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

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

[1]Massachusetts Institute of Technology, Cambridge, MA, USA
[2]University of Melbourne, Parkville, VIC, Australia
[3]ARC Center of Excellence for the Weather of the 21st Century, Australia
[4]atdepth MRV, Cambridge, MA, USA
[5]University of Maryland, College Park, MD, USA
[6]University of Cambridge, Cambridge, United Kingdom
[7]University of Waterloo, Waterloo, ON, Canada
[8]Flatiron Institute, New York, NY, USA

Key Points:

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

Abstract

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

Plain Language Summary

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

1 Introduction

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

This paper describes new ocean modeling software written in the Julia programming language (Bezanson et al., 2017) called Oceananigans. Oceananigans is being developed by the Climate Modeling Alliance (along with heroic external collaborators) as part of a larger effort to develop a climate model automatically-calibrated to observations and high resolution simulations, and with quantified uncertainty. Oceananigans development is motivated primarily by the need for new capabilities. The most materially pressing is the need to implement a hierarchical approach to climate model development (Held, 2005), wherein nonhydrostatic large eddy simulations are used to generate synthetic data for calibrating parameterizations, followed by the refinement of parameters in a hydrostatic global context against observations. Starting from scratch also allowed us to target GPUs and CPUs and to lower the bar for future accelerator support, by leveraging the performance 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 (Chassignet & 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 numerous namelists, and source code modifications to change the model equations in ways not accessible through a change of parameters.

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

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

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

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

1.3 Outline of this paper

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

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

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

2 Oceananigans, the library

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

2.1 Hello, ocean

The way to learn new ocean modeling software is by building simulations with it. Our first example in listing 1 sets up, runs, and visualizes a simulation of two-dimensional turbulence. The 22 lines of listing 1 illustrate one of Oceananigans' main achievements: a numerical experiment may be completely described by a single script. To execute the code in listing 1, we need to copy into a file (call this, for example, `hello_ocean.jl`) and executed by typing `julia hello_ocean.jl` at a terminal.
Oceananigans scripts organize into four sections. The first three define the "grid" "model", and "simulation", and conclude with execution of the simulation. The fourth section, often implemented separately for complex or expensive simulations, performs post-processing and analysis. In listing 1, the grid defined on lines 4–7 determines the problem geometry, spatial resolution, and machine architecture. To use a CPU instead of a GPU, one writes `CPU()` in place of `GPU()` on line 5: no other changes to the script are required.

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

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

```julia
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.

```julia
function circling_source(x, y, t)
    δ, ω, r = 0.1, 2π/3, 2
    dx = x + r * cos(ω * t)
    dy = y + r * sin(ω * t)
    return exp(-(dx^2 + dy^2) / 2δ^2)
end

forcing = (; c = circling_source)
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 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

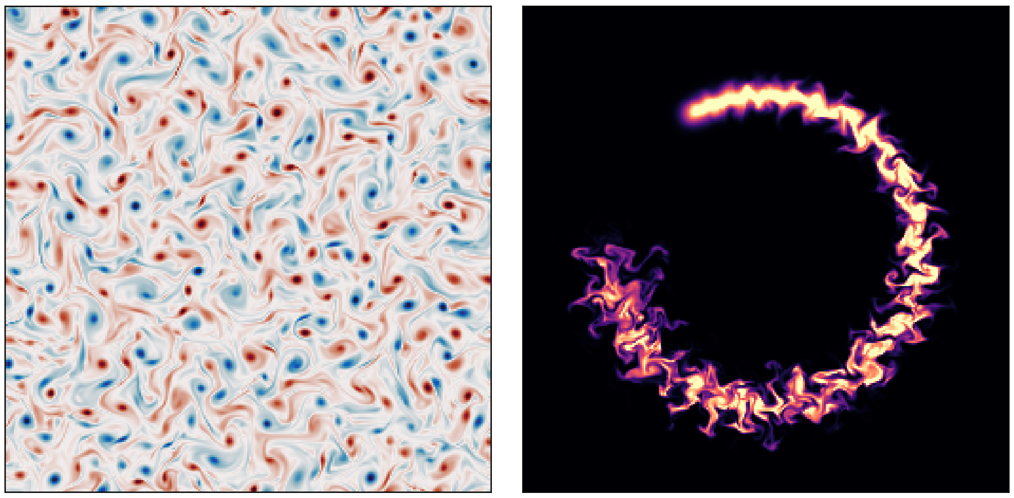

Figure 1: Vorticity after $t = 10$ (left) and a passive tracer injected by a moving source at $t = 2.5$ (right) in a simulation of two-dimensional turbulence using an implicitly-dissipative advection scheme.

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 $Z = \int_0^{2\pi} \zeta^2 \, \mathrm{d}x$ are shown in listing 3.

```
u, v, w = model.velocities

# Lazy expression trees and reductions representing computations:
ζ = ∂x(v) - ∂y(u)
s = √(u^2 + v^2)
Z = Integral(ζ^2, dims=1)
```

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 ρ_0 and a small dynamic perturbation ρ',

$$\rho(\boldsymbol{x}, t) = \rho_0 + \rho'(\boldsymbol{x}, t) \qquad \text{where} \qquad \rho' \ll \rho_0 \, , \tag{1}$$

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

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

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \, , \tag{2}$$

where

$$\boldsymbol{u} \overset{\text{def}}{=} u \hat{\boldsymbol{x}} + v \hat{\boldsymbol{y}} + w \hat{\boldsymbol{z}} \, , \tag{3}$$

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

$$b \overset{\text{def}}{=} -\frac{g\rho'}{\rho_0}\,, \tag{4}$$

where g is gravitational acceleration. The "buoyancy" b acts in the direction of gravity.

The total dynamic pressure P is decomposed into

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

where $\rho_0 g z$ is the static contribution to pressure that opposes the gravitational force associated with the reference density ρ_0, and $\rho_0 p$ represents the dynamic anomaly. p is called the kinematic pressure.

3.1 The NonhydrostaticModel

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

3.1.1 The NonhydrostaticModel momentum equation

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

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

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

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

Using (2), advection in the NonhydrostaticModel is formulated in the "flux form", which is conveniently expressed with indicial notation,

$$\text{advection} = u_j \partial_j u_i + u_{g_j} \partial_j u_i + u_j \partial_j u_{g_i} = \partial_j \Big[(u_j + u_{g_j}) u_i + u_j u_{g_i} \Big]\,, \tag{7}$$

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

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

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

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

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

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

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

3.1.2 The NonhydrostaticModel tracer conservation equation

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

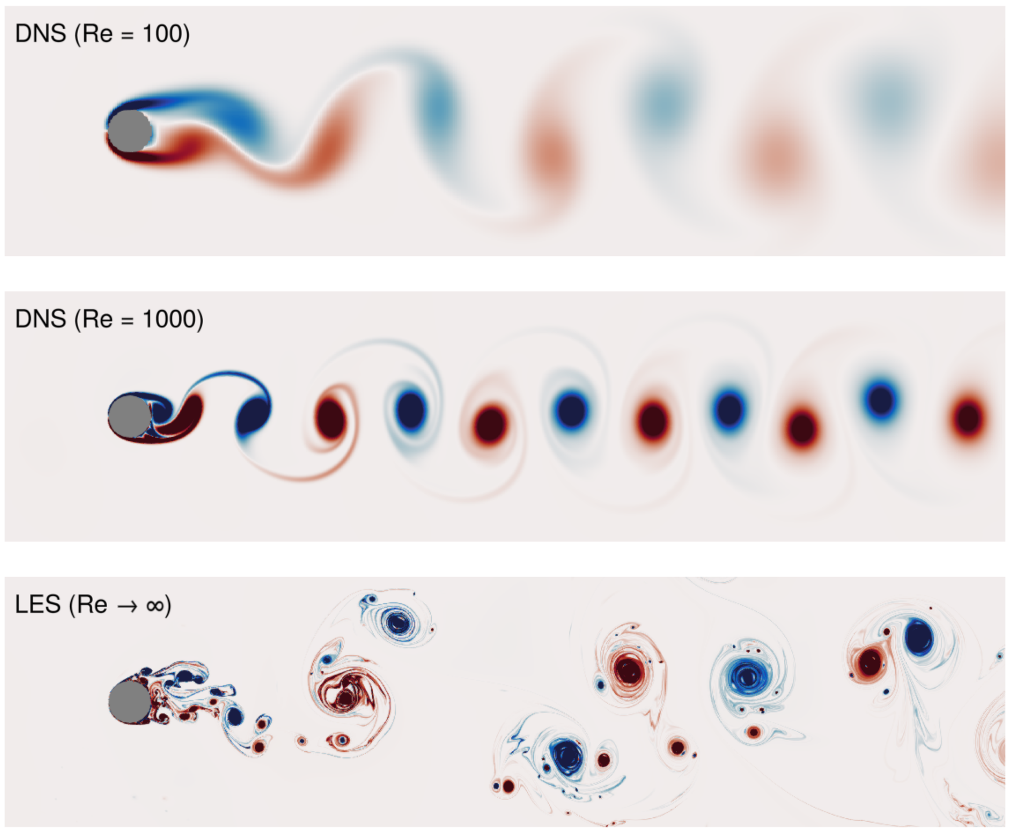

Figure 2: Vorticity snapshots in simulations of flow around a cylinder. The top two panels show vorticity in direct numerical simulations (DNS) that use a molecular ScalarDiffusivity closure and Centered(order=2) advection. The bottom panel shows a large eddy simulation (LES) with no closure and a WENO(order=9)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

```
r, U, Re, Ny = 1/2, 1, 1000, 2048

grid = RectilinearGrid(GPU(), size=(2Ny, Ny), x=(-3, 21), y=(-6, 6),
                              topology=(Periodic, Bounded, Flat))

cylinder(x, y) = (x^2 + y^2) ≤ r^2
grid = ImmersedBoundaryGrid(grid, GridFittedBoundary(cylinder))

closure = ScalarDiffusivity(ν=1/Re)

no_slip = FieldBoundaryConditions(immersed=ValueBoundaryCondition(0))
boundary_conditions = (u=no_slip, v=no_slip)

# Implement a sponge layer on the right side of the domain that
# relaxes v → 0 and u → U over a region of thickness δ
@inline mask(x, y, δ=3, x₀=21) = max(zero(x), (x - x₀ + δ) / δ)
Fu = Relaxation(target=U; mask, rate=1)
Fv = Relaxation(target=0; mask, rate=1)

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 $\boldsymbol{u} = u_\infty \hat{\boldsymbol{x}}$ with $u_\infty = 1$. The experiment can be converted to a large eddy simulation (thereby sending $Re \to \infty$) 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.

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

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

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

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

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 Λ in thermal wind balance with a constant meridional buoyancy gradient $f\Lambda$, such that

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

```julia
grid = RectilinearGrid(GPU(), topology = (Bounded, Flat, Bounded),
                       size = (4096, 1024), x = (0, 2), z = (-0.5, 0))

closure = ScalarDiffusivity(ν=1.15e-6, κ=1e-7)

using SeawaterPolynomials: TEOS10EquationOfState
equation_of_state = TEOS10EquationOfState(reference_density=1000)

buoyancy = SeawaterBuoyancy(gravitational_acceleration = 9.81;
                            constant_salinity = 0, # set S=0 and simulate T only
                            equation_of_state)

model = NonhydrostaticModel(; grid, buoyancy, closure, tracers=:T)

Tᵢ(x, z) = z > -0.25 ? 7.55 : 1
Ξᵢ(x, z) = 1e-2 * randn()
set!(model, T=Tᵢ, u=Ξᵢ, v=Ξᵢ, w=Ξᵢ)
```

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

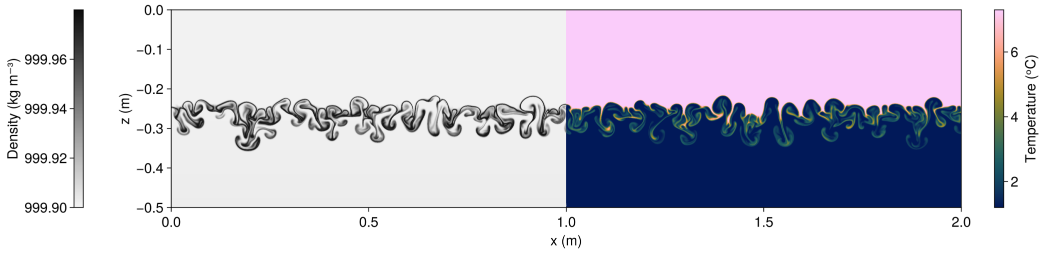

Figure 3: Density and temperature at $t = 1$ minute in a direct numerical simulation of cabelling in freshwater. Note that both fields span from $x = 0$ to $x = 2$ meters; only the left half of the density field and the right half of the temperature field are shown.

```
grid = RectilinearGrid(GPU(); size = (1024, 1024, 64),
                              x = (0, 4096), y = (0, 4096), z = (0, 128),
                              topology=(Periodic, Periodic, Bounded))

f, N², Ri = 1e-4, 1e-7, 1
parameters = (f=f, Λ=sqrt(N²/Ri)) # U = Λz, so Ri = N² / ∂z(U) = N² / Λ and Λ = N / √Ri.

@inline U(x, y, z, t, p) = + p.Λ * z
@inline B(x, y, z, t, p) = - p.f * p.Λ * y

background_fields = (u = BackgroundField(U; parameters),
                     b = BackgroundField(B; parameters))

model = NonhydrostaticModel(; grid, background_fields,
                              advection = WENO(order=9), coriolis = FPlane(; f),
                              tracers = :b, buoyancy = BuoyancyTracer())

Δz = minimum_zspacing(grid)
bᵢ(x, y, z) = N² * z + 1e-2 * N² * Δz * (2rand() - 1)
set!(model, b=bᵢ)
```

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

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 $N^2 = 10^{-7}\,\mathrm{s}^{-2}$ 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 Λ 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 $4\,\mathrm{km} \times 4\,\mathrm{km} \times 128\,\mathrm{m}$ 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 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

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

where $\delta = L/2$ represents the scale of the bathymetry, L is the half-channel width in y (the total width is $2L$), and $H = 128\,\mathrm{m}$ 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

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

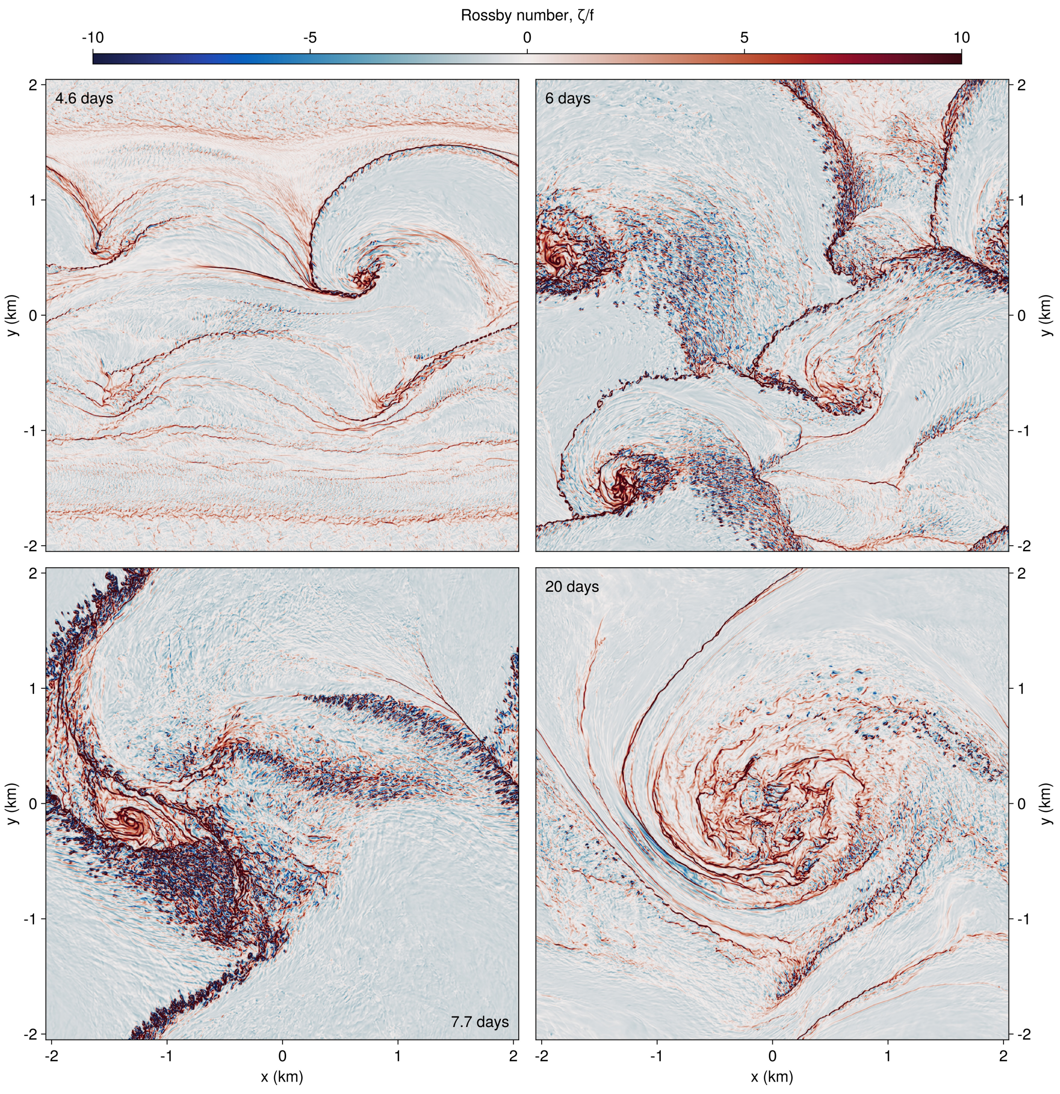

Figure 4: Surface vertical vorticity in a large eddy simulation of the Eady problem with $Ri = 1$ initially, after $t = 4.6$, 6, 7.7, and 20 days. The grid spacing is $4 \times 4 \times 2$ meters in x, y, z. Part of the script that produces this simulation is show in listing 6.

```julia
H, L = 256meters, 1024meters
δ = L / 2
x, y, z = (-3L, 3L), (-L, L), (-H, 0)
Nz = 64

grid = RectilinearGrid(GPU(); size=(6Nz, 2Nz, Nz), halo=(6, 6, 6),
                       x, y, z, topology=(Bounded, Bounded, Bounded))

wedge(x, y) = -H *(1 + (y + abs(x)) / δ)
grid = ImmersedBoundaryGrid(grid, GridFittedBottom(wedge))

T₂ = 12.421hours
U₂ = 0.1 # m/s

@inline Fu(x, y, z, t, p) = 2π * p.U₂ / p.T₂ * cos(2π * t / p.T₂)
@inline U(x, y, z, t, p) = p.U₂ * sin(2π * t / p.T₂)
@inline U(y, z, t, p) = U(zero(y), y, z, t, p)

open_bc = PerturbationAdvectionOpenBoundaryCondition(U; inflow_timescale = 2minutes,
                                                        outflow_timescale = 2minutes,
                                                        parameters=(; U₂, T₂))

u_bcs = FieldBoundaryConditions(east = open_bc, west = open_bc)

@inline ambient_temperature(x, z, t, H) = 12 + 4z/H
@inline ambient_temperature(x, y, z, t, H) = ambient_temperature(x, z, t, H)
ambient_temperature_bc = ValueBoundaryCondition(ambient_temperature; parameters = H)
T_bcs = FieldBoundaryConditions(east = ambient_temperature_bc,
                                west = ambient_temperature_bc)

ambient_salinity_bc = ValueBoundaryCondition(32)
S_bcs = FieldBoundaryConditions(east = ambient_salinity_bc, west = ambient_salinity_bc)

buoyancy = SeawaterBuoyancy(equation_of_state=TEOS10EquationOfState())

model = NonhydrostaticModel(; grid, buoyancy,
                              tracers = (:T, :S),
                              advection = WENO(order=9),
                              coriolis = FPlane(latitude=47.5),
                              boundary_conditions = (; T=T_bcs, u = u_bcs, S = S_bcs))

Tᵢ(x, y, z) = ambient_temperature(x, y, z, 0, H)

set!(model, T=Tᵢ, S=32, u=U(0, 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).

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

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

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 HydrostaticFreeSurface-Model, means that the vertical momentum equation used by NonhydrostaticModel, $\hat{\boldsymbol{z}} \cdot (6)$, is replaced by a statement of hydrostatic balance,

$$\partial_z p = b\,, \tag{14}$$

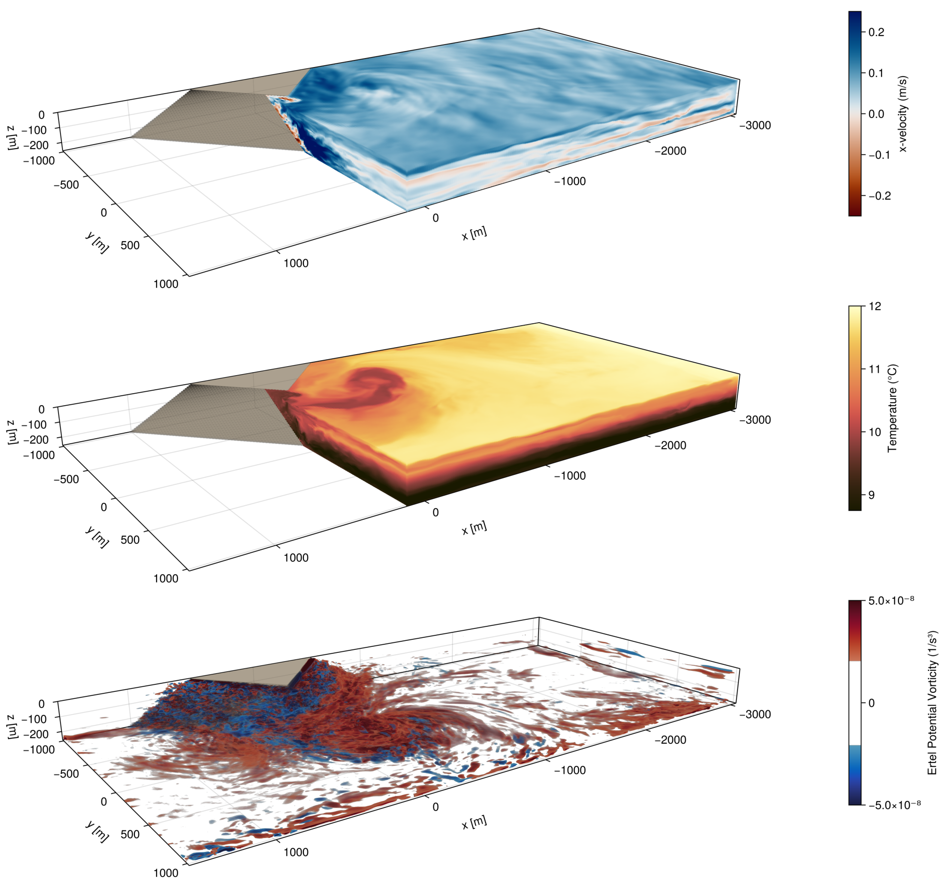

Figure 5: Along-channel velocity, temperature, and Ertel potential vorticity in a tidally-forced flow past an idealized headland with open boundaries. The tidal flow occurs in the x-directions and the snapshot depicts the flow just after the tide has turned to the negative-x direction.

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

$$\partial_z w = -\boldsymbol{\nabla}_h \cdot \boldsymbol{u}_h \, . \tag{15}$$

As a result, time-stepping the HydrostaticFreeSurfaceModel does require solving a three-dimensional Poisson equation for pressure. Moreover, the HydrostaticFreeSurfaceModel introduces a free surface displacement η, which obeys the linearized equation

$$\partial_t \eta = w|_{z=0} \, . \tag{16}$$

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

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

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

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

$$(\boldsymbol{u} \cdot \boldsymbol{\nabla}) \cdot \boldsymbol{u}_h = \begin{cases} \boldsymbol{\nabla} \cdot (\boldsymbol{u}\, \boldsymbol{u}_h) & \text{“flux form”} \, , \\ \zeta\, \hat{\boldsymbol{z}} \times \boldsymbol{u}_h + w\, \partial_z \boldsymbol{u}_h + \boldsymbol{\nabla}_h \tfrac{1}{2} |\boldsymbol{u}_h|^2 & \text{VectorInvariant} \, , \\ \zeta\, \hat{\boldsymbol{z}} \times \boldsymbol{u}_h - \boldsymbol{u}_h\, \partial_z w + \partial_z\, (w\, \boldsymbol{u}_h) + \boldsymbol{\nabla}_h \tfrac{1}{2} |\boldsymbol{u}_h|^2 & \text{WENOVectorInvariant} \, , \end{cases} \tag{18}$$

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

Tracer evolution is governed by the conservation law

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

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

Listing 8 implements a simulation of tidally-forced stratified flow over a series of randomly-positioned Gaussian seamounts. Results are plotted 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 $O(100\,\mathrm{m})$ to $O(100\,\mathrm{km})$.

```julia
using Oceananigans, Oceananigans.Units

grid = RectilinearGrid(size = (2000, 200),
                       x = (-1000kilometers, 1000kilometers),
                       z = (-2kilometers, 0),
                       halo = (4, 4),
                       topology = (Periodic, Flat, Bounded))

h₀ = 100          # typical mountain height (m)
δ  = 20kilometers # mountain width (m)
seamounts = 42
W = grid.Lx - 4δ
x₀ = W .* (rand(seamounts) .- 1/2) # mountains' positions ∈ [-Lx/2+2δ, Lx/2-2δ]
h  = h₀ .* (1 .+ rand(seamounts))  # mountains' heights ∈ [h₀, 2h₀]

bottom(x) = -grid.Lz + sum(h[s] * exp(-(x - x₀[s])^2 / 2δ^2) for s = 1:seamounts)
grid = ImmersedBoundaryGrid(grid, GridFittedBottom(bottom))

T₂ = 12.421hours # period of M₂ tide constituent
@inline tidal_forcing(x, z, t, p) = p.U₂ * 2π / p.T₂ * sin(2π / p.T₂ * t)
u_forcing = Forcing(tidal_forcing, parameters=(; U₂=0.1, T₂=T₂))

model = HydrostaticFreeSurfaceModel(; grid, tracers=:b, buoyancy=BuoyancyTracer(),
                                      momentum_advection = WENO(),
                                      tracer_advection = WENO(),
                                      forcing = (; u = u_forcing))

bᵢ(x, z) = 1e-5 * z
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.

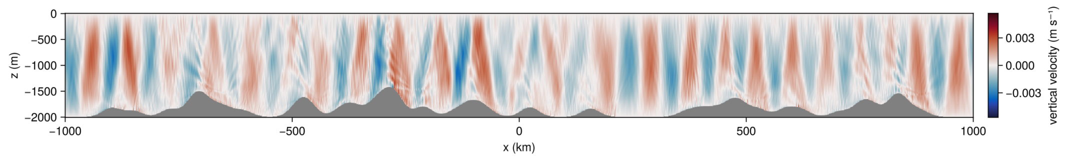

Figure 6: Vertical velocity of an internal wave field excited by tidally-forced stratified flow over superposition of randomly-positioned Gaussian seamounts, after 16 tidal periods.

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-ϵ (Umlauf & Burchard, 2005), which has two additional equations for TKE and the TKE dissipation rate. Figure 7 plots the result, showing how k-ϵ 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-ϵ parameters are calibrated one-by-one by referencing laboratory experiments and observations of increasing complexity (Umlauf & Burchard, 2003). Calibrating k-ϵ 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

```julia
using Oceananigans
using Oceananigans.Units

function vertical_mixing_simulation(closure; N²=1e-5, Jb=1e-7, tx=-5e-4)
    grid = RectilinearGrid(size=50, z=(-200, 0), topology=(Flat, Flat, Bounded))
    buoyancy = BuoyancyTracer()

    b_bcs = FieldBoundaryConditions(top=FluxBoundaryCondition(Jb))
    u_bcs = FieldBoundaryConditions(top=FluxBoundaryCondition(tx))

    if closure isa CATKEVerticalDiffusivity
        tracers = (:b, :e)
    elseif closure isa TKEDissipationVerticalDiffusivity
        tracers = (:b, :e, :ϵ)
    end

    model = HydrostaticFreeSurfaceModel(; grid, closure, tracers, buoyancy,
                                        boundary_conditions=(u=u_bcs, b=b_bcs))

    bᵢ(z) = N² * z
    set!(model, b=bᵢ)

    simulation = Simulation(model, Δt=1minute, stop_time=24hours)
    return run!(simulation)
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.

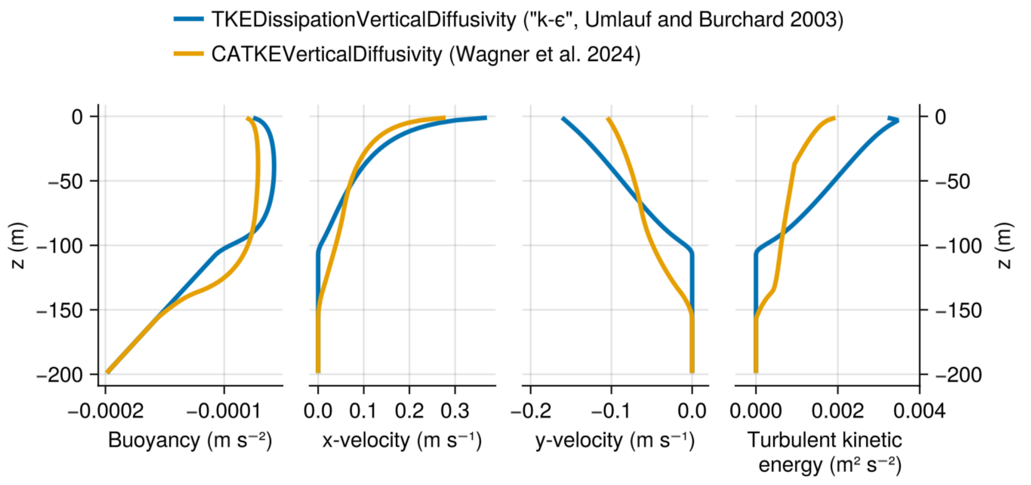

Figure 7: Results from two vertical mixing parameterizations: CATKE and k-ϵ, implemented as described in Listing 9.

```julia
Nx, Ny, Nz = 4320, 1800, 40 # 1/12th degree
z_faces = ClimaOcean.exponential_z_faces(; Nz, depth=6000)
partition = Partition(8) # Distribute simulation across 8 GPUs
arch = Distributed(GPU(); partition)
grid = LatitudeLongitudeGrid(arch; size=(Nx, Ny, Nz), halo=(7, 7, 7),
                             longitude=(0, 360), latitude=(-75, 75), z=z_faces)

bathymetry = ClimaOcean.regrid_bathymetry(grid) # based on ETOPO1
grid = ImmersedBoundaryGrid(grid, GridFittedBottom(bathymetry))

# Build an ocean simulation initialized to the ECCO state estimate on Jan 1, 1993
ocean = ClimaOcean.ocean_simulation(grid)
date  = CFTime.DateTimeProlepticGregorian(1993, 1, 1)
set!(ocean.model, T = ClimaOcean.ECCOMetadata(:temperature; date),
                  S = ClimaOcean.ECCOMetadata(:salinity; date))

# Near-global ocean simulation without no sea ice, forced by JRA55 reanalysis
backend = ClimaOcean.JRA55NetCDFBackend(41))
atmosphere = ClimaOcean.JRA55_prescribed_atmosphere(arch; backend)
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 & Obukhov, 1954) 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 $c = \mathrm{e}^x y$ on three different grids that cover the unit square. At the finest resolution, each cell-averaged value c_{ij}^{fine} is computed approximately using set! to evaluate $\mathrm{e}^x y$ 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}^{medium} and c_{ij}^{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,

$$\int c \, \mathrm{d}x \, \mathrm{d}y = \sum_{i,j}^{1024,1024} \mathcal{V}_{ij}^{\mathrm{fine}} c_{ij}^{\mathrm{fine}} = \sum_{i,j}^{16,16} \mathcal{V}_{ij}^{\mathrm{medium}} c_{ij}^{\mathrm{medium}} = \sum_{i,j}^{4,4} \mathcal{V}_{ij}^{\mathrm{coarse}} c_{ij}^{\mathrm{coarse}} \,, \tag{20}$$

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.

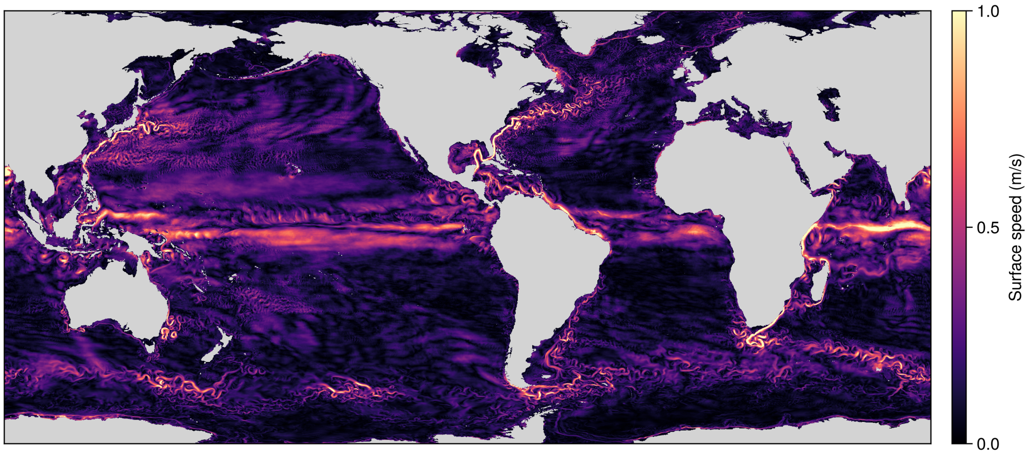

Figure 8: Surface speed in a near-global ocean simulation at 1/12th degree forced by JRA55 atmospheric reanalysis (Tsujino et al., 2018) initialized from the ECCO state estimate (Forget et al., 2015). Oceananigans can also cover the entirety of Earth's global ocean using a tripolar grid (Murray, 1996).

```julia
topology = (Bounded, Bounded, Flat)
x = y = (0, 1)
c(x, y) = exp(x) * y

fine_grid = RectilinearGrid(size=(1024, 1024); x, y, topology)
c_fine = CenterField(fine_grid)
set!(c_fine, c)

medium_grid = RectilinearGrid(size=(16, 16); x, y, topology)
c_medium = CenterField(medium_grid)
regrid!(c_medium, c_fine)

coarse_grid = RectilinearGrid(size=(4, 4); x, y, topology)
c_coarse = CenterField(coarse_grid)
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 $\boldsymbol{\nabla} \cdot \boldsymbol{u}$ are co-located, and cells for velocity components $\boldsymbol{u} = (u, v, w)$ 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.

```julia
using Oceananigans

grid = RectilinearGrid(topology=(Bounded, Flat, Flat), size=4, x=[0, 0.2, 0.3, 0.7, 1])

u = Field{Face, Center, Center}(grid)
c = Field{Center, Center, Center}(grid)

xnodes(u)        # [0.0, 0.2, 0.3, 0.7, 1.0]
xnodes(c)        # [0.1, 0.25, 0.5, 0.85]
location(∂x(c)) # (Face, Center, Center)
```

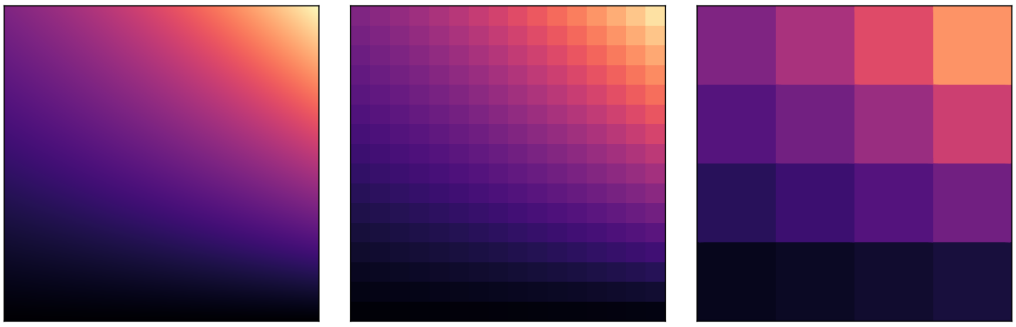

Figure 9: Finite volume discretization of $e^x y$ on the unit square at three different resolutions.

Listing 12: A one-dimensional staggered grid.

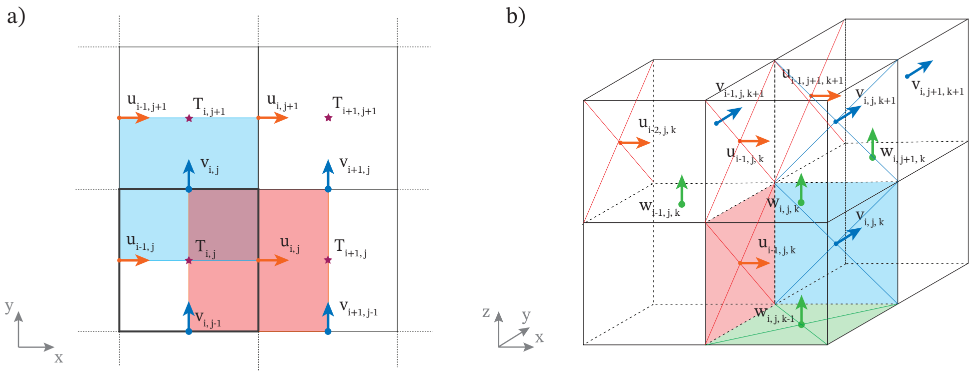

Figure 10: Locations of cell centers and interfaces on a two-dimensional (a) and three-dimensional (b) staggered grid. In (a), the red and blue shaded regions highlight the volumes in the dual u-grid and v-grid, located at (Face, Center, Center) and (Center, Face, Center), respectively. In (b), the shaded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

4.1 A system of composable operators

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,

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

where superscripts denote the location of the *result* of the operation. For example, the difference δ_x^{fcc} acts on fields located at ccc (meaning cell Center in the x, y and z directions respectively). Complementary to the difference operators are reconstruction of "interpolation" operators,

```julia
ℑxᶠᶜᶜ(i, j, k, grid, c) = (c[i, j, k] + c[i-1, j, k]) / 2
ℑxᶜᶜᶜ(i, j, k, grid, u) = (u[i+1, j, k] + u[i, j, k]) / 2
```

The prefix arguments `i, j, k, grid` are more than convention: the prefix enables system for *composing* operators. For example, defining

```julia
δxᶠᶜᶜ(i, j, k, grid, f::Function, args...) =
    f(i, j, k, grid, args...) - f(i-1, j, k, grid, args...)

δxᶜᶜᶜ(i, j, k, grid, f::Function, args...) =
    f(i+1, j, k, grid, args...) - f(i, j, k, grid, args...)
```

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

```julia
δ²xᶜᶜᶜ(i, j, k, grid, f::Function, a...) = δxᶜᶜᶜ(i, j, k, grid, δxᶠᶜᶜ, f, a...)
```

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

4.2 Tracer flux divergences, advection schemes, and reconstruction

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

$$\boldsymbol{\nabla \cdot J} \approx \frac{1}{\mathcal{V}_c}\Big[\delta_x\big(\underbrace{\mathcal{A}_x J_x}_{\text{fcc}}\big) + \delta_y\big(\underbrace{\mathcal{A}_y J_y}_{\text{cfc}}\big) + \delta_z\big(\underbrace{\mathcal{A}_z J_z}_{\text{ccf}}\big)\Big] \, , \tag{21}$$

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

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

$$\boldsymbol{u \cdot \nabla} c = \boldsymbol{\nabla \cdot} (\boldsymbol{u} c) \approx \frac{1}{\mathcal{V}_c}\Big[\delta_x\big(\mathcal{A}_x u \lfloor c \rceil_x\big) + \delta_y\big(\mathcal{A}_y v \lfloor c \rceil_y\big) + \delta_z\big(\mathcal{A}_z w \lfloor c \rceil_z\big)\Big] \, , \tag{22}$$

where $\lfloor c \rceil_x$ denotes a *reconstruction* of c in the x-direction from its native location ccc to the tracer cell interface at fcc; $\lfloor c \rceil_y$ and $\lfloor c \rceil_z$ in (22) are defined similarly.

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

$$\langle c \rangle_i = \tfrac{1}{2}\left(c_i + c_{i-1}\right) \, , \tag{23}$$

where $\langle c \rangle_i$ denotes the centered reconstruction of c on the interface at $x = x_{i-1/2}$. Also in (23) the j, k indices are implied and we have suppressed the direction x to lighten the 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

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

where $[c]_x^N$ denotes N^{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

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

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

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

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 $u > 0$,

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

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

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

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

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

```julia
using Oceananigans

grid = RectilinearGrid(size=128; x=(-4, 8), halo=6, topology=(Periodic, Flat, Flat))
advection = WENO(order=9) # Centered(order=2), UpwindBiased(order=3)
velocities = PrescribedVelocityFields(u=1)
model = HydrostaticFreeSurfaceModel(; grid, velocities, advection, tracers=:c)

top_hat(x) = abs(x) > 1 ? 0 : 1
set!(model, c = top_hat)

simulation = Simulation(model, Δt=1/grid.Nx, stop_time=4)
run!(simulation)
```

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).

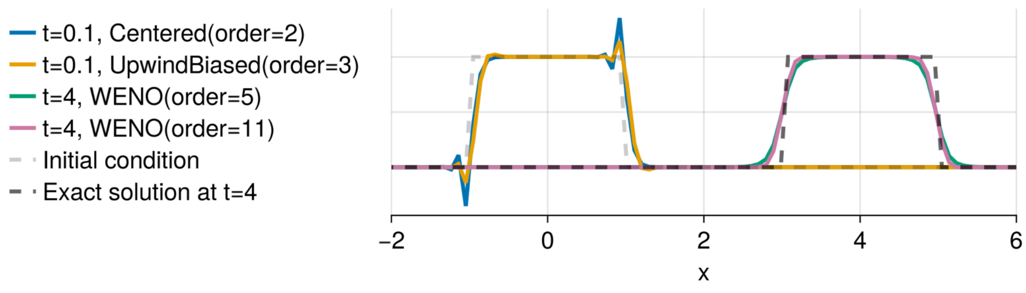

Figure 11: Advection of a top hat tracer distribution in one-dimension using various advection schemes. Centered and Upwind

4.2.1 Discretization of momentum advection

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

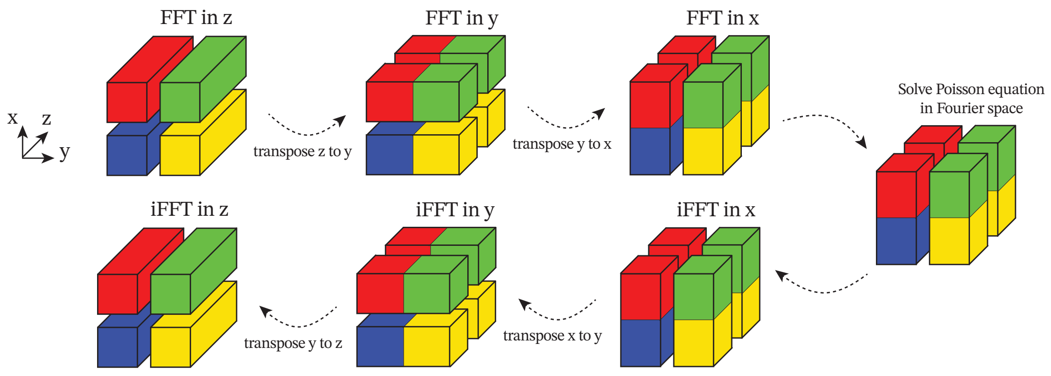

Figure 12: A schematic showing the distributed Poisson solver procedure with a pencil parallelization that divides the domain in two ranks in both x and y. The schematic highlights the data layout in the ranks during each operation.

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

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

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

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

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

Appendix A Time stepping and time discretization

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

A1 Time discretization for tracers

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

$$\partial_t c = G_c + \partial_z \left(\kappa_z \partial_z c \right) \, , \tag{A1}$$

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

$$\left(1 - \Delta t \, \partial_z \, \kappa_z^m \partial_z \right) c^{m+1} = c^m + \int_{t^m}^{t^{m+1}} G_c \, \mathrm{d}t \, , \tag{A2}$$

where $\Delta t \overset{\text{def}}{=} t^{m+1} - t^m$. The tendency integral $\int_{t^m}^{t^{m+1}} G_c \, \mathrm{d}t$ is evaluated either using a "quasi"-second order Adams-Bashforth scheme (QAB2, which is actually first-order *lets add a reference*), or a low-storage third-order Runge-Kutta scheme (RK3). For QAB2, the integral in (A2) spans the entire time-step and takes the form

$$\frac{1}{\Delta t} \int_{t^m}^{t^{m+1}} G_c \, \mathrm{d}t \approx \left(\tfrac{3}{2} + \chi\right) G_c^m - \left(\tfrac{1}{2} + \chi\right) G_c^{m-1} \, , \tag{A3}$$

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

$$\frac{1}{\Delta t} \int_{t^m}^{t^{m+1}} G_c \, \mathrm{d}t \approx \gamma^m G_c^m - \zeta^m G_c^{m-1} \, , \tag{A4}$$

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

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

```
using Oceananigans

grid = RectilinearGrid(size=20, z=(-2, 2), topology=(Flat, Flat, Bounded))
time_discretization = VerticallyImplicitTimeDiscretization()
κ(z, t) = exp(-z^2)
closure = VerticalScalarDiffusivity(time_discretization; κ)
model = HydrostaticFreeSurfaceModel(; grid, closure, tracers=:c)
```

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

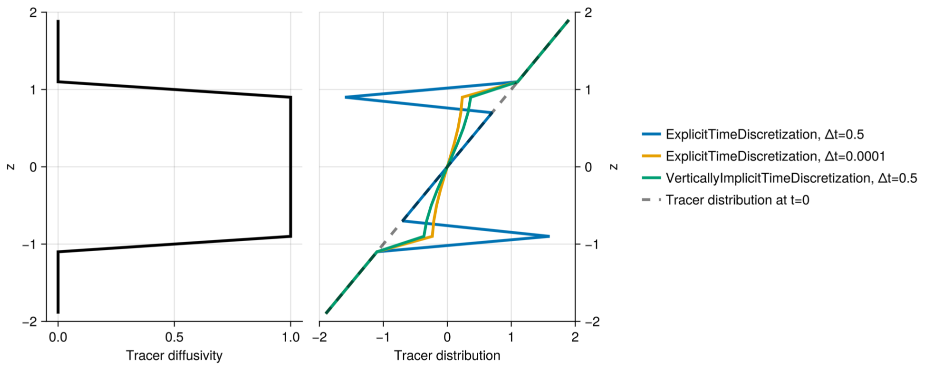

Figure A1: Simulations of tracer diffusion by a top hat diffusivity profile using various choices of time-discretization and time-step size. With a long time-step of $\Delta t = 0.5$, ExplicitTimeDiscretization is unstable while VerticallyImplicitTimeDiscretization is stable. Let the vertically-implicit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

A2 The pressure correction method for momentum in NonhydrostaticModel

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

$$\partial_t \boldsymbol{u} = -\boldsymbol{\nabla} p + b \hat{\boldsymbol{z}} + \boldsymbol{G}_u + \partial_z \left(\nu_z \partial_z \boldsymbol{u} \right) , \tag{A5}$$

where, if specified, ν_z is the vertical component of the viscosity that will be treated with a vertically-implicit time discretization, $\boldsymbol{\nabla} p$ is the total pressure gradient, and $\boldsymbol{G}_u$ is the rest of the momentum tendency. We decompose p into a "hydrostatic anomaly" p' tied to the density anomaly ρ', and a nonhydrostatic component $\tilde{p}$, such that

$$p = \tilde{p} + p' , \qquad \text{where} \qquad \partial_z p' \overset{\text{def}}{=} b \, . \tag{A6}$$

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

$$\boldsymbol{u}^{m+1} = \boldsymbol{u}^m + \int_{t^m}^{t^{m+1}} \left[\boldsymbol{G}_u - \boldsymbol{\nabla} \tilde{p} + \partial_z \left(\nu_z \partial_z \boldsymbol{u} \right) \right] \mathrm{d}t \, . \tag{A7}$$

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

$$\left(1 - \Delta t \, \partial_z \nu_z^m \partial_z \right) \tilde{\boldsymbol{u}} = \boldsymbol{u}^m + \int_{t^m}^{t^{m+1}} \boldsymbol{G}_u \, \mathrm{d}t \, , \tag{A8}$$

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

$$\int_{t^m}^{t^{m+1}} \partial_z \left(\nu_z \partial_z \boldsymbol{u} \right) \mathrm{d}t \approx \Delta t \, \partial_z \left(\nu_z^m \partial_z \tilde{\boldsymbol{u}} \right) . \tag{A9}$$

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

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

$$\int_{t^m}^{t^{m+1}} \boldsymbol{\nabla} \tilde{p} \, \mathrm{d}t \approx \Delta t \, \boldsymbol{\nabla} \tilde{p}^{m+1} , \tag{A10}$$

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

$$\boldsymbol{u}^{m+1} - \tilde{\boldsymbol{u}} = -\Delta t \, \boldsymbol{\nabla} \tilde{p}^{m+1} \, . \tag{A11}$$

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

$$\boldsymbol{\nabla}^2 \tilde{p}^{m+1} = \frac{\boldsymbol{\nabla} \cdot \tilde{\boldsymbol{u}}}{\Delta t} \, . \tag{A12}$$

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

On RectilinearGrids, we solve (A12) using an eigenfunction expansion of the discrete second-order Poisson operator $\boldsymbol{\nabla}^2$ evaluated via the fast Fourier transform (FFT) in equispaced directions (Schumann & Sweet, 1988) plus a tridiagonal solve in variably-spaced directions. With the FFT-based solver, boundary conditions on $\tilde{p}^{m+1}$ are accounted for by enforcing $\hat{\boldsymbol{n}} \cdot \tilde{\boldsymbol{u}} = \hat{\boldsymbol{n}} \cdot \boldsymbol{u}^{m+1}$ on boundary cells — which is additional and separate from the definition $\tilde{\boldsymbol{u}}$ in (A9). This alteration of $\tilde{\boldsymbol{u}}$ on the boundary implicitly contributes the appropriate terms that account for inhomogeneous boundary-normal pressure gradients $\hat{\boldsymbol{n}} \cdot \boldsymbol{\nabla} \tilde{p}^{m+1} \neq 0$ to the right-hand-side of (A12) during the computation of $\boldsymbol{\nabla} \cdot \tilde{\boldsymbol{u}}$.

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

A3 Time discretization of the HydrostaticFreeSurfaceModel

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

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

$$\boldsymbol{U}_h \overset{\text{def}}{=} \int_{-H}^{\eta} \boldsymbol{u}_h \, \mathrm{d}z \,, \tag{A13}$$

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

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

$$\partial_t \boldsymbol{U}_h = -g(H+\eta) \boldsymbol{\nabla}_h \eta + \int_{-H}^{\eta} \boldsymbol{G}_{uh} \, \mathrm{d}z \,, \tag{A14}$$

where $\boldsymbol{G}_{uh}$ includes all the tendency terms that evolve "slowly" compared to the barotropic mode:

$$\boldsymbol{G}_{uh} = -(\boldsymbol{u} \cdot \boldsymbol{\nabla}) \boldsymbol{u}_h - \boldsymbol{f} \times \boldsymbol{u} - \boldsymbol{\nabla} \cdot \boldsymbol{\tau} + \boldsymbol{F}_h \,. \tag{A15}$$

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

$$\partial_t \eta = -\boldsymbol{\nabla}_h \cdot \boldsymbol{U}_h \,. \tag{A16}$$

The pair of equations (A14) and (A16) characterize the evolution of the barotropic mode, which involves faster time-scales than the baroclinic mode evolution described by equations (17). To resolve both modes, we use a split-explicit algorithm where the barotropic mode is advanced in time using a smaller time-step than the one used for three-dimensional baroclinic variables. In particular, a predictor three-dimensional velocity is evolved without accounting 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.

$$(1 - \Delta t \, \partial_z \nu_z^m \partial_z) \tilde{\boldsymbol{u}}_h - \boldsymbol{u}_h^m \approx \int_{t^m}^{t^{m+1}} \boldsymbol{G}_{uh} \, \mathrm{d}t \,. \tag{A17}$$

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 $\Delta\tau = \Delta t / N$,

$$\eta^{n+1} - \eta^n = -\Delta\tau \boldsymbol{\nabla}_h \cdot \boldsymbol{U}_h^n \,, \tag{A18}$$

Table B1: DNS: Direct Numerical Simulation. LES: Large Eddy Simulation.

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

$$\boldsymbol{U}_h^{n+1} - \boldsymbol{U}_h^n = -\Delta\tau \left[g(H+\eta)\boldsymbol{\nabla}_h \eta^{n+1} - \frac{1}{\Delta t}\int_{-H}^{\eta}\int_{t^m}^{t^{m+1}} \boldsymbol{G}_{uh}\,\mathrm{d}t\,\mathrm{d}z \right] . \tag{A19}$$

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

$$\bar{\boldsymbol{U}}_h = \sum_{n=1}^{M} a_n \boldsymbol{U}_h^n \,, \quad \bar{\eta} = \sum_{n=1}^{M} a_n \eta^n \,, \tag{A20}$$

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

$$\boldsymbol{u}_h^{m+1} = \tilde{\boldsymbol{u}}_h + \frac{1}{H+\eta}\left(\bar{\boldsymbol{U}}_h - \int_{-H}^{\eta} \tilde{\boldsymbol{u}}_h \,\mathrm{d}z \right) . \tag{A21}$$

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

Appendix B Table of numerical examples

Open Research Section

Oceananigans is available at the GitHub repository github.com/CliMA/Oceananigans.jl and ClimaOcean is available at github.com/CliMA/ClimaOcean.jl. Oceananigans documentation lives at clima.github.io/OceananigansDocumentation. All the scripts that reproduce the simulations and figures in this paper are available at the GitHub repository github.com/glwagner/OceananigansPaper. Visualizations were made using Makie.jl (Danisch & Krumbiegel, 2021).

Acknowledgments
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. R., 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.

Edson, J. B., Jampana, V., Weller, R. A., Bigorre, S. P., Plueddemann, A. J., Fairall, C. W., ... Hersbach, H. (2014). On the exchange of momentum over the open ocean. *Journal of Physical Oceanography*, *44*(9), 1589.

Forget, G., Campin, J.-M., Heimbach, P., Hill, C., Ponte, R., & Wunsch, C. (2015). ECCO version 4: An integrated framework for non-linear inverse modeling and global ocean state estimation. *Geoscientific Model Development*, *8*(10), 3071–3104.

Ghosh, D., & Baeder, J. D. (2012). High-order accurate incompressible Navier–Stokes algorithm for vortex-ring interactions with solid wall. *AIAA journal*, *50*(11), 2408–2422.

Godunov, S. K. (1959). A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations. *Matematicheskii Sbornik*, *47*, 271–306. (Translated by US Joint Publications Research Service, JPRS 7226, 1969)

Grachev, A. A., Andreas, E. L., Fairall, C. W., Guest, P. S., & Persson, P. O. G. (2007). SHEBA flux–profile relationships in the stable atmospheric boundary layer. *Boundary-layer meteorology*, *124*, 315–333.

Griffies, S. M., Adcroft, A., & Hallberg, R. W. (2020). A primer on the vertical lagrangian-remap method in ocean models based on finite volume generalized vertical coordinates. *Journal of Advances in Modeling Earth Systems*, *12*(10), e2019MS001954.

Griffies, S. M., Pacanowski, R. C., & Hallberg, R. W. (2000). Spurious diapycnal mixing associated with advection in a z-coordinate ocean model. *Monthly Weather Review*, *128*(3), 538–564.

Griffies, S. M., Stouffer, R. J., Adcroft, A. J., Bryan, K., Dixon, K. W., Hallberg, R., ... Rosati, A. (2015). A historical introduction to MOM. *URL https://www. gfdl. noaa. gov/wp-content/uploads/2019/04/mom_history_2017*, *9*.

Häfner, D., Nuterman, R., & Jochum, M. (2021). Fast, cheap, and turbulent—global ocean modeling with GPU acceleration in Python. *Journal of Advances in Modeling Earth Systems*, *13*(12), e2021MS002717.

Halliwell, G. R. (2004). Evaluation of vertical coordinate and vertical mixing algorithms in the HYbrid-Coordinate Ocean Model (HYCOM). *Ocean Modelling*, *7*(3-4), 285–322.

Held, I. M. (2005). The gap between simulation and understanding in climate modeling. *Bulletin of the American Meteorological Society*, *86*(11), 1609–1614.

Huang, N. E. (1979). On surface drift currents in the ocean. *Journal of Fluid Mechanics*, *91*(1), 191–208.

Kärnä, T., Kramer, S. C., Mitchell, L., Ham, D. A., Piggott, M. D., & Baptista, A. M. (2018). Thetis coastal ocean model: discontinuous Galerkin discretization for the three-dimensional hydrostatic equations. *Geoscientific Model Development*, *11*(11), 4359–4382.

Kiss, A. E., Hogg, A. M., Hannah, N., Boeira Dias, F., Brassington, G. B., Chamberlain, M. A., ... others (2020). Access-om2 v1. 0: a global ocean–sea ice model at three resolutions. *Geoscientific Model Development*, *13*(2), 401–442.

Klöwer, M., Gelbrecht, M., Hotta, D., Willmert, J., Silvestri, S., Wagner, G. L., ... others (2024). Speedyweather. jl: Reinventing atmospheric general circulation models towards interactivity and extensibility. *Journal of Open Source Software*, *9*(98), 6323.

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., ... others (2024). Neural general circulation models for weather and climate. *Nature*, *632*(8027), 1060–1066.

Korn, P., Brüggemann, N., Jungclaus, J. H., Lorenz, S., Gutjahr, O., Haak, H., ... others (2022). ICON-O: The Ocean Component of the ICON Earth System Model—Global Simulation Characteristics and Local Telescoping Capability. *Journal of Advances in*

Modeling Earth Systems, *14*(10), e2021MS002952.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., ... others (2023). Learning skillful medium-range global weather forecasting. *Science*, *382*(6677), 1416–1421.

Leclair, M., & Madec, G. (2011). z-Coordinate, an Arbitrary Lagrangian–Eulerian coordinate separating high and low frequency motions. *Ocean Modelling*, *37*(3-4), 139–152.

Lilly, D. K. (1983). Stratified turbulence and the mesoscale variability of the atmosphere. *Journal of the Atmospheric Sciences*, *40*(3), 749–761.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., & Heisey, C. (1997). A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research: Oceans*, *102*(C3), 5753–5766.

Marshall, J., Hill, C., Perelman, L., & Adcroft, A. (1997). Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *Journal of Geophysical Research: Oceans*, *102*(C3), 5733–5752.

McDougall, T. J., & Barker, P. M. (2011). Getting started with TEOS-10 and the Gibbs Seawater (GSW) oceanographic toolbox. *Scor/iapso WG*, *127*(532), 1–28.

Molemaker, M. J., McWilliams, J. C., & Capet, X. (2010). Balanced and unbalanced routes to dissipation in an equilibrated eady flow. *Journal of Fluid Mechanics*, *654*, 35–63.

Monin, A. A., & Obukhov, A. M. (1954). Basic laws of turbulent mixing in the surface layer of the atmosphere. *Contrib. Geophys. Inst. Acad. Sci. USSR*, *151*(163), e187.

Moses, W. S., Churavy, V., Paehler, L., Hückelheim, J., Narayanan, S. H. K., Schanen, M., & Doerfert, J. (2021). Reverse-mode automatic differentiation and optimization of gpu kernels via enzyme. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–16).

Murray, R. J. (1996). Explicit generation of orthogonal grids for ocean models. *Journal of Computational Physics*, *126*(2), 251–273.

National Energy Research Scientific Computing Center. (2025). *Perlmutter architecture.* Retrieved from https://docs.nersc.gov/systems/perlmutter/architecture/ (Accessed: 2025-02-18)

Pawlak, G., MacCready, P., Edwards, K., & McCabe, R. (2003). Observations on the evolution of tidal vorticity at a stratified deep water headland. *Geophysical Research Letters*, *30*(24).

Pearson, B. (2018). Turbulence-induced anti-Stokes flow and the resulting limitations of large-eddy simulation. *Journal of Physical Oceanography*, *48*(1), 117–122.

Petersen, M. R., Jacobsen, D. W., Ringler, T. D., Hecht, M. W., & Maltrud, M. E. (2015). Evaluation of the arbitrary Lagrangian–Eulerian vertical coordinate method in the MPAS-Ocean model. *Ocean Modelling*, *86*, 93–113.

Phillips, N. A. (1956). The general circulation of the atmosphere: A numerical experiment. *Quarterly Journal of the Royal Meteorological Society*, *82*(352), 123–164.

Pressel, K. G., Kaul, C. M., Schneider, T., Tan, Z., & Mishra, S. (2015). Large-eddy simulation in an anelastic framework with closed water and entropy balances. *Journal of Advances in Modeling Earth Systems*, *7*(3), 1425–1456.

Pressel, K. G., Mishra, S., Schneider, T., Kaul, C. M., & Tan, Z. (2017). Numerics and subgrid-scale modeling in large eddy simulations of stratocumulus clouds. *Journal of Advances in Modeling Earth Systems*, *9*(2), 1342–1365.

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

Ringler, T., Petersen, M., Higdon, R. L., Jacobsen, D., Jones, P. W., & Maltrud, M. (2013). A multi-resolution approach to global ocean modeling. *Ocean Modelling*, *69*, 211–232.

Roquet, F., Madec, G., Brodeau, L., & Nycander, J. (2015). Defining a simplified yet "realistic" equation of state for seawater. *Journal of Physical Oceanography*, *45*(10), 2564–2579.

Roquet, F., Madec, G., McDougall, T. J., & Barker, P. M. (2015). Accurate polynomial expressions for the density and specific volume of seawater using the TEOS-10 standard.

Ocean Modelling, *90*, 29–43.

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

Schumann, U., & Sweet, R. A. (1988). Fast Fourier transforms for direct solution of Poisson's equation with staggered boundary conditions. *Journal of Computational Physics*, *75*(1), 123–137.

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. *Journal of Marine Research*, *61*.

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*.