂y(u)
19
20 using CairoMakie
21 heatmap(ζ, colormap=:balance, axis=(; aspect=1))

```

Listing 1: A Julia script that uses Oceananigans and the Julia plotting library CairoMakie to set up, run, and visualize a simulation of two-dimensional turbulence on a Graphics Processing Unit (GPU). The initial velocity field, defined on lines 11-12, consists of random numbers uniformly-distributed between  $-1$  and  $1$ . The vorticity  $\zeta = \partial_x v - \partial_y u$  is defined on line 18. The solution is visualized in figure 1.

```

1 function circling_source(x, y, t)
2     δ, ω, r = 0.1, 2π/3, 2
3     dx = x + r * cos(ω * t)
4     dy = y + r * sin(ω * t)
5     return exp(-(dx^2 + dy^2) / 2δ^2)
6 end
7
8 forcing = (; c = circling_source)
9 model = NonhydrostaticModel(; grid, advection=WENO(order=9), tracers=:c, forcing)

```

Listing 2: Implementation of a moving source of passive tracer with a function in a two-dimensional turbulence simulation. These lines of code replace the model definition on line 9 in listing 1.

## 2.2 Incorporating user code

With a programmable interface and aided by Julia’s just-in-time compilation, user functions specifying domain geometry, forcing, boundary conditions, and initial conditions can be incorporated directly into models without a separate programming environment. To illustrate function-based forcing, we modify listing 1 with code that adds a passive tracer which is forced by a moving source that depends on  $x, y, t$ . A visualization of the vorticity and tracer field generated by listings 1 and 2 are shown in figure 1.

Users can also insert arbitrary functions for more general tasks into the time-stepping loop. This supports things as mundane as printing a summary of the current model status or writing output, to more exotic tasks like nudging state variables or updating a diffusion coefficient based on an externally-implemented model.

## 2.3 Abstractions for arithmetic and discrete calculus

Abstractions representing unary, binary, and calculus operators produce a system for building “lazy” expression trees, whose evaluation is delayed until their result is needed to be

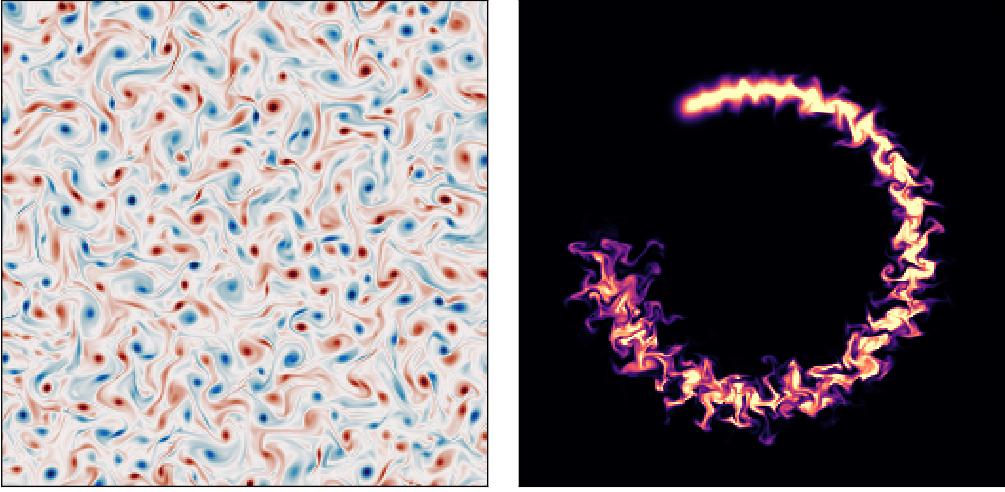


Figure 1: Vorticity after  $t = 10$  (left) and a passive tracer injected by a moving source at  $t = 2.5$  (right) in a simulation of two-dimensional turbulence using an implicitly-dissipative advection scheme.

saved to disk during a simulation. Example calculations representing vorticity,  $\zeta = \partial_x v - \partial_y u$ , speed  $s = \sqrt{u^2 + v^2}$ , and the  $x$ -integral of enstrophy  $Z = \int_0^{2\pi} \zeta^2 dx$  are shown in listing 3.

```

235
236 1 u, v, w = model.velocities
237 2
238 3 # Lazy expression trees and reductions representing computations:
239 4 ζ = ∂x(v) - ∂y(u)
240 5 s = √(u^2 + v^2)
241 6 Z = Integral(ζ^2, dims=1)
242
243

```

Listing 3: “Lazy” abstractions for expression trees and reductions — abstractions that *represent* computations to be performed at some future time as needed — support custom online diagnostics.

### 3 Governing equations and physical parameterizations

Oceananigans implements two “models” for ocean-flavored fluid dynamics: the HydrostaticFreeSurfaceModel, and the NonhydrostaticModel. Each represents a template for equations that govern the evolution of momentum and tracers. Both models are incompressible and make the Boussinesq approximation, which means that the density of the modeled fluid is decomposed into a constant reference  $\rho_0$  and a small dynamic perturbation  $\rho'$ ,

$$\rho(\mathbf{x}, t) = \rho_0 + \rho'(\mathbf{x}, t) \quad \text{where} \quad \rho' \ll \rho_0, \quad (1)$$

and  $\mathbf{x} = (x, y, z)$  is position and  $t$  is time.

The relative smallness of  $\rho'$  reduces conservation of mass to a statement of incompressibility called the continuity equation,

$$\nabla \cdot \mathbf{u} = 0, \quad (2)$$

where

$$\mathbf{u} \stackrel{\text{def}}{=} u \hat{\mathbf{x}} + v \hat{\mathbf{y}} + w \hat{\mathbf{z}}, \quad (3)$$

is the three-dimensional velocity field. Within the Boussinesq approximation, the momentum  $\rho_0 \mathbf{u}$  varies only with the velocity  $\mathbf{u}$ . The effect of density variations is encapsulated by a

257 buoyant acceleration,

$$b \stackrel{\text{def}}{=} -\frac{g\rho'}{\rho_0}, \quad (4)$$

258 where  $g$  is gravitational acceleration. The “buoyancy”  $b$  acts in the direction of gravity.

259 The total dynamic pressure  $P$  is decomposed into

$$P = \rho_0 g z + \rho_0 p(\mathbf{x}, t), \quad (5)$$

260 where  $\rho_0 g z$  is the static contribution to pressure that opposes the gravitational force associated  
261 with the reference density  $\rho_0$ , and  $\rho_0 p$  represents the dynamic anomaly.  $p$  is called the  
262 kinematic pressure.

### 263 3.1 The NonhydrostaticModel

264 The NonhydrostaticModel represents the Boussinesq equations formulated *without*  
265 making the hydrostatic approximation typical to general circulation models. The Nonhydro-  
266 staticModel has a three-dimensional prognostic velocity field.

#### 267 3.1.1 The NonhydrostaticModel momentum equation

268 The NonhydrostaticModel’s momentum equation incorporates advection by a background  
269 velocity field, Coriolis forces, surface wave effects via the Craik-Leibovich asymptotic model  
270 ([Craik & Leibovich, 1976](#); [Huang, 1979](#)), a buoyancy term allowed to be a nonlinear function  
271 of tracers and depth, a stress divergence derived from molecular friction or a turbulence  
272 closure, and a user-defined forcing term. Using the Boussinesq approximation in (1) and  
273 the pressure decomposition in (5), the generic form of NonhydrostaticModel’s momentum  
274 equation is

$$\begin{aligned} \partial_t \mathbf{u} = & -\underbrace{\nabla p - (\mathbf{u} \cdot \nabla) \mathbf{u} - (\mathbf{u}_g \cdot \nabla) \mathbf{u} - (\mathbf{u} \cdot \nabla) \mathbf{u}_g}_{\text{advection}} - \underbrace{\mathbf{f} \times \mathbf{u}}_{\text{Coriolis}} \\ & + \underbrace{(\nabla \times \mathbf{u}_s) \times \mathbf{u} + \partial_t \mathbf{u}_s}_{\text{Stokes drift}} - \underbrace{b \hat{\mathbf{g}}}_{\text{buoyancy}} - \underbrace{\nabla \cdot \boldsymbol{\tau}}_{\text{closure}} + \underbrace{\mathbf{F}_u}_{\text{forcing}}, \end{aligned} \quad (6)$$

275 where  $\mathbf{u}_g$  is a prescribed “background” velocity field,  $p$  is the kinematic pressure,  $\mathbf{f}$  is the  
276 background vorticity associated with a rotating frame of reference,  $\mathbf{u}_s$  is the Stokes drift  
277 profile associated with a prescribed surface wave field,  $b$  is buoyancy,  $\hat{\mathbf{g}}$  is the gravitational  
278 unit vector (usually pointing downwards, that is,  $\hat{\mathbf{g}} = -\hat{\mathbf{z}}$ ),  $\boldsymbol{\tau}$  is the stress tensor associated  
279 with molecular viscous or subgrid turbulent momentum transport, and  $\mathbf{F}_u$  is a body force.

280 To integrate equation (6) while enforcing (2), we use a pressure correction method  
281 that requires solving a three-dimensional Poisson equation to find  $p$ , which can be derived  
282 from  $\nabla \cdot (6)$ . This Poisson equation is often a computational bottleneck in curvilinear or  
283 irregular domains, and its elimination is the main motivation for making the hydrostatic  
284 approximation when formulating the HydrostaticFreeSurfaceModel, as described in section 3.2.  
285 For rectilinear grids, we solve the Poisson equation using a fast, direct, mixed FFT-tridiagonal  
286 solver ([Schumann & Sweet, 1988](#)), providing substantial acceleration over MITgcm’s conjugate  
287 gradient pressure solver ([Marshall, Adcroft, et al., 1997](#)). In irregular domains, we either use a  
288 masking method that permits an approximate solution of the pressure Poisson equation with  
289 the FFT-based method, or a rapidly-converging conjugate gradient iteration that leverages  
290 the FFT-based solver as a preconditioner. The pressure correction scheme is described  
291 further in appendix A2.

292 Using (2), advection in the NonhydrostaticModel is formulated in the “flux form”, which  
293 is conveniently expressed with indicial notation,

$$\text{advection} = u_j \partial_j u_i + u_{g,j} \partial_j u_i + u_j \partial_j u_{g,i} = \partial_j [(u_j + u_{g,j}) u_i + u_j u_{g,i}], \quad (7)$$

294 where, for example, the  $i$ -th component of the advection term is  $[(\mathbf{u} \cdot \nabla) \mathbf{u}]_i = u_j \partial_j u_i$ .

295 The formulation of the Stokes drift terms means that  $\mathbf{u}$  is the Lagrangian-mean velocity  
 296 when Stokes drift effects are included (see, for example, Wagner et al., 2021). With a  
 297 Lagrangian-mean formulation, equations (2) and (6) are consistent only when  $\mathbf{u}_s$  is non-  
 298 divergent — or equivalently, when  $\mathbf{u}_s$  is obtained by projecting the divergence out of the  
 299 usual Stokes drift (Vanneste & Young, 2022). As discussed by Wagner et al. (2021), the  
 300 Lagrangian-mean formulation of (6) means that closures for LES strictly destroy kinetic  
 301 energy, avoiding the inconsistency between resolved and subgrid fluxes affecting typical LES  
 302 formulated in terms of the Eulerian-mean velocity (see also Pearson, 2018).

303 The labeled terms in (6) are controlled by arguments to NonhydrostaticModel invoked  
 304 in both of listings 1 and 2. For example, “advection” chooses a numerical scheme to  
 305 approximate the advection term in (6) and (7). As another example, we consider configuring  
 306 the closure term in (6) to represent (i) molecular diffusion by a constant-coefficient Laplacian  
 307 ScalarDiffusivity, (ii) turbulent stresses approximated by the SmagorinskyLilly eddy viscosity  
 308 model (Smagorinsky, 1963; Lilly, 1983) for large eddy simulation, or (iii) omitting it entirely,  
 309 which we use with WENO advection schemes (and which is also our default setting). In  
 310 these three cases, the closure flux divergence  $\nabla \cdot \boldsymbol{\tau} = \partial_m \tau_{nm}$  in indicial notation becomes

$$-\partial_m \tau_{nm} = \begin{cases} \partial_m (\nu \partial_m u_n) & \text{(ScalarDiffusivity)} \\ 0 & \text{(nothing)} \\ \partial_m \left( 2 \underbrace{C_s \Delta^2 |\Sigma|}_{\nu_e} \Sigma_{nm} \right) & \text{(SmagorinskyLilly)} \end{cases} \quad (8)$$

311 where  $\nu$  is the Laplacian diffusion coefficient,  $\Sigma_{nm} = \partial_m u_n + \partial_n u_m$  is the strain rate tensor,  
 312  $|\Sigma|$  is the magnitude of the strain rate tensor,  $C_s$  is the SmagorinskyLilly model constant,  
 313  $\Delta$  scales with the local grid spacing, and  $\nu_e$  is the eddy viscosity. (ScalarDiffusivity diffusion  
 314 coefficients may also vary in time- and space. Other closure options include fourth-order  
 315 ScalarBiharmonicDiffusivity, various flavors of DynamicSmagorinsky (Bou-Zeid et al., 2005),  
 316 and the AnisotropicMinimumDissipation turbulence closure (Rozema et al., 2015; Vreugdenhil  
 317 & Taylor, 2018) for large eddy simulations.)

318 Listing 4 implements a direct numerical simulation of uniform flow past a cylinder  
 319 with no-slip boundary conditions, a molecular ScalarDiffusivity, and a centered second-  
 320 order advection scheme. Lines 6–7 embed a cylindrical mask in a RectilinearGrid using a  
 321 GridFittedBoundary, which generalizes to arbitrary three-dimensional shapes. The no-slip  
 322 condition is implemented with ValueBoundaryCondition (a synonym for “Dirichlet” boundary  
 323 conditions) on lines 11–12. Other choices include GradientBoundaryCondition (Neumann),  
 324 FluxBoundaryCondition (direct imposition of fluxes), and OpenBoundaryCondition (for  
 325 non-trivial boundary-normal velocity fields).

326 Results obtained with listing 4 for  $Re = 100$ ,  $Re = 1000$ , and a modified version of  
 327 listing 4 for large eddy simulation ( $Re \rightarrow \infty$ ) are visualized in figure 2. To adapt listing 4  
 328 for LES, the closure is eliminated in favor of a 9th-order WENO advection scheme, and the  
 329 no-slip boundary condition is replaced with a quadratic drag boundary condition with a drag  
 330 coefficient estimated from similarity theory using a constant estimated roughness length.

### 331 3.1.2 The NonhydrostaticModel tracer conservation equation

332 The buoyancy term in (6) requires tracers, and can be formulated to use buoyancy  
 333 itself as a tracer, or to depend on temperature  $T$  and salinity  $S$ . For seawater, a 54-  
 334 term polynomial approximation TEOS10EquationOfState (McDougall & Barker, 2011;  
 335 Roquet, Madec, McDougall, & Barker, 2015) is implemented in the auxiliary package  
 336 SeawaterPolynomials, along with quadratic approximations to TEOS-10 (Roquet, Madec,  
 337 Brodeau, & Nycander, 2015) and a LinearEquationOfState. All tracers — either “active”

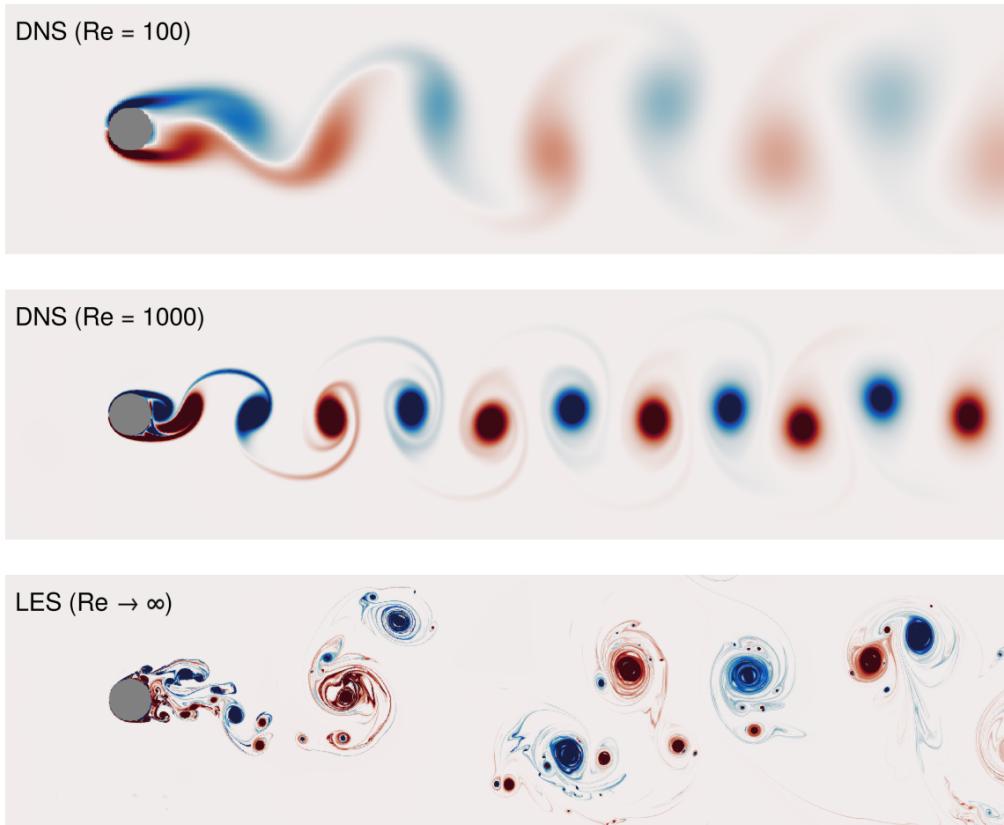


Figure 2: Vorticity snapshots in simulations of flow around a cylinder. The top two panels show vorticity in direct numerical simulations (DNS) that use a molecular ScalarDiffusivity closure and Centered(order=2) advection. The bottom panel shows a large eddy simulation (LES) with no closure and a WENO(order=9) advection scheme.

```

1 r, U, Re, Ny = 1/2, 1, 1000, 2048
2
3 grid = RectilinearGrid(GPU(), size=(2Ny, Ny), x=(-3, 21), y=(-6, 6),
4 topology=(Periodic, Bounded, Flat))
5
6 cylinder(x, y) = (x^2 + y^2) ≤ r^2
7 grid = ImmersedBoundaryGrid(grid, GridFittedBoundary(cylinder))
8
9 closure = ScalarDiffusivity(ν=1/Re)
10
11 no_slip = FieldBoundaryConditions(immersed=ValueBoundaryCondition(0))
12 boundary_conditions = (u=no_slip, v=no_slip)
13
14 # Implement a sponge layer on the right side of the domain that
15 # relaxes v → 0 and u → U over a region of thickness δ
16 @inline mask(x, y, δ=3, x₀=21) = max(zero(x), (x - x₀ + δ) / δ)
17 Fu = Relaxation(target=U; mask, rate=1)
18 Fv = Relaxation(target=0; mask, rate=1)
19
20 model = NonhydrostaticModel(; grid, closure, boundary_conditions, forcing=(u=Fu, v=Fv))

```

Listing 4: Direct numerical simulation of flow past a cylinder at various Reynolds numbers  $Re$ . The domain is periodic in  $x$  and a sponge layer on the right side of relaxes the solution to  $\mathbf{u} = u_\infty \hat{\mathbf{x}}$  with  $u_\infty = 1$ . The experiment can be converted to a large eddy simulation (thereby sending  $Re \rightarrow \infty$ ) by replacing the no-slip boundary conditions with an appropriate drag model and either (i) using an appropriate turbulence closure or (ii) using the WENO(order=9) advection scheme with no turbulence closure. Visualizations of the DNS and LES cases are shown in figure 2.

338 tracers required to compute the buoyancy term, as well as additional user-defined passive  
 339 tracers — obey the tracer conservation equation

$$\partial_t c = \underbrace{-(\mathbf{u} \cdot \nabla) c - (\mathbf{u}_g \cdot \nabla) c_g}_{\text{advection}} - \underbrace{\nabla \cdot \mathbf{J}_c}_{\text{closure}} + \underbrace{S_c}_{\text{biogeochemistry}} + \underbrace{F_c}_{\text{forcing}}, \quad (9)$$

340 where  $c$  represents any tracer,  $c_g$  represents a prescribed background tracer concentration  
 341 for  $c$ ,  $\mathbf{J}_c$  is a tracer flux associated with molecular diffusion or subgrid turbulence,  $S_c$  is a  
 342 source or sink term associated with biogeochemical transformations (provided, for example,  
 343 by external packages like OceanBioME; Strong-Wright et al., 2023), and  $F_c$  is a user-defined  
 344 source or sink.

345 A simulation with a passive tracer having a user-defined source term is illustrated by  
 346 listing 2 and figure 1. For a second example, we consider freshwater cabling. Cabling  
 347 occurs when two water masses of similar density mix to form a new water mass which,  
 348 due to the nonlinearity of the equation of state, is denser than either of its constituents.  
 349 Freshwater, for example, is densest at 4 degrees Celsius, while 1- and 7.55-degree water are  
 350 lighter with roughly the same density. We implement a direct numerical simulation in which  
 351 7.55-degree water overlies 1-degree water, using the TEOS10EquationOfState provided by  
 352 the auxiliary package SeawaterPolynomials. The script is shown in listing 5. The resulting  
 353 density and temperature fields after 1 minute of simulation are shown in figure 3. Note that  
 354 the TEOS10EquationOfState typically depends on both temperature and salinity tracers,  
 355 but listing 5 specifies a constant salinity  $S = 0$  and thus avoids allocating memory for or  
 356 simulating salinity directly.

```

357
358 1 grid = RectilinearGrid(GPU(), topology = (Bounded, Flat, Bounded),
359           size = (4096, 1024), x = (0, 2), z = (-0.5, 0))
360
361 2 closure = ScalarDiffusivity(ν=1.15e-6, κ=1e-7)
362
363 3
364 4 using SeawaterPolynomials: TEOS10EquationOfState
365 5
366 6 equation_of_state = TEOS10EquationOfState(reference_density=1000)

```

```

366   8
367   9 buoyancy = SeawaterBuoyancy(gravitational_acceleration = 9.81);
368   10           constant_salinity = 0, # set S=0 and simulate T only
369   11           equation_of_state)
370   12
371   13 model = NonhydrostaticModel(; grid, buoyancy, closure, tracers=:T)
372   14
373   15 T_i(x, z) = z > -0.25 ? 7.55 : 1
374   16 Ξ_i(x, z) = 1e-2 * randn()
375   17 set!(model, T=T_i, u=Ξ_i, v=Ξ_i, w=Ξ_i)
376

```

Listing 5: Direct numerical simulation of convective turbulence driven by cabbeling between 1- and 7.55-degree freshwater.  $\nu$  denotes viscosity and  $\kappa$  denotes the tracer diffusivity. The diffusivity may also be set independently for each tracer.

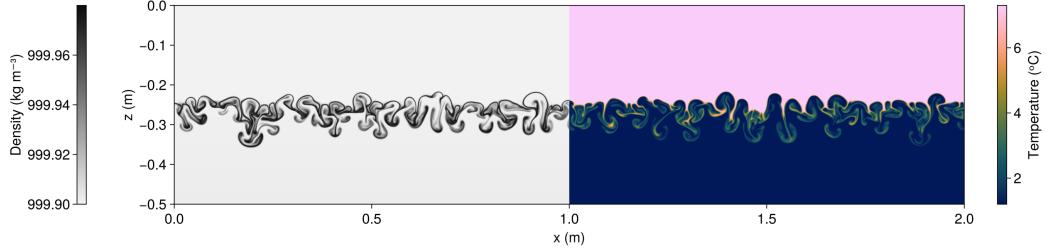


Figure 3: Density and temperature at  $t = 1$  minute in a direct numerical simulation of cabelling in freshwater. Note that both fields span from  $x = 0$  to  $x = 2$  meters; only the left half of the density field and the right half of the temperature field are shown.

We next consider a large eddy simulation of the Eady problem (Eady, 1949). In the Eady problem, perturbations evolve around a basic state with constant shear  $\Lambda$  in thermal wind balance with a constant meridional buoyancy gradient  $f\Lambda$ , such that

$$u = \underbrace{\Lambda z}_{\stackrel{\text{def}}{=} U} + u', \quad \text{and} \quad b = \underbrace{-f\Lambda y}_{\stackrel{\text{def}}{=} B} + b'. \quad (10)$$

We use Oceananigans' `BackgroundFields` to simulate the nonlinear evolution of  $(u', v, w)$  and  $b'$  expanded around  $U$  and  $B$  in a doubly-periodic domain. We impose an initially stable density stratification with  $b' = N^2 z$  and  $N^2 = 10^{-7} \text{ s}^{-2}$  superposed with random noise. The Richardson number of the initial condition is  $Ri = N^2 / \partial_z U = N^2 / \Lambda$ ; we choose mean shear  $\Lambda$  so that  $Ri = 1$ , which guarantees the basic state is unstable to baroclinic instability but stable to symmetric and Kelvin-Helmholtz instability (Stone, 1971). A portion of the script is shown in listing 6.

Our Eady simulation uses fully-turbulence-resolving resolution with 4 meter horizontal spacing and 2 meter vertical spacing in a  $4 \text{ km} \times 4 \text{ km} \times 128 \text{ m}$  domain and simulates 30 days on a single Nvidia H100 GPU. Four snapshots of vertical vorticity normalized by  $f$  (the Rossby number) are shown in figure 4, illustrating the growth of kilometer-scale vortex motions amid bursts of meter-scale three-dimensional turbulence that develop along thin filaments of vertical vorticity and vertical shear. This simple configuration captures a competition between baroclinic instability, which acts to “restratify” or strengthen boundary layer stratification, and three-dimensional turbulent mixing driven either by a forward cascade from kilometer-scale motions (Molemaker et al., 2010; Dong et al., 2024) or atmospheric storms (Boccaletti et al., 2007; Callies & Ferrari, 2018).

Finally, we illustrate Oceananigans' capabilities for realistic, three-dimensional large eddy simulations in complex geometries by simulating temperature- and salinity-stratified

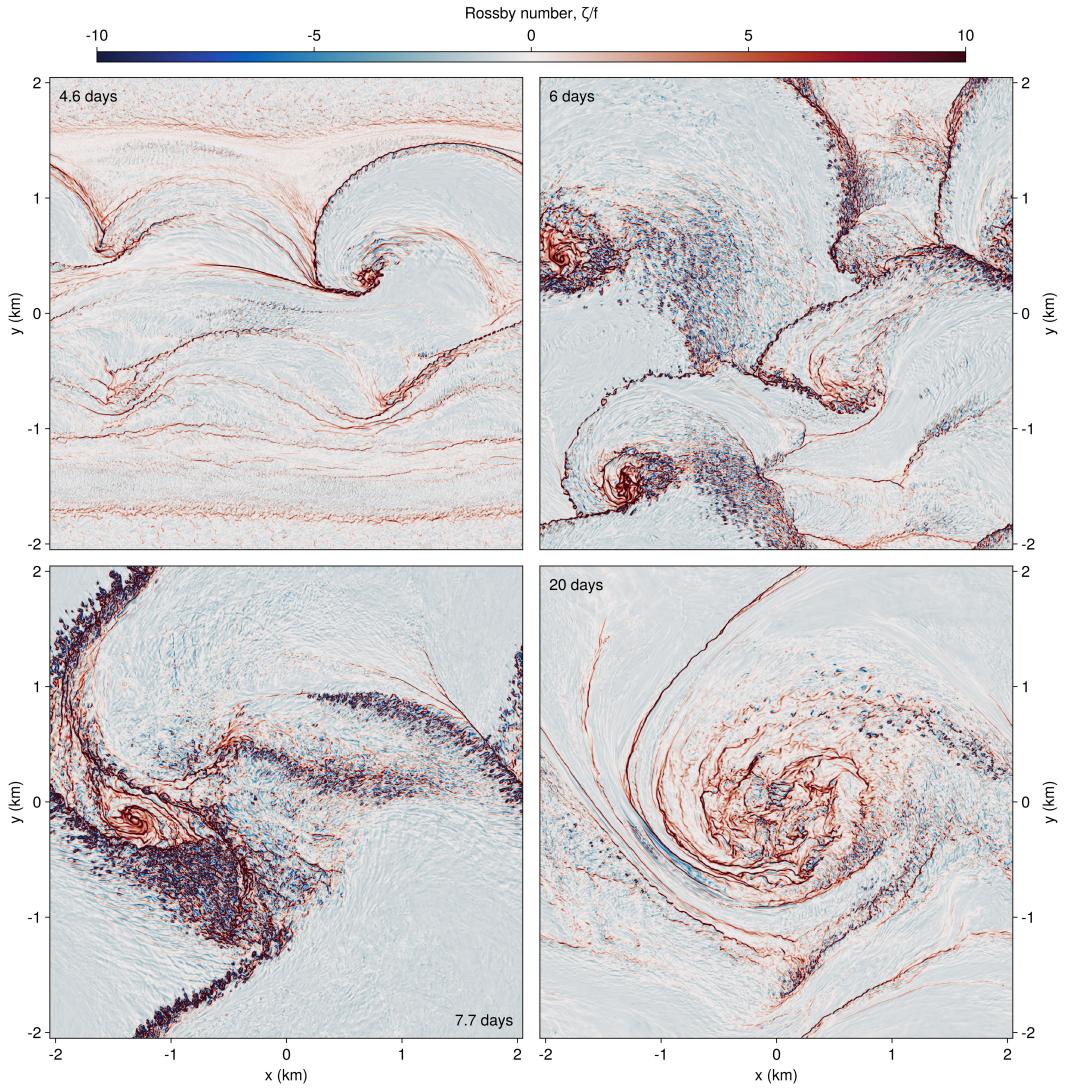


Figure 4: Surface vertical vorticity in a large eddy simulation of the Eady problem with  $Ri = 1$  initially, after  $t = 4.6, 6, 7.7$ , and  $20$  days. The grid spacing is  $4 \times 4 \times 2$  meters in  $x, y, z$ . Part of the script that produces this simulation is shown in listing 6.

```

1 grid = RectilinearGrid(GPU()); size = (1024, 1024, 64),
2           x = (0, 4096), y = (0, 4096), z = (0, 128),
3           topology=(Periodic, Periodic, Bounded)
4
5 f, N2, Ri = 1e-4, 1e-7, 1
6 parameters = (f=f, Λ=sqrt(N2/Ri)) # U = Λz, so Ri = N2 / ∂z(U) = N2 / Λ and Λ = N / √Ri.
7
8 @inline U(x, y, z, t, p) = + p.Λ * z
9 @inline B(x, y, z, t, p) = - p.f * p.Λ * y
10
11 background_fields = (u = BackgroundField(U; parameters),
12                         b = BackgroundField(B; parameters))
13
14 model = NonhydrostaticModel(; grid, background_fields,
15                               advection = WENO(order=9), coriolis = FPlane(; f),
16                               tracers = :b, buoyancy = BuoyancyTracer())
17
18 Δz = minimum_zspacing(grid)
19 bi(x, y, z) = N2 * z + 1e-2 * N2 * Δz * (2rand() - 1)
20 set!(model, b=bi)

```

Listing 6: Large eddy simulation of the Eady problem expanded around the background geostrophic shear with  $Ri = 1$ .

tidal flow past a headland, reminiscent of an extensively observed and modeled flow past Three Tree Point in Puget Sound in the Pacific Northwest of the United States (Pawlak et al., 2003; Warner & MacCready, 2014). The bathymetry involves a sloping wedge that juts from a square-sided channel, such that

$$z_b(x, y) = -H \left( 1 + \frac{y + |x|}{\delta} \right), \quad (11)$$

where  $\delta = L/2$  represents the scale of the bathymetry,  $L$  is the half-channel width in  $y$  (the total width is  $2L$ ), and  $H = 128$  m is the depth of the channel, and  $z = z_b(x, y)$  is the height of the bottom. The flow is driven by a tidally-oscillating boundary velocity

$$U(t) = U_2 \sin \left( \frac{2\pi t}{T_2} \right) \quad (12)$$

imposed at the east and west boundaries. Here,  $T_2 = 12.421$  hours is the period of the semi-diurnal lunar tide, and  $U_2 = 0.15$  m s<sup>-1</sup> is the characteristic tidal velocity around Three Tree Point. The initial temperature and salinity are

$$T|_{t=0} = 12 + 4 \frac{z}{H} {}^\circ\text{C}, \quad \text{and} \quad S|_{t=0} = 32 \text{ g kg}^{-1}. \quad (13)$$

A portion of the script that implements this simulation is shown in listing 7.

The oscillatory, turbulent flow is visualized in figure 5. The calculation of Ertel Potential Vorticity shown in figure 5c uses the companion package Oceanostics (Chor et al., 2025).

### 3.2 Hydrostatic model with a free surface

The HydrostaticFreeSurfaceModel solves the *hydrostatic*, rotating Boussinesq equations with a free surface. The hydrostatic approximation, inherent to the HydrostaticFreeSurfaceModel, means that the vertical momentum equation used by NonhydrostaticModel,  $\hat{z} \cdot (6)$ , is replaced by a statement of hydrostatic balance,

$$\partial_z p = b, \quad (14)$$

while the vertical velocity is obtained diagnostically from the continuity equation,

$$\partial_z w = -\nabla_h \cdot \mathbf{u}_h. \quad (15)$$

```

1 H, L = 256meters, 1024meters
2 δ = L / 2
3 x, y, z = (-3L, 3L), (-L, L), (-H, 0)
4 Nz = 64
5
6 grid = RectilinearGrid(GPU()); size=(6Nz, 2Nz, Nz), halo=(6, 6, 6),
7 x, y, z, topology=(Bounded, Bounded, Bounded))
8
9 wedge(x, y) = -H * (1 + (y + abs(x)) / δ)
10 grid = ImmersedBoundaryGrid(grid, GridFittedBottom(wedge))
11
12 T2 = 12.421hours
13 U2 = 0.1 # m/s
14
15 @inline Fu(x, y, z, t, p) = 2π * p.U2 / p.T2 * cos(2π * t / p.T2)
16 @inline U(x, Y, z, t, p) = p.U2 * sin(2π * t / p.T2)
17 @inline U(y, z, t, p) = U(zero(y), y, z, t, p)
18
19 open_bc = PerturbationAdvectionOpenBoundaryCondition(U; inflow_timescale = 2minutes,
20 outflow_timescale = 2minutes,
21 parameters=(; U2, T2))
22
23 u_bcs = FieldBoundaryConditions(east = open_bc, west = open_bc)
24
25 @inline ambient_temperature(x, z, t, H) = 12 + 4z/H
26 @inline ambient_temperature(x, y, z, t, H) = ambient_temperature(x, z, t, H)
27 ambient_temperature_bc = ValueBoundaryCondition(ambient_temperature; parameters = H)
28 T_bcs = FieldBoundaryConditions(east = ambient_temperature_bc,
29 west = ambient_temperature_bc)
30
31 ambient_salinity_bc = ValueBoundaryCondition(32)
32 S_bcs = FieldBoundaryConditions(east = ambient_salinity_bc, west = ambient_salinity_bc)
33
34 buoyancy = SeawaterBuoyancy(equation_of_state=TEOS10EquationOfState())
35
36 model = NonhydrostaticModel(; grid, buoyancy,
37 tracers = (:T, :S),
38 advection = WENO(order=9),
39 coriolis = FPlane(latitude=47.5),
40 boundary_conditions = (; T=T_bcs, u = u_bcs, S = S_bcs))
41
42 Ti(x, y, z) = ambient_temperature(x, y, z, 0, H)
43
44 set!(model, T=Ti, S=32, u=U(0, 0, 0, (; U2, T2)))

```

Listing 7: Large eddy simulation of flow past a headland reminiscent of Three Tree Point in the Pacific Northwest (see Pawlak et al., 2003; Warner & MacCready, 2014).

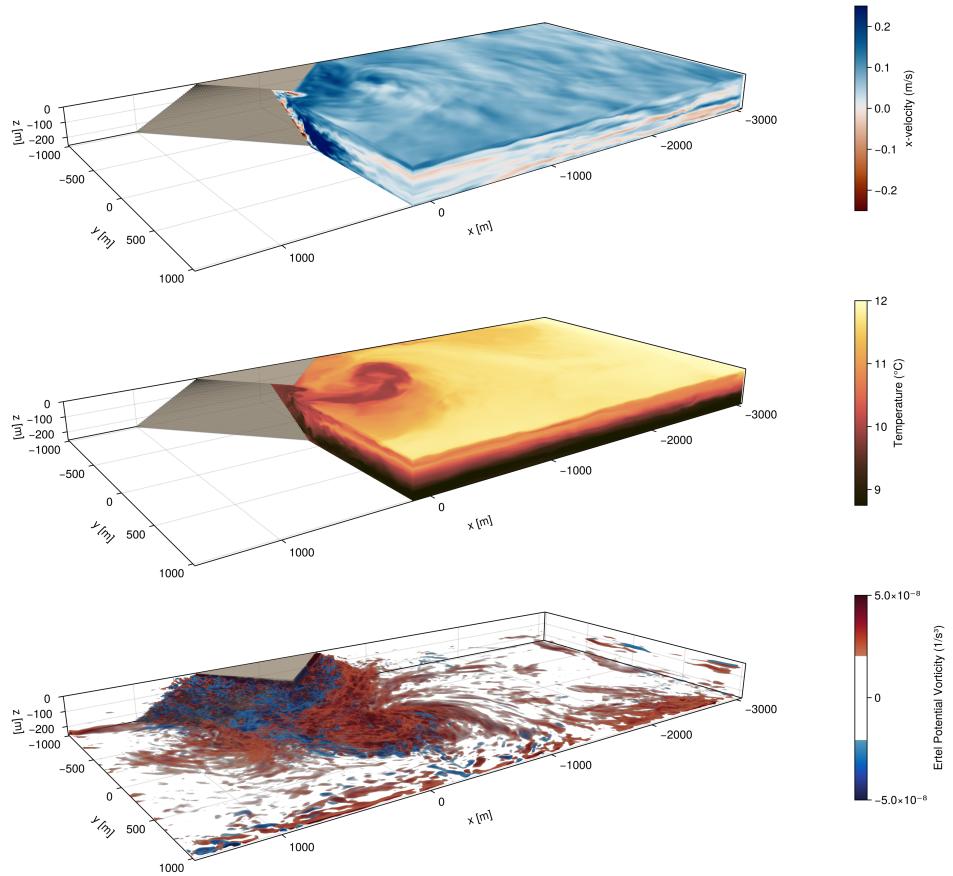


Figure 5: Along-channel velocity, temperature, and Ertel potential vorticity in a tidally-forced flow past an idealized headland with open boundaries. The tidal flow occurs in the  $x$ -directions and the snapshot depicts the flow just after the tide has turned to the negative- $x$  direction.

419 As a result, time-stepping the HydrostaticFreeSurfaceModel does require solving a three-  
 420 dimensional Poisson equation for pressure. Moreover, the HydrostaticFreeSurfaceModel  
 421 introduces a free surface displacement  $\eta$ , which obeys the linearized equation

$$\partial_t \eta = w|_{z=0}. \quad (16)$$

422 Equation (16) replaces the rigid-lid impenetrability condition  $w|_{z=0} = 0$  typically applied at  
 423 top boundaries in the NonhydrostaticModel. The numerical algorithms and computational  
 424 performance of the HydrostaticFreeSurfaceModel are described in more detail by Silvestri,  
 425 Wagner, Constantinou, et al. (2024).

426 In the HydrostaticFreeSurfaceModel, the horizontal momentum  $\mathbf{u}_h = u\hat{\mathbf{x}} + v\hat{\mathbf{y}}$  evolves  
 427 according to

$$\partial_t \mathbf{u}_h = -\nabla_h p - \underbrace{g\nabla_h \eta}_{\text{free surface}} - \underbrace{(\mathbf{u} \cdot \nabla) \mathbf{u}_h}_{\text{momentum advection}} - \underbrace{\mathbf{f} \times \mathbf{u}}_{\text{Coriolis}} - \underbrace{\nabla \cdot \boldsymbol{\tau}}_{\text{closure}} + \underbrace{\mathbf{F}_{uh}}_{\text{forcing}}, \quad (17)$$

428 where  $p$  is the hydrostatic kinematic pressure anomaly,  $\eta$  is the free surface displacement,  
 429  $\mathbf{u} = u\hat{\mathbf{x}} + v\hat{\mathbf{y}} + w\hat{\mathbf{z}}$  is the three-dimensional velocity,  $\mathbf{f}$  is the background vorticity associated  
 430 with a rotating frame of reference,  $\boldsymbol{\tau}$  is the stress associated with subgrid turbulent horizontal  
 431 momentum transport, and  $\mathbf{F}_{uh}$  is a body force. Momentum advection can be formulated in  
 432 three ways,

$$(\mathbf{u} \cdot \nabla) \cdot \mathbf{u}_h = \begin{cases} \nabla \cdot (\mathbf{u} \mathbf{u}_h) & \text{"flux form"}, \\ \zeta \hat{\mathbf{z}} \times \mathbf{u}_h + w \partial_z \mathbf{u}_h + \nabla_h \frac{1}{2} |\mathbf{u}_h|^2 & \text{VectorInvariant}, \\ \zeta \hat{\mathbf{z}} \times \mathbf{u}_h - \mathbf{u}_h \partial_z w + \partial_z (w \mathbf{u}_h) + \nabla_h \frac{1}{2} |\mathbf{u}_h|^2 & \text{WENOVectorInvariant}, \end{cases} \quad (18)$$

433 where the "flux form" treats momentum advection in the same way as for the NonhydrostaticModel.  
 434 The numerical implementation of the WENOVectorInvariant formulation, which  
 435 leverages Weighted Essentially Non-Oscillatory (WENO) reconstructions to selectively and  
 436 minimally dissipate enstrophy and the variance of divergence (see section 4), is described by  
 437 Silvestri, Wagner, Campin, et al. (2024).

438 Tracer evolution is governed by the conservation law

$$\partial_t c = - \underbrace{(\mathbf{u} \cdot \nabla) c}_{\text{tracer advection}} - \underbrace{\nabla \cdot \mathbf{J}_c}_{\text{closure}} + \underbrace{S_c}_{\text{biogeochemistry}} + \underbrace{\mathbf{F}_c}_{\text{forcing}}, \quad (19)$$

439 which is identical to NonhydrostaticModel except that background fields are not supported.  
 440 Additionally, the velocity field  $\mathbf{u}$  can be prescribed rather than evolved.

441 Listing 8 implements a simulation of tidally-forced stratified flow over a series of  
 442 randomly-positioned Gaussian seamounts. Results are plotted in figure 6.

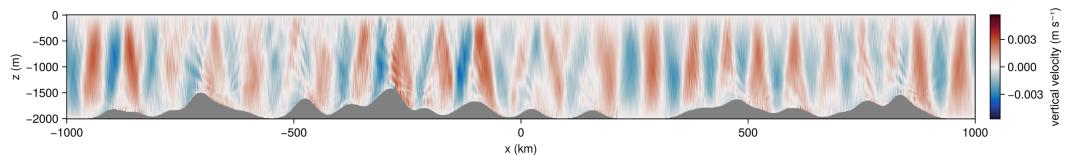


Figure 6: Vertical velocity of an internal wave field excited by tidally-forced stratified flow over superposition of randomly-positioned Gaussian seamounts, after 16 tidal periods.

```

1 using Oceananigans, Oceananigans.Units
2
3 grid = RectilinearGrid(size = (2000, 200),
4                         x = (-1000kilometers, 1000kilometers),
5                         z = (-2kilometers, 0),
6                         halo = (4, 4),
7                         topology = (Periodic, Flat, Bounded))
8
9 h₀ = 100           # typical mountain height (m)
10 δ = 20kilometers # mountain width (m)
11 seamounts = 42
12 W = grid.Lx - 4δ
13 x₀ = W .* (rand(seamounts) .- 1/2) # mountains' positions ∈ [-Lx/2+2δ, Lx/2-2δ]
14 h = h₀ .* (1 .+ rand(seamounts)) # mountains' heights ∈ [h₀, 2h₀]
15
16 bottom(x) = -grid.Lz + sum(h[s] * exp(-(x - x₀[s])^2 / 2δ^2) for s = 1:seamounts)
17 grid = ImmersedBoundaryGrid(grid, GridFittedBottom(bottom))
18
19 T₂ = 12.421hours # period of M₂ tide constituent
20 @inline tidal_forcing(x, z, t, p) = p.U₂ * 2π / p.T₂ * sin(2π / p.T₂ * t)
21 u_forcing = Forcing(tidal_forcing, parameters=(; U₂=0.1, T₂=T₂))
22
23 model = HydrostaticFreeSurfaceModel(; grid, tracers=:b, buoyancy=BuoyancyTracer(),
24                                       momentum_advection = WENO(),
25                                       tracer_advection = WENO(),
26                                       forcing = (; u = u_forcing))
27
28 bᵢ(x, z) = 1e-5 * z
29 set!(model, b=bᵢ)

```

Listing 8: Two-dimensional simulation of tidally-forced stratified flow over a superposition of randomly-positioned Gaussian seamounts. Results are shown in Figure 6.

### 3.2.1 Vertical mixing parameterizations

Oceananigans' vertical mixing parameterizations are closures that predict the vertical fluxes of tracers and momentum. Depending on the parameterization, the evolution of auxiliary tracers like turbulent kinetic energy and the turbulent kinetic energy dissipation rate may also be simulated. Vertical mixing parameterizations are useful for hydrostatic simulations where vertical mixing is otherwise unresolved due to a coarse horizontal grid spacing. For example, such regional and global configurations, horizontal grid spacing typically varies from  $O(100\text{ m})$  to  $O(100\text{ km})$ .

Listing 9 implements a simulation of wind-driven vertical mixing in a single column model using two parameterizations: CATKE (Wagner, Hillier, et al., 2025), which has one additional equation for the evolution of turbulent kinetic energy (TKE), and  $k$ - $\epsilon$  (Umlauf & Burchard, 2005), which has two additional equations for TKE and the TKE dissipation rate. Figure 7 plots the result, showing how  $k$ - $\epsilon$  undermixes compared to CATKE. This discrepancy in mixing rates is likely due to differences in how the models are calibrated. While all of CATKE's parameters are jointly calibrated to 35 large eddy simulations (LES) that include surface wave effects (Wagner, Hillier, et al., 2025),  $k$ - $\epsilon$  parameters are calibrated one-by-one by referencing laboratory experiments and observations of increasing complexity (Umlauf & Burchard, 2003). Calibrating  $k$ - $\epsilon$  parameters similarly to CATKE is an interesting direction for future work.

## 3.3 Global ocean simulations with ClimaOcean

The HydrostaticFreeSurfaceModel can be used to simulate regional or global ocean circulation on rectilinear grids, latitude-longitude grids, and the tripolar grid (Murray, 1996) to cover the entirety of Earth's global ocean. To illustrate global simulation with the HydrostaticFreeSurfaceModel, we implement a near-global simulation on a latitude-longitude

```

1 using Oceananigans
2 using Oceananigans.Units
3
4 function vertical_mixing_simulation(closure; N2=1e-5, Jb=1e-7, tx=-5e-4)
5   grid = RectilinearGrid(size=50, z=(-200, 0), topology=(Flat, Flat, Bounded))
6   buoyancy = BuoyancyTracer()
7
8   b_bcs = FieldBoundaryConditions(top=FluxBoundaryCondition(Jb))
9   u_bcs = FieldBoundaryConditions(top=FluxBoundaryCondition(tx))
10
11  if closure isa CATKEVerticalDiffusivity
12    tracers = (:b, :e)
13  elseif closure isa TKEDissipationVerticalDiffusivity
14    tracers = (:b, :e, :epsilon)
15  end
16
17  model = HydrostaticFreeSurfaceModel(; grid, closure, tracers, buoyancy,
18                                      boundary_conditions=(u=u_bcs, b=b_bcs))
19
20  b_i(z) = N2 * z
21  set!(model, b=b_i)
22
23  simulation = Simulation(model, Δt=1minute, stop_time=24hours)
24  return run!(simulation)
25 end

```

Listing 9: Comparison of two vertical mixing parameterizations in the evolution of an initially linearly stratified boundary layer subjected to stationary surface fluxes of buoyancy and momentum. Results are shown in Figure 7.

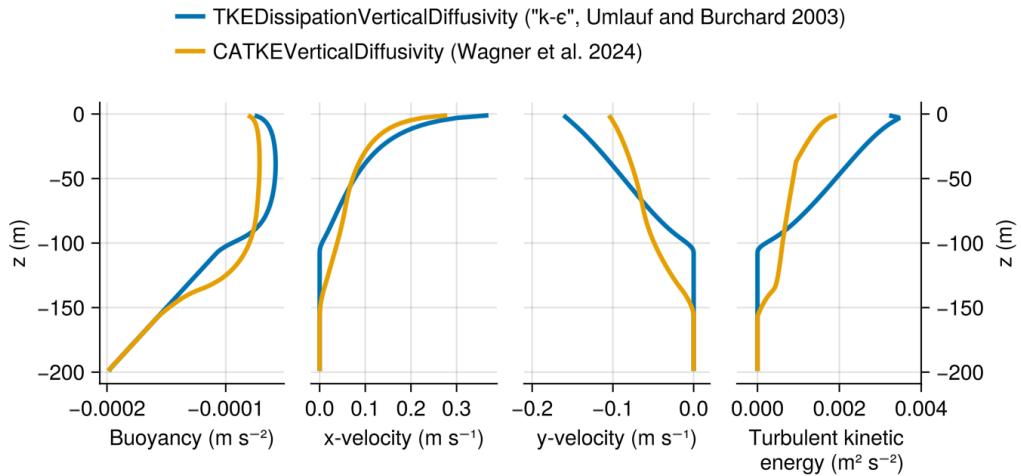


Figure 7: Results from two vertical mixing parameterizations: CATKE and  $k-\epsilon$ , implemented as described in Listing 9.

```

1 Nx, Ny, Nz = 4320, 1800, 40 # 1/12th degree
2 z_faces = ClimaOcean.exponential_z_faces(; Nz, depth=6000)
3 partition = Partition(8) # Distribute simulation across 8 GPUs
4 arch = Distributed(GPU()); partition
5 grid = LatitudeLongitudeGrid(arch; size=(Nx, Ny, Nz), halo=(7, 7, 7),
6                             longitude=(0, 360), latitude=(-75, 75), z=z_faces)
7
8 bathymetry = ClimaOcean.regrid_bathymetry(grid) # based on ETOPO1
9 grid = ImmersedBoundaryGrid(grid, GridFittedBottom(bathymetry))
10
11 # Build an ocean simulation initialized to the ECCO state estimate on Jan 1, 1993
12 ocean = ClimaOcean.ocean_simulation(grid)
13 date = CFTime.DateTimeProlepticGregorian(1993, 1, 1)
14 set!(ocean.model, T = ClimaOcean.ECCOMetadata(:temperature; date),
15       S = ClimaOcean.ECCOMetadata(:salinity; date))
16
17 # Near-global ocean simulation without no sea ice, forced by JRA55 reanalysis
18 backend = ClimaOcean.JRA55NetCDFBackend(41)
19 atmosphere = ClimaOcean.JRA55_prescribed_atmosphere(arch; backend)
20 coupled_model = ClimaOcean.OceanSeaIceModel(ocean; atmosphere)

```

Listing 10: A near-global simulation on a LatitudeLongitudeGrid distributed across 8 GPUs, leveraging ClimaOcean.

grid using ClimaOcean (Wagner, Silvestri, et al., 2025), which is a second package that provides capabilities to compute interfacial fluxes between a prescribed atmosphere, a sea ice model, and a hydrostatic ocean simulation implemented using Oceananigans. In ClimaOcean, turbulent interfacial fluxes are computed using Monin-Obhukov similarity theory (Monin, n.d.) following (Edson et al., 2014) for air-sea fluxes and (Grachev et al., 2007) for air-ice fluxes. ClimaOcean additionally provides utilities for downloading and interfacing with JRA55 reanalysis data (Tsujino et al., 2018), building grids based on Earth bathymetry and initializing simulations from the ECCO state estimate (Forget et al., 2015).

Part of a code that implements a near-global simulation with horizontal resolution of 1/12th degree, distributed over 8 GPUs, forced by JRA55 reanalysis and initialized from the ECCO state estimate is shown in listing 10. The surface speed generated after 180 days of simulation time is shown in figure 8. For more information about Oceananigans GPU performance in global configurations, see Silvestri, Wagner, Constantinou, et al. (2024).

## 4 Finite volume spatial discretization

Oceananigans uses a finite volume method in which fields are represented discretely by their average value over small local regions or “finite volumes” of the domain. Listing 11 discretizes  $c = e^x y$  on three different grids that cover the unit square. At the finest resolution, each cell-averaged value  $c_{ij}^{\text{fine}}$  is computed approximately using `set!` to evaluate  $e^x y$  at the center of each finite volume, where  $i, j$  denote the  $x$  and  $y$  indices of the finite volumes. At medium and coarse resolution, the  $c_{ij}^{\text{medium}}$  and  $c_{ij}^{\text{coarse}}$  are computed by averaging or “regridding” fields discretized at a higher resolution. This computation produces three fields with identical integrals over the unit square. For example, integrals are computed exactly by summing discrete fields over all cells,

$$\int c \, dx \, dy = \sum_{i,j}^{1024,1024} \mathcal{V}_{ij}^{\text{fine}} c_{ij}^{\text{fine}} = \sum_{i,j}^{16,16} \mathcal{V}_{ij}^{\text{medium}} c_{ij}^{\text{medium}} = \sum_{i,j}^{4,4} \mathcal{V}_{ij}^{\text{coarse}} c_{ij}^{\text{coarse}}, \quad (20)$$

where  $\mathcal{V}_{ij}$  is the “volume” of the cell with indices  $i, j$  (more accurately an “area” in this two-dimensional situation). Figure 9 visualizes the three fields.

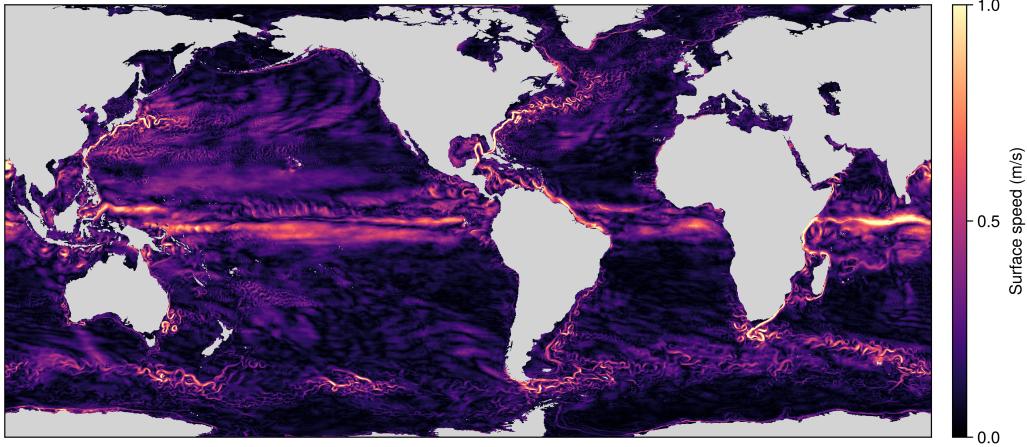


Figure 8: Surface speed in a near-global ocean simulation at 1/12th degree forced by JRA55 atmospheric reanalysis (Tsujino et al., 2018) initialized from the ECCO state estimate (Forget et al., 2015). Oceananigans can also cover the entirety of Earth’s global ocean using a tripolar grid (Murray, 1996).

```

1 topology = (Bounded, Bounded, Flat)
2 x = y = (0, 1)
3 c(x, y) = exp(x) * y
4
5 fine_grid = RectilinearGrid(size=(1024, 1024); x, y, topology)
6 c_fine = CenterField(fine_grid)
7 set!(c_fine, c)
8
9 medium_grid = RectilinearGrid(size=(16, 16); x, y, topology)
10 c_medium = CenterField(medium_grid)
11 regrid!(c_medium, c_fine)
12
13 coarse_grid = RectilinearGrid(size=(4, 4); x, y, topology)
14 c_coarse = CenterField(coarse_grid)
15 regrid!(c_coarse, c_medium)

```

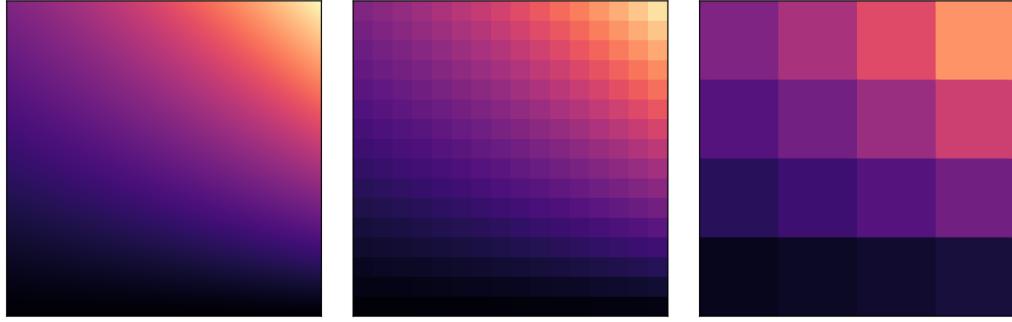
Listing 11: Finite volume discretization of  $e^x y$  on three grids over the unit square. The fields are visualized in figure 9. The meaning of the “Center” in “CenterField” is discussed below.

The discrete calculus and arithmetic operations required to solve the governing equations of the NonhydrostaticModel and HydrostaticFreeSurfaceModel use the system of “staggered grids” described by Arakawa (1977). Both models use “C-grid” staggering, where cells for tracers, pressure, and the divergence of the velocity field  $\nabla \cdot \mathbf{u}$  are co-located, and cells for velocity components  $\mathbf{u} = (u, v, w)$  are staggered by half a cell width in the  $x$ -,  $y$ -, and  $z$ -direction, respectively. Listing 12 illustrates grid construction and notation for a one-dimensional staggered grid with unevenly-spaced cells. Figure 10 visualizes 2- and 3-dimensional staggered grids, indicating the location of certain variables.

```

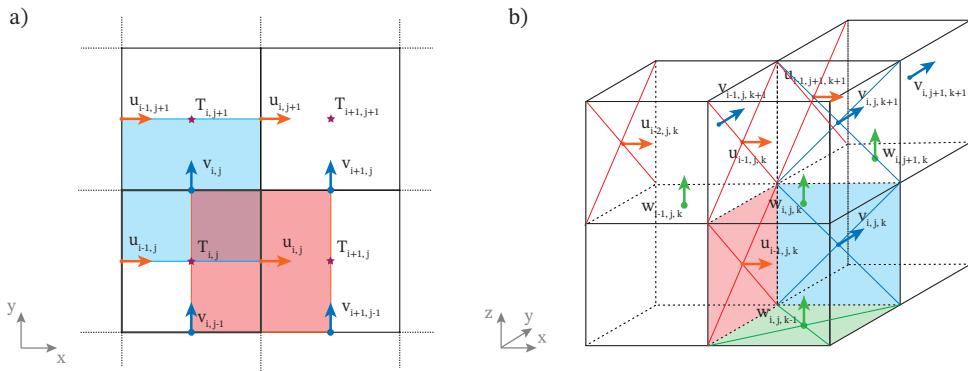
500
501 1 using Oceananigans
502
503 2
504 3 grid = RectilinearGrid(topology=(Bounded, Flat, Flat), size=4, x=[0, 0.2, 0.3, 0.7, 1])
505
506 4
507 5 u = Field(Face, Center, Center)(grid)
508 6 c = Field(Center, Center, Center)(grid)
509
510 7
511 8 xnodes(u)      # [0.0, 0.2, 0.3, 0.7, 1.0]
512 9 xnodes(c)      # [0.1, 0.25, 0.5, 0.85]
513 10 location(∂x(c)) # (Face, Center, Center)

```

Figure 9: Finite volume discretization of  $e^x y$  on the unit square at three different resolutions.

513

Listing 12: A one-dimensional staggered grid.

Figure 10: Locations of cell centers and interfaces on a two-dimensional (a) and three-dimensional (b) staggered grid. In (a), the red and blue shaded regions highlight the volumes in the dual  $u$ -grid and  $v$ -grid, located at (Face, Center, Center) and (Center, Face, Center), respectively. In (b), the shaded regions highlight the facial areas used in the fluxes computations, denoted with  $\mathcal{A}_x$ ,  $\mathcal{A}_y$ , and  $\mathcal{A}_z$ .

514

#### 4.1 A system of composable operators

515  
516  
517  
518

A convention for indexing is associated with staggered locations. Face indices are “left” of cell indices. This means that difference operators acting on fields at cells differ from those that act on face fields. To illustrate this we introduce Oceananigans-like difference operators,

519  
520  
521  
522  
523

```
1 δxfcc(i, j, k, grid, c) = c[i, j, k] - c[i-1, j, k]
2 δxccc(i, j, k, grid, u) = u[i+1, j, k] - u[i, j, k]
```

525  
526

where superscripts denote the location of the *result* of the operation. For example, the difference  $\delta_x^{fcc}$  acts on fields located at ccc (meaning cell Center in the  $x$ ,  $y$  and  $z$  directions

527 respectively). Complementary to the difference operators are reconstruction of “interpolation”  
 528 operators,

```
529
530 1 Sxfcc(i, j, k, grid, c) = (c[i, j, k] + c[i-1, j, k]) / 2
531 2 Sxccc(i, j, k, grid, u) = (u[i+1, j, k] + u[i, j, k]) / 2
532
533
```

535 The prefix arguments  $i, j, k, \text{grid}$  are more than convention: the prefix enables  
 536 system for *composing* operators. For example, defining

```
537
538 1 delta_xfcc(i, j, k, grid, f::Function, args...) =
539     f(i, j, k, grid, args...) - f(i-1, j, k, grid, args...)
540
541 2 delta_xccc(i, j, k, grid, f::Function, args...) =
542     f(i+1, j, k, grid, args...) - f(i, j, k, grid, args...)
543
544
```

546 leads to a concise definition of the second-difference operator:

```
547
548 1 delta2xccc(i, j, k, grid, f::Function, a...) = delta_xccc(i, j, k, grid, delta_xfcc, f, a...)
```

552 Operator composition is used throughout Oceananigans source code to implement stencil  
 553 operations.

## 554 4.2 Tracer flux divergences, advection schemes, and reconstruction

555 The divergence of a tracer flux  $\mathbf{J} = J_x \hat{x} + J_y \hat{y} + J_z \hat{z}$  is discretized conservatively by  
 556 the finite volume method via

$$\nabla \cdot \mathbf{J} \approx \frac{1}{V_c} \left[ \delta_x \left( \underbrace{\mathcal{A}_x J_x}_{\text{fcc}} \right) + \delta_y \left( \underbrace{\mathcal{A}_y J_y}_{\text{ccf}} \right) + \delta_z \left( \underbrace{\mathcal{A}_z J_z}_{\text{ccf}} \right) \right], \quad (21)$$

557 where  $\delta_x, \delta_y, \delta_z$  are difference operators in  $x, y, z$ ,  $V_c$  denotes the volume of the tracer cells,  
 558  $\mathcal{A}_x, \mathcal{A}_y$ , and  $\mathcal{A}_z$  denote the areas of the tracer cell faces with surface normals  $\hat{x}, \hat{y}$ , and  $\hat{z}$ ,  
 559 respectively. Equation (21) indicates the location of each flux component: fluxes into tracers  
 560 cell at ccc are computed at the cell faces located at fcc, cfc, and ccf.

561 The advective tracer flux in (9) is written in “conservative form” using incompressibility  
 562 (2), and then discretized similarly to (21) to form

$$\mathbf{u} \cdot \nabla c = \nabla \cdot (\mathbf{u}c) \approx \frac{1}{V_c} \left[ \delta_x (\mathcal{A}_x u [c]_x) + \delta_y (\mathcal{A}_y v [c]_y) + \delta_z (\mathcal{A}_z w [c]_z) \right], \quad (22)$$

563 where  $[c]_x$  denotes a *reconstruction* of  $c$  in the  $x$ -direction from its native location ccc to  
 564 the tracer cell interface at fcc;  $[c]_y$  and  $[c]_z$  in (22) are defined similarly.

565 The advective fluxes  $\mathbf{u}c$  must be computed on interfaces between tracer cells, where  
 566 the approximate value of  $c$  must be reconstructed. (Velocity components like  $u$  must also  
 567 be reconstructed on interfaces. Within the C-grid framework, we approximate  $u$  on tracer  
 568 cell interfaces directly using the values  $u_{ijk}$ , which represent  $u$  averaged over a region  
 569 encompassing the interface.) The simplest kind of reconstruction is Centered(order=2),  
 570 which is equivalent to taking the average between adjacent cells,

$$\langle c \rangle_i = \frac{1}{2} (c_i + c_{i-1}), \quad (23)$$

571 where  $\langle c \rangle_i$  denotes the centered reconstruction of  $c$  on the interface at  $x = x_{i-1/2}$ . Also  
 572 in (23) the  $j, k$  indices are implied and we have suppressed the direction  $x$  to lighten the  
 573 notation. Reconstructions stencils for Center(order=N) are automatically generated for

even  $N$  up to  $N_{\max} = 12$ , where  $N_{\max}$  is an adjustable parameter in the source code. All subsequent reconstructions are described in the  $x$ -direction only.

Centered schemes should be used when explicit dissipation justified by a *physical* rationale dominates at the grid scale. In scenarios where dissipation is needed solely for artificial reasons, we find applications for UpwindBiased schemes, which use an odd-order stencil biased against the direction of flow. For example, UpwindBiased(order=1) and UpwindBiased(order=3) schemes are written

$$u[c]_x^1 = \begin{cases} u c_{i-1} & \text{if } u > 0, \\ u c_i & \text{if } u < 0, \end{cases} \quad \text{and} \quad u[c]_x^3 = \begin{cases} u \frac{1}{6} (-c_{i-2} + 5c_{i-1} + 2c_i) & \text{if } u > 0, \\ u \frac{1}{6} (2c_{i-1} + 5c_i - c_{i+1}) & \text{if } u < 0, \end{cases} \quad (24)$$

where  $[c]_x^N$  denotes  $N^{\text{th}}$ -order upwind reconstruction in the  $x$ -direction. (Note that  $u[c]_x^N = 0$  if  $u = 0$ .)

The compact form of equations (24) demonstrates how upwind schemes introduce variance dissipation through numerical discretization. In particular, an UpwindBiased(order=1) reconstruction can be rewritten as a sum of a Centered(order=2) discrete advective flux and a discrete diffusive flux

$$u[c]_x^1 = u \frac{c_i + c_{i-1}}{2} - \kappa_1 \frac{c_i - c_{i-1}}{\Delta x}, \quad \text{where} \quad \kappa_1 = \frac{|u|\Delta x}{2}. \quad (25)$$

Reordering the UpwindBiased(order=3) advective flux in the same manner recovers a sum of a Centered(order=4) advective flux and a 4th-order hyperdiffusive flux, equivalent to a finite volume approximation of

$$uc + \kappa_3 \frac{\partial^3 c}{\partial x^3}, \quad \text{where} \quad \kappa_3 = \frac{|u|\Delta x^3}{12}. \quad (26)$$

UpwindBiased reconstruction can be always reordered to expose a sum of Centered reconstruction and a high-order diffusive flux with a velocity-dependent diffusivity. The diffusive operator associated with UpwindBiased(order=1) and UpwindBiased(order=3) is enough to offset the dispersive errors of the Centered component and, therefore, eliminate the artificial explicit diffusion needed for stability purposes. However, this approach does not scale to high order since the diffusive operator associated with a high order UpwindBiased scheme (5th, 7th, and so on), becomes quickly insufficient to eliminate spurious errors associated with the Centered component (Godunov, 1959).

The inability to achieve high order and, therefore, low dissipation motivated the implementation of Weighted, Essentially Non-Oscillatory (WENO) reconstruction (C. Shu, 1997; C.-W. Shu, 2009). WENO is a non-linear reconstruction scheme that combines a set of odd-order linear reconstructions obtained by stencils that are shifted by a value  $s$  relative to the canonical UpwindBiased stencil, using a weighting scheme for each stencil that depends on the smoothness of the reconstructed field  $c$ . Since the constituent stencils are lower-order than the WENO order, this strategy yields a scheme whose order of accuracy adapts depending on the smoothness of the reconstructed field. In smooth regions high-order is retained, while the order quickly decreases in the presence of noisy regions, decreasing the order of the associated diffusive operator. WENO proves especially useful for high-resolution, turbulence-resolving simulations (either at meter or planetary scales) without requiring any additional explicit artificial dissipation (Pressel et al., 2017; Silvestri, Wagner, Campin, et al., 2024).

To illustrate how WENO works we consider a fifth-order WENO scheme for  $u > 0$ ,

$$\{c\}^5 = \gamma_0 [c]^{3,0} + \gamma_1 [c]^{3,1} + \gamma_2 [c]^{3,2}, \quad (27)$$

612 where the notation  $[c]^{3,s}$  denotes an UpwindBiased stencil *shifted* by  $s$  indices, such that  
 613  $[c]^3 \stackrel{\text{def}}{=} [c]^{3,0}$ . The shifted upwind stencils  $[c]_i^{N,s}$  evaluated at index  $i$  are defined

$$[c]_i^{3,s} = \frac{1}{6} \begin{cases} -c_{i-1} + 5c_i + 2c_{i+1} & \text{for } s = -1, \\ 2c_{i-2} + 5c_{i-1} - c_i & \text{for } s = 0, \\ 2c_{i-3} - 7c_{i-2} + 11c_{i-1} & \text{for } s = 2. \end{cases} \quad (28)$$

614 The weights  $\gamma_s(c)$  are determined by a smoothness metric that produces  $\{c\}^5 \approx [c]^5$  when  $c$  is  
 615 smooth, but limits to the more diffusive  $\{c\}^5 \approx [c]^3$  when  $c$  changes abruptly. Thus WENO  
 616 adaptively introduces dissipation as needed based on the smoothness of  $c$ , yielding stable  
 617 simulations with a high effective resolution that require no artificial dissipation. WENO can  
 618 alternatively be interpreted as adding an implicit hyperviscosity that adapts from low- to  
 619 high-order depending on the local nature of the solution. To compute the weights  $\gamma_s(c)$ , we  
 620 use the WENO-Z formulation (Balsara & Shu, 2000).

621 The properties of Centered, UpwindBiased, and WENO reconstruction are investigated  
 622 by listing 13, which simulates the advection of a top hat tracer distribution. The results are  
 623 plotted in figure 11.

```

624
625 1 using Oceananigans
626 2
627 3 grid = RectilinearGrid(size=128; x=(-4, 8), halo=6, topology=(Periodic, Flat, Flat))
628 4 advection = WENO(order=9) # Centered(order=2), UpwindBiased(order=3)
629 5 velocities = PrescribedVelocityFields(u=1)
630 6 model = HydrostaticFreeSurfaceModel(; grid, velocities, advection, tracers=:c)
631 7
632 8 top_hat(x) = abs(x) > 1 ? 0 : 1
633 9 set!(model, c = top_hat)
634 10
635 11 simulation = Simulation(model, Δt=1/grid.Nx, stop_time=4)
636 12 run!(simulation)
637
638
639
```

Listing 13: A script that advects a top hat tracer profile in one-dimension with a constant prescribed velocity.  
 We use halo=6 to accommodate schemes up to WENO(order=11).

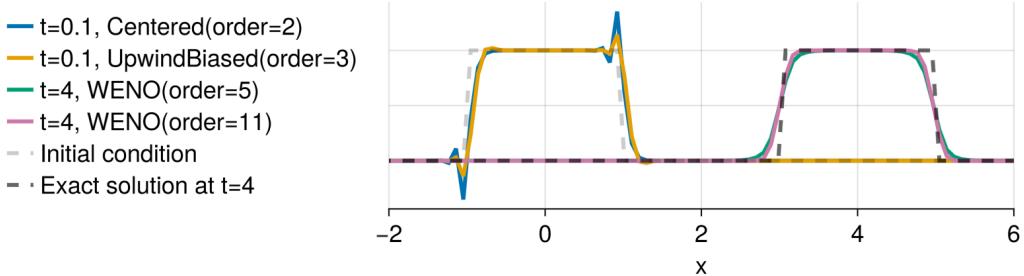


Figure 11: Advection of a top hat tracer distribution in one-dimension using various advection schemes. Centered and Upwind

#### 640 4.2.1 Discretization of momentum advection

641 The discretization of momentum advection with a flux form similar to (22) is more  
 642 complex than the tracer case because both the advecting velocity and advected velocity  
 643 require reconstruction. We use the method described by Ghosh and Baeder (2012) and Pressel

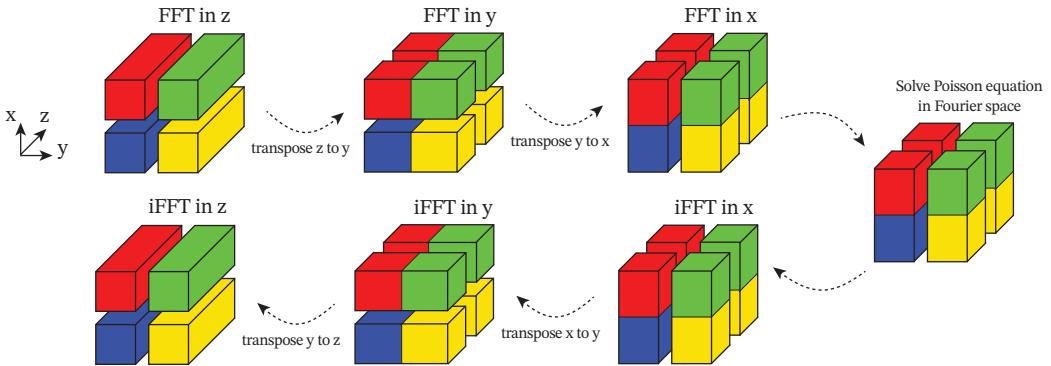


Figure 12: A schematic showing the distributed Poisson solver procedure with a pencil parallelization that divides the domain in two ranks in both  $x$  and  $y$ . The schematic highlights the data layout in the ranks during each operation.

et al. (2015), wherein advecting velocities are constructed with a high-order Centered scheme when the advected velocity component is reconstructed with a high-order UpwindBiased or WENO scheme. We have also developed a novel WENO-based method for discretizing momentum advection in the rotational or “vector invariant” form especially appropriate for representing mesoscale and submesoscale turbulent advection on curvilinear grids (Silvestri, Wagner, Campin, et al., 2024).

## 5 Parallelization

Oceananigans supports distributed computations with slab and pencil domain decomposition. The interior domain is extended using “halo” or “ghost” cells that hold the results of interprocessor boundaries. “halo” cells are updated before the computation of tendencies through asynchronous send / receive operations using the message passing interface (MPI) Julia library (Byrne et al., 2021). For a detailed description of the parallelization strategy of the HydrostaticFreeSurfaceModel; see Silvestri, Wagner, Constantinou, et al. (2024). The NonhydrostaticModel implements the same overlap of communication and computation for halo exchange before the calculation of tendencies. For the FFT-based three-dimensional pressure solver, we implement a transpose algorithm that switches between  $x$ -local,  $y$ -local, and  $z$ -local configurations to compute efficiently the discrete transforms. The transpose algorithm for the distributed FFT solver is shown in figure 12.

## 6 Conclusions

This paper describes GPU-based ocean modeling software called “Oceananigans” written in the high-level Julia programming language. Oceananigans enables high resolution simulations of oceanic motion at any scale with an innovative user interface design that makes simple simulations easy and complex, creative simulations possible. The current state of Oceananigans realizes a particular strategy for improving dynamical cores: simple, C-grid, WENO numerics for turbulence resolving simulations coupled to the raw power of GPU acceleration.

Using GPUs enables high-resolution simulations on few resources — such as a single GPU instance on the cloud — increasing access to ocean modeling. But it also enables a new class of very high resolution simulations. For example, on the Perlmutter supercomputer (National Energy Research Scientific Computing Center, 2025), it is currently possible to complete a 100-member ensemble of century-long global ocean simulations at 10 kilometer resolution in

675 10 days of wall time — thereby resolving mesoscale turbulent mixing, a prominent bias in  
 676 ocean models and a fundamental process missing from most climate simulations today. These  
 677 new capabilities address uncertainty in ocean heat and carbon uptake in climate projections.

678 Oceananigans is designed for composition with external packages, which has fostered the  
 679 development of a number of auxiliary packages. For example, OceanBioME ([Strong-Wright  
 et al., 2023](#)) implements Oceananigans-compatible biogeochemistry models, oriented towards  
 680 ecosystem dynamics and compatible with both the hydrostatic and nonhydrostatic models.  
 681 A second biogeochemistry package is also under development for climate simulations. The  
 682 Oceanostics ([Chor et al., 2025](#)) package implements complex diagnostics in Oceananigans  
 683 syntax, useful for online and offline analysis of large eddy simulations.

685 A next step is to couple ocean models built with Oceananigans to prognostic atmosphere  
 686 models, including the Climate Modeling Alliance atmosphere dynamical core ([Yatunin et al.,  
 687 2025](#)) and the simpler SpeedyWeather ([Klöwer et al., 2024](#)). A further possibility, enabled  
 688 by Oceananigans GPU capabilities, is to couple to hybrid physics/AI atmosphere models  
 689 ([Kochkov et al., 2024](#)), or fully-AI atmosphere models like ACE ([Watt-Meyer et al., 2023,  
 690 2024](#)) and GraphCast ([Lam et al., 2023](#)). A sea ice model called ClimaSeaIce, which uses  
 691 the same finite volume engine underpinning Oceananigans, is under active development to  
 692 support coupled ocean-sea-ice simulations. There is also an ongoing effort to use Enzyme  
 693 ([Moses et al., 2021](#)) to develop an adjoint for Oceananigans, and to more generally use  
 694 autodifferentiation to compute the gradients of cost functions that invoke Oceananigans  
 695 simulations.

696 Each achievement — groundbreaking performance, physics flexibility, or an innovative  
 697 design — would, on their own, enable scientific breakthroughs. This matters because ocean  
 698 modeling software will have to continue to evolve rapidly to keep pace with the advancing  
 699 state of computational science to remain cutting-edge: to continue to use the world’s largest  
 700 supercomputers, to present the most productive possible abstractions for both users and  
 701 developers, and to enable the next generation of parameterizations and AI-based model  
 702 components.

## 703 Appendix A Time stepping and time discretization

704 In this section we describe time stepping methods and time discretization options for  
 705 the NonhydrostaticModel and the HydrostaticFreeSurfaceModel.

### 706 A1 Time discretization for tracers

707 Tracers are stepped forward with similar schemes in the NonhydrostaticModel and the  
 708 HydrostaticFreeSurfaceModel, each of which includes optional implicit treatment of vertical  
 709 diffusion terms. Equation (9) is abstracted into two components,

$$\partial_t c = G_c + \partial_z (\kappa_z \partial_z c) , \quad (\text{A1})$$

710 where, if specified,  $\kappa_z$  is the vertical diffusivity of  $c$  to be treated with a VerticallyImplicitTimeDiscretization,  
 711 and  $G_c$  is the remaining component of the tracer tendency from  
 712 equation 9. (Vertical diffusion treated with an ExplicitTimeDiscretization is also absorbed  
 713 into  $G_c$ .) We apply a semi-implicit time discretization of vertical diffusion to approximate  
 714 integral of (A1) from  $t^m$  to  $t^{m+1}$ ,

$$(1 - \Delta t \partial_z \kappa_z^m \partial_z) c^{m+1} = c^m + \int_{t^m}^{t^{m+1}} G_c dt , \quad (\text{A2})$$

715 where  $\Delta t \stackrel{\text{def}}{=} t^{m+1} - t^m$ . The tendency integral  $\int_{t^m}^{t^{m+1}} G_c dt$  is evaluated either using a  
 716 “quasi”-second order Adams-Bashforth scheme (QAB2, which is actually first-order *lets add a  
 717 reference*), or a low-storage third-order Runge-Kutta scheme (RK3). For QAB2, the integral

718 in (A2) spans the entire time-step and takes the form

$$\frac{1}{\Delta t} \int_{t^m}^{t^{m+1}} G_c dt \approx \left(\frac{3}{2} + \chi\right) G_c^m - \left(\frac{1}{2} + \chi\right) G_c^{m-1}, \quad (\text{A3})$$

719 where  $\chi$  is a small parameter, chosen by default to be  $\chi = 0.1$ . QAB2 requires one tendency  
720 evaluation per time-step. For RK3, the indices  $m = (1, 2, 3)$  correspond to *substages*, and  
721 the integral in (A2) takes the form

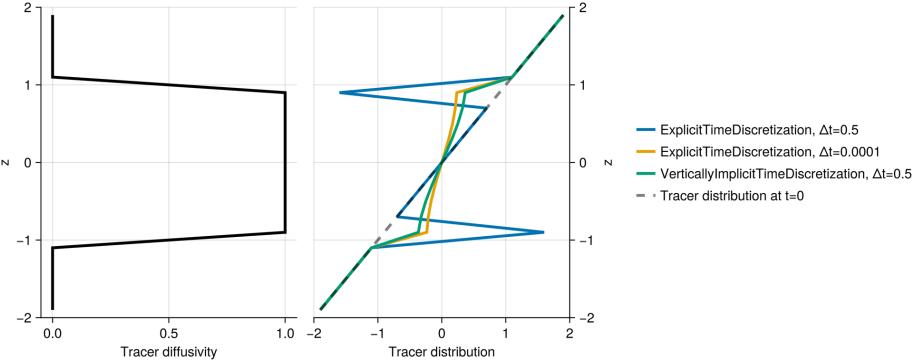
$$\frac{1}{\Delta t} \int_{t^m}^{t^{m+1}} G_c dt \approx \gamma^m G_c^m - \zeta^m G_c^{m-1}, \quad (\text{A4})$$

722 where  $\gamma = (8/15, 5/12, 3/4)$  and  $\zeta = (0, 17/60, 5/12)$  for  $m = (1, 2, 3)$  respectively. RK3  
723 requires three evaluations of the tendency  $G_c$  per time-step. RK3 is self-starting because  
724  $\zeta^1 = 0$ , while QAB2 must be started with a forward-backwards Euler step (the choice  
725  $\chi = -1/2$  in (A3)). Equation (A2) is solved with a tridiagonal algorithm following a second-  
726 order spatial discretization of  $\partial_z \kappa_z^n \partial_z c^{m+1}$  — either once per time-step for QAB2, or three  
727 times for each of the RK3’s three stages.

728 VerticallyImplicitTimeDiscretization permits longer time-steps when using fine vertical  
729 spacing. Listing 14 illustrates the differences between vertically-implicit and explicit time  
730 discretization using one-dimensional diffusion of by a top-hat diffusivity profile. The results  
731 are shown in figure A1.

```
732
733 1 using Oceananigans
734 2
735 3 grid = RectilinearGrid(size=20, z=(-2, 2), topology=(Flat, Flat, Bounded))
736 4 time_discretization = VerticallyImplicitTimeDiscretization()
737 5 κ(z, t) = exp(-z^2)
738 6 closure = VerticalScalarDiffusivity(time_discretization; κ)
739 7 model = HydrostaticFreeSurfaceModel(; grid, closure, tracers=:c)
```

740 Listing 14: Diffusion of a tracer by a top hat tracer diffusivity profile using various time steps and time  
741 discretizations.



742 Figure A1: Simulations of tracer diffusion by a top hat diffusivity profile using various choices of time-  
743 discretization and time-step size. With a long time-step of  $\Delta t = 0.5$ , ExplicitTimeDiscretization is unstable  
744 while VerticallyImplicitTimeDiscretization is stable. Let the vertically-implicit solution depends on the long  
745 time-step  $\Delta t = 0.5$ , as revealed by comparison with ExplicitTimeDiscretization using  $\Delta t = 10^{-4}$ .

743      **A2 The pressure correction method for momentum in NonhydrostaticModel**

744      The NonhydrostaticModel uses a pressure correction method for the momentum equation  
 745      (6) that ensures  $\nabla \cdot \mathbf{u} = 0$ . We rewrite (6) as

$$\partial_t \mathbf{u} = -\nabla p + b \hat{\mathbf{z}} + \mathbf{G}_u + \partial_z (\nu_z \partial_z \mathbf{u}), \quad (\text{A5})$$

746      where, if specified,  $\nu_z$  is the vertical component of the viscosity that will be treated with a  
 747      vertically-implicit time discretization,  $\nabla p$  is the total pressure gradient, and  $\mathbf{G}_u$  is the rest  
 748      of the momentum tendency. We decompose  $p$  into a ‘‘hydrostatic anomaly’’  $p'$  tied to the  
 749      density anomaly  $\rho'$ , and a nonhydrostatic component  $\tilde{p}$ , such that

$$p = \tilde{p} + p', \quad \text{where} \quad \partial_z p' \stackrel{\text{def}}{=} b. \quad (\text{A6})$$

750      By computing  $p_h$  in (A6), we recast (A5) without  $b$  and with  $\nabla p = \nabla p_n + \nabla_h p_h$ . Next,  
 751      integrating (A5) in time from  $t^m$  to  $t^{m+1}$  yields

$$\mathbf{u}^{m+1} = \mathbf{u}^m + \int_{t^m}^{t^{m+1}} [\mathbf{G}_u - \nabla \tilde{p} + \partial_z (\nu_z \partial_z \mathbf{u})] dt. \quad (\text{A7})$$

752      Next we introduce the predictor velocity  $\tilde{\mathbf{u}}$ , defined such that

$$(1 - \Delta t \partial_z \nu_z^m \partial_z) \tilde{\mathbf{u}} = \mathbf{u}^m + \int_{t^m}^{t^{m+1}} \mathbf{G}_u dt, \quad (\text{A8})$$

753      or in other words, defined as a velocity-like field that cannot feel nonhydrostatic pressure  
 754      gradient  $\nabla \tilde{p}$ . Equation (A8) uses a semi-implicit treatment of vertical momentum diffusion  
 755      which is similar but slightly different to the treatment of tracer diffusion in (A2),

$$\int_{t^m}^{t^{m+1}} \partial_z (\nu_z \partial_z \mathbf{u}) dt \approx \Delta t \partial_z (\nu_z^m \partial_z \tilde{\mathbf{u}}). \quad (\text{A9})$$

756      The integral in (A8) is evaluated with the same methods used for tracers — either (A3) for  
 757      QAB2 or (A4) when using RK3. With a second-order discretization of vertical momentum  
 758      diffusion, the predictor velocity in (A8) may be computed with a tridiagonal solver.

759      Introducing a fully-implicit time discretization for  $\tilde{p}$ ,

$$\int_{t^m}^{t^{m+1}} \nabla \tilde{p} dt \approx \Delta t \nabla \tilde{p}^{m+1}, \quad (\text{A10})$$

760      and inserting (A10) into (A8), we derive the pressure correction to the predictor velocity,

$$\mathbf{u}^{m+1} - \tilde{\mathbf{u}} = -\Delta t \nabla \tilde{p}^{m+1}. \quad (\text{A11})$$

761      The final ingredient needed to complete the pressure correction scheme is an equation  
 762      for the nonhydrostatic pressure  $\tilde{p}_n^{m+1}$ . For this we form  $\nabla \cdot$  (A11) and use  $\nabla \cdot \mathbf{u}^{m+1} = 0$  to  
 763      obtain a Poisson equation for  $\tilde{p}_n^{m+1}$ ,

$$\nabla^2 \tilde{p}_n^{m+1} = \frac{\nabla \cdot \tilde{\mathbf{u}}}{\Delta t}. \quad (\text{A12})$$

764      Boundary conditions for equation (A12) may be derived by evaluating  $\hat{\mathbf{n}} \cdot$  (A7) on the  
 765      boundary of the domain.

766      On RectilinearGrids, we solve (A12) using an eigenfunction expansion of the discrete  
 767      second-order Poisson operator  $\nabla^2$  evaluated via the fast Fourier transform (FFT) in eq-  
 768      uispaced directions (Schumann & Sweet, 1988) plus a tridiagonal solve in variably-spaced  
 769      directions. With the FFT-based solver, boundary conditions on  $\tilde{p}^{m+1}$  are accounted for  
 770      by enforcing  $\hat{\mathbf{n}} \cdot \tilde{\mathbf{u}} = \hat{\mathbf{n}} \cdot \mathbf{u}^{m+1}$  on boundary cells — which is additional and separate from

771 the definition  $\tilde{\mathbf{u}}$  in (A9). This alteration of  $\tilde{\mathbf{u}}$  on the boundary implicitly contributes the  
 772 appropriate terms that account for inhomogeneous boundary-normal pressure gradients  
 773  $\hat{\mathbf{n}} \cdot \nabla \tilde{p}^{m+1} \neq 0$  to the right-hand-side of (A12) during the computation of  $\nabla \cdot \tilde{\mathbf{u}}$ .

774 A preconditioned conjugate gradient iteration may be used on non-rectilinear grids,  
 775 including complex domains. For domains that immerse an irregular boundary within a  
 776 RectilinearGrid, we have implemented an efficient, rapidly-converging preconditioner that  
 777 leverages the FFT-based solver with masking applied to immersed cells. The FFT-based  
 778 preconditioner for solving the Poisson equation in irregular domains will be described in a  
 779 forthcoming paper.

### 780 A3 Time discretization of the HydrostaticFreeSurfaceModel

781 The HydrostaticFreeSurfaceModel uses a linear free surface formulation paired with  
 782 a geopotential vertical coordinate that may be integrated in time using either a fully  
 783 ExplicitFreeSurface, an ImplicitFreeSurface utilizing a two-dimensional elliptical solve, or a  
 784 SplitExplicitFreeSurface. The latter free surface solver can also be used to solve the primitive  
 785 equations with a non-linear free surface formulation and a free-surface following vertical  
 786 coordinate (the  $z^*$  vertical coordinate, Adcroft & Campin, 2004). For brevity, we describe  
 787 here only the SplitExplicitFreeSurface, which is the most generally useful method. The  
 788 SplitExplicitFreeSurface substeps the depth-integrated or “barotropic” horizontal velocity  
 789  $\mathbf{U}_h$  along with the free surface displacement  $\eta$  using a short time step while the depth-  
 790 dependent, “baroclinic” velocities, along with tracers, are relatively stationary.

791 The barotropic horizontal transport  $\mathbf{U}_h$  is defined

$$792 \mathbf{U}_h \stackrel{\text{def}}{=} \int_{-H}^{\eta} \mathbf{u}_h \, dz, \quad (\text{A13})$$

793 where  $\mathbf{u}_h = (u, v)$  is the total horizontal velocity and  $H$  is the depth of the fluid.

794 Similarly integrating the horizontal momentum equations (17) from  $z = -H$  to  $z = \eta$   
 yields an evolution equation for  $\mathbf{U}_h$ ,

$$795 \partial_t \mathbf{U}_h = -g(H + \eta) \nabla_h \eta + \int_{-H}^{\eta} \mathbf{G}_{uh} \, dz, \quad (\text{A14})$$

796 where  $\mathbf{G}_{uh}$  includes all the tendency terms that evolve “slowly” compared to the barotropic  
 mode:

$$797 \mathbf{G}_{uh} = -(\mathbf{u} \cdot \nabla) \mathbf{u}_h - \mathbf{f} \times \mathbf{u} - \nabla \cdot \boldsymbol{\tau} + \mathbf{F}_h. \quad (\text{A15})$$

798 The evolution equation for the free surface is obtained by integrating the continuity equation  
 (15) in  $z$  to obtain  $\nabla \cdot \mathbf{U}_h = -w|_{z=\eta}$ , and inserting this into (16) to find

$$799 \partial_t \eta = -\nabla_h \cdot \mathbf{U}_h. \quad (\text{A16})$$

800 The pair of equations (A14) and (A16) characterize the evolution of the barotropic mode,  
 801 which involves faster time-scales than the baroclinic mode evolution described by equations  
 802 (17). To resolve both modes, we use a split-explicit algorithm where the barotropic mode is  
 803 advanced in time using a smaller time-step than the one used for three-dimensional baroclinic  
 804 variables. In particular, a predictor three-dimensional velocity is evolved without accounting  
 805 for the barotropic mode evolution, using the QAB2 scheme described by A3. We denote this  
 “predictor” velocity, again, with a tilde as done in section A2.

$$806 (1 - \Delta t \partial_z \nu_z^m \partial_z) \tilde{\mathbf{u}}_h - \mathbf{u}_h^m \approx \int_{t^m}^{t^{m+1}} \mathbf{G}_{uh} \, dt. \quad (\text{A17})$$

807 We then compute the barotropic mode evolution by sub-stepping  $M$  times the barotropic  
 equations using a forward-backward time-stepping scheme and a time-step  $\Delta\tau = \Delta t/N$ ,

$$808 \eta^{n+1} - \eta^n = -\Delta\tau \nabla_h \cdot \mathbf{U}_h^n, \quad (\text{A18})$$

808

$$\mathbf{U}_h^{n+1} - \mathbf{U}_h^n = -\Delta\tau \left[ g(H + \eta) \nabla_h \eta^{n+1} - \frac{1}{\Delta t} \int_{-H}^{\eta} \int_{t^m}^{t^{m+1}} \mathbf{G}_{uh} dt dz \right]. \quad (\text{A19})$$

809

The slow tendency terms are frozen in time during substepping. The barotropic quantities  
810 are averaged within the sub-stepping with

$$\bar{\mathbf{U}}_h = \sum_{n=1}^M a_n \mathbf{U}_h^n, \quad \bar{\eta} = \sum_{n=1}^M a_n \eta^n, \quad (\text{A20})$$

811

where  $M$  is the number of substeps per baroclinic step, and  $a_n$  are the weights calculated  
812 from the provided averaging kernel. The default choice of averaging kernel is the minimal  
813 dispersion filters developed by Shchepetkin and McWilliams (2005). The number of substeps  
814  $M$  is calculated to center the averaging kernel at  $t^{m+1}$ . As a result, the barotropic subcycling  
815 overshoots the baroclinic step, i.e.  $M > N$  with a maximum of  $M = 2N$ . Finally, the  
816 barotropic mode is reconciled to the baroclinic mode with a correction step

$$\mathbf{u}_h^{m+1} = \tilde{\mathbf{u}}_h + \frac{1}{H + \eta} \left( \bar{\mathbf{U}}_h - \int_{-H}^{\eta} \tilde{\mathbf{u}}_h dz \right). \quad (\text{A21})$$

817

The barotropic variables are then reinitialized for evolution in the next barotropic mode  
818 evolution using the time-averaged  $\bar{\eta}$  and  $\bar{\mathbf{U}}_h$ .

819

## Appendix B Table of numerical examples

Description	Code	Visualization
2D turbulence using WENO(order=9) advection	listing 1	fig. 1
2D turbulence with moving tracer source	listing 2	fig. 1
DNS and LES of flow around a cylinder at various $Re$	listing 4	fig. 2
DNS of cabbeling in freshwater	listing 5	fig. 3
LES of the Eady problem with WENO(order=9)	listing 6	fig. 4
Tidally-oscillating flow past Three Tree Point	listing 7	fig. 5
Internal waves generated by tidal forcing over bathymetry	listing 8	fig. 6
Comparison of vertical mixing parameterizations	listing 9	fig. 7
Near-global ocean simulation with ClimaOcean	listing 10	fig. 8
Visualization of the finite volume discretization	listing 11	fig. 9
One-dimensional advection of a top-hat tracer profile	listing 13	fig. 11
Tracer diffusion with various time discretizations	listing 14	fig. A1

821

## Open Research Section

822

Oceananigans is available at the GitHub repository [github.com/CliMA/Oceananigans.jl](https://github.com/CliMA/Oceananigans.jl).  
823 Oceananigans documentation lives at <https://clima.github.io/OceananigansDocumentation>.  
824 All the scripts that reproduce the simulations and figures in this paper are available at the  
825 GitHub repository [github.com/glwagner/OceananigansPaper](https://github.com/glwagner/OceananigansPaper). Visualizations were made  
826 using Makie.jl (Danisch & Krumbiegel, 2021).

827

## Acknowledgments

828

This project is supported by Schmidt Sciences, LLC and by the National Science Foundation

grant AGS-1835576. N.C.C. is additionally supported by the Australian Research Council under the Center of Excellence for the Weather of the 21st Century CE230100012 and the Discovery Project DP240101274.

## References

- Adcroft, A., & Campin, J.-M. (2004). Rescaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 7(3-4), 269–284.
- Arakawa, A. (1977). Computational design of the basic dynamical processes of the UCLA general circulation model. *Methods in Computational Physics/Academic Press*.
- Balsara, D., & Shu, C. (2000). Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy. *Journal of Computational Physics*, 160(2), 405–452. doi: 10.1006/jcph.2000.6443
- Besard, T., Foket, C., & De Sutter, B. (2018). Effective extensible programming: unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 30(4), 827–841.
- Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1), 65–98.
- Boccaletti, G., Ferrari, R., & Fox-Kemper, B. (2007). Mixed layer instabilities and restratification. *Journal of Physical Oceanography*, 37(9), 2228–2250.
- Bou-Zeid, E., Meneveau, C., & Parlange, M. (2005). A scale-dependent Lagrangian dynamic model for large eddy simulation of complex turbulent flows. *Physics of fluids*, 17(2).
- Bryan, K. (1969). A numerical method for the study of the circulation of the world ocean. *Journal of Computational Physics*, 135(2), 154–169.
- Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D., & Brown, B. P. (2020). Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2(2), 023068.
- Byrne, S., Wilcox, L. C., & Churavy, V. (2021). MPI.jl: Julia bindings for the Message Passing Interface. In *Proceedings of the JuliaCon Conferences* (Vol. 1, p. 68). doi: 10.21105/jcon.00068
- Callies, J., & Ferrari, R. (2018). Baroclinic instability in the presence of convection. *Journal of Physical Oceanography*, 48(1), 45–60.
- Chassignet, E. P., & Xu, X. (2017). Impact of horizontal resolution (1/12 to 1/50) on Gulf Stream separation, penetration, and variability. *Journal of Physical Oceanography*, 47(8), 1999–2021.
- Chassignet, E. P., & Xu, X. (2021). On the importance of high-resolution in large-scale ocean models. *Advances in Atmospheric Sciences*, 38, 1621–1634.
- Chor, T., Constantinou, N. C., Bisits, J. I., Wagner, G. L., Ramadhan, A., Zheng, Z., & Whitley, V. (2025). *Oceanostics.jl*. Zenodo. Retrieved from <https://doi.org/10.5281/zenodo.8280754> doi: 10.5281/zenodo.8280754
- Churavy, V. (2024). *KernelAbstractions.jl*. Zenodo. Retrieved from <https://doi.org/10.5281/zenodo.13773520> doi: 10.5281/zenodo.13773520
- Cox, M. D. (1984). *A primitive equation, 3-dimensional model of the ocean* (Tech. Rep. No. 1). Princeton, NJ: NOAA Geophysical Fluid Dynamics Laboratory.
- Craik, A. D., & Leibovich, S. (1976). A rational model for Langmuir circulations. *Journal of Fluid Mechanics*, 73(3), 401–426.
- Danilov, S., Sidorenko, D., Wang, Q., & Jung, T. (2017). The finite-volume sea ice–ocean model (FESOM2). *Geoscientific Model Development*, 10(2), 765–789.
- Danisch, S., & Krumbiegel, J. (2021). Makie.jl: Flexible high-performance data visualization for Julia. *Journal of Open Source Software*, 6(65), 3349. doi: 10.21105/joss.03349
- Dong, J., Fox-Kemper, B., Wenegrat, J. O., Bodner, A. S., Yu, X., Belcher, S., & Dong, C. (2024). Submesoscales are a significant turbulence source in global ocean surface boundary layer. *Nature Communications*, 15(1), 9566.
- Eady, E. T. (1949). Long waves and cyclone waves. *Tellus*, 1(3), 33–52.

- 881 Edson, J. B., Jampana, V., Weller, R. A., Bigorre, S. P., Plueddemann, A. J., Fairall, C. W.,  
 882 ... Hersbach, H. (2014). On the exchange of momentum over the open ocean. *Journal  
 883 of Physical Oceanography*, *44*(9), 1589.
- 884 Forget, G., Campin, J.-M., Heimbach, P., Hill, C., Ponte, R., & Wunsch, C. (2015). ECCO  
 885 version 4: An integrated framework for non-linear inverse modeling and global ocean  
 886 state estimation. *Geoscientific Model Development*, *8*(10), 3071–3104.
- 887 Ghosh, D., & Baeder, J. D. (2012). High-order accurate incompressible Navier–Stokes  
 888 algorithm for vortex-ring interactions with solid wall. *AIAA journal*, *50*(11), 2408–  
 889 2422.
- 890 Godunov, S. K. (1959). A difference scheme for numerical solution of discontinuous solution  
 891 of hydrodynamic equations. *Matematicheskii Sbornik*, *47*, 271–306. (Translated by US  
 892 Joint Publications Research Service, JPRS 7226, 1969)
- 893 Grachev, A. A., Andreas, E. L., Fairall, C. W., Guest, P. S., & Persson, P. O. G. (2007). Sheba  
 894 flux–profile relationships in the stable atmospheric boundary layer. *Boundary-layer  
 895 meteorology*, *124*, 315–333.
- 896 Griffies, S. M., Adcroft, A., & Hallberg, R. W. (2020). A primer on the vertical lagrangian–  
 897 remap method in ocean models based on finite volume generalized vertical coordinates.  
 898 *Journal of Advances in Modeling Earth Systems*, *12*(10), e2019MS001954.
- 899 Griffies, S. M., Pacanowski, R. C., & Hallberg, R. W. (2000). Spurious diapycnal mixing  
 900 associated with advection in a z-coordinate ocean model. *Monthly Weather Review*,  
 901 *128*(3), 538–564.
- 902 Griffies, S. M., Stouffer, R. J., Adcroft, A. J., Bryan, K., Dixon, K. W., Hallberg, R., ...  
 903 Rosati, A. (2015). A historical introduction to MOM. URL [https://www.gfdl.noaa.gov/wp-content/uploads/2019/04/mom\\_history\\_2017.pdf](https://www.gfdl.noaa.gov/wp-content/uploads/2019/04/mom_history_2017.pdf), 9.
- 904 Häfner, D., Nuterman, R., & Jochum, M. (2021). Fast, cheap, and turbulent—global ocean  
 905 modeling with GPU acceleration in Python. *Journal of Advances in Modeling Earth  
 906 Systems*, *13*(12), e2021MS002717.
- 907 Halliwell, G. R. (2004). Evaluation of vertical coordinate and vertical mixing algorithms in  
 908 the HYbrid-Coordinate Ocean Model (HYCOM). *Ocean Modelling*, *7*(3–4), 285–322.
- 909 Held, I. M. (2005). The gap between simulation and understanding in climate modeling.  
 910 *Bulletin of the American Meteorological Society*, *86*(11), 1609–1614.
- 911 Huang, N. E. (1979). On surface drift currents in the ocean. *Journal of Fluid Mechanics*,  
 912 *91*(1), 191–208.
- 913 Kärnä, T., Kramer, S. C., Mitchell, L., Ham, D. A., Piggott, M. D., & Baptista, A. M.  
 914 (2018). Thetis coastal ocean model: discontinuous Galerkin discretization for the  
 915 three-dimensional hydrostatic equations. *Geoscientific Model Development*, *11*(11),  
 916 4359–4382.
- 917 Kiss, A. E., Hogg, A. M., Hannah, N., Boeira Dias, F., Brassington, G. B., Chamberlain,  
 918 M. A., ... others (2020). Access-om2 v1. 0: a global ocean–sea ice model at three  
 919 resolutions. *Geoscientific Model Development*, *13*(2), 401–442.
- 920 Klöwer, M., Gelbrecht, M., Hotta, D., Willmert, J., Silvestri, S., Wagner, G. L., ... others  
 921 (2024). Speedyweather. jl: Reinventing atmospheric general circulation models towards  
 922 interactivity and extensibility. *Journal of Open Source Software*, *9*(98), 6323.
- 923 Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., ... others  
 924 (2024). Neural general circulation models for weather and climate. *Nature*, *632*(8027),  
 925 1060–1066.
- 926 Korn, P., Brüggemann, N., Jungclaus, J. H., Lorenz, S., Gutjahr, O., Haak, H., ... others  
 927 (2022). Icon-o: The ocean component of the icon earth system model—global simulation  
 928 characteristics and local telescoping capability. *Journal of Advances in Modeling Earth  
 929 Systems*, *14*(10), e2021MS002952.
- 930 Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., ...  
 931 others (2023). Learning skillful medium-range global weather forecasting. *Science*,  
 932 *382*(6677), 1416–1421.
- 933 Leclair, M., & Madec, G. (2011). z-coordinate, an arbitrary lagrangian–eulerian coordinate  
 934 separating high and low frequency motions. *Ocean Modelling*, *37*(3–4), 139–152.

- 936 Lilly, D. K. (1983). Stratified turbulence and the mesoscale variability of the atmosphere.  
 937 *Journal of the Atmospheric Sciences*, 40(3), 749–761.
- 938 Marshall, J., Adcroft, A., Hill, C., Perelman, L., & Heisey, C. (1997). A finite-volume,  
 939 incompressible Navier Stokes model for studies of the ocean on parallel computers.  
 940 *Journal of Geophysical Research: Oceans*, 102(C3), 5753–5766.
- 941 Marshall, J., Hill, C., Perelman, L., & Adcroft, A. (1997). Hydrostatic, quasi-hydrostatic, and  
 942 nonhydrostatic ocean modeling. *Journal of Geophysical Research: Oceans*, 102(C3),  
 943 5733–5752.
- 944 McDougall, T. J., & Barker, P. M. (2011). Getting started with TEOS-10 and the Gibbs  
 945 Seawater (GSW) oceanographic toolbox. *Scor/iaipso WG*, 127(532), 1–28.
- 946 Molemaker, M. J., McWilliams, J. C., & Capet, X. (2010). Balanced and unbalanced routes  
 947 to dissipation in an equilibrated eady flow. *Journal of Fluid Mechanics*, 654, 35–63.
- 948 Monin, A. (n.d.). Basic laws of turbulent mixing in the surface layer of the atmosphere.
- 949 Moses, W. S., Churavy, V., Paehler, L., Hückelheim, J., Narayanan, S. H. K., Schanen, M.,  
 950 & Doerfert, J. (2021). Reverse-mode automatic differentiation and optimization of gpu  
 951 kernels via enzyme. In *Proceedings of the international conference for high performance  
 952 computing, networking, storage and analysis* (pp. 1–16).
- 953 Murray, R. J. (1996). Explicit generation of orthogonal grids for ocean models. *Journal of  
 954 Computational Physics*, 126(2), 251–273.
- 955 National Energy Research Scientific Computing Center. (2025). *Perlmutter archi-  
 956 tecture*. Retrieved from [https://docs.nersc.gov/systems/perlmutter/  
 957 architecture/](https://docs.nersc.gov/systems/perlmutter/architecture/) (Accessed: 2025-02-18)
- 958 Pawlak, G., MacCready, P., Edwards, K., & McCabe, R. (2003). Observations on the  
 959 evolution of tidal vorticity at a stratified deep water headland. *Geophysical Research  
 960 Letters*, 30(24).
- 961 Pearson, B. (2018). Turbulence-induced anti-Stokes flow and the resulting limitations of  
 962 large-eddy simulation. *Journal of Physical Oceanography*, 48(1), 117–122.
- 963 Petersen, M. R., Jacobsen, D. W., Ringler, T. D., Hecht, M. W., & Maltrud, M. E. (2015).  
 964 Evaluation of the arbitrary Lagrangian–Eulerian vertical coordinate method in the  
 965 MPAS-Ocean model. *Ocean Modelling*, 86, 93–113.
- 966 Phillips, N. A. (1956). The general circulation of the atmosphere: A numerical experiment.  
 967 *Quarterly Journal of the Royal Meteorological Society*, 82(352), 123–164.
- 968 Pressel, K. G., Kaul, C. M., Schneider, T., Tan, Z., & Mishra, S. (2015). Large-eddy  
 969 simulation in an anelastic framework with closed water and entropy balances. *Journal  
 970 of Advances in Modeling Earth Systems*, 7(3), 1425–1456.
- 971 Pressel, K. G., Mishra, S., Schneider, T., Kaul, C. M., & Tan, Z. (2017). Numerics and  
 972 subgrid-scale modeling in large eddy simulations of stratocumulus clouds. *Journal of  
 973 Advances in Modeling Earth Systems*, 9(2), 1342–1365.
- 974 Ramadhan, A., Wagner, G., Hill, C., Campin, J.-M., Churavy, V., Besard, T., ... Marshall,  
 975 J. (2020). Oceananigans.jl: Fast and friendly geophysical fluid dynamics on GPUs.  
 976 *Journal of Open Source Software*, 5(53).
- 977 Ringler, T., Petersen, M., Higdon, R. L., Jacobsen, D., Jones, P. W., & Maltrud, M. (2013).  
 978 A multi-resolution approach to global ocean modeling. *Ocean Modelling*, 69, 211–232.
- 979 Roquet, F., Madec, G., Brodeau, L., & Nycander, J. (2015). Defining a simplified yet  
 980 “realistic” equation of state for seawater. *Journal of Physical Oceanography*, 45(10),  
 981 2564–2579.
- 982 Roquet, F., Madec, G., McDougall, T. J., & Barker, P. M. (2015). Accurate polynomial  
 983 expressions for the density and specific volume of seawater using the TEOS-10 standard.  
 984 *Ocean Modelling*, 90, 29–43.
- 985 Rozema, W., Bae, H. J., Moin, P., & Verstappen, R. (2015). Minimum-dissipation models  
 986 for large-eddy simulation. *Physics of Fluids*, 27(8).
- 987 Schumann, U., & Sweet, R. A. (1988). Fast Fourier transforms for direct solution of Poisson’s  
 988 equation with staggered boundary conditions. *Journal of Computational Physics*,  
 989 75(1), 123–137.
- 990 Shchepetkin, A. F., & McWilliams, J. C. (2005). The regional oceanic modeling system

- (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean modelling*, 9(4), 347–404.
- Shu, C. (1997). *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws* (ICASE Report No. 97-65). Institute for Computer Applications in Science and Engineering, NASA Langley Research Center.
- Shu, C.-W. (2009). High order weighted essentially nonoscillatory schemes for convection dominated problems. *SIAM review*, 51(1), 82–126.
- Silvestri, S., Wagner, G. L., Campin, J.-M., Constantinou, N. C., Hill, C. N., Souza, A., & Ferrari, R. (2024). A new WENO-based momentum advection scheme for simulations of ocean mesoscale turbulence. *Journal of Advances in Modeling Earth Systems*, 16(7), e2023MS004130.
- Silvestri, S., Wagner, G. L., Constantinou, N. C., Hill, C. N., Campin, J.-M., Souza, A. N., ... Ferrari, R. (2024). A GPU-based ocean dynamical core for routine mesoscale-resolving climate simulations. *Authorea Preprints*. doi: 10.22541/essoar.171708158.82342448/v1
- Smagorinsky, J. (1963). General circulation experiments with the primitive equations: I. The basic experiment. *Monthly weather review*, 91(3), 99–164.
- Stone, P. H. (1971). Baroclinic stability under non-hydrostatic conditions. *Journal of Fluid Mechanics*, 45(4), 659–671.
- Strong-Wright, J., Chen, S., Constantinou, N. C., Silvestri, S., Wagner, G. L., & Taylor, J. R. (2023). OceanBioME.jl: A flexible environment for modelling the coupled interactions between ocean biogeochemistry and physics. *Journal of Open Source Software*, 8(90), 5669.
- Tsujino, H., Urakawa, S., Nakano, H., Small, R. J., Kim, W. M., Yeager, S. G., ... others (2018). JRA-55 based surface dataset for driving ocean–sea-ice models (JRA55-do). *Ocean Modelling*, 130, 79–139.
- Umlauf, L., & Burchard, H. (2003). A generic length-scale equation for geophysical turbulence models.
- Umlauf, L., & Burchard, H. (2005). Second-order turbulence closure models for geophysical boundary layers. a review of recent work. *Continental Shelf Research*, 25(7-8), 795–827.
- Vanneste, J., & Young, W. R. (2022). Stokes drift and its discontents. *Philosophical Transactions of the Royal Society A*, 380(2225), 20210032.
- Vreugdenhil, C. A., & Taylor, J. R. (2018). Large-eddy simulations of stratified plane Couette flow using the anisotropic minimum-dissipation model. *Physics of Fluids*, 30(8).
- Wagner, G. L., Chini, G. P., Ramadhan, A., Gallet, B., & Ferrari, R. (2021). Near-inertial waves and turbulence driven by the growth of swell. *Journal of Physical Oceanography*, 51(5), 1337–1351.
- Wagner, G. L., Hillier, A., Constantinou, N. C., Silvestri, S., Souza, A. N., Burns, K., ... others (2025). Formulation and calibration of CATKE, a one-equation parameterization for microscale ocean mixing. *Authorea Preprints*. doi: 10.48550/arXiv.2306.13204
- Wagner, G. L., Silvestri, S., Constantinou, N. C., Strong-Wright, J., Byrne, S., Bozzola, G., ... Churavy, V. (2025, February). *CliMA/ClimaOcean.jl: v0.4.0*. Zenodo. Retrieved from <https://doi.org/10.5281/zenodo.14890032> doi: 10.5281/zenodo.14890032
- Warner, S. J., & MacCready, P. (2014). The dynamics of pressure and form drag on a sloping headland: Internal waves versus eddies. *Journal of Geophysical Research: Oceans*, 119(3), 1554–1571.
- Watt-Meyer, O., Dresdner, G., McGibbon, J., Clark, S. K., Henn, B., Duncan, J., ... others (2023). ACE: A fast, skillful learned global atmospheric model for climate prediction. *arXiv preprint arXiv:2310.02074*.
- Watt-Meyer, O., Henn, B., McGibbon, J., Clark, S. K., Kwa, A., Perkins, W. A., ... Bretherton, C. S. (2024). ACE2: Accurately learning subseasonal to decadal atmospheric variability and forced responses. *arXiv preprint arXiv:2411.11268*.
- Yatunin, D., Byrne, S., Kawczynski, C., Kandala, S., Bozzola, G., Sridhar, A., ... others (2025). The Climate Modeling Alliance Atmosphere Dynamical Core: Concepts, Numerics, and Scaling. *Authorea Preprints*.