DJ4Earth: Differentiable, and Performance-portable Earth System Modeling via Program Transformations

William S. Moses[1], Gong Cheng[2], Valentin Churavy[3], Maximilian Gelbrecht[4], Milan Klöwer[5], Joseph Kump[6], Mathieu Morlighem[2], Sarah Williamson[6], Dhruv Apte[6], Paul Berg[7], Mosè Giordano[8], Christopher Hill[9], Nora Loose[10], Alexis Montoison[11], Sri Hari Krishna Narayanan[11], Avik Pal[9], Michel Schanen[11], Simone Silvestri [9,12], Greg Wagner[9], and Patrick Heimbach[6]

[1]University of Illinois Urbana-Champaign, IL, USA
[2]Dartmouth College, NH, USA
[3]Johannes Gutenberg University Mainz & University of Augsburg, Germany
[4]Technical University of Munich & Potsdam Institute for Climate Impact Research, Germany
[5]University of Oxford, UK
[6]University of Texas at Austin, TX, USA
[7]Bern University of Applied Sciences, Switzerland
[8]University College London, UK
[9]Massachusetts Institute of Technology, MA, USA
[10][C]Worthy, LLC, USA
[11]Argonne National Laboratory, IL, USA
[12]Politecnico di Torino, Italy

Key Points:

- Four Earth system model components are successfully differentiated using the reverse mode of the automatic differentiation tool Enzyme.
- The Julia-based, GPU-enabled models use bespoke numerics, with finite-volume, finite-element, and spectral spatial discretization schemes.
- The compiler transpilation tool Reactant enables optimized, portable performance across diverse ML-customized HPC architectures.

Corresponding author: Patrick Heimbach, heimbach@utexas.edu

Abstract

Differentiable Earth system models (ESMs) enable powerful applications such as sensitivity analysis, gradient-based calibration, state estimation, boundary flux inversions, uncertainty quantification, and online machine learning. Reverse-mode automatic differentiation (AD) efficiently provides gradients for such tasks, yet models have rarely included this capability because of complex, bespoke numerical algorithms. As part of the *Differentiable programming in Julia for Earth system modeling (DJ4Earth)* initiative, we present enabling features that make general-purpose AD tractable and efficient for full-fledged ESM components written in Julia. The approach leverages the AD framework Enzyme.jl and the compiler transpilation tool Reactant.jl, augmented by sophisticated checkpointing algorithms. Operating at the Low-Level Virtual Machine (LLVM) intermediate representation or Multi-Level Intermediate Representation (MLIR) compiler levels, these frameworks support mutable memory, custom kernels, and compiler optimizations before and after differentiation. Julia-specific challenges related to just-in-time compilation and garbage collection are handled efficiently. Reactant further enables automatic performance portability across CPUs, GPUs, and TPUs, facilitating use of emerging AI-customized high-performance computing architectures. We demonstrate these frameworks on four Julia-based ESM components featuring diverse spatial discretizations and numerical algorithms: (i) the rotating-sphere shallow water model ShallowWaters.jl, (ii) the finite-volume ocean model Oceananigans.jl, (iii) the ice sheet model DJUICE.jl, and (iv) the spectral atmospheric model SpeedyWeather.jl. Across these ESM components, our tools compute efficient and correct gradients. These results establish a foundation for differentiable, high-performance and performance-portable ESMs that can integrate neural networks for unresolved processes, trained online, enabling next-generation hybrid physics–machine learning ESMs constrained by physical dynamics and observations.

Plain Language Summary

Earth system models are computer programs that simulate how Earth's atmosphere, ocean, ice, and biosphere interact and evolve. These models consist of millions of lines of code and rely on uncertain inputs. To improve accuracy, scientists adjust these inputs to minimize the difference between simulations and observations, measured by a "cost function". Another computer program can efficiently determine how changes in each input affect the outcome. This calculation, called the gradient of the cost function, would be extremely time-consuming to code manually. Instead, we use an automatic differentiation (AD) tool called Enzyme, which computes these gradients efficiently and updates them automatically whenever the model changes. As computing systems evolve rapidly, especially those optimized for artificial intelligence (AI), another tool called Reactant enables models to run efficiently across different hardware, from CPUs to GPUs and AI accelerators. We demonstrate these methods on four Earth system model components written in the modern programming language Julia: a shallow water model, an ocean model, an ice sheet model, and an atmospheric model. For each, the code generated via AD produces correct gradients of the cost function. This work lays the foundation for combining these differentiated models with machine learning to improve model accuracy efficiently.

1 Introduction

Earth system models (ESMs) provide a comprehensive framework for simulating weather, climate, hydrological resources, biogeochemical cycles, and cryospheric changes across a range of spatial and temporal scales (e.g., Randall et al. (2019)). These models prove useful in quantifying magnitudes and patterns of natural climate variability, determining the impact of climate change, and providing likely scenarios of the planet's

future climate to policy makers. ESMs consist of submodels or components representing the atmosphere, ocean, cryosphere, and biosphere. These components typically solve partial differential equations representing the conservation and constitutive laws for the component's state on a discretized space of the rotating planet. However, ESM components rely on parameterizations for subgrid-scale processes, such as turbulent mixing or unresolved mesoscale eddies in the ocean, air-sea fluxes of heat, humidity, momentum, or biogeochemical tracers (Christensen & Zanna, 2022). In the atmosphere, parameterizations also represent microphysics such as cloud formation and precipitation, processes that cannot be resolved even with higher resolution. Ice sheet models have to prescribe basal boundary conditions that cannot be estimated from remote sensing. These parameterizations and poorly constrained boundary conditions are sources of structural and parametric uncertainty. Their calibration relies on observational data or high-fidelity simulations. In the context of ESMs, parameter calibration or tuning has so far been conducted in a somewhat ad hoc fashion because of the computational cost and the underlying complexity of the problem (e.g., Hourdin et al. (2016); Balaji et al. (2022)). Ad hoc parameter calibration, together with initial condition uncertainty, is prerceived to be the primary reason ESM simulations have suffered persistent biases that may obscure predictive skill on weather to decadal time scales (Eyring et al., 2019).

Rigorous methods for model calibration whereby models "learn from data" have been underexplored in climate or Earth system modeling (Schneider et al., 2017, 2023). They rely either on ensemble methods (i.e., sampling) or gradient-based optimization, or a combination thereof. Each of these faces a distinct set of computational challenges. While ensemble methods have become the method of choice in various ESM applications, they suffer from a number of potential drawbacks: (1) in the context of comprehensive ESMs, they have mainly been applied to tackle initial condition uncertainty; (2) they suffer from the "curse of dimensionality": when initial condition and parametric uncertainty exhibit spatial structure, as is generally the case in geoscience applications, ensemble methods become computationally intractable as the ensemble size becomes excessively large; (3) many of the ensemble approaches used in ESMs (with the exception of rigorous data assimilation, such as Kalman filter or inversion) do not "learn from data" for calibration; and (4) structural model uncertainty is dealt with only in an ad hoc manner via multimodel or stochastic ensemble methods.

1.1 The Case for Differentiable ESMs

Some of the shortcomings listed above may be overcome through the use of gradient-based optimization, which is the subject of the well-established field of inverse estimation and control methods (Bryson & Ho, 1975; Tarantola, 2005; Wunsch, 2006). At its heart is the use of adjoint models, namely, models that efficiently compute the sensitivity of some scalar-valued model-data misfit or quantity of interest to a high-dimensional space of uncertain input or control variables, such as initial conditions, boundary conditions, or model parameters. Optimal input variables are then obtained through iterative nonlinear gradient-based optimization. The underlying adjoint model is the formal transpose of the tangent linear model of the (generally nonlinear) parent model. It can be obtained by hand-coding, as has been done, for example, in numerical weather prediction (Rabier et al., 2000) or regional ocean modeling (Moore et al., 2004), or through the use of automatic differentiation (AD) tools. AD computes derivatives by applying the chain rule of differentiation to elementary operations (e.g., Griewank & Walther (2008); Margossian (2019)). Reverse-mode AD generates the adjoint model (which computes gradients) rather than the tangent linear model (which computes directional derivatives), making gradient-based methods computationally tractable for large-scale applications. A key advantage of AD-generated over hand-coded adjoints is the ability to keep the adjoint model up to date with respect to ongoing developments of the parent model. Differentiable programming in the context of optimal estimation and control (or inverse)

methods consists of writing the parent model in a way that is amenable to efficient adjoint code generation using AD (Blondel & Roulet, 2024; Sapienza et al., 2025).

The advent or revival of machine learning (ML) techniques has introduced new strategies for "learning" subgrid-scale parameterizations and model calibration (Zanna & Bolton, 2020; Yuval et al., 2021; Espinosa et al., 2022), emulating ESM components (Lam et al., 2023; Bi et al., 2023; Perkins et al., 2023; Dheeshjith et al., 2025) and improving forecasting on a broad range of time scales (He et al., 2021). The key computational ingredient driving many of these ML techniques is backpropagation through neural network (NN) architectures, which is conceptually identical to propagating sensitivity information through the use of adjoint operators for physics-based models (Baydin et al., 2018). Whereas adjoints efficiently compute the derivative of model-data misfit functions or quantities of interest with respect to input or control variables, backpropagation efficiently computes the derivative of the loss function with respect to NN weights and biases. Both are in fact structurally the same and are implemented via reverse-mode AD, but they have evolved as different terminologies in the simulation-based science and machine learning domains (Griewank, 2012). Differentiable programming is essential in that it enables rapid and accurate construction of the backpropagation operator of the NN architecture or of the adjoint operator of the physical model using AD (Chizat et al., 2019; Sapienza et al., 2025).

In a hybrid framework, the two differentiable programming applications discussed in the preceding paragraph are seamlessly integrated: the physical model's adjoint and the NN's backpropagation operator. Here, the role of the neural network is typically to replace or augment a subgrid-scale parameterization scheme. During the *online* or *full-model* training, gradients are propagated through the NN via standard backpropagation, while the sensitivities of the model's state variables are computed through the adjoint. The high-dimensional input space which necessitates adjoint approaches is now composed of (or includes) the space of NN weights. This integrated training strategy ensures that the NN learns corrections that remain dynamically consistent with the governing physical equations. By contrast, *offline* training does not use the model adjoint and optimizes the NN weights in isolation, producing solutions that may generalize less robustly across regimes and conditions.

Driven by the rise in machine learning applications, several novel AD tools have been developed in recent years, including the JAX framework (Bradbury et al., 2018) and Enzyme (W. Moses & Churavy, 2020). These systems benefit from compiler optimizations and offer an easy interface for potential GPU acceleration and integration of ML into the ESM.

Equipping ESM components with AD enables:

1. Comprehensive parameter calibration through gradient-based optimization (e.g., Stammer (2005); Larour et al. (2014)).
2. Smoother-based, dynamically and kinematically consistent state estimation (e.g., Wunsch & Heimbach (2007); Badgeley et al. (2025)).
3. Comprehensive, time-resolved, and spatially resolved boundary flux inversion from interior observations (e.g., Kaminski et al. (2013); Liang & Yu (2016)).
4. More general sensitivity analyses of (usually scalar-valued) quantities of interest or model metrics to a range of spatially and temporally resolved input variables (e.g., Errico & Vukicevic (1992); Fukumori et al. (2015); Pillar et al. (2016); Kostov et al. (2021)).
5. Derivative-based, that is, Hessian-based, uncertainty quantification (e.g., Isaac et al. (2015); Kaminski et al. (2018); Loose & Heimbach (2021)).
6. Combination of adjoint and backpropagation operators in a hybrid approach, whereby a neural network is embedded within an ESM component (e.g., Kochkov et al. (2021)).

We emphasize that, while the last point is our main motivation for developing differentiable ESM components that embed ML architectures, such as subgrid-scale surrogate models that learn from data to provide better-calibrated simulations, the purpose of this work is not (yet) to showcase such a hybrid learning approach. Instead, we here demonstrate the feasibility of general-purpose reverse-mode AD on a range of ESM components to produce correct and efficient gradients, thus setting the stage for hybrid learning approaches as described above.

1.2 What Makes Development of Differentiable Models Hard

Whereas some individual components of entire ESMs have been rendered differentiable (e.g., Marotzke et al. (1999); Heimbach et al. (2002); Stammer et al. (2002); Kaminski et al. (2013); Morlighem et al. (2021)), no fully differentiable coupled ESM yet exists (Gelbrecht et al., 2023; Shen et al., 2023). At its core, the difficulty of whole-model differentiation stems from both the significant computational demands of ESMs and the need to support differentiable versions of all the complex features in modern programming languages. ESMs are not written by individuals but are the effort or large teams, connecting model components (atmosphere, ocean, land, etc.) that themselves are often the product of decade-old legacy software without a coherent programming paradigm or differentiability in mind.

ESMs run on large supercomputers producing data at a rate of gigabytes per second. Data and computation at this scale necessitate that the simulation code be written in a computationally efficient fashion that obscures the mathematical structure that the code represents. In practice this means that simulations must be written "in place" to minimize memory usage, rely on control flow, ideally leverage just-in-time compilation (a feature rarely used in current ESMs), and employ numerous custom kernels for central processing units (CPUs), graphics processing units (GPUs), or tensor processing units (TPUs) for execution. All these features break modern and traditional differentiation tools such as JAX (Bradbury et al., 2018), PyTorch (Paszke et al., 2019), and Tapenade (Hascoet & Pascual, 2013).

Beyond the difficulties presented by the code structure (Hückelheim et al., 2024), the structure of the computation presents further challenges to differentiation. Typical usage of ESMs involves simulating for millions of time steps, each of which fully overwrites the current state of the model. Reverse-mode differentiation of a time-stepping loop, however, requires either storage of all previous time steps—asymptotically increasing the memory requirements of the derivative—or recomputing the current state, either of which in its pure form is prohibitive for comprehensive ESMs. Checkpointing balances storing and recomputing. It reduces or limits the memory load by storing a subset of the gradient computations (Griewank & Walther, 2008). This comes at the cost of increased computational load, however, since the states in the intermediate steps need to be recomputed. Thus, checkpointing requires a delicate balance between memory efficiency and computational speed (Alhashim et al., 2025).

Another difficulty faced by differentiable ESMs is the chaotic nature of the climate system. Pires et al. (1996); Lea et al. (2000); Metz et al. (2021) discuss how such systems render gradients computed by AD unstable, resulting in gradient explosion and, subsequently, ill-conditioned Jacobians and large eigenvalues. The difference in the timescales of the different processes also induces stiffness in the differential equations that may lead to errors. Recent work is pointing to ways in which these issues may be alleviated (Kennedy et al., 2025).

1.3 DJ4Earth

The *Differentiable programming in Julia for Earth system modeling (DJ4Earth)* initiative is a new framework to enable differentiable Earth system models in Julia. The purpose of this paper is to describe a number of algorithmic developments required to render an initial set of recently developed Julia-based ESM components differentiable for the DJ4Earth framework. Because each of these components uses bespoke numerical algorithms, general-purpose reverse-mode AD has been the method of choice to generate derivative codes. The AD tool used is Enzyme and its Julia-specific binding Enzyme.jl (W. Moses & Churavy, 2020; W. S. Moses et al., 2021, 2022). Section 2 describes algorithmic developments, notably Reactant.jl, that were essential to handle Julia-specific issues and to generate a Multi-Level Intermediate Representations (MLIRs) in order to generate robust, efficient, and performance-portable derivative code. Further requirements for iterative or time-evolving algorithms were the implementation of checkpointing schemes, at both the Julia level and the MLIR level, in order to mitigate storage-related memory issues that are ubiquitous in reverse-mode AD.

These technical developments are showcased in four application case studies representing ESM components that implement a range of numerical algorithms and spatial discretization schemes, including finite-volume, finite-element, and spectral schemes. They comprise the shallow water model ShallowWaters.jl (Section 3); a full-fledged ocean general circulation model Oceananigans.jl, which forms the ocean component of the Climate Modeling Alliance (CliMA) model (Section 4); the ice sheet model DJUICE.jl (Section 5); and the atmospheric general circulation model SpeedyWeather.jl with parameterized physics (Section 6). A concluding discussion is given in Section 7.

2 Techniques for Efficient Differentiable Earth System Modeling

ESMs are large and complex pieces of software that contain many different components and numerical algorithms. Users and developers of ESMs need to be able to explore different configurations and model compositions. As an example, the Oceananigans code (see Section 4) may be used as a high-resolution large eddy simulation model or as a global general circulation model. Utilizing a dynamic high-level programming language allows the model configuration to evolve beyond the traditional run-file approach to a program as the configuration approach, enabling developers to quickly explore and alter model configuration or to provide customization through user functions. The Julia programming language is such a high-level dynamic programming language, with a host of capabilities that make it particularly attractive for ESM applications. Julia uses an LLVM-based just-in-time (JIT) compiler that can natively target common accelerators, allowing user functions to be inlined into the computational kernels.

In order to enable whole-model differentiation of ESMs or ESM components, several novel computational algorithms and techniques needed to be developed that take advantage of Julia capabilities and overcome some of the challenges created by this flexibility and extensibility. The following section describes the development of the automatic differentiation framework Enzyme.jl; the tracing compiler Reactant.jl; and Checkpointing.jl, an implementation of checkpointing algorithms. A high-level workflow of how these frameworks interact for a modern ESM component (here, an ocean model) is given in Fig. 1.

2.1 Automatic Differentiation: Enzyme.jl

AD is a technique for computing the mathematical derivatives of computer programs (Griewank, 2003). The most important derivative programs are tangent linear models, which compute directional derivatives (i.e., the impact of changing one input on all outputs), and adjoint models, which compute gradients (i.e., the sensitivity of one out-

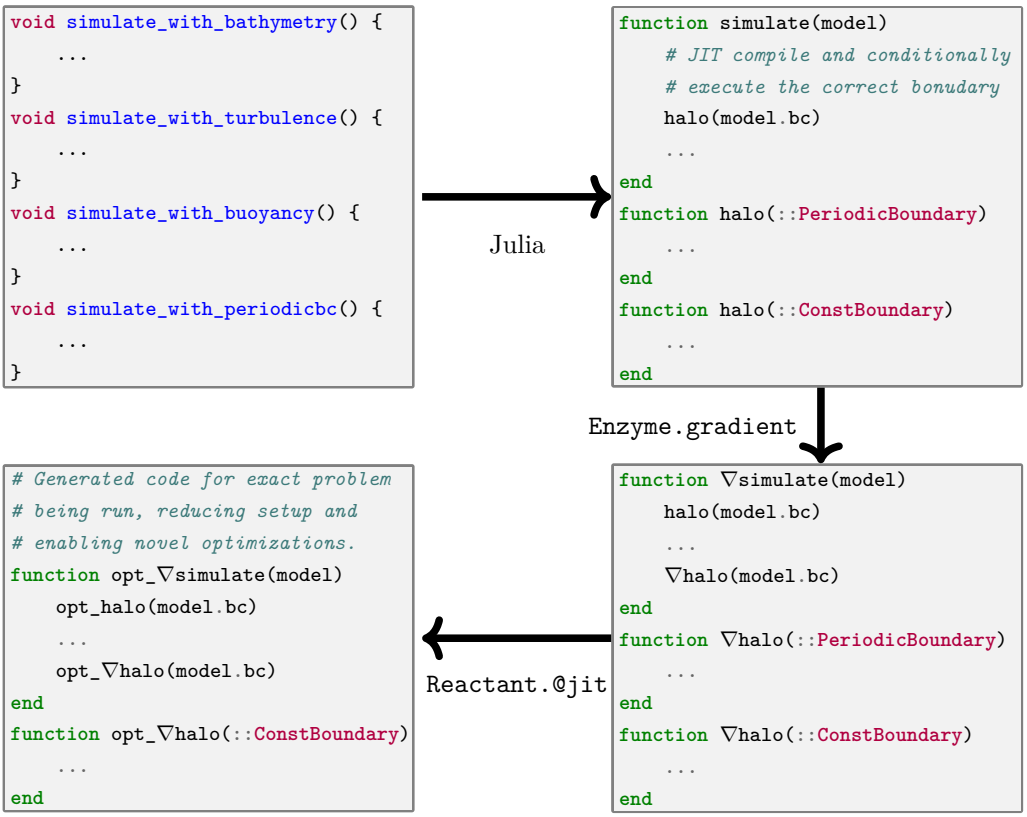

Figure 1: ***Top Left:*** C++-style code of prior ocean simulation models, containing many separate variations of the simulation for each potential specialization. Top Right: Julia-style ocean model program in which a single simulation is written, with each feature conditionally enabled via just-in-time (JIT) compilation. Bottom Right: Enzyme-generated derivatives of the simulation code. Bottom Left: Reactant-optimized simulation code in which the exact problem being run is known and excess code can be removed and additional optimizations specific to the simulation at hand can be applied.

put with respect to changes in all inputs). The former are implemented via so-called forward-mode AD, whereas the latter via reverse-mode AD, a concept equivalent to backpropagation in machine learning (e.g., Rumelhart et al. (1986); Griewank (2012)). For details, we refer to monographs on the subject, such as Griewank & Walther (2008); Naumann et al. (2015).

ESMs contain numerous challenges to differentiation stemming from both the necessary structure of ESM application code and the structure of the computation itself. Production-quality ESMs push the limit of what can be efficiently computed on modern hardware. They often consume all system memory, requiring the simulation to be written in a form that mutates data in place. They require vast amounts of computation and are written with custom kernels to efficiently run on modern systems such as CPUs, GPUs, and TPUs. Despite these efforts, current-generation ESMs can achieve only around 5% peak performance on today's high-performance computing (HPC) architectures (e.g., Zhang et al. (2020); see Balaji et al. (2016) for a detailed discussion of ESM performance metrics). They are often memory- and compute-bound, and the many different algorithms operating consecutively with varying large arrays are difficult to optimize collectively without reaching diminishing returns on some of them (Amdahl's law). To support the numerous combinations of model features, ESM code bases feature control flow to dynamically enable certain code paths. Modern ESMs increasingly leverage JIT compilation to avoid wasting time preparing to use features that are not required to execute a particular model. Moreover, ESM application codes are large, leveraging nearly all features of the programming language(s) they are written in.

Most of the work to date on differentiable ESM components has relied on hand-coded adjoints. These are essentially a second copy of the simulation code that instead computes the derivative. Examples include the tangent linear and adjoint components of ECMWF's weather forecast model (Rabier et al., 2000; Janisková & Lopez, 2013) and the Regional Ocean Modeling System (Moore et al. (2004, 2011)). Although this idea is simple in principle, in practice it leads to several issues. Given the size and complexity of ESM code bases, writing a second version of the application is a difficult endeavor that is costly in money, personnel, and development time. Moreover, it presents a significant maintenance and correctness burden. Whenever the original simulation (the primal calculation) is modified, great care must be taken to update the corresponding derivative code base to reflect these changes accurately in the corresponding gradient computation. If the inverse or control problem is changed, for example, from a pure state to a parameter estimation problem (or a combination thereof), the structure of the derivative code may change fundamentally; simply put, for $f = a \cdot x$, we have $df(x) = a \cdot dx$, or $df(a) = da \cdot x$, or $df(a, x) = da \cdot x + a \cdot dx$, each of which results in different derivative code.

In parallel, tools to automatically generate the derivatives were developed (Giering & Kaminski, 1998). However, these tools were limited in the features of the language they support. For example, the AD tools ADIFOR, TAF, or Tapenade took many years to extend their capabilities from Fortran77 to Fortran90/95 language features. Meanwhile, Fortran is continually evolving (e.g., Kedward et al. (2022); Magnin et al. (2023)). To analyze existing code to generate derivatives, source-transformation tools must understand how to parse and perform semantic analysis from scratch, before they even start differentiation. The extraordinary difficulty of this initial analysis task cannot be overstated. For example, the draft ISO C++ Standard published in 2020 (`https://isocpp.org/files/papers/N4860.pdf`) contains 1,841 pages of text, most of which is comprehensible only to programming language experts. Compliant compilers, such as Clang/LLVM, are maintained as a collaboration between several large technology companies. Over the span of a single month (as of August 2025), the LLVM project had 4,385 active pull requests from 805 unique programmers, resulting in 1,049,370 lines of code being added to over 12,748 files. Consequently, these initial general-purpose tools were extremely lim-

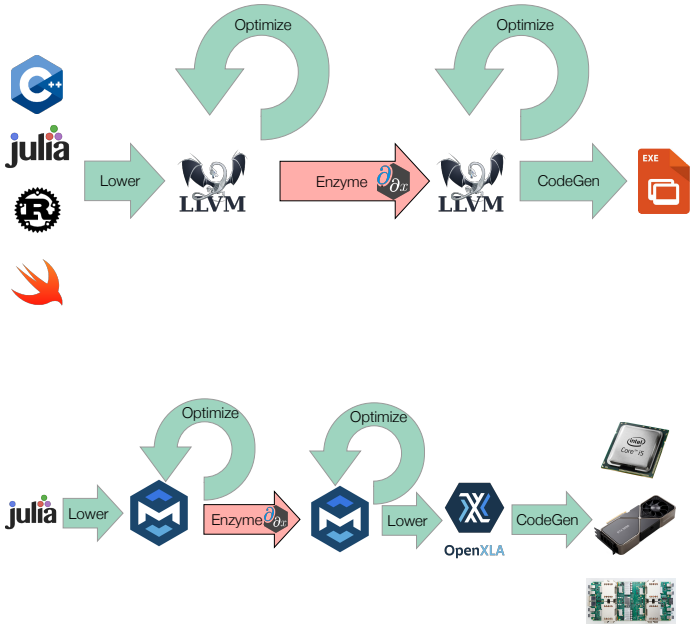

Figure 2: Top: The Enzyme compiler pipeline. Programs of a variety of languages are first compiled to an LLVM and optimized, prior to and after differentiation. Bottom: The Reactant compiler pipeline. Reactant first lowers into the stablehlo/tensor dialect within MLIR and performs linear algebra optimizations. Reactant then performs automatic differentiation with Enzyme on MLIR, before a second round of tensor optimizations. Finally, Reactant lowers the MLIR for execution by XLA on any number of CPUs, GPUs, or TPUs.

ited in the features they supported, and codes needed to be adapted accordingly. Structure types, pointers, control flow, templates, and more all present difficulties to automated tools.

Modern AD tools, such as JAX, PyTorch, and TensorFlow, define a fixed subset of primitives useful for a particular domain, usually machine learning. These domain-specific languages (DSLs) tend to support differentiation of nearly all the tensor-specific runtime functions within their library, but this support comes with a new constraint: all code must be written in said DSL. These tools work well if the DSL closely mirrors the operations being performed, such as native convolution or attention layers making it easy to perform machine learning. Unfortunately, they are not designed with the primitives applicable to ESMs, necessitating significant code rewriting. In particular, these tools tend to lack support for custom kernels (required for high-performance primal computations), mutable memory (required for large ESMs), and control flow (required for easy switching between different models).

Instead of writing tools at the frontend level that have to deal with all the complexity of the input language, Enzyme performs differentiation within the compiler (Fig. 2, upper pipeline). This approach enables Enzyme to leverage the existing production compilers for their host language (here Julia) and needs to support only a smaller fixed set of operations. For example, as of August 2025, LLVM contains 68 unique instruction types. As a result, Enzyme can differentiate any program in any language with an LLVM-compatible compiler. Working directly on programs instead of traces further enables Enzyme to natively handle control flow, mutation, and custom kernels. Moreover, unlike other tools

```
# Compute magnitude in O(N)
function mag(x) end
function norm(out, x)
    # res = mag(x) code motion optimization can move outside the loop
    for i in 1:N
        out[i] = x[i]/mag(x)
    end
end
```

```
# LICM, then AD, O(N)
function grad_norm(out, d_out,
                   x,   d_x)
  res = mag(x)
  for i in 1:N
    out[i] = x[i]/res
  end
  d_res = 0.0
  for i in N:-1:1
    d_res += -x[i]*x[i]/res * d_out[i]
    d_x[i] += d_out[i]/res
  end
  grad_mag(x, d_x, d_res)
end
```

```
# AD, then LICM O(N^2)
function grad_norm(out, d_out,
                   x,   d_x)
  float res = mag(x);
  for i in 1:N
    out[i] = in[i]/res
  end
  d_res = 0.0
  for i in N:-1:1
    d_res = -x[i]*x[i]/res * d_out[i]
    d_x[i] += d_out[i]/res
    grad_mag(x, d_x, d_res)
  end
end
```

Figure 3: Top: An $O(N^2)$ function `norm` that normalizes a vector. Running loop-invariant code-motion (LICM) (Muchnick, 1997, Sec. 13.2) moves the $O(N)$ call to `mag` outside the loop, reducing `norm`'s runtime to $O(N)$. Left: An $O(N)$ `grad_norm` resulting from running LICM before AD. Both `mag` and its adjoint `grad_mag` are outside the loop. Right: An $O(N^2)$ `grad_norm` resulting from running LICM after AD. `grad_mag` remains inside the loop as it uses a value computed inside the loop, making LICM illegal.

that must perform differentiation on source code, Enzyme can perform program optimizations before and after differentiation. Prior work on Enzyme has demonstrated that combining program optimization with differentiation (Fig. 3) results in significantly improved derivative code. In particular, Enzyme has demonstrated a 4.2× geometric mean speedup on CPU code when enabling optimization before AD (W. Moses & Churavy, 2020), orders-of-magnitude speedups on GPU programs (W. S. Moses et al., 2021), and optimal program scaling on distributed and task-parallel programs (W. S. Moses et al., 2022).

Applying differentiation in a dynamic language such as Julia, however, presents several core challenges: dynamism, customized algorithms, and automatic memory management (garbage collection). For many algorithmic pieces of an ESM optimal adjoints are known, and we developed facilities in Enzyme.jl to provide custom differentiation rules. To appreciate the issues in the context of rendering ESMs differentiable or extending the AD tool capabilities, we briefly outline them in the following.

2.1.1 Dynamism

Julia's execution model poses additional challenges. Julia is a dynamic programming language utilizing multiple dispatch. This means that at each call site, the target method of a function is computed utilizing the concrete types of all arguments. To execute programs faster, Julia compiles methods just before their execution and caches the result; during the compilation phase, it uses abstract interpretation to discover the types of all variables inside a method from the types of the arguments. A *type instability* in a Julia program is a failure during the compilation process to infer the specific type of a variable; this allows Julia to represent uncertainty about variables that will be resolved during runtime. Using abstract interpretation, Julia recovers a partially static and par-

tially dynamic call graph of a program. Unlike dynamic function calls in statically compiled languages such as C++ or Fortran, Julia defers the resolution of dynamic function calls to runtime using its JIT compiler, thus not emitting the corresponding code immediately. In contrast, Enzyme requires all relevant functions and their LLVM intermediate representation to be available for differentiation. Enzyme.jl works around this by first extracting the static subset of the current program and differentiating code within this compilation unit. If there are dynamic JIT calls, these will be marked with corresponding Julia runtime functions such as `jl_apply_generic`, with function arguments that describe the function to be dynamically compiled and executed. Leveraging Enzyme's handler for custom calls, Enzyme.jl defines the derivative of a dynamic function dispatch to instead perform a dynamic dispatch to a modified function, which will again call into Enzyme to extract and differentiate the target code, and then JIT-compile the result. This process will repeat recursively until all the dynamic dispatches that are actually required by the program have been executed. Enzyme.jl thus follows the execution model of the host language, delaying the compilation of the derivative code until execution necessitates it.

2.1.2 Custom Differentiation Rules

Sometimes the automatically generated derivative code is far from optimal and not the code one wants to run. For example, when differentiating the determinant of a unitary matrix, the derivative is always zero. Rather than wasting time adding up values from the implementation of the determinant which will eventually compute zero, Enzyme can simply avoid performing the computation entirely. As another example, one may want to change how Enzyme decides to save or recompute certain values to improve performance (e.g., checkpointing; Section 2.3.2 utilizes custom rules for this purpose).

Enzyme enables this functionality by providing support for custom differentiation rules of any user-defined function. In particular, users should override the method `Enzyme.forward` with a specialization for any function `f` they want to define a rule for. Whenever Enzyme sees a call to `f`, instead of differentiating it directly, Enzyme will JIT-compile the user-provided implementation within `Enzyme.forward`. When Enzyme is used to differentiate entire applications, this means that Enzyme will use the user-defined rules when specified and automatically generate the corresponding derivative routines for all other code.

2.1.3 Automated Memory Management (Garbage Collection)

The Julia runtime maintains control of all allocations performed within the language. This enables users to avoid considering the lifetime of their memory allocations, preventing a large class of potential bugs. The decision of when to free memory is made by a garbage collector (GC) that tracks all allocations, freeing them when there is provably no remaining user of the memory. This presents a new challenge for reverse-mode AD. Some data must be preserved from the original forward pass evaluation for use in the reverse pass. For example, when differentiating `A * B`, the corresponding derivative of `A * dB + dA * B` requires both `A` and `B` to be available during differentiation. Consequently, Enzyme needs to extend the lifetime of these values from the forward pass to the reverse pass. Enzyme may also generate new differentiation-specific memory. This includes storage for `dA` and `dB`. Enzyme consequently must inform the GC about any memory that it creates or whose lifetime needs to be changed. To do so, it places references onto a data tape and generates a descriptor for the data tape that allows the GC to mark this subtape.

2.2 Automatic Device Scheduling and Distribution: Reactant.jl

The use of a dynamic language such as Julia provides many benefits, including ease of development. Dynamic dispatch makes it easy to write flexible code that can be reused

but consequently may make it difficult to perform whole-model optimization. A tracing compiler can partially evaluate the simulation code and overcome the loss of information induced by dynamic dispatch, reducing the amount of code to analyze for automatic differentiation and opening opportunities for additional performance optimizations.

As we saw before, optimizing a simulation results in compound performance gains for the derivative simulation (Fig. 3). Reactant.jl is a new compiler framework for Julia that leverages the MLIR (Lattner et al., 2021) and the Accelerated Linear Algebra (XLA) compiler to perform domain-specific optimization. Unlike LLVM, which has a fixed instruction set that corresponds to individual scalar integer and floating-point operations, one can define operations with arbitrary high-level meaning. For example, Reactant directly preserves the high-level tensors and linear algebra operations from Julia within a dialect of MLIR, StableHLO, which contains primitive instructions for matrix multiplication, convolution, and more.

Reactant begins by mapping the corresponding instructions within Julia with high-level tensor operations within the StableHLO dialect (Fig. 2, lower pipeline). This mapping involves partially evaluating out any sources of type instability, such as discussed above. Reactant then performs a series of linear algebra optimizations on the tensor code. For example, if Reactant detects that one intends to compute `transpose(x .+ transpose(x))`, it will instead choose to optimize it as simply `x .+ transpose(x)`. In isolation, these linear algebra optimizations have been demonstrated to provide significant speedups to tensor programs, including double-digit improvements in ML training (Lücke et al., 2025). Subsequently, Enzyme performs differentiation on the program, now on MLIR rather than LLVM. Finally, Reactant lowers the program into XLA for execution, which enables the final program to be run on CPU, GPU, or TPU—including distributed clusters thereof—without any rewriting required.

While the need for Reactant in our workflow to differentiate ESMs is primarily to remove type instabilities and other performance pitfalls, it comes with a number of additional performance benefits. Scientific codes, such as ESMs, maintain hundreds of hand-written kernels, preventing them from using the advanced tensor capabilities of modern ML accelerators. Yet the core computations within such kernels are often similar to ML workloads. For example, a simple stencil kernel is roughly analogous to a convolution. Reactant enables these existing stencil kernels to efficiently leverage the ML-specific hardware features, such as tensor cores on NVIDIA GPUs or Google TPUs.

2.3 Automatic Memory Reduction: Checkpointing

In general, a numerical model is implemented as a function $y = f(x)$, where x are the inputs and y are the outputs. For calculating sensitivities, we can apply calculus and derive the adjoint model $\bar{x} = f(x, \bar{y})$, where the adjoint $\bar{x}$ is computed with respect to the input x and input adjoint $\bar{y}$. When f is applied iteratively over N iterations as $y_t = f(x_t)$, the adjoint model imposes a computational reversal $\bar{x}_t = \nabla f(x_{t-1}, \bar{y}_t)$, where x needs to be provided in reverse order of the original *forward* model f execution. Practical applications of ESMs at state-of-the-art resolution of 25 km globally can consist of $O(10^6)$ timesteps (e.g., 100 years at 10-minute time steps), each requiring O(10 GB) (e.g., 1,000,000 horizontal grid points, 100 vertical layers, 20 variables including scratch arrays) making it prohibitively large to hold all time steps simultaneously in memory (Gaikwad et al., 2025). In AD, this data flow reversal is known as the *checkpointing problem* (Griewank & Walther, 2000). It can be described as a mixed-integer programming problem where the fastest way of computing the adjoint is determined under constraints such as available memory space and the latency to read and write data. Several checkpointing strategies exist, including square root (periodic) checkpointing (Fig. 4), multilevel checkpointing, and binomial checkpointing.

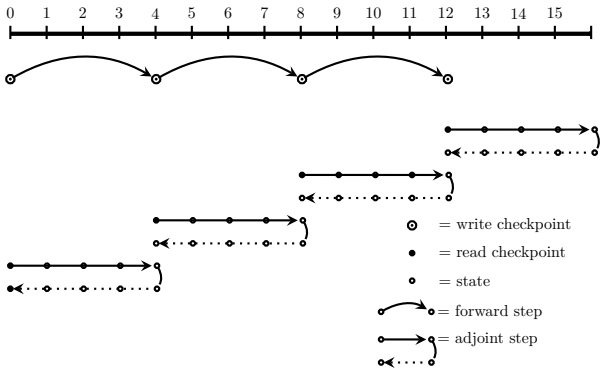

Figure 4: Square root checkpointing schedule for $l = 16$ time steps (0-15). The forward computation stores checkpoints at timesteps 0, 4, 8, and 12. The adjoint computation for steps 12-15 uses the checkpoint stored at at 12. Then the adjoint computation for steps 8-11 using checkpoint at 8. Then the adjoint computation for steps 4-7 using checkpoint at 4. Finally, the adjoint computation for steps 0-3 uses checkpoint at 0.

We have made checkpointing transparent to the user and implemented two complementary strategies: (1) a low-level implementation integrated directly into Enzyme and (2) a higher-level approach that leverages the Julia metaprogramming macro feature to checkpoint iterative loops (Schanen et al., 2023), provided through a native Julia package, Checkpointing.jl.

2.3.1 Enzyme MLIR Checkpointing

The low-level scheme is directly integrated into EnzymeMLIR to make checkpointing directly embedded into the device codes. Checkpointing in EnzymeMLIR implements a form of periodic checkpointing called square root checkpointing (Fig. 4). Here, checkpoints for N time steps are taken at a period of $\sqrt{N}$ time steps. The state to be checkpointed is determined automatically by Enzyme's analyses, and the checkpoints are stored in memory. This also enables program optimization to occur prior to checkpointing, potentially reducing the number of variables that must be preserved.

2.3.2 Checkpointing.jl

In contrast to the low-level approach described above, Checkpointing.jl is implemented natively in Julia and has access to all language features. It is split into three areas: checkpointing algorithm, storage device (RAM, disk), and rules (ChainRules, EnzymeRules). As opposed to the MLIR implementation, we support multiple checkpointing algorithms (periodic, revolve, online), and with the rules support we target nearly all AD tools in Julia. This accessible implementation was largely made possible through Julia's multiple dispatch and metaprogramming features. This allows us to automatically and transparently transform loop iterations into differentiated loops.

3 Application 1: Shallow Water Model in a Rotating System

The first example used to demonstrate the capabilities of general-purpose AD in Julia with Enzyme is a shallow water model for a fluid in a rotating Cartesian coordinate system on a β-plane (Vallis, 2017), representing the idealized surface circulation of the North Atlantic. Contained in the package ShallowWaters.jl (Klöwer et al., 2020, 2022),

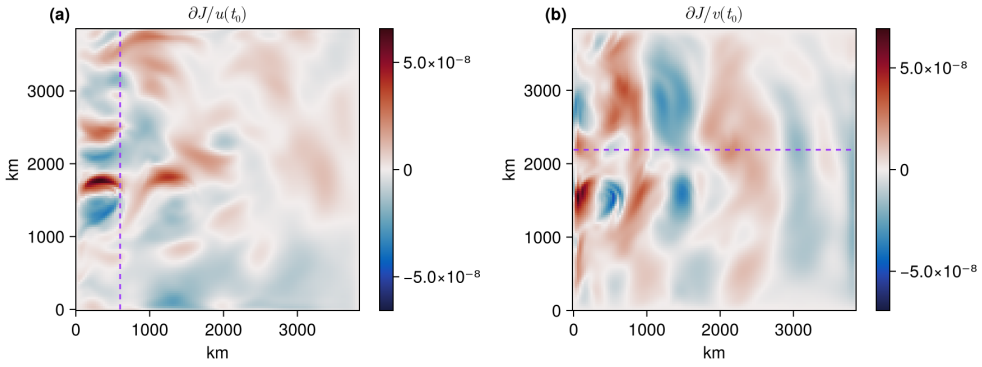

Figure 5: Derivative of the quantity of interest $\mathcal{J}$ (Eq. (3)) with respect to the initial conditions (a) $u(x, y, t_0)$ and (b) $v(x, y, t_0)$. In both panels a dashed purple line shows where derivatives are checked in Fig. 6.

the model solves the conservation equations for momentum and volume

$$\begin{aligned} \frac{\partial}{\partial t}u + u\frac{\partial}{\partial x}u + v\frac{\partial}{\partial y}u - fv &= -g\frac{\partial}{\partial x}\eta + M_x + F_x \\ \frac{\partial}{\partial t}v + u\frac{\partial}{\partial x}v + v\frac{\partial}{\partial y}v + fu &= -g\frac{\partial}{\partial y}\eta + M_y + F_y \\ \frac{\partial}{\partial t}\eta + \frac{\partial}{\partial x}(uh) + \frac{\partial}{\partial y}(vh) &= 0 \end{aligned} \tag{1}$$

for the prognostic variables $\mathbf{u} = (u, v)^T$, and η. The former define the x and y components of the velocity vector, and the latter is the sea-surface displacement from rest. The right-hand side of the momentum equations represents horizontal pressure gradients, surface wind stress $\mathbf{F} = (F_x, F_y)^T$, and the combined effects of turbulent mixing and bottom drag denoted by $\mathbf{M} = (M_x, M_y)^T$. The Coriolis force f is computed with a β-plane approximation at a latitude of 45°N, and gravitational acceleration is set to $g = 9.81$ m/s^2.

Equation (1) is solved on a square domain with sides of length $L_x = L_y = 3840$ km and a single-layer depth of $H_0 = 500$ m at rest. The grid is set at 30 km resolution, corresponding to a discretized domain with 128 x 128 cells. Equation (1) is solved by using a fourth-order Runge–Kutta time integration with time step $\Delta t = 385$ s. The circulation is driven by a sinusoidal wind stress function in the x direction that varies solely with latitude y, given by

$$F_x(y) = \frac{F_0}{\rho H_0}\left\{\cos\left(2\pi\left(\frac{y}{L_y} - \frac{1}{2}\right)\right) + 2\sin\left(2\pi\left(\frac{y}{L_y} - \frac{1}{2}\right)\right)\right\} \tag{2}$$

and shown in Fig. 1(b) in the supplemental material. Here the water density is $\rho = 1000$ kg/m^3, and the forcing strength is $F_0 = 0.12$ Pa. There is no wind forcing in the y direction ($F_y = 0$). The time-averaged sea-surface displacement η exposes two gyres, basin-wide closed circulations (Fig. 1(a) in the Supporting Information). Experiments are conducted following a ten-year model spinup.

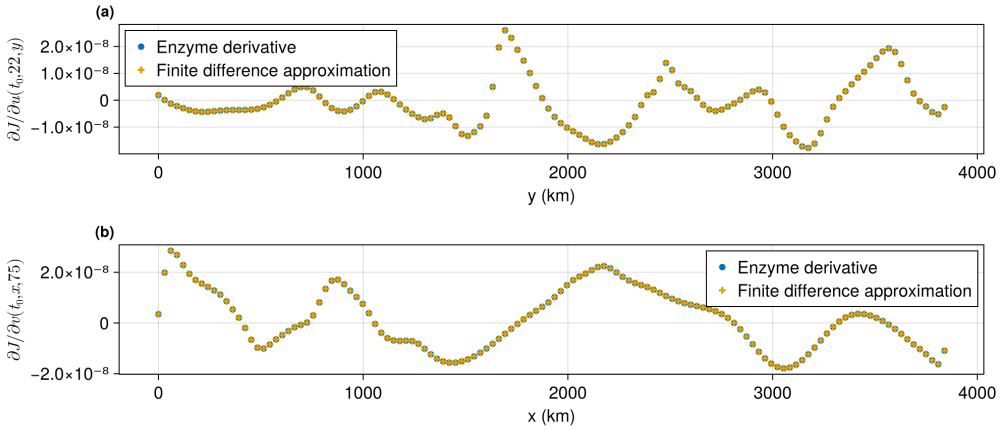

Figure 6: Derivative of $\mathcal{J}$ (Eq. (3)), computed with Enzyme reverse-mode AD (blue dots) versus a finite-difference approximation (yellow crosses). (a) Derivatives with respect to $u(x_0 = 22, y, t_0)$, where $x_0 = 22$ corresponds to 600 km. (b) Derivatives with respect to $v(x, y_0 = 75, t_0)$, where $y_0 = 75$ corresponds to 2190 km.

3.1 Sensitivity Analysis

Our first example demonstrating correct and efficient derivative code generation with Enzyme is a sensitivity analysis. Our quantity of interest is

$$\mathcal{J}(\mathbf{u}(x, y, t)) = \frac{1}{N} \sum_{x,y} \left\{ u(x, y, t_f)^2 + v(x, y, t_f)^2 \right\}, \tag{3}$$

where t_f is the final time step of the integration and $N = n_x \cdot n_y$, with n_x, n_y is the number of cells in the x and y directions, respectively. $\mathcal{J}$ thus defines a measure of the average kinetic energy at the end of the integration window. To compute derivatives of $\mathcal{J}$, ShallowWaters is integrated for ten days beyond the ten-year spinup, after which the backwards problem is run with Enzyme and Checkpointing for $t_f - t_0 = 10$ days (or roughly 2,250 time steps). Two sample derivative fields are shown in Fig. 5, representing the gradient of $\mathcal{J}$ with respect to u and v at initial time t_0. Values of these gradients were verified by using a finite-difference calculation, results of which are provided for specific x- and y-coordinates in Fig. 6. The location of the derivative checks is shown via dashed purple lines in Fig. 5; for $\partial\mathcal{J}/\partial u(t_0)$ the x-coordinate is fixed at 600 km, and for $\partial\mathcal{J}/\partial v(t_0)$ the y-coordinate is fixed at 2190 km. The gradients computed via reverse-mode AD versus a finite-difference approximation show excellent agreement.

3.2 Data Assimilation

Another important use of reverse-mode AD is data assimilation, showcased in our second example. Here, data assimilation is used to seek improved initial conditions $\mathbf{x}(x, y, t_0)$ by minimizing the loss

$$\mathcal{J} = \sum_{t=t_1}^{t_f} \underbrace{\sum_{x,y} \{ \tilde{\mathbf{x}}(x, y, t) - \mathbf{d}(x, y, t) \}^2}_{\mathcal{J}_t}, \tag{4}$$

where $\tilde{\mathbf{x}}$ indicates the predicted model state (a vector of u, v, and η) and $\mathbf{d}$ the available data. The data $\mathbf{d}$ are daily state snapshots at each model point, obtained from a "truth"

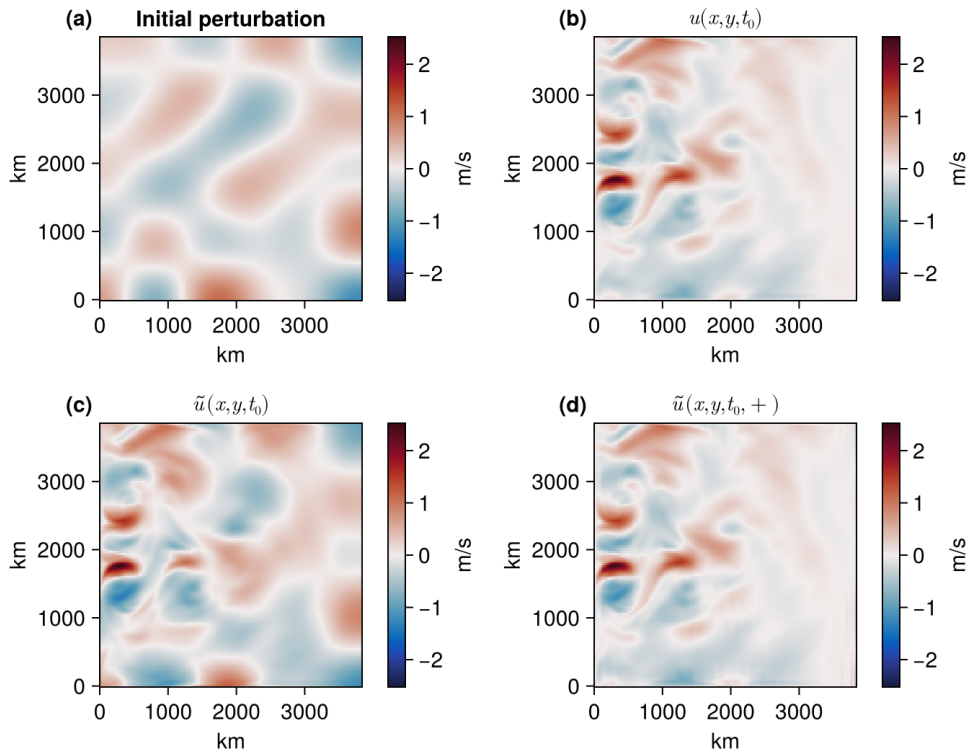

Figure 7: Data assimilation results shown for the zonal velocity component at the initial time t_0. (a) Perturbation applied to initial zonal velocity; (b) unperturbed initial zonal velocity, $u(x, y, t_0)$; (c) perturbed initial zonal velocity, $\tilde{u}(x, y, t_0)$; (d) optimized initial zonal velocity, $\tilde{u}(x, y, t_0, +)$

integration. A long wavelength Gaussian perturbation (Fig. 7(a)) of the form

$$\delta\mathbf{u}(x,y,t_0) = \sum_{m=1}^{5}\sum_{n=1}^{5}\{a_{nm}\cos(k_n x)\cos(k_m y) + b_{nm}\cos(k_n x)\sin(k_m y) + c_{nm}\sin(k_n x)\cos(k_m y) + d_{nm}\sin(k_n x)\sin(k_n y)\}$$

with wavenumbers $k_n = \pi n/L$, $k_m = \pi m/L$ and random numbers a_{nm}, b_{nm}, c_{nm}, $d_{nm} \sim \mathcal{N}(0, 0.1)$ is applied to the true initial conditions $u(x, y, t_0)$, $v(x, y, t_0)$ (Fig. 7(b)), resulting in an incorrect predicted model state at time t_0 (Fig. 7(c)). The data assimilation is run over a 10-day integration, using the L-BFGS algorithm implemented in MadNLP.jl (Pacaud et al. (2024), Shin et al. (2021)) for the optimization. The algorithm successfully converges to an optimized initial state $\tilde{u}(x, y, t_0, +)$ (Fig. 7(d)), which closely resembles the true initial conditions (Fig. 7(b)). The value of the loss function decreases by three orders of magnitude over the first 50 iterations and another order of magnitude over the following 150 iterations.

The optimized initial state greatly improves the accuracy of the model output after ten days of integration, seen in Fig. 8. The result of the model beginning from the perturbed initial state (Fig. 8(b)) deviates from the truth (Fig. 8(a)) despite being integrated for only ten days. With an optimized initial condition, the result of the integration (Fig. 8(c)) closely resembles the true final state. The value of the non-accumulated

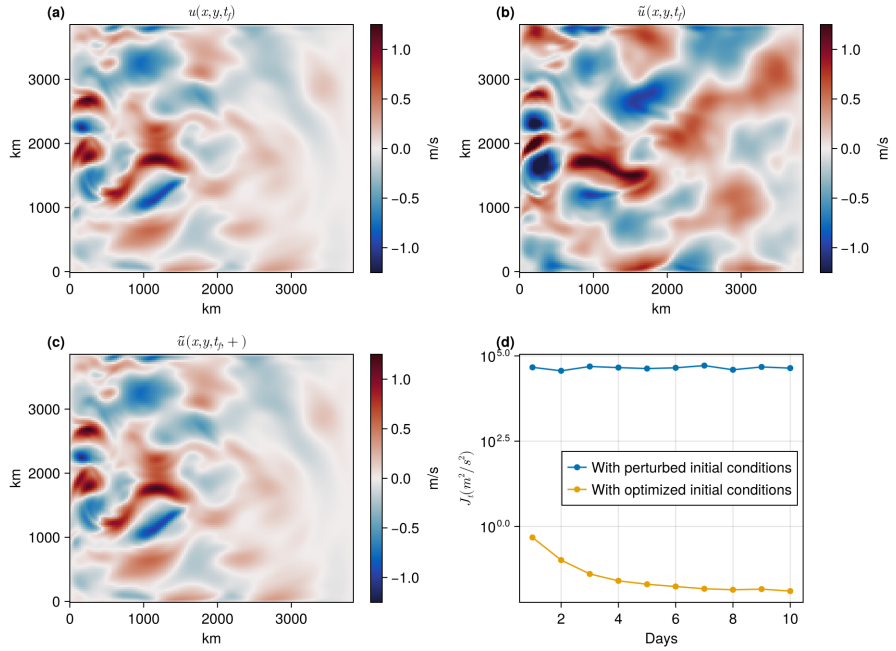

Figure 8: Effect of data assimilation on the evolving model state up to the final time t_f = 10 days. (a) True final zonal velocity component, $u(x, y, t_f)$; (b) predicted final zonal velocity component, $\tilde{u}(x, y, t_f)$ from the perturbed initial condition (Fig. 7(c)); (c) predicted final zonal velocity component, $u(x, y, t_f, +)$ from the optimized initial condition (Fig. 7(d)); (d) non-accumulated loss J_t (Eq. (4)) for each day of the integration, computed using the perturbed initial state (blue line) and optimized initial state (yellow line).

loss function $\mathcal{J}_t$ (Eq. (4)) remains consistently lower for each day of integration in the optimized model (yellow line in Fig. 8(d)) than in the perturbed model (blue line in Fig. 8(d)).

3.3 Performance

Figure 9 compares execution time and memory utilization as a function of integration length for the adjoint sensitivity analysis without (yellow curves) and with (blue curves) checkpointing under the revolve checkpointing scheme. With checkpointing, metrics are computed for integrations of up to approximately 22,000 time steps (100 days). Without checkpointing, the simulation can be run only for about 4,500 time steps (20 days) before the memory required to store the time-evolving state exceeds the laptop's available system capacity.

Checkpointing allows one to compute sensitivities for time windows beyond 20 days while maintaining a minimal memory footprint (Fig. 9b). The amount of memory allocated to store checkpoints typically is configured to be machine dependent and constant. Here it is configured to be proportional to the square root of the number of time steps. In contrast, using Enzyme AD alone requires storing each model state during the forward pass, resulting in a drastic increase in memory utilization. Starting at around 1,000 time steps (around 5 days), the checkpointed reverse-mode AD becomes faster than using AD alone (Fig. 9a). The reason is that, beyond that point, more time is spent allocating memory to compute model derivatives than on the derivative computation itself. Despite the fact that ShallowWaters is a relatively simple model, this result demon-

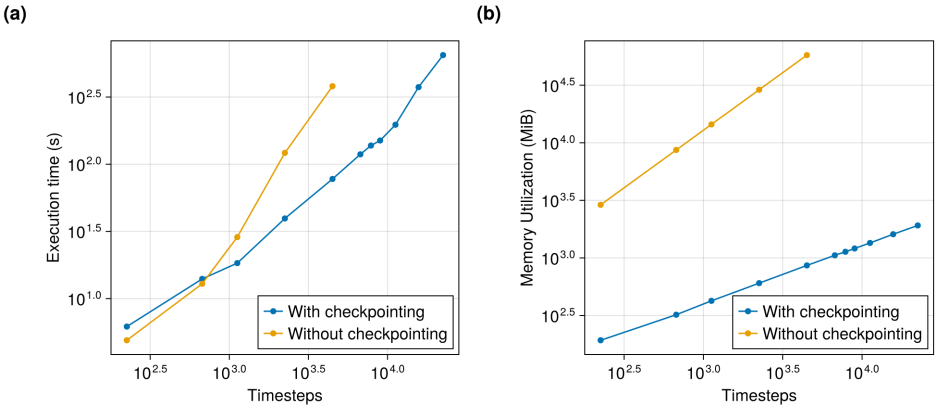

Figure 9: Comparison of (a) derivative computation execution time and (b) memory utilization, with and without checkpointing for the sensitivity analysis (Section 3.1).

strates that implementing a checkpointing scheme alongside AD is essential to feasibly lay the framework for differentiable ocean models.

4 Application 2: Ocean General Circulation Model in a Re-entrant Channel Configuration

Our second application features Oceananigans.jl, a Julia-based software package for finite-volume simulations of the ocean general circulation, designed to run efficiently using CPUs or GPUs (Silvestri et al. (2025); Wagner et al. (2025), hereafter referred to as Oceananigans). This package forms the ocean model component of the Climate Modeling Alliance. For our example, we construct a re-entrant channel configuration of an idealized Southern Ocean circulation, similar to the setup in Abernathey et al. (2011).

We solve the Boussinesq and hydrostatic approximations of the incompressible Navier–Stokes equations of a fluid on a rotating sphere, using conservation of momentum

$$\begin{aligned} \partial_t \mathbf{u} + (\mathbf{v} \cdot \nabla)\mathbf{u} + \mathbf{f} \times \mathbf{u} = & -\nabla_h (p + g\eta) - \nabla \cdot \tau + \mathbf{F}_\mathbf{u} \\ 0 = & -\partial_z p + b\,, \end{aligned} \tag{5}$$

conservation of volume

$$\nabla_h \mathbf{u} + \partial_z w = 0, \tag{6}$$

as well as conservation of heat and salt. Here $\mathbf{u} = (u, v)$ and w are the horizontal and vertical components of the three-dimensional velocity field $\mathbf{v}(x, y, z)$; τ is the hydrostatic kinematic stress tensor; $\mathbf{F}_\mathbf{u}$ is the external forcing of $\mathbf{u}$; p is kinematic pressure; η is free surface displacement (i.e., sea surface height); $\mathbf{f}$ is the Coriolis parameter associated with rotation; and $b = -g\rho'/\rho_0$ is the buoyancy computed from the density $\rho = \rho' + \rho_0$, where ρ_0 is a constant reference density, ρ' is the density perturbation, and g is gravitational acceleration (for details see Silvestri et al. (2025); Wagner et al. (2025)).

Following Abernathey et al. (2011), our model has dimensions 1000 km × 2000 km × 2187 m. It is discretized by using a rectilinear Arakawa C-grid with 80×160 evenly spaced horizontal cells at a 12.5 km resolution and 32 vertical levels of varying thicknesses, ranging from 10 m at the surface to approximately 214 m at the bottom. We use Oceananigans's `HydrostaticFreeSurfaceModel` on GPU architecture to numerically solve our re-entrant channel model. Our setup features periodic boundary conditions in the zonal

(east-west) direction, a sponge layer at the northern boundary, a heat flux that loosely approximates observed buoyancy fluxes in the Southern Ocean, and an idealized mid-latitude westerly surface zonal wind stress.

We make some modifications to the Abernathey et al. (2011) configuration. Most notably, we add a wall topography with a gap from $y = 400$ km to $y = 1000$ km that provides effects analogous to the Drake Passage in the real Antarctic Circumpolar Current. We also replace the implicit free surface with a split-explicit free surface and make use of a flux-form weighted essentially non-oscillatory method (WENO) for our advection schemes. There is no vertical mixing scheme, although the vertical diffusivity is increased in the top five surface layers (approximately the upper 60 m). Example figures of the spun-up state are deferred to Section 2 of the Supporting Information.

Sensitivity Analysis

In our re-entrant channel model, the quantity of interest $\mathcal{J}$ is the zonal volume transport across the gap present in the model's topography that mimics the Drake Passage,

$$\mathcal{J}(u(x_0, y, z, t)) \;=\; U(x_0, t) \;=\; \sum_{y,z} u(x_0, y, z, t)\, \Delta y\, \Delta z. \tag{7}$$

Here the location of the passage is $x_0 = 500$ km, and $\Delta y\, \Delta z$ is the cross-sectional area element in the $y{-}z$-plane. To showcase the range of sensitivities that can be computed with the adjoint, we seek sensitivities of $\mathcal{J}$ with respect to the initial state, surface boundary conditions, and model parameters.

Our first investigation concerns the sensitivity of zonal volume transport to wind stress, $\nabla_\tau \mathcal{J} = (\partial\mathcal{J}/\partial\tau_x,\, \partial\mathcal{J}/\partial\tau_y)$. Figure 10a,b depict the sensitivity of $\mathcal{J}$ to changes in zonal and meridional wind stress 14 days prior to evaluation of $\mathcal{J}$, corresponding to a 14-day adjoint integration. Surface wind stress drives large-scale horizontal momentum input to the ocean through the Ekman layer. This sensitivity helps describe how the wind stress drives eastward volume flow through the gap in our topography. Note that within Oceananigans, wind stresses are negative-east (zonal) and negative-north (meridional), so a negative gradient suggests an eastward or northward wind stress in that location increases zonal volume transport.

Sensitivity values for zonal wind stress τ_x are highest within the gap and progressively decrease further away from it, especially to the north and south. This sensitivity pattern is explained by the fact that eastward τ_x upstream of the gap (noting that the configuration is periodic) directly accelerates the upper ocean eastward, funneling it through the gap and increasing zonal transport. In general, τ_x gradients have the expected sign and magnitude.

Although our forward model configuration features only an idealized zonal wind stress, we may also consider the derivative of $\mathcal{J}$ with respect to meridional wind stress τ_y. Again, these gradients follow a reasonably expected pattern when accounting for sign conventions. On the west side of the topography they have opposite signs north and south of the gap, which reflect how τ_y controls the pressure difference across the gap via Ekman transport and surface map divergence. Similar, but opposite, sign values are seen in the gradients downstream of the gap. They produce weaker gradients in magnitude since they are not positioned directly upstream of the gap, although they still exert influence due to the periodic boundary conditions. Wind stress sensitivity patterns similar to those computed here have been obtained in MITgcm adjoint simulations with "realistic" Drake Passage topography (e.g., Fig. 6 of Losch & Heimbach (2007) but used longer integrations and opposite sign convention, or Fig. 4 of Kalmikov & Heimbach (2014)).

Sensitivities of the zonal volume transport across the gap to changes in initial temperature at two depth levels, $z = 15$ m and $z = 504$ m, are depicted in Fig. 11a,b. Re-

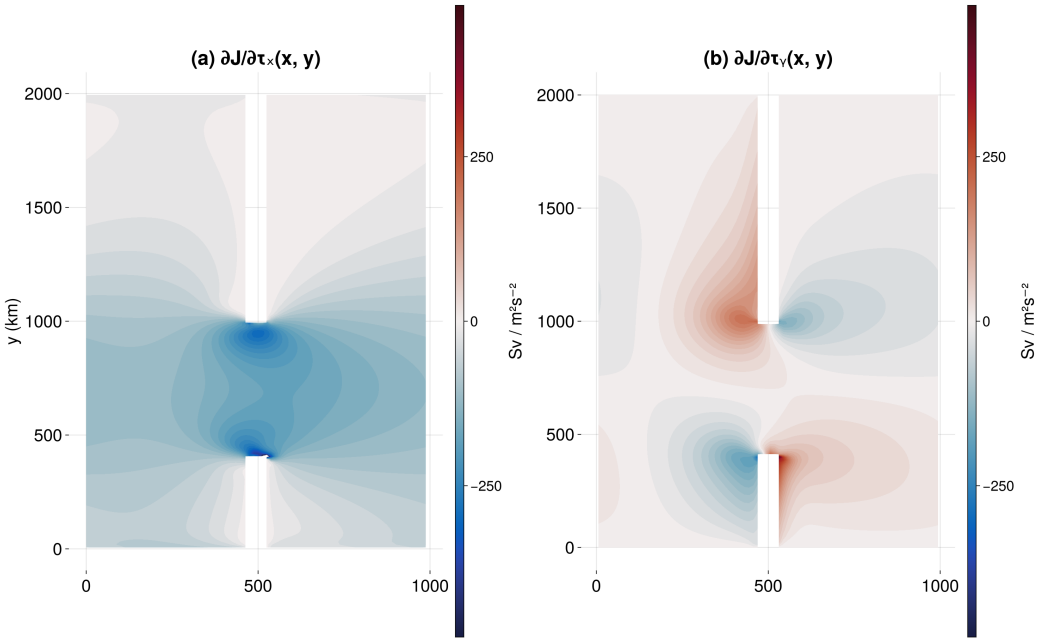

Figure 10: Sensitivities of zonal volume transport through the topography gap (Eq. (7)) with respect to zonal (a) and meridional (b) wind stress, τ_x and τ_y. This was a run of 8,100 time steps (approximately 14 days). Within Oceananigans, wind stresses are negative-east (zonal) and negative-north (meridional), so a negative gradient suggests an eastward or northward momentum flux (out of the atmosphere) in that location increases zonal volume transport.

lated, full-depth sensitivities are shown in Fig. 12 for a meridional section at the longitude of the gap ($x_0 = 550$ km, panel a) and for a zonal section at a latitude near the northern end of the gap ($y_0 = 1000$ km). The dipole pattern that builds near the northern end and upstream of the gap is visible in the zonal section and amplified at depth. Similarly, sensitivities are amplified at depth for the meridional section, both south ($y <$ 500 km) and north ($y > 1000$ km) of the gap. There are a couple reasonable explanations: we know that local warming creates a steeper meridional density gradient across the gap, which itself creates vertical shear in u via thermal wind ($\partial u/\partial z \propto \partial \rho/\partial y$). Moreover, the density gradient also raises steric height, which changes horizontal pressure gradients that drive zonal flow. Furthermore, the narrowing at the gap (and presence of topography) means the same horizontal pressure change creates a larger change in bottom pressure and forms stress that affects the momentum balance.

As a third category of sensitivities besides surface boundary condition and initial condition sensitivities, panels (c) and (d) of Fig. 11 showcase sensitivities of $\mathcal{J}$ to changes in the vertical diffusivity model parameter. Again, a spatially highly non-uniform impact of changes in vertical mixing on the transport is evident. A similarity in pattern between this sensitivity and initial temperature sensitivity is apparent, which, over the limited duration of the adjoint calculation is physically sensible. While the initial temperature sets the background stratification, the diffusivity field contributes to how it evolves. Altering the diffusivity which generally acts to even out tracer gradients will alter the baroclinic structure of the water column thus contributing to changes in thermal wind

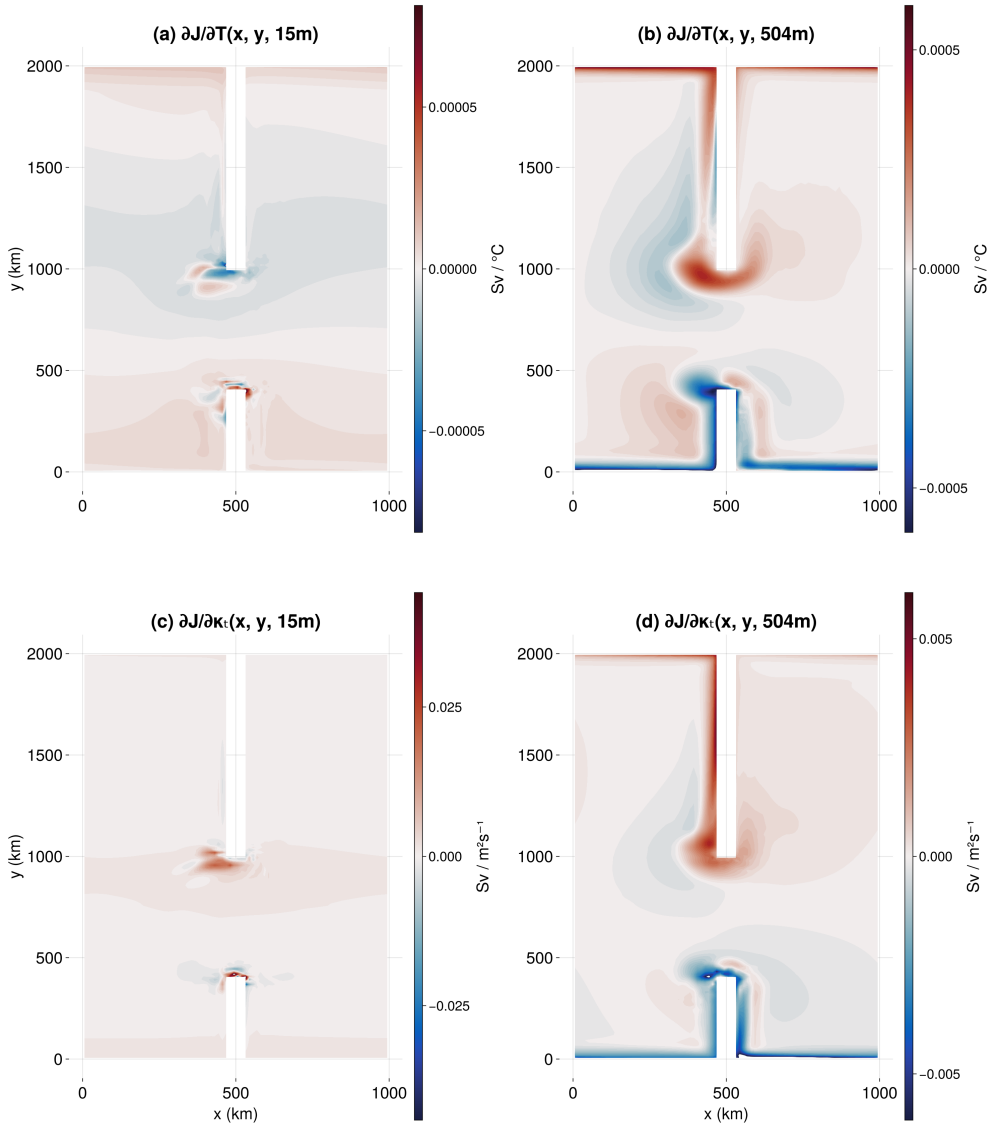

Figure 11: Sensitivities of zonal volume transport through the topography gap (Eq. (7)) with respect to initial temperature T (a and b), and vertical temperature diffusivity κ_T (c and d) at select depths.

shear and steric height (differentiating over a longer run may reveal new patterns in the diffusivity sensitivities). Similarly to the previous application (Fig. 6), finite-difference "gradient checks" have been conducted to verify the gradient computed with the adjoint for a representative range of elements of the different control variables (not shown).

Producing the gradients presented in the section required end-to-end differentiation of Oceananigans using Enzyme and Reactant. This, in turn, involved successful AD of a hydrostatic free-surface model featuring WENO momentum and tracer (temperature and salinity) advection, linear equations of state for buoyancy, volumetric forcings and flux boundary conditions, harmonic and biharmonic Smagorinsky-like turbulence

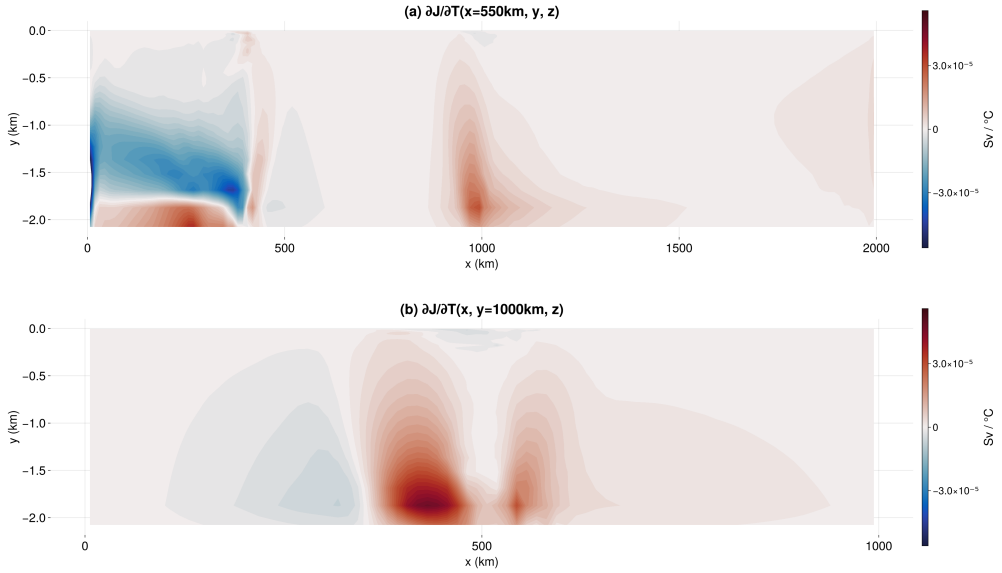

Figure 12: Sensitivities of $\mathcal{J}$ with respect to the initial temperature, shown across the full depth range as cross sections. Gradients are divided by the thickness of their associated layer. Top is along 550 km in the zonal direction (right of the gap in the ridge topography); bottom is along 1000 km in the meridional direction. Sensitivity values are divided by layer thickness to account for uneven cell thicknesses.

closures, and a periodic domain with masking by an idealized passage. A recurring problem with differentiating the Oceananigans code was type instability. Although the computationally intensive portions of the Oceananigans code base are type stable, the package features an extensive array of configuration options stored in nested tuples and other type-unstable structures. These configuration options do not impact the package's computational performance but do pose problems for Enzyme AD (see Section 2). Use of Reactant first produced type-stable model code that could then be successfully differentiated. Reactant also improved the runtime for CPU-based models by an order of magnitude, which helped with development, although the runs presented here were computed and differentiated by using a GPU backend.

5 Application 3: Ice Sheet Model

For our third example we employ the Differentiable JUlia ICE sheet model (`DJUICE.jl`). This model is essentially a carbon copy of the finite-element C++ Ice-Sheet and Sea-level System Model (ISSM, Larour et al., 2012). DJUICE follows ISSM's object-oriented structure, which requires a number of mutable structures, and has a large number of dynamic memory allocations. These two aspects make automatic differentiation particularly challenging. We show that `Enzyme` is able to differentiate static and transient models.

5.1 Inferring Basal Friction

First, we explore a standard problem in glaciology that involves inferring basal conditions, which typically cannot be measured, from surface observations (MacAyeal, 1992;

Morlighem et al., 2013). Ice sheet flow is modeled by using the Shelfy Stream Approximation (MacAyeal, 1989):

$$\begin{aligned}
\frac{\partial}{\partial x}\left(4H\mu\frac{\partial u}{\partial x}+2H\mu\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial y}\left(H\mu\frac{\partial u}{\partial y}+H\mu\frac{\partial v}{\partial x}\right)&=\rho g H\frac{\partial s}{\partial x}+\alpha^2 N u\\
\frac{\partial}{\partial y}\left(4H\mu\frac{\partial v}{\partial y}+2H\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial x}\left(H\mu\frac{\partial u}{\partial y}+H\mu\frac{\partial v}{\partial x}\right)&=\rho g H\frac{\partial s}{\partial y}+\alpha^2 N v,
\end{aligned} \tag{8}$$

where H is the ice thickness, u and v are the two components of the horizontal ice velocity vector, μ is the nonlinear ice viscosity that follows Glen's flow law (Glen, 1955), s is the ice surface elevation, N is the effective pressure at the base of the ice, and α is the unknown friction coefficient. To infer the spatially varying $\alpha(x, y)$, we typically minimize a cost function that measures the misfit between the modeled velocity, $\mathbf{u} = (u, v)$, and the satellite-derived observed ice velocity, $\mathbf{u}^{\text{obs}} = (u^{\text{obs}}, v^{\text{obs}})$:

$$\mathcal{J}(\alpha(x,y)) = \int_\Omega \frac{1}{2}\left\{ \left(u-u^{\text{obs}}\right)^2 + \left(v-v^{\text{obs}}\right)^2 \right\} d\Omega, \tag{9}$$

where Ω is the model domain. Automatic differentiation is used to determine the gradient of this cost function with respect to the spatial distribution of the basal friction coefficient $\alpha(x, y)$, which then feeds a standard gradient descent algorithm to infer an optimal field for $\alpha(x, y)$.

We apply this approach to Pine Island Glacier in West Antarctica. Our model has 18,227 elements on a two-dimensional unstructured mesh, with element sizes varying from 1 km to 20 km. We adopt the model configuration of Seroussi et al. (2014). The initial ice geometry is taken from BedMachine Antarctica (Morlighem et al., 2011) and the observed ice velocity from Rignot et al. (2011).

For comparison, we run an identical experiment with ISSM. Figure 13 compares the sensitivity $\frac{\partial \mathcal{J}}{\partial \alpha}$ obtained with ISSM and DJUICE, along with their difference. The root mean square difference between the two sensitivity fields is 7.87×10^{-5}. Notably, we used a relatively loose tolerance for the nonlinear solver, 0.01 for the relative residual, to achieve faster solves. Even under this setting the two packages agree to $\mathcal{O}(10^{-3})$.

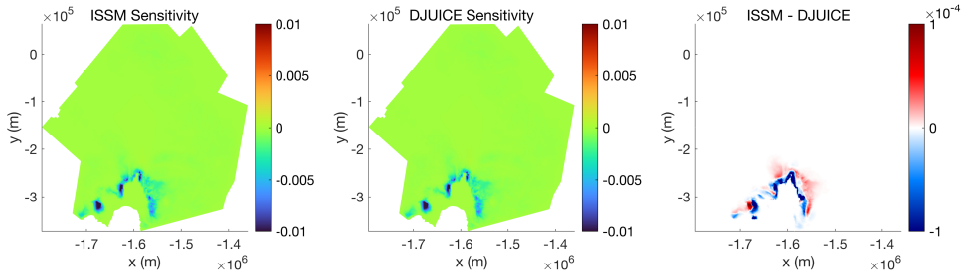

Figure 13: Sensitivity map $\frac{\partial \mathcal{J}}{\partial \alpha}$ ($\text{m}^{5/2}\text{s}^{-5/2}$) of the squared misfit between simulated and observed ice velocities, $\mathcal{J}(\alpha(x, y))$, to changes in the basal friction coefficient $\alpha(x, y)$, for Pine Island Glacier computed by using ISSM (left), DJUICE (middle), and their difference (right).

5.2 Sensitivity Mapping for a Transient Model

In addition to computing sensitivities of model-data misfit functions used for gradient-based optimization (preceding section), automatic differentiation can be used to map sensitivities of a wide range of quantities of interest. For example, Morlighem et al. (2021)

used ISSM and STREAMICE to map the sensitivity of Pine Island Glacier's future volume above floatation to basal friction and basal melt under the floating ice shelf. We apply the same experiment but with DJUICE instead of ISSM. The model mesh has 23,767 elements. We solve for the Shallow Shelf Approximation, and the geometry evolves in time based on the conservation of mass. We use a similar depth-dependent parameterization for basal melt:

$$\dot{m}(x,y) = m(x,y) + \begin{cases} 0 & \text{if } z \geq 0, \\ -\frac{1}{10}z & \text{if } 0 > z > -500, \\ 50 & \text{if } z \leq -500, \end{cases} \tag{10}$$

where z is the base elevation of the ice. Following Morlighem et al. (2021), we are interested in quantifying the spatial sensitivity of the volume above flotation (V) to perturbations in basal melting. For example, the Gâteaux derivative of V, $\mathcal{D}V(m)$, with respect to ocean melting, m, is

$$\forall \delta m \in \mathcal{H}^1(\Omega) \quad \langle \mathcal{D}V(m), \delta m \rangle = \lim_{\epsilon \to 0} \frac{V(m + \epsilon\, \delta m) - V(m)}{\epsilon}, \tag{11}$$

where δm indicates a perturbation in m, $\langle \cdot, \cdot \rangle$ is the inner product, and $\mathcal{H}^1(\Omega)$ denotes the space of square-integrable functions whose first derivatives are also square integrable on the model domain, Ω.

Enzyme computes the gradient of $\mathcal{J} = V$ with respect to m at each vertex of the mesh, and we recover $\mathcal{D}V(m)$ on the $\mathcal{H}^1(\Omega)$ space by multiplying this output by the mass matrix inverse. This procedure avoids mesh-dependency sensitivities, as described in Morlighem et al. (2021).

Instead of running the model for 20 years, we perform only 5 time iterations (half-year) given the computational cost of the model. The sensitivity maps on the ice shelf are shown in Fig. 14. The root mean square difference between the two sensitivity fields is 2.7368×10^3. Notably, we used the same loose tolerance, 0.01, for the relative residual in the nonlinear solver, as the experiment in Section 5.1.

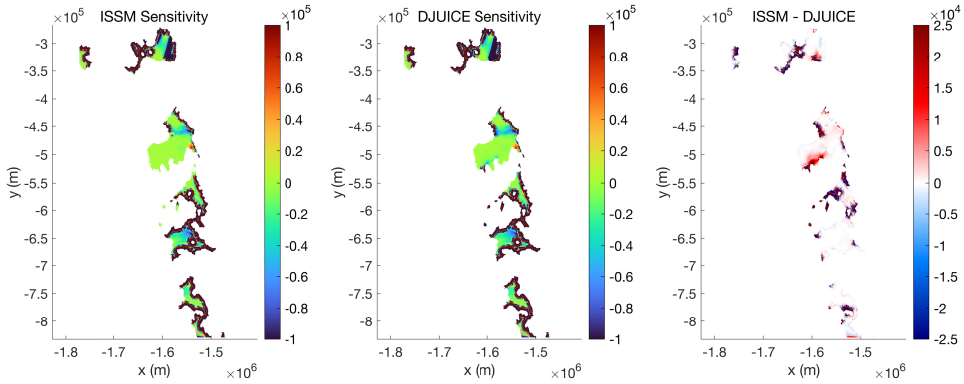

Figure 14: Sensitivity map $\mathcal{D}V(m)$ of the volume above flotation, $V(m(x,y))$, to changes in the melting perturbation $m(x,y)$ for the Amundsen Sea Embayment computed by using ISSM (left), DJUICE (middle), and their difference (right), in the unit of $\mathrm{m}^3/(\mathrm{m}^3/\mathrm{s})$.

6 Application 4: Atmospheric General Circulation Model

For our fourth technical example we analyze the general circulation of the atmosphere as simulated by SpeedyWeather.jl (Klöwer et al., 2024). To adapt it to usage with Enzyme.jl, we had to implement only minor changes. Type stability has been a central

programming paradigm of SpeedyWeather from the start, but Enzyme also requires this for performance-irrelevant code where we were less consistent. Then we slightly revised our state variable and scratch memory handling. The experiment shown here uses Enzyme.jl in combination with Checkpointing.jl.

As a spectral atmospheric model, SpeedyWeather.jl uses spherical harmonics in combination with a grid, discretizing the so-called primitive equations, which are widely used in numerical weather prediction, on the sphere. Each time step performs numerous spherical harmonic transforms to transfer variables between the gridpoint and spectral space. We use a horizontal resolution of T31 (spherical harmonics up to degree and order 31) combined with an octahedral Gaussian grid of 96 latitudes, corresponding to a 3.75° resolution at the equator (about 400 km globally) and eight vertical layers. The time step is 40 min using a semi-implicit filtered leapfrog scheme. The prognostic variables $\mathbf{P}$ are the relative vorticity $\zeta = \nabla \times \mathbf{u}$ and divergence $\mathcal{D} = \nabla \cdot \mathbf{u}$ of the horizontal wind vector $\mathbf{u}$, the logarithm of surface pressure $\ln p_s$, temperature T, and specific humidity q, each discretized in spectral space horizontally and in sigma coordinates (fraction of surface pressure) vertically. The primitive equations are

$$
\begin{aligned}
\frac{\partial \zeta}{\partial t} &= \nabla \times (\mathcal{P}_\mathbf{u} + (f+\zeta)\mathbf{u}_\perp - W(\mathbf{u}) - R_d T_v \nabla \ln p_s) \\
\frac{\partial \mathcal{D}}{\partial t} &= \nabla \cdot (\mathcal{P}_\mathbf{u} + (f+\zeta)\mathbf{u}_\perp - W(\mathbf{u}) - R_d T_v \nabla \ln p_s) - \nabla^2 \left(\frac{1}{2}(u^2+v^2) + \Phi \right) \\
\frac{\partial \ln p_s}{\partial t} &= -\frac{1}{p_s} \nabla \cdot \int_0^{p_s} \mathbf{u} \, \mathrm{d}p \\
\frac{\partial T}{\partial t} &= \mathcal{P}_T - \nabla \cdot (\mathbf{u}T) + T\mathcal{D} - W(T) + \frac{R_d}{c_p} T_v \frac{D \ln p}{Dt} \\
\frac{\partial q}{\partial t} &= \mathcal{P}_q - \nabla \cdot (\mathbf{u}q) + q\mathcal{D} - W(q),
\end{aligned}
\tag{12}
$$

with Coriolis parameter f, dry gas constant R_d, virtual temperature T_v, geopotential Φ, heat capacity c_p, and vertical advection operator W. Many atmospheric processes are summarized in $\mathcal{P}_\mathbf{u}$ (drag in the planetary boundary layer) and $\mathcal{P}_\mathbf{T}, \mathcal{P}_\mathbf{q}$ (e.g., radiation, convection, large-scale condensation, surface fluxes with land and ocean). SpeedyWeather's primitive equation model is coupled to a simple thermodynamic model of the ocean (a so-called slab ocean model), a thermodynamic sea-ice model, and a 2-layer land surface bucket model.

Sensitivity Analysis

We demonstrate the differentiability of SpeedyWeather by conducting a sensitivity analysis of the temperature $\mathcal{J} = T_f$ of the lowest atmospheric layer at a grid point in Denmark (55° N, 11° E) over a short integration of the model. We compute derivatives $\frac{\partial T_f}{\partial \mathbf{P}_0}$ of the final temperature T_f after 6 hours and 2 days of integration with respect to the initial conditions of the prognostic variables $\mathbf{P}_0 = \{\zeta_0, \ln p_{s0}, T_0\}$. For the sake of brevity we show only a few selected variables and layers in Fig. 15. As expected, the sensitivities decrease with distance from the selected grid point (locality principle in classical physics). They are more localized for the short 6-hour integration (Fig. 15 a-d) and spread during the course of the longer 2-day integration (Fig. 15 e-h). The vorticity and surface pressure of the 2-day integration, in particular, exhibit a sensitivity pattern that is consistent with the underlying westerly wind over the Atlantic causing an eastward transport (see arrows in Fig. 15).

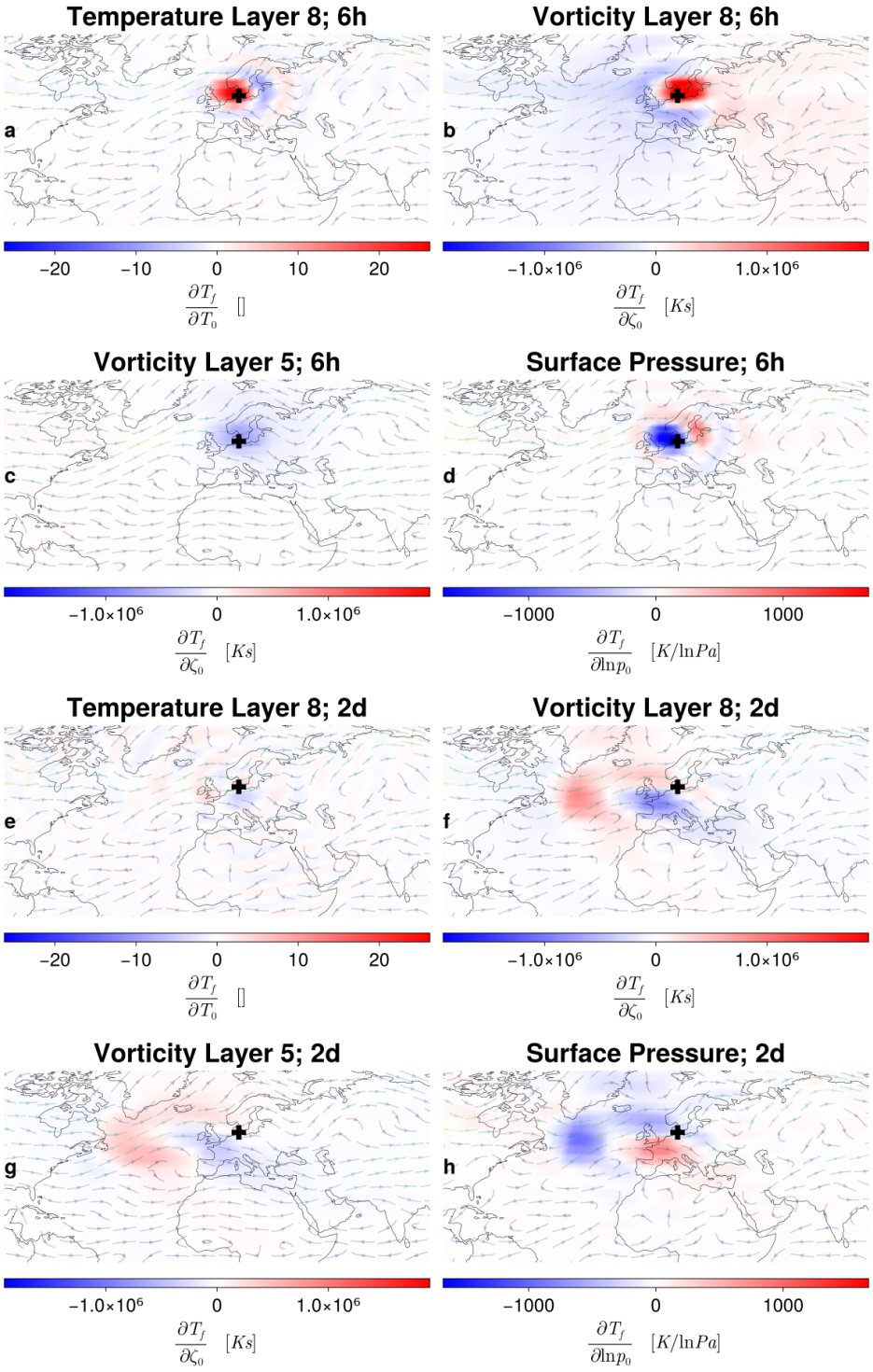

Figure 15: Sensitivities of the temperature of the lowest atmospheric layer in (55° N, 11° E) over Denmark, marked with a cross, with respect to the initial conditions of a 6-hour (a–d) and 2-day (e–h) integration of the SpeedyWeather.jl global atmospheric model. Arrows depict the wind vector field of the respective layer of the initial condition. Layer 8 corresponds to $\sigma = 0.9375$ (near-surface) and layer 5 to $\sigma = 0.5625$ (mid-troposphere), where σ is a fraction of surface pressure used as vertical coordinate.

7 Discussion

We have successfully differentiated four ESM components written in the Julia programming language. These models implement a range of spatial discretization methods, including finite-volume, finite-element, and spectral schemes, and bespoke numerical algorithms.

At the heart of this work is the use of the general-purpose AD tool Enzyme and its reverse mode. Other approaches exist for achieving differentiable models. Specifically within Julia, the SciML package (Rackauckas et al., 2020) is based on composable algorithms and solvers that make the availability of differentiable models notionally more straightforward. However, high-end and highly performant ESMs typically rely on highly customized algorithms that do not easily fit within such frameworks. Similar issues arise in the context of other customized programming languages such as JAX. A main motivation for exploring the general-purpose AD route within Julia was the already existing ESM components, in particular Oceananigans.jl and ClimaOcean.jl (Ramadhan et al., 2020; Silvestri et al., 2025; Wagner et al., 2025) that are being developed as part of CliMA (Yatunin et al., 2025), as well as the flexible, light-weight atmospheric general circulation model (GCM) SpeedyWeather.jl (Klöwer et al., 2024). A new ice sheet model. DJUICE.jl, was rewritten from an existing C++ code to complement the Julia-based ESM components. None of this software was written for Enzyme, and only relatively small changes had to be implemented to use Enzyme successfully. This contrasts with models written in JAX, such as the ocean model *Veros* (Häfner et al., 2021) and the atmospheric model *NeuralGCM* (Kochkov et al., 2024), which often demand more extensive adaption. Nevertheless, achieving full end-to-end differentiation with Enzyme required several important extensions and tool developments, available and reusable now. These efforts focused on key features of the Julia programming language, including JIT compilation, dynamic dispatch, and memory management via garbage collection.

Two major aspects that favored the choice of the emerging AD tool Enzyme over existing tools such as Zygote.jl were the requirement to efficiently handle mutable arrays, which are ubiquitous in time-stepping ESM components, and Enzyme's performance characteristics. A major novelty of Enzyme over existing AD tools is that it acts at the LLVM compiler's intermediate representation level, thus enabling code optimization both before and after algorithmic differentiation takes place. This has shown to deliver more efficient derivative calculations compared with other AD tools.

The ESM components also necessitated work to integrate reverse-mode checkpointing algorithms. This was achieved in two ways: (i) integration of Checkpointing.jl (Schanen et al., 2023) within Enzyme and (ii) development of checkpointing algorithms at the MLIR level. The latter was required for workflows that use Reactant in combination with Enzyme (see Application 2 showcased in Section 4).

The value of the tight collaboration between Earth system model developers and computer scientists cannot be overestimated in driving significant improvements and maturation of the capabilities of software transformation tools featured here, Enzyme and Reactant, that were critical to the work. In particular, both of these software packages aim to consume and rewrite generic programs for either differentiation or improved performance/portability, respectively. Rewriting general code is a major task, especially in the context of comprehensive ESMs, and likely to fail if attempted all at once. Instead, both software projects adopted an incremental approach: they began with a limited set of features, ensured full support for these, and gradually expanded the feature set until all functionalities required by the various ESM components were covered. Co-developing the scientific simulation features alongside the Enzyme and Reactant software tools that support them was key to the success of all projects. Arguably, such tight collaborations are easier to achieve in smaller communities like those around the Julia programming language.

In terms of applications, we worked through a hierarchy of ESM components, at each step creating minimal reproducible examples (MREs) to unblock AD tool limitations that were encountered at the time. A first application (not presented here) used a simple three-box model of the ocean's thermohaline circulation inspired by Stommel (Stommel, 1961; Tziperman & Ioannou, 2002). This work motivated the initial development of Checkpointing.jl (Schanen et al., 2023) and drove the support for handling Julia's dynamic dispatch within Enzyme.

Moving up in terms of model complexity, we subjected a shallow water model for a fluid on the rotating beta plane to Enzyme and checkpointing to investigate scalability and performance aspects (Section 3). The work on the shallow water model helped identify bottlenecks in the early Enzyme versions through the provision of MREs that provided rapid tool fixes. In this way, it also supported the differentiation of the comprehensive finite-volume, vertical height-coordinate ocean general circulation model Oceananigans, which we conducted in parallel with the shallow water model work.

The power of the Reactant tool was exposed in the work on Oceananigans, our second application. The Reactant pipeline offers reduction in code complexity through a tracing approach along with automated performance portability across different HPC hardware (in our case using CPUs and GPUs). The Multi-Level Intermediate Representation created by this tool provides more stable code that Enzyme can transform robustly and efficiently. An added benefit of investing in the Reactant pipeline is the ability to lower both the parent and the Enzyme-differentiated code to the XLA compiler, which provides code optimization for high-performance execution across different compute hardware including CPUs, GPUs, TPUs, and emerging ML accelerators. Given the rapid ML-driven hardware development, this work offers the prospect of automated performance portability across a range of emerging HPC platforms that will become available for ESM simulations in both research and industry, particularly within the ML-driven sector.

Our third application, DJUICE, relies heavily on mutable arrays and mutable structures, which make other AD tools such as Zygote.jl and Diffractor.jl, which do not support mutation, impractical for this application. Mutation is essential for large climate models, since reallocating memory at every update would quickly exhaust resources and hinder GPU acceleration. Significant developments were required for Enzyme.jl to support mutation. Another important development necessary to differentiate DJUICE was to properly handle the differentiation of the backslash operator. DJUICE uses implicit solvers and requires solving large linear systems. One remaining point of development is to support sparse arrays. Currently Enzyme.jl supports only standard arrays, likely limiting the performance of the adjoint. We are working on adding support for `SparseArray.jl` in order to further improve the performance of the code, as the left-hand side of the linear systems (i.e., the stiffness matrices) are highly sparse. A related study by Utkin et al. (2025) used Enzyme.jl to generate the adjoint of a simple glacier model based on a depth-averaged shallow ice approximation to simulate Alpine mountain valley glaciers, further demonstrating the versatility of the AD tool.

The developments of Enzyme.jl and Checkpointing.jl that enabled differentiability of the shallow water, ocean, and ice sheet models described above subsequently facilitated their application to the spectral atmospheric general circulation model SpeedyWeather. Similarly to the other showcased models, SpeedyWeather's computations rely heavily on mutating data structures. Adapting it to the usage with other AD tools would therefore have been prohibitively impractical. Adapting it to Enzyme.jl, on the other hand, required fairly minor revisions: ensuring type stability throughout the model, slightly restructuring how variables are handled during time stepping, and defining two differentiation rules for the transforms used. The sensitivity analysis shown here is just the first step demonstrating successful gradient calculation via reverse-mode AD.

The availability of differentiable ESM components offers a range of exciting opportunities for advancing data-constrained, data-driven, and mixed modeling approaches. A main incentive of this development has been the recognition of the conceptual and algorithmic similarity between adjoint-based inverse methods and backpropagation-based neural network learning. Combining these two approaches enables the embedding of NN architectures within physics-based models where the goal is to faithfully represent conservation laws but to learn empirical subgrid-scale parameterization schemes. This holds for climate applications, in particular, which rely on long integration that requires stable schemes, and where property conservation plays an essential role to detect small residuals in the climate change signal within the noise of natural variability. Differentiable ESMs offer the prospect of better utilizing "training data" through gradient-based optimization, whether derived from observations of the climate system, associated climatologies, or high-fidelity data from higher-resolution or more complex simulations. This approach has been referred to as "online learning," "full-model learning," or "a posteriori learning" in the recent literature and has been investigated in a number of idealized quasi-geostrophic simulations (e.g., Frezat et al. (2022); Maddison (2024); Yan et al. (2025)). Our work represents a breakthrough in that it makes these approaches feasible for a range of high-end ESMs. To date, only one study has demonstrated this approach with NeuralGCM, a spectral model using similar numerics to SpeedyWeather but written in JAX (Kochkov et al., 2024). The ability to conduct such approaches efficiently on AI-customized compute hardware (GPUs, TPUs) further unleashes the potential of seamless integration of physics-based and ML algorithms for ESM learning. We hope this work encourages wider adoption of such methods in the modeling community, leading to a greater use of observations for constraining and more rigorously calibrating ESMs.

Open Research Section

The frameworks used in this work are Enzyme.jl, Reactant.jl, and Checkpointing.jl. These were applied to four ESM components: ShallowWater.jl, Oceananigans.jl, DJUICE.JL, and SpeedyWeather.jl. Because of the different provenances of these software packages, we are making them available as sub-modules through a central GitHub repository at `https://github.com/DJ4Earth/differentiable-esm-components-2025`. Scripts to reproduce the simulations and figures are contained in the sub-modules for each application. All software packages are open-source.

Conflict of Interest declaration

The authors declare there are no conflicts of interest for this manuscript.

Acknowledgments

This work was supported by NSF CSSI grants #2103942, 2147601, 2103791, 2104068, and 2103804 (Collaborative Research: Frameworks: Convergence of Bayesian inverse methods and scientific machine learning in Earth system models through universal differentiable programming). Additional support was provided by the Applied Mathematics activity within the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Applied Mathematics under Contract No. DE-AC02-06CH11357. MG acknowledges funding from the Volkswagen Foundation. MK acknowledges funding from the Natural Environment Research Council under grant number UKRI191.

References

Abernathey, R., Marshall, J., & Ferreira, D. (2011). The dependence of Southern Ocean meridional overturning on wind stress. *Journal of Physical Oceanography*,

41(12), 2261–2278.

Alhashim, M. G., Hausknecht, K., & Brenner, M. P. (2025). Control of flow behavior in complex fluids using automatic differentiation. *Proceedings of the National Academy of Sciences*, *122*(8), e2403644122.

Badgeley, J. A., Morlighem, M., & Seroussi, H. (2025). Increased sea-level contribution from northwestern Greenland for models that reproduce observations. *Proceedings of the National Academy of Sciences*, *122*(25), e2411904122. doi: 10.1073/pnas.2411904122

Balaji, V., Couvreux, F., Deshayes, J., Gautrais, J., Hourdin, F., & Rio, C. (2022). Are general circulation models obsolete? *Proceedings of the National Academy of Sciences*, *119*(47), e2202075119. doi: 10.1073/pnas.2202075119

Balaji, V., Maisonnave, E., Zadeh, N., Lawrence, B. N., Biercamp, J., Fladrich, U., … Wright, G. (2016). CPMIP: measurements of real computational performance of Earth system models in CMIP6. *Geoscientific Model Development*, *10*(1), 19–34. doi: 10.5194/gmd-10-19-2017

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, *18*, 1–43. Retrieved from http://jmlr.org/papers/v18/17-468.html

Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., & Tian, Q. (2023, July). Accurate medium-range global weather forecasting with 3D neural networks. *Nature*, *619*(7970), 533–538. Retrieved 2024-03-12, from https://www.nature.com/articles/s41586-023-06185-3 (Publisher: Nature Publishing Group) doi: 10.1038/s41586-023-06185-3

Blondel, M., & Roulet, V. (2024). The Elements of Differentiable Programming. *arXiv*. Retrieved from https://doi.org/10.48550/arXiv.2403.14606 doi: 10.48550/arxiv.2403.14606

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., … others (2018). JAX: composable transformations of Python+ NumPy programs. *http://github.com/google/jax*.

Bryson, A. E., & Ho, Y.-C. (1975). *Applied optimal control: optimization, estimation and control*. Taylor and Francis.

Chizat, L., Oyallon, E., & Bach, F. (2019). On lazy training in differentiable programming. *Advances in neural information processing systems*, *32*.

Christensen, H., & Zanna, L. (2022). Parametrization in Weather and Climate Models. *Oxford Research Encyclopedia of Climate Science*. doi: 10.1093/acrefore/9780190228620.013.826

Dheeshjith, S., Subel, A., Adcroft, A., Busecke, J., Fernandez-Granda, C., Gupta, S., & Zanna, L. (2025). Samudra: An AI global ocean emulator for climate. *Geophysical Research Letters*, *52*(10), e2024GL114318.

Errico, R. M., & Vukicevic, T. (1992). Sensitivity analysis using an adjoint of the PSU-NCAR mesoseale model. *Monthly Weather Review*, *120*(8), 1644–1660. Retrieved from http://journals.ametsoc.org/doi/abs/10.1175/1520-0493%281992%29120%3C1644%3ASAUAAO%3E2.0.CO%3B2 doi: 10.1175/1520-0493(1992)120<1644:sauaao>2.0.co;2

Espinosa, Z. I., Sheshadri, A., Cain, G. R., Gerber, E. P., & DallaSanta, K. J. (2022). Machine learning gravity wave parameterization generalizes to capture the QBO and response to increased CO2. *Geophysical Research Letters*, *49*(8), e2022GL098174.

Eyring, V., Cox, P. M., Flato, G. M., Gleckler, P. J., Abramowitz, G., Caldwell, P., … Williamson, M. S. (2019). Taking climate model evaluation to the next level. *Nature Climate Change*, *9*(2), 102 – 110. Retrieved from https://www.nature.com/articles/s41558-018-0355-y doi: 10.1038/s41558-018-0355-y

Frezat, H., Sommer, J. L., Fablet, R., Balarac, G., & Lguensat, R. (2022). A posteri-

ori learning for quasi-geostrophic turbulence parametrization. *Journal of Advances in Modeling Earth Systems*, *14*(11). doi: 10.1029/2022ms003124

Fukumori, I., Wang, O., Llovel, W., Fenty, I., & Forget, G. (2015). A near-uniform fluctuation of ocean bottom pressure and sea level across the deep ocean basins of the Arctic Ocean and the Nordic Seas. *Progress in Oceanography*, *134*(C), 152–172. Retrieved from http://dx.doi.org/10.1016/j.pocean.2015.01.013 doi: 10.1016/j.pocean.2015.01.013

Gaikwad, S. S., Narayanan, S. H. K., Hascoët, L., Campin, J.-M., Pillar, H., Nguyen, A., … Heimbach, P. (2025). MITgcm-AD v2: Open source tangent linear and adjoint modeling framework for the oceans and atmosphere enabled by the automatic differentiation tool Tapenade. *Future Generation Computer Systems*, *163*, 107512. doi: https://doi.org/10.1016/j.future.2024.107512

Gelbrecht, M., White, A., Bathiany, S., & Boers, N. (2023). Differentiable programming for Earth system modeling. *Geoscientific Model Development*, *16*(11), 3123–3135. doi: 10.5194/gmd-16-3123-2023

Giering, R., & Kaminski, T. (1998). Recipes for adjoint code construction. *ACM Transactions on Mathematical Software (TOMS)*, *24*(4), 437–474. Retrieved from https://doi.org/10.1145/293686.293695 doi: 10.1145/293686.293695

Glen, J. W. (1955). The creep of polycrystalline ice. *Proc. R. Soc. A*, *228*(1175), 519–538.

Griewank, A. (2003). A mathematical view of automatic differentiation. *Acta Numerica*, *12*, 321–398. Retrieved from https://www.cambridge.org/core/product/identifier/S0962492902000132/type/journal_article doi: 10.1017/s0962492902000132

Griewank, A. (2012). Who invented the reverse mode of differentiation? *Documenta Math.*, 389–400.

Griewank, A., & Walther, A. (2000, March). Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, *26*(1), 19–45. doi: 10.1145/347837.347846

Griewank, A., & Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics (SIAM). Retrieved from https://doi.org/10.1137/1.9780898717761 doi: 10.1137/1.9780898717761

Hascoet, L., & Pascual, V. (2013). The Tapenade automatic differentiation tool: principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, *39*(3), 1–43.

He, S., Li, X., DelSole, T., Ravikumar, P., & Banerjee, A. (2021). Sub-seasonal climate forecasting via machine learning: Challenges, analysis, and advances. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 35, pp. 169–177).

Heimbach, P., Hill, C., & Giering, R. (2002). Automatic Generation of Efficient Adjoint Code for a Parallel Navier-Stokes Solver. In (Vol. 2330, pp. 1019 – 1028). Retrieved from http://link.springer.com/10.1007/3-540-46080-2_107 doi: 10.1007/3-540-46080-2_107

Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., … Williamson, D. (2016). The art and science of climate model tuning. *Bulletin of the American Meteorological Society*, *98*(3), 589–602. Retrieved from http://journals.ametsoc.org/doi/10.1175/BAMS-D-15-00135.1 doi: 10.1175/bams-d-15-00135.1

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). doi: 10.1029/2021ms002717

Hückelheim, J., Menon, H., Moses, W., Christianson, B., Hovland, P., & Hascoët, L. (2024). A taxonomy of automatic differentiation pitfalls. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. doi: 10.1002/widm.1555

Isaac, T., Petra, N., Stadler, G., & Ghattas, O. (2015). Scalable and efficient algorithms for the propagation of uncertainty from data through inference to prediction for large-scale problems, with application to flow of the Antarctic ice sheet. *Journal of Computational Physics*, *296*(C), 348–368. Retrieved from http://dx.doi.org/10.1016/j.jcp.2015.04.047 doi: 10.1016/j.jcp.2015.04.047

Janisková, M., & Lopez, P. (2013). Data assimilation for atmospheric, oceanic and hydrologic applications (Vol. II). *Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications (Vol. II)*, 251–286. doi: 10.1007/978-3-642-35088-7_11

Kalmikov, A. G., & Heimbach, P. (2014). A Hessian-Based Method for Uncertainty Quantification in Global Ocean State Estimation. *SIAM Journal on Scientific Computing*, *36*(5), S267 – S295. Retrieved from http://epubs.siam.org/doi/abs/10.1137/130925311 doi: 10.1137/130925311

Kaminski, T., Kauker, F., Pedersen, L. T., Voßbeck, M., Haak, H., Niederdrenk, L., … Gråbak, O. (2018). Arctic Mission Benefit Analysis: impact of sea ice thickness, freeboard, and snow depth products on sea ice forecast performance. *The Cryosphere*, *12*(8), 2569–2594. Retrieved from https://www.the-cryosphere.net/12/2569/2018/ doi: 10.5194/tc-12-2569-2018

Kaminski, T., Knorr, W., Schürmann, G., Scholze, M., Rayner, P. J., Zaehle, S., … Ziehn, T. (2013). The BETHY/JSBACH Carbon Cycle Data Assimilation System: experiences and challenges. *Journal of Geophysical Research: Biogeosciences*, *118*(4), 1414–1426. doi: 10.1002/jgrg.20118

Kedward, L. J., Aradi, B., Čertík, O., Curcic, M., Ehlert, S., Engel, P., … Vandenplas, J. (2022). The state of Fortran. *Computing in Science & Engineering*, *24*(2), 63–72. doi: 10.1109/mcse.2022.3159862

Kennedy, P. D., Banerjee, A., Köhl, A., & Stammer, D. (2025). Long-window tandem variational data assimilation methods for chaotic climate models tested with the Lorenz 63 system. *Nonlinear Processes in Geophysics*, *32*(3), 353–365.

Klöwer, M., Düben, P. D., & Palmer, T. N. (2020, August). Number formats, error mitigation, and scope for 16-bit arithmetics in weather and climate modeling analyzed with a shallow water model. *Journal of Advances in Modeling Earth Systems*, *12*(10), e2020MS002246. doi: 10.1029/2020MS002246

Klöwer, M., Hatfield, S., Croci, M., Düben, P. D., & Palmer, T. N. (2022). Fluid simulations accelerated with 16 bits: Approaching 4x speedup on A64FX by squeezing ShallowWaters.jl into Float16. *Journal of Advances in Modeling Earth Systems*, *14*(2), e2021MS002684.

Klöwer, M., Gelbrecht, M., Hotta, D., Willmert, J., Silvestri, S., Wagner, G. L., … Hill, C. (2024). SpeedyWeather.jl: Reinventing atmospheric general circulation models towards interactivity and extensibility. *Journal of Open Source Software*, *9*(98), 6323. Retrieved from https://doi.org/10.21105/joss.06323 doi: 10.21105/joss.06323

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., & Hoyer, S. (2021). Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, *118*(21), e2101784118.

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., … Hoyer, S. (2024). Neural general circulation models for weather and climate. *Nature*, 1–7. Retrieved from https://doi.org/10.1038/s41586-024-07744-y doi: 10.1038/s41586-024-07744-y

Kostov, Y., Johnson, H. L., Marshall, D. P., Heimbach, P., Forget, G., Holliday, N. P., … Smith, T. (2021). Distinct sources of interannual subtropical and subpolar Atlantic overturning variability. *Nature Geoscience*, *14*(7), 491–495. Retrieved from http://dx.doi.org/10.1038/s41561-021-00759-4 doi: 10.1038/s41561-021-00759-4

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., … Battaglia, P. (2023, December). Learning skillful medium-range

global weather forecasting. *Science*. Retrieved 2024-03-05, from https://www.science.org/doi/10.1126/science.adi2336 (Publisher: American Association for the Advancement of Science) doi: 10.1126/science.adi2336

Larour, E., Seroussi, H., Morlighem, M., & Rignot, E. (2012, Mar). Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM). *J. Geophys. Res.*, *117*(F01022), 1-20. doi: 10.1029/2011JF002140

Larour, E., Utke, J., Csatho, B., Schenk, A., Seroussi, H., Morlighem, M., … Khazendar, A. (2014). Inferred basal friction and surface mass balance of the Northeast Greenland Ice Stream using data assimilation of ICESat (Ice Cloud and land Elevation Satellite) surface altimetry and ISSM (Ice Sheet System Model). *The Cryosphere*, *8*(6), 2335–2351. Retrieved from http://www.the-cryosphere.net/8/2335/2014/ doi: 10.5194/tc-8-2335-2014

Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., … Zinenko, O. (2021). Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 ieee/acm international symposium on code generation and optimization (cgo)* (pp. 2–14).

Lea, D. J., Allen, M. W., & Haine, T. W. N. (2000). Sensitivity analysis of the climate of a chaotic system. *Tellus A*, *52*(5), 523–532. doi: 10.1034/j.1600-0870.2000.01137.x

Liang, X., & Yu, L. (2016). Variations of the global net air–sea heat flux during the "Hiatus" Period (2001–10). *Journal of Climate*, *29*(10), 3647–3660. Retrieved from http://journals.ametsoc.org/doi/10.1175/JCLI-D-15-0626.1 doi: 10.1175/jcli-d-15-0626.1

Loose, N., & Heimbach, P. (2021). Leveraging uncertainty quantification to design ocean climate observing systems. *Journal of Advances in Modeling Earth Systems*, *13*(4), e2020MS002386.

Losch, M., & Heimbach, P. (2007). Adjoint Sensitivity of an Ocean General Circulation Model to Bottom Topography. *Journal of Physical Oceanography*, *37*(2), 377–393. doi: 10.1175/jpo3017.1

Lücke, M. P., Zinenko, O., Moses, W. S., Steuwer, M., & Cohen, A. (2025). The MLIR transform dialect: Your compiler is more powerful than you think. In *Proceedings of the 23rd acm/ieee international symposium on code generation and optimization* (pp. 241–254).

MacAyeal, D. R. (1989, APR 10). Large-scale ice flow over a viscous basal sediment: Theory and application to Ice Stream B, Antarctica. *J. Geophys. Res.*, *94*(B4), 4071–4087.

MacAyeal, D. R. (1992, JAN 10). The basal stress distribution of Ice Stream E, Antarctica, Inferred by control methods. *J. Geophys. Res.*, *97*(B1), 595–603.

Maddison, J. R. (2024). Online learning in idealized ocean gyres. *arXiv*. doi: 10.48550/arxiv.2412.06393

Magnin, Alves, Arnoud, Markus, & Marzino, E. (2023). Fortran... et puis quoi encore ? *Bulletin 1024*(22), 143–161. doi: 10.48556/sif.1024.22.143

Margossian, C. C. (2019). A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *9*(4), 1 – 19. Retrieved from https://doi.org/10.1002/widm.1305 doi: 10.1002/widm.1305

Marotzke, J., Giering, R., Zhang, K. Q., Stammer, D., Hill, C., & Lee, T. (1999). Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research: Oceans*, *104*(C12), 29529–29547. Retrieved from https://doi.org/10.1029/1999JC900236 doi: 10.1029/1999jc900236

Metz, L., Freeman, C. D., Schoenholz, S. S., & Kachman, T. (2021). Gradients are not all you need. *arXiv preprint arXiv:2111.05803*.

Moore, A. M., Arango, H. G., Broquet, G., Powell, B. S., Weaver, A. T., & Zavala-Garay, J. (2011). The Regional Ocean Modeling System (ROMS) 4-dimensional variational data assimilation systems: Part I - System overview and formulation. *Progress in Oceanography*, *91*(1), 34 – 49. Retrieved from http://dx.doi.org/10.1016/j.pocean.2011.05.004 doi: 10.1016/j.pocean.2011.05.004

Moore, A. M., Arango, H. G., Lorenzo, E. D., Cornuelle, B. D., Miller, A. J., & Neilson, D. J. (2004). A comprehensive ocean prediction and analysis system based on the tangent linear and adjoint of a regional ocean model. *Ocean Modelling*, *7*(1-2), 227–258. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S146350030300057X doi: 10.1016/j.ocemod.2003.11.001

Morlighem, M., Goldberg, D., Dias dos Santos, T., Lee, J., & Sagebaum, M. (2021). Mapping the sensitivity of the Amundsen Sea Embayment to changes in external forcings using automatic differentiation. *Geophys. Res. Lett.*, *48*(23), e2021GL095440. Retrieved from https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021GL095440 doi: 10.1029/2021GL095440

Morlighem, M., Rignot, E., Seroussi, H., Larour, E., Ben Dhia, H., & Aubry, D. (2011). A mass conservation approach for mapping glacier ice thickness. *Geophysical Research Letters*, *38*(19). Retrieved from https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2011GL048659 doi: https://doi.org/10.1029/2011GL048659

Morlighem, M., Seroussi, H., Larour, E., & Rignot, E. (2013, SEP). Inversion of basal friction in Antarctica using exact and incomplete adjoints of a higher-order model. *J. Geophys. Res.*, *118*(3), 1746–1753. doi: 10.1002/jgrf.20125

Moses, W., & Churavy, V. (2020). Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. *Advances in Neural Information Processing Systems*, *33*, 12472–12485.

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).

Moses, W. S., Hari Krishna Narayanan, S., Paehler, L., Churavy, J., Valentinand Hückelheim, Schanen, M., Doerfert, J., & Hovland, P. (2022). Scalable automatic differentiation of multiple parallel paradigms through compiler augmentation. In *SC '22: Proceedings of the international conference for high performance computing, networking, storage and analysis.* New York, NY, USA: Association for Computing Machinery.

Muchnick, S. S. (1997). *Advanced compiler design and implementation.* Morgan Kaufmann.

Naumann, U., Lotz, J., Leppkes, K., & Towara, M. (2015). Algorithmic differentiation of numerical methods. *ACM Transactions on Mathematical Software (TOMS)*, *41*(4), 1–21. doi: 10.1145/2700820

Pacaud, F., Shin, S., Montoison, A., Schanen, M., & Anitescu, M. (2024). Condensed-space methods for nonlinear programming on GPUs. *arXiv preprint arXiv:2405.14236*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

Perkins, W. A., Brenowitz, N. D., Bretherton, C. S., & Nugent, J. M. (2023). Emulation of cloud microphysics in a climate model. *Authorea Preprints*.

Pillar, H. R., Heimbach, P., Johnson, H. L., & Marshall, D. P. (2016). Dynamical attribution of recent variability in Atlantic overturning. *Journal of Climate*, *29*(9), 3339–3352. Retrieved from http://journals.ametsoc.org/doi/10.1175/JCLI-D-15-0727.1 doi: 10.1175/jcli-d-15-0727.1

Pires, C., Vautard, R., & Talagrand, O. (1996). On extending the limits of variational assimilation in nonlinear chaotic systems. *Tellus A*, *48*(1), 96–121. Retrieved from http://tellusa.net/index.php/tellusa/article/view/11634 doi: 10.3402/tellusa.v48i1.11634

Rabier, F., Järvinen, H., Klinker, E., Mahfouf, J. F., & Simmons, A. (2000). The ECMWF operational implementation of four-dimensional variational assimilation. I: Experimental results with simplified physics. *Quarterly Journal of the Royal Meteorological Society*, *126*(564), 1143–1170. Retrieved from https://doi.org/10.1002/qj.49712656415 doi: 10.1002/qj.49712656415

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., … Edelman, A. (2020). Universal differential equations for scientific machine learning. *arXiv*, 1 – 45. Retrieved from https://arxiv.org/abs/2001.04385v3 doi: 10.48550/arxiv.2001.04385

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), 2018. Retrieved from https://joss.theoj.org/papers/10.21105/joss.02018 doi: 10.21105/joss.02018

Randall, D. A., Bitz, C. M., Danabasoglu, G., Denning, A. S., Gent, P. R., Gettelman, A., … Thuburn, J. (2019). 100 years of Earth system model development. *Meteorological Monographs*, *59*, 12.1–12.66. Retrieved from http://journals.ametsoc.org/doi/10.1175/AMSMONOGRAPHS-D-18-0018.1 doi: 10.1175/amsmonographs-d-18-0018.1

Rignot, E., Mouginot, J., & Scheuchl, B. (2011). Ice flow of the Antarctic ice sheet. *Science*, *333*(6048), 1427–1430. Retrieved from https://www.science.org/doi/abs/10.1126/science.1208336 doi: 10.1126/science.1208336

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. Retrieved from https://doi.org/10.1038/323533a0 doi: 10.1038/323533a0

Sapienza, F., Bolibar, J., Schäfer, F., Groenke, B., Pal, A., Boussange, V., … Rackauckas, C. (2025). Differentiable programming for differential equations: A review. *SIAM Review*, *in press*. Retrieved from https://doi.org/10.48550/arXiv.2406.09699 doi: 10.48550/arxiv.2406.09699

Schanen, M., Narayanan, S. H. K., Williamson, S., Churavy, V., Moses, W. S., & Paehler, L. (2023). Transparent checkpointing for automatic differentiation of program loops through expression transformations. In *Computational science - iccs 2023: 23rd international conference, prague, czech republic, july 3-5, 2023, proceedings, part iii* (pp. 483–497). Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/978-3-031-36024-4_37

Schneider, T., Behera, S., Boccaletti, G., Deser, C., Emanuel, K., Ferrari, R., … Yamagata, T. (2023). Harnessing AI and computing to advance climate modelling and prediction. *Nature Climate Change*, *13*(9), 887–889. doi: 10.1038/s41558-023-01769-3

Schneider, T., Lan, S., Stuart, A., & Teixeira, J. (2017). Earth system modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, *44*(24), 12,396–12,417. Retrieved from http://doi.wiley.com/10.1002/2017GL076101 doi: 10.1002/2017gl076101

Seroussi, H., Morlighem, M., Rignot, E., Mouginot, J., Larour, E., Schodlok, M., & Khazendar, A. (2014). Sensitivity of the dynamics of Pine Island Glacier, West Antarctica, to climate forcing for the next 50 years. *The Cryosphere*, *8*(5), 1699–1710. Retrieved from https://tc.copernicus.org/articles/8/1699/2014/ doi: 10.5194/tc-8-1699-2014

Shen, C., Appling, A. P., Gentine, P., Bandai, T., Gupta, H., Tartakovsky, A., … Lawson, K. (2023). Differentiable modelling to unify machine learning and phys-

ical models for geosciences. *Nature Reviews Earth & Environment*, 1–16. doi: 10.1038/s43017-023-00450-9

Shin, S., Coffrin, C., Sundar, K., & Zavala, V. M. (2021). Graph-based modeling and decomposition of energy infrastructures. *IFAC-PapersOnLine*, *54*(3), 693–698.

Silvestri, S., Wagner, G. L., Constantinou, N. C., Hill, C. N., Campin, J., Souza, A. N., … Ferrari, R. (2025). A GPU-based ocean dynamical core for routine mesoscale-resolving climate simulations. *Journal of Advances in Modeling Earth Systems*, *17*(4). doi: 10.1029/2024ms004465

Stammer, D. (2005). Adjusting internal model errors through ocean state estimation. *Journal of Physical Oceanography*, *35*(6), 1143–1153. Retrieved from http://journals.ametsoc.org/doi/abs/10.1175/JPO2733.1 doi: 10.1175/jpo2733.1

Stammer, D., Wunsch, C., Giering, R., Eckert, C., Heimbach, P., Marotzke, J., … Marshall, J. (2002). Global ocean circulation during 1992–1997, estimated from ocean observations and a general circulation model. *Journal of Geophysical Research: Oceans*, *107*(C9), 1-1–1-27. Retrieved from http://doi.wiley.com/10.1029/2001JC000888 doi: 10.1029/2001jc000888

Stommel, H. (1961). Thermohaline convection with two stable regimes of flow. *Tellus*, *13*(2), 224–230. Retrieved from http://doi.org/10.1111/j.2153-3490.1961.tb00079.x doi: 10.1111/j.2153-3490.1961.tb00079.x

Tarantola, A. (2005). *Inverse problem theory and methods for model parameter estimation*. SIAM.

Tziperman, E., & Ioannou, P. J. (2002). Transient growth and optimal excitation of thermohaline variability. *Journal of Physical Oceanography*, *32*(12), 3427–3435. Retrieved from http://journals.ametsoc.org/doi/abs/10.1175/1520-0485(2002)032%3C3427:TGAOEO%3E2.0.CO%3B2 doi: 10.1175/1520-0485(2002)032<3427:tgaoeo>2.0.co;2

Utkin, I., Chen, Y., Räss, L., & Werder, M. A. (2025). Snapshot and time-dependent inversions of basal sliding using automatic generation of adjoint code on graphics processing units. *Journal of Glaciology*, *71*, e72. doi: 10.1017/jog.2025.40

Vallis, G. K. (2017). *Atmospheric and oceanic fluid dynamics: Fundamentals and large-scale circulation* (2nd ed.). Cambridge: Cambridge University Press. doi: 10.1017/9781107588417

Wagner, G. L., Silvestri, S., Constantinou, N. C., Ramadhan, A., Campin, J.-M., Hill, C., … Ferrari, R. (2025). *High-level, high-resolution ocean modeling at all scales with Oceananigans.* Retrieved from https://arxiv.org/abs/2502.14148

Wunsch, C. (2006). *Discrete inverse and state estimation problems*. Cambridge University Press. Retrieved from https://doi.org/10.1017/CBO9780511535949 doi: 10.1017/CBO9780511535949

Wunsch, C., & Heimbach, P. (2007). Practical global oceanic state estimation. *Physica D: Nonlinear Phenomena*, *230*(1-2), 197–208. Retrieved from https://doi.org/10.1016/j.physd.2006.09.040 doi: 10.1016/j.physd.2006.09.040

Yan, F. E., Frezat, H., Sommer, J. L., Mak, J., & Otness, K. (2025). Adjoint-based online learning of two-layer quasi-geostrophic baroclinic turbulence. *Journal of Advances in Modeling Earth Systems*, *17*(7). doi: 10.1029/2024ms004857

Yatunin, D., Byrne, S., Kawczynski, C., Kandala, S., Bozzola, G., Sridhar, A., … Schneider, T. (2025). The Climate Modeling Alliance Atmosphere Dynamical Core: Concepts, Numerics, and Scaling. *ESS Open Archive*. doi: 10.22541/essoar.173940262.23304403/v1

Yuval, J., O'Gorman, P. A., & Hill, C. N. (2021). Use of neural networks for stable, accurate and physically consistent parameterization of subgrid atmospheric processes with good performance at reduced precision. *Geophysical Research Letters*,

48(6), e2020GL091363.

Zanna, L., & Bolton, T. (2020). Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters*, *47*(17), e2020GL088376. Retrieved 2020-08-28, from https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020GL088376 (_eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2020GL088376) doi: 10.1029/2020GL088376

Zhang, S., Fu, H., Wu, L., Li, Y., Wang, H., Zeng, Y., … Guo, Y. (2020). Optimizing high-resolution Community Earth System Model on a heterogeneous many-core supercomputing platform. *Geoscientific Model Development*, *13*(10), 4809–4829. doi: 10.5194/gmd-13-4809-2020